IP Address Handoff in the MANET

Hongbo Zhou and Matt W. Mutka
Dept. of Computer Science & Engineering
Michigan State University
East Lansing, Michigan, USA
{zhouhon1, mutka}@cse.msu.edu

Lionel M. Ni
Dept. of Computer Science
Hong Kong University of Science & Technology
Hong Kong SAR, China
ni@cs.ust.hk

***Abstract*—When compared with a fixed host that is connected to a hardwired network, a mobile node in the MANET may change its IP address more frequently due to the deployment of autoconfiguration, global connectivity, and hierarchical addressing schemes. When an IP address changes, the performance of unicast routing protocols and real-time communications may degrade, and privacy may be compromised within the MANET. Although there have been some autoconfiguration algorithms proposed for the assignment of unique IP addresses to mobile nodes, the overhead resulting from address changes has not been carefully examined. Based on studies of the overhead caused by address change, an IP address handoff solution, which extends the unicast routing protocol and Network Address Translation (NAT) scheme, is proposed in the paper. The proposed approach is able to offset the overhead of broken routing fabrics and on-going communications, which is supported by our analysis and a prototype implementation.**

Keywords-handoff; IP address allocation; MANET; security

I. INTRODUCTION

A Mobile Ad-hoc Network (MANET) is a temporary wireless network composed of mobile nodes without an infrastructure. A MANET may be suitable for networks within airports, meeting rooms, and open spaces due to both economical and technological feasibilities. Because the MANET is based upon IP protocol suites, a node in the MANET cannot take part in unicast communications until it is configured with a free IP address.

Although it is simple to set IP addresses of mobile nodes in a small scale MANET, it becomes desirable for the procedure to be automatic for a large scale open MANET where mobile nodes are free to join and leave, which has motivated research efforts into the study of autoconfiguration in MANETs [1]-[8].

Another problem associated with IP address assignment of a mobile node is that the IP address may change during its session in the MANET. IP address change is not a serious problem in hardwired networks because the IP address of a host is either statically configured or dynamically allocated by a DHCP server. It usually does not change its IP address during a session unless it reboots. However, because the nodes in the MANET are free to move arbitrarily, IP address change happens more frequently when applied with autoconfiguration, global connectivity, and hierarchical addressing schemes.

There are several scenarios in which a mobile node will change its IP address:

The work was supported in part by NSF Grants No. 0334035, 0098017, 9911074, NIH Grant No. EB002238-01, Hong Kong RGC Grants HKUST6161/03E and DAG02/03.EG02, and a grant from Microsoft Research.

(1) Merger of two partitions of a network

If some mobile nodes in the MANET move out of the transmission range of the other nodes, the network becomes partitioned as illustrated in Fig. 1(a). Because these nodes may not be aware of the partition, they may still use the previous allocated addresses. If the IP address of a node (say node A) in one partition is allocated to the new node (say node B) in the other partition, address conflict occurs when these two partitions become connected, as illustrated in Fig.1 (b). One example is when some attendants leave a meeting room for a short period and then return during a presentation session. The prophet address allocation is insensitive to this scenario [6] [7], while the nodes in one partition may need to change their addresses with DDHCP [5].

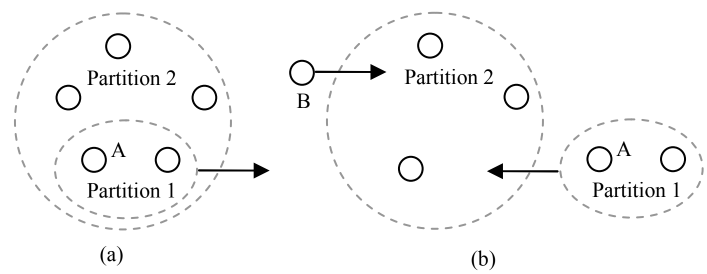


Figure 1. A network is partitioned and then merged later

(2) Merger of two independent MANETs

The second scenario is that two independently configured MANETs merge. Because these two networks are autoconfigured separately, there may be some duplicate addresses in both networks, such as node A in MANET 1 and node B in MANET 2 in Fig. 2. Thus, one needs to change its address due to the merger.

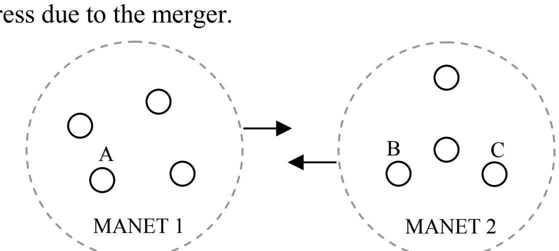


Figure 2. Merger of two independent MANETs

(3) Merger of a MANET with a LAN

The third scenario is that a MANET merges with a LAN that has an "ad-hoc" mode Access Point (AP)[1]. The mobile nodes (such as node R in Fig. 3) that are within transmission range of the AP of the LAN may want to use the configuration information (e.g., a free IP address in the LAN and the default router) broadcast by the AP to configure itself and function as a relay node. As a result, the MANET becomes connected to the Internet [9] [10]. Furthermore, if the MANET and the LAN use the same private address range, there may be duplicate addresses in both the MANET and LAN, such as node A and node B in Fig. 3. Because the hardwired host in the LAN may not be willing to release its address, the node in the MANET will have to change its address.

