Location Privacy in Mobile Internetworking

Alberto Escudero Pascual <alberto@it.kth.se>
Martin Hedenfalk <e97_mhe@e.kth.se>
Per Heselius <d97-phe@nada.kth.se>

Royal Institute of Technology (KTH)
Stockholm, Sweden

May 17, 2001

Abstract

The Freedom System is a pseudonymous IP network that provides privacy protection by hiding the user's real IP addresses, email addresses, and other personal identifying information from communication partners and eavesdroppers. The following paper describes a set of protocol extensions to the Freedom System architecture to permit a *mobile node* to seamlessly roam among IP subnetworks and media types while remaining untraceable and pseudonymous.

These extensions make it possible to support transparency above the IP layer, including the maintenance of active TCP connections and UDP port bindings in the same way that MobileIP$_{v4}$ does but with the addition that the home and foreign network are unlinkable. We call this extension the *Flying Freedom System.*

Contents

List of Figures

List of Tables

1 Introduction

There are several important issues regarding security in wireless networks. As in all computer communications, these include message integrity, authentication, and confidentiality. Message integrity means that the message is transmitted without alteration, authentication means that the sending/receiving user is the one he claims to be, and confidentiality means that no one other than the intended party is able to read the transmitted message. In wireless networks, where users move between different networks and media types, another issue becomes equally important: **location privacy**. Location-aware services take advantage of the user's or terminal's location information, but what happens if the user doesn't want to be located? This means that it should be impossible to locate where a mobile user is currently working, if he/she so desires. [1]

In cellular mobile systems, such as GSM/GPRS or UMTS, it is also possible to locate users based on the cell they are in or in some cases even where within the cell the user is. In the future, a customer may choose whether this should be possible or not. Service providers could offer location privacy services as an add-on service for their customers.

1.1 Location privacy while seamlessly roaming

This paper presents a set of protocol extensions to the Freedom System [8] which provides similar functionality as in MobileIP$_{v4}$ [2] and also includes location privacy [5]. The Freedom System has been developed by the Canadian company Zero Knowledge Systems Inc.

MobileIP allows users to move between different networks, while maintaining the same IP address. This is done by associating a care-of-address with the mobile node when it is away from home. All traffic to the mobile node is intercepted in the home network by a home agent that tunnels the data to the care-of-address.

When providing location privacy to the mobile node we need to ensure that:

- The home network should have no knowledge about which foreign network the mobile node is currently connected to.
- Similarly, the foreign or "roaming" network should have no knowledge about the mobile node's home network.
- An eavesdropper or man-in-the-middle should not be able to tell who the communicating parties are.
- In addition, all the usual communication security constraints must apply; ie message integrity, authentication and confidentiality.

2 A Pseudonymous IP Network: Freedom overview

This section is a quick overview of the Freedom System architecture and has been written with the intention of providing sufficient information to understand our protocol extensions to the Freedom System. For a detailed look at the entities, systems and protocols that make up the Freedom System we refer the reader to the Freedom Network architecture white papers [7,8,9].

The Freedom System is a *Pseudonymous IP*, or *PIP*, network [10]. The *PIP* network provides privacy protection by hiding the user's real IP addresses, email addresses and other personal identifying information from communication partners and eavesdroppers.

The Freedom System makes it possible for a user to access the Internet without revealing any location or personal information, through the use of so called *Nyms*. The user connects to the Internet via the Freedom System that encrypts the traffic and reroutes it through special servers. Which servers to be used in the routing is determined by the user before the connection is established. Each server only knows the next and the previous server on the route. This way a third person eavesdropping the channel can't find out the source and destination of the connection. Since all traffic is encrypted, the content is not visible to anyone else.

The Freedom System could be seen as an overlay network composed of globally distributed servers that runs on top of the Internet. Freedom routers or **Anonymous Internet Proxies** AIP_is are the core network privacy daemons and they are in charge of passing encapsulated packets between themselves until they reach an exit node AIP_{exit} or AIP wormhole. When a certain AIP_i runs as an AIP_{exit}, it works as a traditional network address translator, NAT.