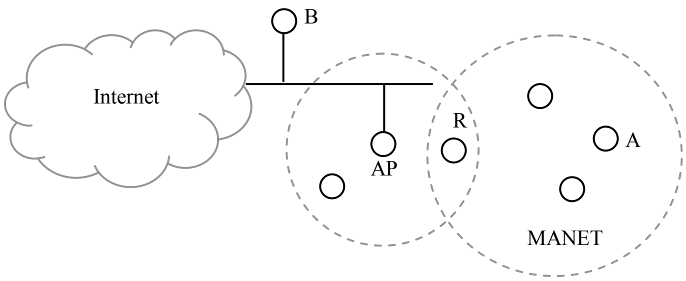


Figure 3. The merger of a MANET and a WLAN

(4) A MANET with a hierarchical addressing scheme

In the network where a hierarchical addressing scheme is deployed [11], the mobile nodes are divided into different clusters, each of which has a unique subnet address. When a mobile node (such as node A in Fig. 4) moves from one cluster to another cluster, it will change its address to one with the corresponding subnet address.

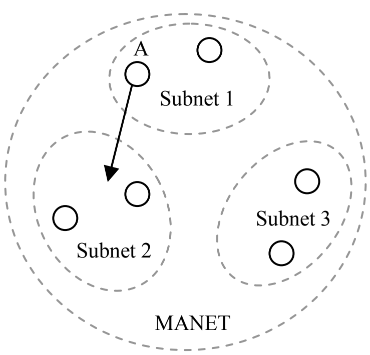


Figure 4. A MANET with hierarchical addressing scheme

Although many autoconfiguration algorithms for MANETs have been proposed to allocate a mobile node a free IP address on its arrival in the MANET without an address conflict, few consider the issue of IP address change. To the best of our knowledge, this paper is the first effort to present a systematic solution for IP address handoff in the MANET.

The paper is structured as follows. Section 2 discusses the overhead caused by IP address change of a mobile node during its session in the MANET. Some related work on handoff schemes is introduced in Section 2 as well. However, none of these solutions can solve the problem systematically and satisfactorily. The solutions to remedy the overhead due to broken routing fabrics and on-going communications are presented in Section 3 and Section 4, respectively. Section 5 gives analytical evaluation of the performance and discusses its limitations, other scenarios and challenges to key management in MANETs. A prototype implementation and test are introduced in Section 6. Section 7 concludes the paper.

II. Issues and Related Work

This section discusses the motivation for IP address handoff and introduces related work.

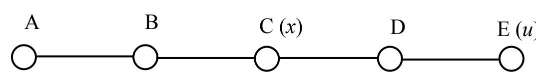
A. Motivation

There are two major issues resulting from IP address change of a mobile node in the MANET:

1) Broken routing fabrics

Unlike the hosts connected to the edge networks, all the nodes in a large-scale MANET have to function as routers (i.e., multi-hop routing). If a node changes its IP address, all the routing entries that point to the node as the downstream next hop will be obsolete.

Fig. 5 gives an example of a simple MANET composed of 5 nodes in a chain with AODV [12][2] as the routing protocol. Suppose that the IP address of node C is x. The routing tables at node B and node D are also shown in Fig. 5. When node C changes its address from x to y, all the routing entries in node B and D will be invalid.



Dest	Next
D	C (x)
E	C (x)

Dest	Next
B	C (x)
A	C (x)

Figure 5. A MANET of 5 nodes in a chain

According to the specification of AODV, after two HELLO message intervals (i.e., 2 seconds), nodes B and D will detect downstream link breakage. If the destination of data packets is within the distance of certain hops, nodes B and D may initiate scoped broadcasts to rebuild the path between B and D; otherwise, they will send a Route Error (RERR) packet back to nodes A and E respectively, which triggers flooding of route rediscovery packets.

In a more complicated topology, node C will have many neighbors and will be on many active paths. Thus, much overhead will incur for route maintenance.

2) Broken on-going communications

While referring to Fig. 5, suppose that node A is communicating with node E. If node E changes its address

[1] For example, the AP from Ericsson is able to support both ad hoc and infrastructure modes simultaneously.

[2] The routing protocols that support multiple paths, such as DSR, are insensitive to the IP address change of a node along the path.

from u to v, the on-going communications between A and E will be broken, which does not meet the requirement of real-time multimedia applications.

Because the address of u will not exist any longer (if the address change is not caused by address conflict), the local repair mechanism of AODV will fail eventually. As a result, the overhead of route rediscovery is inevitable. Even if node A initiates route rediscovery, it will not find the destination, unless the DNS scheme proposed in [13] is combined with the reactive routing protocol.

An ad-hoc approach is that node E resumes the connection actively. However, because node E may be a server of an application (e.g., the user of node E is running a FTP server so that other participants of the meeting can download documents from him), it is not responsible for the initiation of communication. Furthermore, the application may run in the background or the user of node E may not be aware of the broken communication. Thus, we need a better solution.

IP address change may compromise privacy as well. One example is the case of the merger of two MANETs. Suppose that node C is running a VoIP application with node B in MANET 2, as illustrated in Fig. 2. Because node B has the same address as node A in MANET 1, node B will change its address after the merger (for example, MANET 2 has a smaller NID [6] [7]). If node A keeps its address, the route maintenance procedure will rebuild the path between node C and node A. Thus, the voice data packets destined for node B will be redirected to node A. Suppose that node A is also running the VoIP application simultaneously, and that they use the same UDP ports. If the voice data packets are not encrypted, node C will talk with node A for a while until it realizes that it is speaking with a wrong party in the middle of the communication.

B. Related work

Several schemes have been proposed for IP address handoff in hardwired networks and MANETs. However, none of these approaches can solve the issue in the four aforementioned scenarios systematically and satisfactorily.

1) Mobile IP

Mobile IP intends to provide basic support for mobile hosts in a LAN [14]. According to the scheme, a mobile host is assigned a permanent home address that is bound with its home agent. When it becomes connected to a foreign network, it receives a temporary care-of address and other information (e.g., the subnet mask and default router) from the foreign agent. The mobile host registers its current care-of address at its home agent, which then builds a tunnel between itself and the foreign agent. When another host initiates communication with the mobile host, it usually gets the mobile host's home address from DNS query and sends the packets to the home address. The packets will be then forwarded to the mobile host's care-of address by the home agent through the IP tunnel.

Mobile IP is efficient for IP address handoff in a LAN that has an infrastructure. However, because the nodes in the MANET are mobile and instable, none of them can be designated as the home agent or foreign agent for another node. Thus, it cannot be applied in the MANET.

2) Tunneling mechanism

In addition to autoconfiguration in the MANET, the scheme in [8] proposed a solution for the maintenance of communication states after address changes. The node (say node A) that changes its IP address notifies the other end (say node B) with a special Address Error (AERR) message. From then on, they communicate with each other through an IP-in-IP tunnel: the outer IP header contains the A's new address, while the inner IP header contains A's old address. Unlike the IP tunneling in Mobile IP, the communicating nodes A and B are also the end points of the tunnel: the source encapsulates the packet that is decapsulated at the destination.

This approach is able to preserve communication states at both ends, but neglects the overhead caused by broken routing fabrics. Furthermore, it brings a "DoS" problem that will be discussed in Section 4.

III. SOLUTIONS TO BROKEN ROUTING FABRICS

Unlike the link breakage caused by node movement, the overhead of broken routing fabrics is due to IP address change of a node. Because the node may still be within the transmission range of its neighbors, and it is aware of the address change, a simple solution can be implemented to remedy this kind of overhead[3].

We assume AODV as the routing protocol. Suppose that node C changes its address from x to y, as illustrated in Fig. 5. Node C can notify all its neighbors (such as nodes B and E) of the address change. A new routing control packet, namely Route Shift packet, can be introduced into AODV scheme. The packet is a one-hop broadcast packet that contains the source's old address and new address. On receipt of the packet, the neighbors change the next hop from x to y in all the routing entries with node C as the next hop.

However, this solution is vulnerable to IP spoofing attacks, which are difficult to detect and prevent in MANETs that have no infrastructure. With the introduction of the Route Shift packet, a malicious node can impersonate another node and broadcast the packet to undermine the routing fabrics. We need a way to identify the source of the packet.

One solution is to use a cryptographic method such as a digital signature, in which node C signs the Route Shift packet with its private key. All its neighbors contact the Certificate Authority (CA) to get the certificate for node C's public key and validate the Route Shift packet. Although this method can defeat IP spoofing attacks, it brings delay and communication overhead.

Our solution is that node C chooses a random number for its current address x, and puts the hash value of the number in the Route Request packet and Route Reply packet[4] in transit, and periodical HELLO messages. All its neighbors store the

[3] If the address change happens with link breakage simultaneously, the broken routing fabrics will be repaired with route maintenance.

[4] These routing control packets are not destined for node C itself.

hash value in either their neighbor tables or routing tables. When node A changes its address, it puts the random number in the Route Shift packet. Because the packet is a one-hop broadcast packet, which will be received by its neighbors simultaneously, it is not vulnerable to the "man-in-the-middle" attack. If the hash value of the number contained in the Route Shift packet is equal to the stored hash value, the neighbors will be sure of the source of the packet. This method depends on the complexity of the hash function. MD5 [15] is such a good candidate that it is very difficult to determine the number from its hash value.