Symmetric link encryption is applied between node pairs (AIP to AIP $\{AIP_i - AIP_{i+1}\}$ and Freedom client to entry AIP $\{FC_j - AIP_1\}$) to hide the nature and characteristics of the traffic between them.

When a Freedom client with IP address IP_{FC_j} communicates with a correspondent node CN_m via a previously built virtual circuit VC_x in the Freedom System, the correspondent node sees that the traffic as coming from the wormhole IP address $IP_{AIP_{exit}}$ instead of the client's real IP address.

The client creates a virtual circuit inside the Freedom System by sending a route creation packet which contains secrets S_N[1] to be shared with each AIP_i in the chosen chain. The route create packet uses *Nested ElGamal encryption* to securely transmit the shared secrets and to ensure that each AIP_i can only read the part of the *route create packet* destined for itself. Hence, AIP_i only knows the previous AIP_{i-1} and the next AIP_{i+1} in the chain as given in the *route create packet*. The Nested ElGamal encryption is performed using the AIPs' public keys $K_{publicAIP_i}$.

[1] The S_N, named *preKeySeed*, is a *key seed* that is used to generate keys for the three symmetric algorithms (routeCrypt, bckSymAlg and fwdSymAlg) [8].

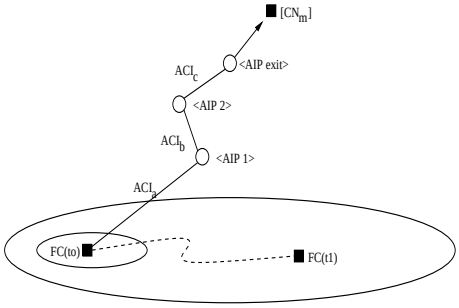

Figure 1: Freedom Overview

The set of encryption layers (multilayer nested encryption) is called "telescope encryption" and it is used to provide *"Freedom client-to-wormhole"* confidentiality to both route creation and data packets.

Once the route VC_x is created from the Freedom client to the wormhole AIP_{exit}, the data packets travel towards the wormhole over the virtual circuit, being link decrypted, telescope unwrapped and finally link encrypted at each point.

The data is routed to the next hop by use of an **Anonymous Circuit Identifier** (ACI) mapping table. The ACIs indicate, along with a packet's source address and port, the next hop in a particular route.

Data coming in over a given $[IP_{AIP_i}, Port_{AIP_i}, ACI_k]$ is first link decrypted and then telescope encrypted with the key generated from S_N. Finally the data is link encrypted and sent on its way to $IP_{AIP_{i+1}}$ with a rewritten ACI value, ACI_{k+1}.

When the ACI_k from the incoming data packet indicates that the packets from that entity need to be sent to the wormhole, the AIP_{exit} acts as a NAT_x for that connection. In this case, the wormhole will map the ACI value ($ACI_k = ACI_{exit}$) with a TCP (or UDP) local port.

2.1 Freedom virtual circuit example.

Let us consider a Freedom client FC_j that wants to communicate with a correspondent node CN_m. The FC_j chooses a set of three AIP_i from the globally distributed Freedom AIPs $\{AIP_1 - AIP_2 - AIP_{exit}\}$. The chosen chain establishes one virtual circuit VC_x between the Freedom client and the wormhole AIP_{exit}.

The Freedom client negotiates a link encryption key with $AIP_1 = AIP_{entry}$ for the $\{FC_j - AIP_1\}$ link.

During the route creation process each AIP_i receives from the FC_j a unique shared secret $S_N = preKeySeed_N$ for that session. The shared secret is mapped to the ACI_k field of the incoming route create packet.

It is also during the route creation when each AIP_i is responsible for choosing a random locally unique ACI_{k+1} that will be used to send packets to the next AIP_{i+1}. The first ACI_1 in the virtual circuit chain is selected by the FC_j.

In figure 1 we can see that:

Table 1: Mappings in AIP_{entry} before and after a handover.

STATE	from	to
$Before$	$FC_j[IP, Port](t_0)$ - $ACI_1(t_0)$	AIP_2 - ACI_2
$After$	$FC_j[IP, Port](t_1)$ - $ACI_1(t_1)$	AIP_2 - ACI_2

- FC_j chooses ACI_a and shared secret S_A to communicate with the Freedom entry AIP, AIP_1.
- After applying link decryption using the previously negotiated key with FC_j; AIP_1 knows that packets coming from IP_{FC_j} address with ACI_a can be decrypted with the key generated from shared secret S_A[2] and have to be link encrypted and forwarded to AIP_2 with rewritten ACI value, ACI_b.
- In the same way AIP_2 (after link decryption) uses S_B to decrypt packets coming from AIP_1 with ACI_b and link encrypts and forwards them to AIP_{exit} with ACI_c.
- After link decryption in AIP_{exit} the last layer of the *telescope encryption* is removed using the key derived from the shared secret S_C. AIP_{exit} also maps the packets coming from AIP_2 with ACI_c and certain port number to a local non routable IP address that will act as the source of a NAT_x session.

3 Protocol extensions to the Freedom System

Our protocol extensions to the Freedom System can be divided into two types. The first type concerns location privacy when the mobile node is run only as a client, ie the mobile node is only making outbound connections *(mobile client location privacy)*. The second set of extensions concerns location privacy when the mobile node also wants to act as a server accepting inbound connections from correspondent hosts *(mobile server location privacy)*.

We have identified three different subcases for mobile client location privacy:

- **Case 0** (full route create): The mobile node sends a new **ROUTE CREATE** message after changing its point of attachment rebuilding the whole virtual circuit but keeping the same AIP_{exit} and ACI_{exit}, ie to preserve TCP connections and UDP port bindings.
- **Case 1** (partial route creating preserving AIP_{entry}): The mobile node sends a **ROUTE**

[2]The S_A is used as key seed material to generate the key for the algorithm *(fwdSymAlg)* that is used to decrypt the corresponding encryption layer when data travels towards the AIP_{exit}.

CREATE$^{v.3}$ message[3] to the AIP_{entry} that updates the partial route. The information in the *route create packet* is used to renew the $[IP_{FC_j}(t), Port_{FC_j}(t), ACI_1(t)]$ parameters while preserving the mapping with $[IP_{AIP_2}, Port_{AIP_2}, ACI_2]$. If we represent the stages before and after a handover with t_o and t_1, then the ACI mappings in an entry AIP (AIP_{entry}) are represented as: [table 1].

- **Case 2** (partial route creating non-preserving AIP_{entry}): The mobile node sends a **ROUTE CREATE**$^{v.3}$ message upwards in the hierarchy of AIPs until the message reaches the "switching" AIP. All the routes under the switching AIP are updated preserving the higher part of the hierarchy [6].

When talking about mobile server location privacy:

- **Case 3**: The mobile server is reachable at IP_{FS_j} and $Port_{FS_j}$ allocated in a chosen AIP_{exit}, but the care-of-address of the server is not known by any AIP_i. In this case the mobile server registers an IP_{FS_j} and $Port_{FS_j}$ served by some AIP_{exit} in the Freedom System. When packets arrive to that IP and port, the data travels back over the network, the information is passed along the route indicated by the ACI_k until it reaches the AIP_{entry} and link encrypted to the care-of-address of the mobile server $IP(coa)_{FS_j}$. The data is transported from the AIP_{exit} to the client in the same way as data packets are transported in the normal mode of operation, ie the data packets are link decrypted, telescope encrypted and finally link encrypted in each AIP_i.

3.1 Mobile client location privacy

One of the possible scenarios looks as follows: A Freedom client FC_j running an IEEE 802.11 wireless interface $IP_{FC_j}(t_0)_{WLAN}$ (while communicating with correspondent node CN_m) is moving from an indoor environment to an outdoor GPRS wireless network. A "vertical handover" is performed from one media type to another and the *mobile node* obtains a new IP address $IP_{FC_j}(t_1)_{GPRS}$ from the GPRS network.

The correspondent node is not aware of the mobility of the *mobile node* and furthermore the AIP_{exit} is also not aware of which foreign networks the mobile node is roaming in. The AIP_{exit} acts as a MobileIP$_{v4}$ home agent for the Freedom client and the AIP_{entry} acts as a MobileIP$_{v4}$ foreign agent. The AIP_{entry} and the AIP_{exit} are unlinkable [4].