IV. SOLUTIONS TO BROKEN COMMUNICATIONS

The second type of overhead due to IP address change is the broken communications between the source and destination. We first provide the reasonable assumptions, and then describe the schemes for route rebuilding and communication states preservation.

A. Assumptions

We assume that the IP layer of the mobile node supports more than one IP address. Because we can bind at least two IP addresses with a NIC in most mainstream operating systems (e.g., Unix and Windows), we can assume that it is the same for mobile devices.

With two IP addresses bound to the same interface, we specify that the new address as primary address and the old address as secondary address, and designate that the node use the primary address in the outgoing IP packets. Therefore, the node can still receive the packets destined for the old address for a short period, but the old address will not be used in the following new connections.

In order not to trigger RERR packets sent from the neighbors of the changing node, we also need to extend the HELLO message to contain both the primary and secondary addresses for several intervals. However, to prevent the data packets destined for another node whose primary address is the same as its secondary address to be forwarded to it, the node must not reply to the Route Request (RREQ) packet for its old address.

The second assumption is that the underlying links are bi-directional. Because most MAC layers deployed in MANETs confirm to 802.11 standard, our assumption can be easily satisfied.

B. Route rebuilding

Suppose that node A is communicating with node B and node A changes its address (say, from x to y) during the communication. Although node A can still receive the packets destined for its old address of x for a short period with the mechanism proposed above, we expect that the following communications be based upon the new address of y. However, because the new address has not been seen before, a broadcast of RREQ[5] from the other end is necessary to build the path towards it.

To save the overhead of route rediscovery, we resort to a gratuitous Route Reply (RREP) packet. Because the path to node B may be still valid, node A can send a RREP packet with its new address to node B, which generates valid routing entries backwards to A in the routing tables in the nodes along the path because underlying links are bi-directional (our assumption). The routing entries towards the old address of x will expire eventually.

C. Communication preservation

The most important problem in handoff processing is the preservation of communication states at end points. This is because the checksum in the transport layer is computed based upon the source and destination IP addresses and the transport layer will check the corresponding header in the IP packet against the communication states before delivering the data to the upper layer.

Suppose that node A is communicating with node B, and that node A changes its address from x to y. We adopt an NAT mechanism running on both nodes to preserve the communication states:

(1) At node A, the new destination address of y in the incoming packets is modified to the old address of x prior to delivering it to the transport layer, and the old source address in outgoing packets is modified to the new address before sending it to the link layer;

(2) At node B, the new source address of y in the incoming packets is modified to the old address of x prior to delivering it to the transport layer, and the old destination address in the outgoing packets is modified to the new address before sending it to the link layer.

Although the packets from node A to node B can contain A's old source address of x, which does not affect forwarding policy and the routing fabrics, we still perform NAT on it for purpose of correct reporting of Route Error packets if the path from A to B becomes invalid due to node mobility.

Compared with the tunneling mechanism proposed in [8], our approach has the following advantages:

(1) The overhead of a second IP header is saved. Although the length of the IP header is only 20 bytes in IPv4 or 40 bytes in IPv6 (without any options), it may lead to fragmentation/defragmentation that is time-consuming;

(2) Because only one address in the IP header is modified in NAT, it will be faster when applied with the improved computation of IP checksum [16];

(3) The tunneling scheme brings a "DoS" problem as illustrated in Fig. 6. Suppose that node A has been communicating with node B before node A changes its address from x to y due to the address conflict with node C. An IP tunnel is built between B and A to redirect packets destined for A's old address of x to its new address of y. If node C begins to communicate with B, the packets from B to C will also be forwarded to A through the tunnel, which means node C will never get any replies. Moreover, suppose that node B runs a TCP server application and listens at a well-known port. When node C initiates a connection from the same client port as node

[5] The RREQ may be combined with a name query message.

A, the connection between node A and node B will be reset due to the incorrect sequence number and TCP flags. Although the IP-in-IP tunnel scheme is simple, it cannot solve these problems.

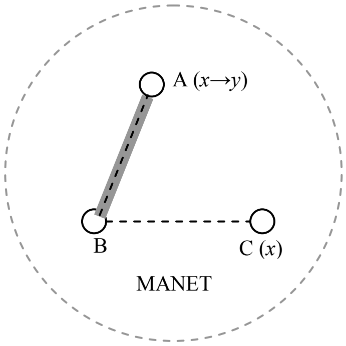


Figure 6. A "DoS" problem caused by IP tunneling

To overcome these problems, our scheme extends NAT to utilize both port numbers and sequence numbers[6] to distinguish different connections at node B, which can be explained with an example below. Suppose that node B is a web server, and that node A is fetching web pages from B when it changes its address from x to y. An NAT table, such as Table 1, is built at node B to store the required information.