3.1.1 Case 0: Full route create

The option for a full recreation of a route maintaining the same TCP/UDP bindings is a present feature in the Freedom 2.x architecture [Fig.2] . The *specACI* field in the **ROUTE CREATE**$^{v.2}$ packet allows the Freedom client to specify the ACI_{exit} that the wormhole AIP_{exit} should use so that a route can be extended or changed using the same exit hop. This feature allows a Freedom client to dynamically add a new AIP in the chain, preserving the previously allocated ACI_{exit} - NAT_x mapping.

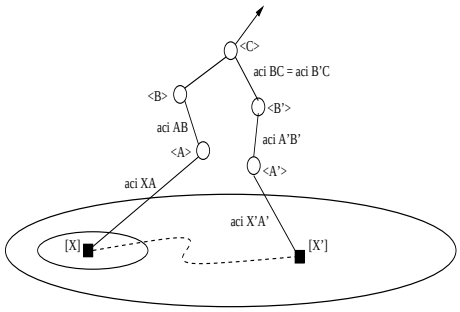

Figure 2: Case 0: ROUTE CREATE $AIP_{exit} = AIP_{switch}$

A successful route creation is completed when the AIP_{exit} checks and validates the **Nym signature**. All packets received by AIP_{exit} from AIP_{exit-1} with a certain source port number and an ACI value (ACI_{exit}) are mapped to a local non routable IP address that will act as the source of a NAT_x session.

This reserved ACI_{exit} in the AIP_{exit} is used to identify the socket used to communicate through TCP/UDP with the corresponding host CN_m. The allocated ACI_{exit} is sent back in the **ROUTE CREATE ACK**$^{v.2}$ answer in response to the client's initial **ROUTE CREATE**$^{v.2}$ message. A **ROUTE CREATE ACK**$^{v.2}$ packet is just a data packet with the data packet type field set to a special value. The payload carries the 2 byte ACI_{exit} number allocated in the AIP_{exit} for that session.

When the client wants to change its point of attachment, it sends a new **ROUTE CREATE**$^{v.2}$ message using the *specACI* field that is set to the $ACI_{exit}(t_0)$. The $ACI_{exit}(t_0)$ is acquired from the **ROUTE CREATE ACK**$^{v.2}$ message from the previous route creation. This way the TCP connection and UDP port bindings between the AIP_{exit} and the CN_m are preserved, and thus the connections between the mobile client and the corresponding host.

The whole route is rebuilt between the AIP_{entry} and AIP_{exit}, even where those routes are unchanged. The **Nym signature** is also rechecked at the AIP_{exit}.

From the point of view of all applications running on the Freedom client the connection looks unchanged though the client IP address has changed ($IP_{FC_j}(t_0) \neq IP_{FC_j}(t_1)$).

3.1.2 Case 1: Route creating a preserving AIP_{entry}

This is the first of the proposed extensions of the Freedom System, to be able to change the point of attachment, while preserving TCP/UDP connections, but

[3]The ROUTE CREATE and ROUTE CREATE ACK messages described in case 1-3 are not supported in current Freedom 2.x and they are part of our protocol extensions proposal.

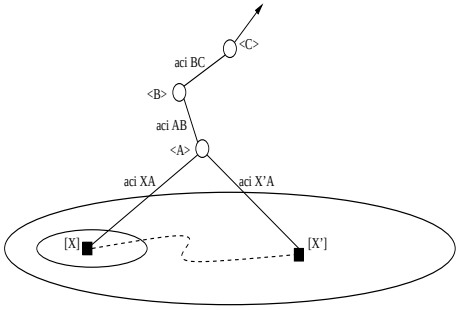

Figure 3: Case 1: ROUTE CREATE $AIP_{entry} = AIP_{switch}$

without rebuilding the whole route [Fig 3]. The Freedom client gets a new IP address $IP_{FC_j}(t_1)$ (perhaps due to a move to another network) but uses the same $AIP_{entry}(t_0) = AIP_{entry}(t_1)$.

As shown in *case 0* the current solution requires that the whole route is rebuilt (except for the socket binding in the AIP_{exit}) and that the **Nym signature** is rechecked.

Case 1 presents an alternative when the mobile node wants to keep using the same entry AIP ($AIP_{entry}(t_0) = AIP_{entry}(t_1)$). In this case, the whole route does not need to be rebuilt.

In order to update the route binding for the mobile node, the AIP_{entry} needs to be notified that:

- the Freedom client has a new IP address $IP_{FC_j}(t_1)$.
- the Freedom client already has had a route binding for the old IP address $[IP_{FC_j}(t_0), Port_{FC_j}(t_0), ACI_1(t_0)]$.

The mobile node first has to exchange a new shared secret with the entry AIP ($AIP_{entry}(t_1)$) to be able to establish new link encryption between the Freedom client and the entry AIP. $\{FC_j(t_1) - AIP_{entry}(t_1)\}$.

The mobile node then sends a **ROUTE CREATE**$^{v.3}$ message, as described in Appendix A, that contains the old IP address $IP_{FC_j}(t_0)$, old port $Port_{FC_j}(t_0)$, old $PreKeySeed_{AIP_{entry}}(t_0)$ and old ACI $ACI_1(t_0)$. The AIP_{entry} then checks the authenticity of the message by checking that the $PreKeySeed_{AIP_{entry}}(t_0)$ sent with the update is the same as the one that was previously exchanged between the client and entry AIP. ($IP_{FC_j}(t_0)$ with $ACI_1(t_0)$). If the message is verified to be correct, it then updates its route binding (uniquely identified with the $[IP_{FC_j}(t_0), Port_{FC_j}(t_0), ACI_1(t_0)]$) with the new $[IP_{FC_j}(t_1), Port_{FC_j}(t_1), ACI_1(t_1)]$ which is extracted from the **ROUTE CREATE**$^{v.3}$ header, see [table 1]

3.1.3 Case 2: Route creating a non-preserving AIP_{entry}

To generalize *case 1*, we introduce the concept of a "switching AIP", AIP_{switch} [Fig. 4].

When the mobile node changes its point of attachment (IP address), it may not want to use the same AIP_{entry}. For example, it may be impossible to use the same AIP_{entry} because it resides in a private network.

However, some $AIPs$ in the route can be the same, so the minimum route that needs to be rebuilt, is the partial route upwards to the first common AIP (AIP_{switch}).

If the mobile node selects $AIP_{switch} = AIP_{entry}$, this would behave as in *case 1*. If $AIP_{switch} = AIP_{exit}$, this case would behave as in *case 0* [table 2].

In any case the mobile node first has to perform a new key exchange with the new $AIP_{entry}(t_1)$ to be able to establish link encryption between the client with the new IP address and the entry AIP $\{FC_j(t_1) - AIP_{entry}(t_1)\}$.

The client sends a **ROUTE CREATE**$^{v.3}$ message along the new specified path, up to the switching AIP, AIP_{switch}. The AIP_{switch} discovers that this is actually an update of an existing route, updates its bindings and disables the old route by sending a tear down message down the old path. If this tear down message is lost, the old route will eventually time out, since no new data will go that way.

In the same way as in *case 1* the **ROUTE CREATE**$^{v.3}$ message verifies to the AIP_{switch} that this message is authorized to update the binding represented by $IP_{AIP_{switch-1}}(t), Port_{AIP_{switch-1}}(t), ACI_{switch}(t)]$ from the values at t_0 to the new ones at t_1.

To succeed with this, the **ROUTE CREATE** $^{v.3}$ contains the old IP address and port of the next lower entity ($IP_{AIP_{switch-1}}(t_0)$)- ($Port_{AIP_{switch-1}}(t_0)$), the old $PreKeySeed_{AIP_{switch}}(t_0)$ and the old ACI $ACI_{switch}(t_0)$.

This means that the client must know all ACI_i used along the route. This can be accomplished by modifying the **ROUTE CREATE ACK**$^{v.3}$ message, sent back from the initial **ROUTE CREATE**$^{v.3}$ message, so that each AIP_i in the chain adds its own ACI_k

Table 2: AIP_{switch} depending on the case.