TABLE I. NAT TABLE AT NODE B

1	2	3	4	5	6
Old remote address	**New remote address**	**Local port**	**Remote port**	**Remote sequence number**	**Next remote sequence number**
x	y	80	2030	228743	22884312
…	…	…	…	…	…

The procedure at node B works as the following:

(1) When node B receives a packet, before it delivers the packet to the transport layer, it checks the columns 2, 3, and 4 in the NAT table with the corresponding fields in the packet. If there is a match, there are three possibilities: a) the sequence number in the TCP header is equal to field 5 in the entry, which means the packet is either a retransmitted packet from node A or the first packet from node A since the entry was inserted into the NAT table. The new source address is modified to be the old address in field 1, and the next remote sequence number column is the sum of the remote sequence number and the payload length; b) if the sequence number in the packet is equal to field 6 in the entry, which means the packet is a new packet from node A, the new address in the packet is modified to be the old address, the value in field 6 is copied to filed 5, and field 6 is increased by the payload length; c) if the sequence number is not equal to either, which means the packet comes from node C that has x as its primary address, the packet is discarded silently. In all the other cases, the packet is delivered to the upper layer intact.

(2) When node B has a packet for the destination address of x, before the packet is sent to the link layer, node B compares columns 1, 3, 4 and 6 in the NAT table and the corresponding fields in the packet to find a match[7]. If there is such an entry, as the one in Table 1, the old destination address of x is changed to the new address of y, together with re-computation of the IP checksum; otherwise, the packet is sent intact.

For UDP communications, because there is no sequence number in the UDP header, only the first four fields in the NAT table are used.

Compared with node B, the processing at node A is simpler. Another NAT table, such as Table 2 is used. If the local port number in the packet is equal to one in the table, the old address is modified to the new address in the outgoing packets, and vice versa for incoming packets.

TABLE II. NAT TABLE AT NODE A

Old address	New address	Port number
x	y	2030
x	y	…

Because node A changes its own address, it is trivial for it to insert the entry when it has a packet to send. The problem remains that how the entry is inserted at node B's NAT table. A special message, Address Change Message (ACM), can be sent from node A to B indicating the creation of the NAT entry, which includes the old address, new address, protocol (UDP or TCP), local port, remote port, and sequence number (TCP only). The next remote sequence number in B's NAT table is initialized as zero, and will be filled by the following TCP data packets. To save communication overhead further, the message can be combined with the gratuitous RREP packet mentioned in route rebuilding.

The ACM packet must be sent before any data packets for node B. If node A is going to send a packet immediately after the address change, the data packet can be buffered before sending of the ACM packet. If node A has nothing to send immediately after address change, it waits for data packets from node B. Although node B has not been informed of A's address change, according to our assumption, the route from node B to node A with the old address will be valid for a short period. Thus, the data packet can still arrive at node A, which triggers an ACM packet sent to B.

To remove the entry, the TCP flag of FIN can be examined. For UDP entries, an expiration time can be associated with each of them. The entry is refreshed with UDP data packets and removed when it expires. The timeout method could also be utilized for TCP entries in case that the FIN packet is lost on transit or the other end shuts down abruptly.

To prevent IP spoofing attacks, the ACM packet must be signed with node A's private key. Therefore, the contents can be validated with A's public key at node B.

[6] Only a TCP header has a sequence number. For a UDP header, the sequence number can be regarded as 0, which is meaningless.

[7] The acknowledge number in the packet is compared with the next remote sequence number in the NAT table.

V. Performance Evaluation and Discussion

This section analyzes the performance, limitations, other scenarios, and challenges of the handoff scheme.

A. Performance analysis

Suppose that there are n nodes in the MANET with l links. The average degree of a node is d (= l/n). Node A, which changes its IP address, has k connections with m nodes and on p active paths. The average length of the path is q. In a MANET where the mobile nodes are distributed evenly, we can assume that $l >> n >> q$, $n >> d$, and $n > m$;

As analyzed before, there are two kinds of overheads:

1) Overhead of broken routing fabrics

Without the handoff scheme, the active paths going through node A will become invalid. As a result, at least one end of each path will initiate route rediscovery if the local repair mechanism is not utilized, which means there will be at least p times of flooding throughout the MANET. If no flooding optimization is adopted, the same packet will be forwarded by all the nodes once. Thus, there will be $2l$ packets for one flooding. As a result, there will be at least $2pl$ packets caused by broken routing fabrics.

With the introduction of the Route Shift packet, which is a one-hop broadcast packet, only d packets are generated.

2) Overhead of broken communications

Without the handoff scheme, the m nodes that communicate with node A will perform route rediscovery. Therefore, $2ml$ packets will be generated to find the path towards node A.

With the introduction of a gratuitous RREP packet and the ACM packet, only $(m+k)q$ unicast packets are necessary to rebuild the route and insert NAT table entries. If RREP and ACM are combined, the number of packets is decreased to kq.

B. Limitations

If node A wants to communicate with node C as in Fig. 6, because C's primary address of x has been bound with A's NIC as the secondary address, all the packets from A to C will be regarded as local packets. Thus, they cannot communicate with each other until node A does not use the secondary address any longer.

As discussed in the section above, to distinguish TCP connections from node A (that changes its address) and node C (that has the same primary address), node B utilizes both port numbers and sequence number in the TCP header. If node C connects to node B's well-known port from the same port number after node A changes its address, the connection will be rejected. If the client port is dynamically allocated, as most application programs do, node A and node C may have different local ports.

On the contrary, node B may connect to C's well-known port after it has connected with A's same port (for example, node B is browsing A and Cs' homepages at the same time) and A has changed its address. If the client port is dynamically allocated, B will have different local port numbers, which will not be affected by our scheme. If node B binds the local port in the program, because the acknowledgement number in the connection with node A is usually different from the one in the connection with node C, the communication will not be affected either. However, if the acknowledgement number is the same, its NAT module will redirect the connection.

With regard to UDP communication, because there is no sequence number in the UDP header, if nodes A and C use the same local client port number, the packets for C will be redirected to A. Furthermore, because there are no connection flags, UDP communication between nodes A and B will be forced to break if there has not been any data exchange for a long period after creation of the NAT entry.

Another limitation is that if there is no data exchange between node A and B for a while immediately after A changes its address, the path from B to A will be invalid due to node mobility. Thus, a route rediscovery combined with name query from node B is inevitable. However, this problem can be solved if node A actively sends an ACM packet for each pre-existing connection.

C. Multiple address changes

Our scheme can be applied to multiple address changes without any significant modifications.

Suppose that the source node changes its IP address twice during the session, without loss of generality. After the first address change, the NAT tables have been built at both the source and destination to change the secondary address in the data packets to primary address and back forth. When the second address change happens, because the IP layer supports more than one IP address, the old primary address can be "pushed back" to be the second secondary address.

The solution for broken routing fabrics can be utilized as usual. As to broken on-going communications, if the communication was initiated with the second secondary address (between the first and the second address change), new NAT entries will be created. However, for the communications initiated with the first secondary address (before the first address change), it must be associated with the up-to-date primary address. To solve this problem, noting that the primary address is mainly used for purpose of routing, we just need to modify the new address field of corresponding entries at both the source and destination.

Thus, for all the existing NAT entries at the source node, when a second address change happens, a special flag can be set for each entry in addition to modification of new address field (Table 2). When an outgoing data packet has a matching local port number, an ACM message is sent to the destination, and the flag can be cleared. On receipt of this second ACM message, the destination modifies its NAT table to reflect the change.

D. Address changes at both source and destination

If both of the source and destination change their IP addresses simultaneously, because their old addresses are bound with them as secondary addresses, the ACM messages or data packets from either node with the old destination addresses will still reach the other end successfully within a short time interval.

However, once the ACM message from the other end is received, a second NAT table is created, in addition to the first table caused by its own IP address change, as the tables illustrated both in Table 1 and in Table 2. Therefore, every data packet will undergo NAT twice[8]:

(1) The outgoing data packets will be performed source NAT with Table 2, and then destination NAT with Table 1;

(2) The incoming data packets will be performed source NAT with Table 1, and then destination NAT with Table 2.

Thus, the communication states at both ends could be preserved in spite of both address changes.

E. Challenge to key management

Address handoff brings challenges to the key management in MANETs because most existing schemes assume that the IP address of a node is fixed, and thus a public/private key pair is bound with an IP address [17] [18]. When applied with autoconfiguration, their assumptions will be invalid for the following two reasons:

(1) The addresses of the nodes are dynamically assigned when they join the MANET.

(2) A node may change its address during its session in the MANET due to the reasons mentioned in Section 1.

Suppose there is a CA in the MANET (either centralized or distributed). When the mobile node joins the network and is assigned with an IP address, it must register the binding of its public key with its current IP address with the CA. When it changes its address, it needs to register itself again with the CA to get a new certificate. However, since the binding of its previous address and its public key may not expire at the CA, its request will be rejected by the CA even it is really the owner of the public key, because it seems that two different nodes have the same public key in the eyes of the CA.

The approach proposed in Section 3 will solve this problem: we can specify that the node generate another random number every time when it is assigned an IP address. If the length of the random number is long enough, two nodes will have different random numbers. During the process of registration, the hash value of the random number is included in the messages. When the node changes its address, the random value associated with the previous address is included in the registration message and encrypted with the other end's public key or the secret key that they have agreed upon. Thus, the source of the messages can be identified.

VI. PROTOTYPE IMPLEMENTATION

A prototype of the handoff scheme is implemented to test the preservation of communication states in a LAN, as illustrated in Fig. 7. After a TCP connection was built between the client of a laptop and the server of a desktop, the client changes its IP address. Although the client is working in the infrastructure mode, it should be the same in the ad-hoc mode. The application is a simple string echo program, in which the server echoes the string typed at the client's terminal.