Case	AIP_{switch}
Case 0	AIP_{exit}
Case 1	AIP_{entry}
Case 2	AIP_i

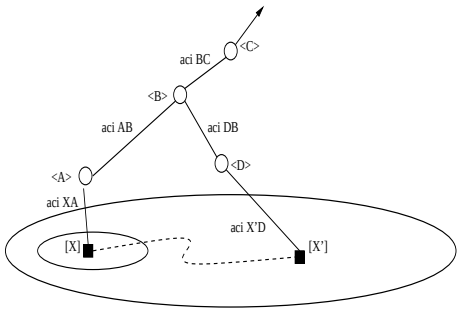

Figure 4: Case 2: ROUTE CREATE $AIP_i = AIP_{switch}$

number to the message before it is passed on along the route. The client already knows the IP addresses of all AIPs (AIP_i), since it is the client's responsibility to choose the chain of AIPs (AIP_i) in the first place.

If the $PreKeySeed_{AIP_{switch}}(t_0)$ is verified to be correct, the AIP_{switch} sends a tear down message down the old route, and updates its bindings to reflect the change.

The **ROUTE CREATE**$^{v.3}$ message is similar to the standard **ROUTE CREATE**$^{v.2}$ message, in the way that new shared secrets $S_N(t_1)$ are exchanged between the client and the new set of AIPs ($AIP_i(t_1)$, $i < switch$), within each layer of the *telescope encryption.*

The multilayer encryption ensures that each secret is only known by the respective AIP_i. The client reuses the secret established with the AIP_{switch} in the initial **ROUTE CREATE**$^{v.3}$ message $S_{switch}(t_0) = S_{switch}(t_1)$.

The AIP_{switch} has to send an acknowledgement that the route actually has been updated. This message is identical to the **ROUTE CREATE ACK**$^{v.3}$ except it only contains the newly chosen ACIs (ACI_i, $i < switch$) in the partial route.

3.1.4 Switching policies

How does the client decide what AIP should be the switching one? Three possible policies are:

- Preserve as much of the old route as possible. This yields a shorter path for the **ROUTE CREATE** message, which in turn yields faster handover.
- Optimize the route length. This yields fewer hops in the route from the client to the destination.
- Change more of the route than actually needed, to increase the privacy level.

3.1.5 Entry AIP discovery

In our scenario we used a mobile client with both a wireless LAN like 802.11b and a GPRS interface. The mobile client wants to roam between different IP networks hiding the mobility from both the correspondent node and the wormhole.

The mobile node needs to know which entry AIPs (AIP_{entry}) are available in the different IP networks it is roaming in.

The mobile client determines which AIP_{entry} to use based on the following discovery procedure:

All AIPs (AIP_i) sends out an "AIP advertisement" periodically. The "AIP advertisement" message is a standard ICMP router advertisement message with a *Freedom AIP advertisement extension,* see Appendix B.1.1.

The client can also force an advertisement by sending out "AIP solicitation" messages. The "AIP solicitation" message is a standard ICMP router solicitation message with TTL set to 1.

Which AIPs to use in the rest of the route created is determined by the user, based on information retrieved from the Freedom System.

One interesting feature of Freedom System that can be used to speed up handovers is that the client can create secure links between itself and *more than one* AIP_i. The $\{FC_j(t_1) - AIP_{entry}(t_1)\}$ encryption link can be established prior a new route creation is requested.

3.1.6 Handover

The mobile node performs handover when a change in point of attachment has been detected. The change can be detected either because the old connection is lost or by the client receiving agent advertisement messages from a new network. If a connection is lost then the client sends an agent solicitation message to trigger an agent advertisement message.

3.2 Mobile server location privacy

With this second type of protocol extension we want to allow an external node to start a connection to a *mobile server*, using the Freedom System, via an IP address IP_{FS_j} and port $Port_{FS_j}$ previously registered in the AIP_{exit}.

The AIP_{exit} acts as a home agent for the mobile server, accepting incoming connections and making the data travel back over the network to the care-of- address that the mobile server is using while moving.

The information is passed along the route indicated by the $ACIs$ until it reaches the AIP_{entry} and then it is link encrypted to the mobile server IP address. The data is transported from the AIP_{exit} to the client in the same way as data packet is transported in the normal mode of operation, ie the data packet is link decrypted, telescope encrypted and finally link encrypted in each AIP. The AIP_{entry} acts a foreign agent for the mobile server. The IP address of the server is in fact its care-of-address $IP(coa)_{FS_j}$.

The mobile server that wants to be reachable via the Freedom System opens a "control connection" to the AIP_{exit} and registers an IP address and port where the AIP_{exit} should listen to incoming connections. This registration is mapped with an ACI_{exit}. This ACI_{exit} binding is created by sending a **ROUTE CREATE**$^{v.3}$ message that includes the number of IP addresses and ports to be registered with the exit AIP and how those IP addresses and ports should be mapped to the remote local ports that the service is listening to on the mobile server.

In the previous cases the AIP_{exit} maps the mobile node port number and ACI_{exit} with a random port selected by the NAT for the outbound connection.[4] In our new case the **ROUTE CREATE**$^{v.3}$ message

[4]In order to guarantee that IP and port are unique for the NAT, the AIP_{exit} allocates one fake non routable IP address per ACI_{exit}. The mobile node local port number and ACI_{exit} are mapped to IP_{fake} and a random port number. IP_{fake} and $Port_{IP_{fake}}$ are used as source of *requests connections* to the NAT.

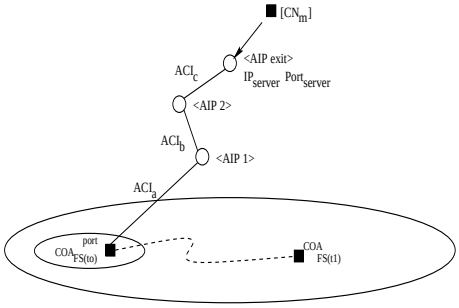

Figure 5: Mobile Server

would explicitly inform the AIP_{exit} that inbound connections to the mobile server's "home address" with *port X* should be mapped to ACI_{exit} and *port X'*.

Each AIP_i keeps a *control timer* for each virtual circuit that is reset when data is transmitted. When the timer expires a *tear down message* is generated to destroy the virtual circuit. If a corresponding host tries to connect when the route has been destroyed, this would of course lead to the inbound connection being lost.

To avoid the destruction of the route we could regularly send "keep alive" traffic. The best way would be to send a **ROUTE KEEPALIVE** message regularly at some time interval suitable compared to the route teardown timer. This message is basically a normal *data packet* with a special type [Appendix A]. Data packets of this type are passed along the route to the ACI_{exit} where they are discarded. Each AIP resets their expiration timers before passing this message to the next AIP.

4 Conclusions

Our protocol extensions of the Freedom System permits a mobile client to seamlessly roam among IP subnetworks and media types while remaining untraceable. These extensions makes it possible to support transparency above the IP layer, including the maintenance of active TCP connections in the same way that MobileIP$_{v4}$ does but with the addition that the **home network and foreign network are unlinkable** [5].

The new proposed routing messages provide the mobile node the **flexibility of rebuilding partial routes** hiding the mobility associated with certain pseudonym.

We have also introduced the possibility of having an **unlocatable mobile server** roaming behind the Freedom System. The mobile server is able to accept incoming connections via a home address/port previously registered in one of the Freedom System's wormholes.

About the authors

Alberto Escudero <alberto@it.kth.se>, is graduate student at the Institution of Microelectronics and Information Technology. (formerly Teleinformatics). KTH. Sweden. As a researcher in IMIT, Alberto is currently focusing on *location privacy* in mobile internetworking and privacy protection of personal information and over the last year was involved in the IT University - Kista Wireless Network, FlyingLinux.NET and The Big Brother "storebror" Project. http://www.it.kth.se/~aep

Martin Hedenfalk <mhe@home.se>, is an undergraduate student in Electrical Engineering at the Royal Institute of Technology, Stockholm, Sweden. Martin has many years of experience with GNU/Linux and has developed networking software for the Open Source community. He currently combines his last year studies of his M.Sc. by working part time as a Linux consultant.

Per Heselius <d97-phe@nada.kth.se>, is an undergraduate student in Computer Science at the Royal Institute of Technology in Stockholm, Sweden. Per is interested in developing systems for Computer Security, Cryptography and Computer Communications and is currently finishing the last year of his M.Sc. in Computer Science and Engineering.