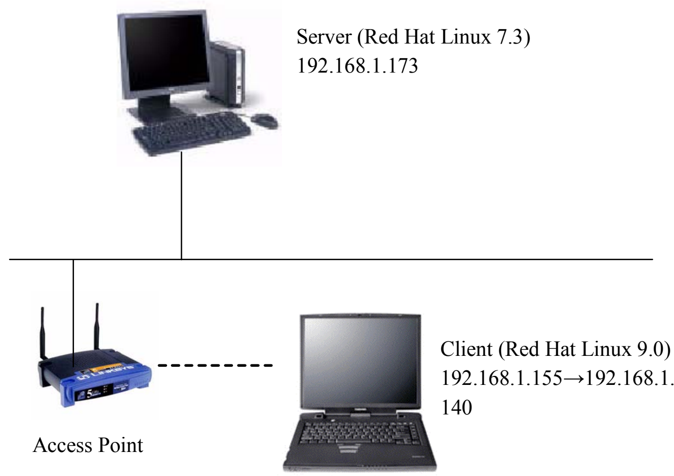


Figure 7. The testbed of handoff scheme

The handoff scheme is implemented as hooks by means of netfilter [19]. The NAT processing of outgoing packets is performed at NF_IP_LOCAL_OUT, while the NAT processing of incoming packets is performed at NF_IP_PRE_ROUTING.

The code illustrated in Fig. 8 shows the procedure of NAT processing of outgoing packets at the client. The processing of incoming packets is similar. To simplify the test, we did not use the aforementioned NAT tables in the prototype because there is only one TCP connection. In a real implementation, a hash table may be utilized to expedite lookup of NAT tables. Other optimizations are also desirable. For example, IP addresses and port numbers should be stored in network order, and the fast computation of IP checksum should be used.

```
#define OLD_ADDRESS 0xC0A8019B  // 192.168.1.155
#define NEW_ADDRESS 0xC0A8018C // 192.168.1.140
static unsigned int handoff_NAT_out(unsigned int hook, struct sk_buff
**pskb, const struct net_device *indev, const struct net_device *outdev, int
(*okfn)(struct sk_buff *)
{
   struct tcphdr* th;
   // Whether we should perform NAT or not
   if ((*pskb)->nh.iph->saddr == htonl(OLD_ADDRESS) && (*pskb)-
>nh.iph->protocol == 6)
    {
      th = (struct tcphdr*)((char*)(*pskb)->nh.iph + (*pskb)->nh.iph->ihl*4);
      if (th->dest == htons(10000))
      {
        // Change (source) address
        (*pskb)->nh.iph->saddr = (DWORD)htonl(NEW_ADDRESS);
        // Recompute IP checksum
        (*pskb)->nh.iph->check = 0;
        (*pskb)->nh.iph->check = in_checksum((WORD*)((*pskb)->nh.iph),
(*pskb)->nh.iph->ihl*4);
      }
   }
   return NF_ACCEPT;
}
```

Figure 8. NAT processing of outgoing packets at changing node

The test is done in the following steps:

[8] The order of source NAT and destination NAT is not important.

(1) The client initiates a connection to the server;

(2) After a while, the IP address of the client is changed from 192.168.1.155 to 192.168.1.140 at the wireless NIC (eth1);

(3) The old address of 192.168.1.155 is bound with the client's wireless NIC as an alias (eth1:0)[9];

(4) Install hooks at both the client and server;

(5) Continue typing strings at the client's terminal, which are shown on the server's terminal.

We used ethereal running on the client to capture all the TCP packets during the test, which are shown in Fig. 9. In Fig. 9, the first three lines are the control packets for TCP handshaking procedure between the client (192.168.1.155) and the server (192.168.1.173). The lines 4-7 are the TCP data packets before the client's address change. Starting from line 8, the client's address is changed from 192.168.1.155 to 192.168.1.140, so the source address of all the following outgoing packets is the new address. However, for the incoming data packets, because the new destination address has been modified by the hook, so the destination address of the captured incoming packets is still shown as the old address. With ethereal running on the server side, the outgoing packets' destination address field still contains the new address of the client.

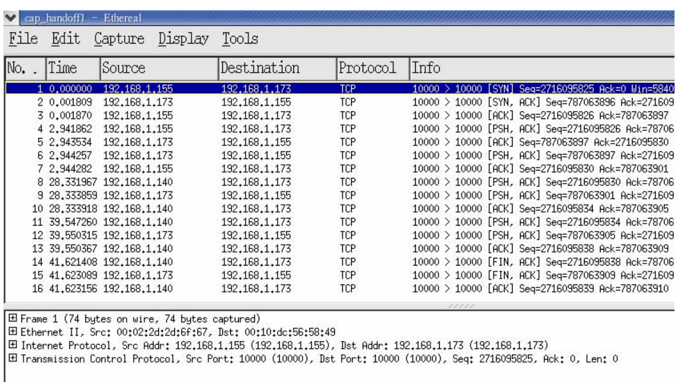


Figure 9. The captured TCP packets during the test

VII. CONCLUSION

With the deployment of autoconfiguration, global connectivity, and hierarchical addressing scheme (in large scale networks), mobile nodes in MANETs may change their IP addresses more frequently than fixed hosts in hardwired networks. However, the issue of address change has not been carefully studied. Based upon the analysis of the overhead caused by address change, we introduced the Route Shift packet, Address Change Message, and NAT scheme to solve the problem. Thus, the overhead caused by broken routing fabrics is saved. Moreover, the both end can continue their TCP/UDP communications, unaware of the address change. The method could be applied to the scenario that the node changes its IP address more that once, and that both the source and destination change their addresses simultaneously, without any significant modifications.

There is still much work ahead. For example, only a quantitative analysis was presented in the paper. The simulation of the schemes needs more research effort. Another aspect is that the prototype implementation only aims at one node's address change during one connection. The tests in the scenarios of more connections and more address changes have not been conducted yet, which will be our future work.

[9] This step is necessary because otherwise the outgoing packets will be dropped at the client's IP layer.

REFERENCES

[1] C. Perkins, J. Malinen, R. Wakikawa, E. Belding-Royer, and Y. Sun, "IP address autoconfiguration for ad hoc networks," draft-ietf-manet-autoconf-01.txt, November 2001 (work in progress)

[2] A. Misra, S. Das, A. McAuley, and S. K. Das, "Autoconfiguration, registration, and mobility management for pervasive computing," IEEE Personal Communication System Magazine, Vol. 8, pp. 24-31, August 2001

[3] K. Weniger and M. Zitterbart, "IPv6 autoconfiguration in large scale mobile ad-hoc networks," In Proceedings of European Wireless 2002, Florence, Italy, February 2002

[4] N. Vaidya, "Duplicate address detection in mobile ad hoc networks," Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'02), Lausanne, Switzerland, June 2002

[5] S. Nesargi and R. Prakash, "MANETconf: configuration of hosts in a mobile ad hoc network," In Proceedings of the 21st Annual Joint Conference of IEEE Computer and Communication Societies (INFOCOM 2002), New York, NY, June 2002

[6] H. Zhou, L. M. Ni, and M. W. Mutka, "Prophet address allocation for large scale MANETs," In Proceedings of the 22nd Annual Joint Conference of IEEE Computer and Communication Societies (INFOCOM 2003), San Francisco, CA, April 2003

[7] H. Zhou, L. M. Ni, and M. W. Mutka, "Prophet address allocation for large scale MANETs," Ad Hoc Networks Journal, Vol. 1, Issue 4, pp 423-434, November 2003

[8] J.-H. Jeong, H.-W. Cha, J.-S. Park, and H.-J. Kim, "Ad hoc IP address autoconfiguration," draft-jeong-adhoc-ip-addr-autoconf-00.txt, May 2003 (work in progress)

[9] R. Wakikawa, J. T. Malinen, C. E. Perkins, A. Nilsson, and A. J. Tuominen, "Global connectivity for IPv6 mobile ad hoc Networks," draft-wakikawa-manet-globalv6-02.txt, November 2002 (work in progress)

[10] E. M. Belding-Royer, Y. Sun, and C. E. Perkins, "Global connectivity for IPv4 mobile ad hoc networks," draft-royer-manet-globalv4-00.txt, November 2001 (work in progress)

[11] G. Pei and M. Gerla, "Mobility management for hierarchical wireless networks," Mobile Networks and Application (MONET), Vol. 6, No. 4, pp 331-337, August 2001

[12] C. Perkins, E. M. Belding-Royer, and S. R. Das, "Ad hoc on-demand distance vector (AODV) Routing," Network Working Group RFC 3561, July 2003

[13] P. Engelstad and G. Egeland, "Name resolution in on-demand MANETS and external IP networks," draft-engelstad-manet-name-resoltuion-00.txt, February 2003 (work in progress)

[14] C. Perkins (editor), "IP mobility support," Network Working Group RFC 2002, October 1996

[15] R. Rivest, "MD5 message-digest algorithm," Network Working Group RFC 1321, April 1992

[16] J. Touch and B. Parham, "Computation of the Internet checksum via incremental update," Network Working Group RFC 1624, May 1994

[17] L. Zhou and Z. J. Haas, "Securing ad hoc networks," IEEE Network, Vol. 13, No. 6, pp. 24-30, 1999

[18] S. Čapkun, L. Buttyán, and J. P. Hubaux, "Self-organized public-key management for ad hoc networks," In IEEE Transactions on Mobile Computing, Vol.2, No. 1, January-March 2003

[19] R. Russell and H. Welte, "Linux netfilter hacking HOWTO," http://www.netfilter.org/documentation, July 2002