References

[1] Forrester Report. *Surviving the privacy revolution.* February 2001

[2] Perkins, C. *IP Mobility support, RFC 2002.* 1996

[3] Deering, S., *ICMP Router Discovery Messages*, RFC 1256, September 1991.

[4] Pfitzmann, A., Köhntopp, M. *Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology.* 2000
http://www.koehntopp.de/marit/publikationen/

[5] Fasbender, A., Kesdogan, D., Kubitz, O. *Variable and Scalable Security: Protection of Location Information in Mobile IP.*

[6] Forsberg, D., Malinen, J.T., Maline, J.K., Weckström, T. *Dynamics - HUT Mobile IP a Technical Document.* 1999
http://www.cs.hut.fi/Research/Dynamics/documents.html

[7] Boucher, P., Shostack, A., Goldberg, I. *Freedom System 2.0 Architecture.* 2000
http://www.freedom.net/info/whitepapers

[8] Goldberg, I., Shostack, A. *Freedom Network 1.0 Architecture and Protocols.* 1999
http://www.freedom.net/info/whitepapers

[9] Back, A., Goldberg, I., Shostack, A. *Freedom 2.0 Security Issues and Analysis.* 2000
http://www.freedom.net/info/whitepapers

[10] Goldberg, I. *A pseudonymous communications infrastructure for the internet.* Fall 2000
http://www.isaac.cs.berkeley.edu/~iang/

A Protocol definitions

The same protocol message is used regardless of the scenario: case 1, 2 and 3. The same route creation message is used for all 3 cases. This has its cost, in the form of message complexity and size. Another approach could have been to have different messages for each of the three cases.

Most of the text explaining the Freedom protocol extension fields is based on [8] and [7]. Most of the advertisement protocols are heavily based on [2] and [3]. The texts have been modified to incorporate our extensions.

A.1 ROUTE_CREATE

The ROUTE_CREATE message is defined in table 3.

```
struct route_create {
    u8 rand0[8];
    u8 vers;
    u8 type;
    u8 pri;
    u8 spec[3];
    u16 ACI;
    struct route_info routes[N];
    struct nym_auth nymAuth;
    u8 ate[8];
    u8 lekv;
};
```

rand0 8 bytes of random data. It plays the role of an initialization vector (IV) for the link encryption.

vers A 1 byte unsigned number denoting the packet format version number. Should be to the constant 3 (the current version).

type A 1 byte unsigned number denoting the type of the packet. Should be set to the constant 2 (a route creation packet).

pri A 1 byte unsigned number denoting the priority of the packet. Lower numbers have higher priority. This field is currently unused.

spec 3 bytes of data specific to the packet type. This field is currently unused.

ACI A 2 byte unsigned number indicating, along with the packet's source address and port, the next hop in a particular route.

routes Subfield describing the routes, see section A.1.1.

nymAuth Subfield describing NYM authentication [8], see section A.1.4. The `nymAuth` part of the message is not used when *updating* the route. When updating the route, this whole subfield is set to zeros.

ate 8 bytes, each containing the number 8.

lekv A 1 byte unsigned value specifying the version of the link encryption key used to link encrypt the packet. This is necessary to allow packets to be decrypted while a new link encryption key is being negotiated. In practice, this value is either 0 or 1.

A.1.1 Route info

The Route info field is defined in table 4.

```
struct route_info {
    u8 preKeySeed[128];
    u8 symKeyAlg;
    u8 encrKeyVers;
    u8 pad0[6];
    struct route_crypt routeCrypt;
};
```

preKeySeed A 128 byte value representing $g^x \ mod \ p$ in a half-certified Diffie-Hellman key agreement. It will become key seed material after it is raised to the power b (all mod p), where b is the entry AIP's private key. The resulting data will be used to generate keys for the three symmetric algorithms listed in the route field.

symKeyAlg A 1 byte unsigned value indicating the symmetric algorithm used to decrypt the remainder of the route field. Its associated key is ultimately generated from `preKeySeed`. Currently always set to 3, indicating 128-bit Blowfish. Other valid values are 4 and 5 indicating 3-key 3DES and DESX, respectively.

encrKeyVers A 1 byte unsigned value specifying the version of the AIP's public encryption key used to encrypt the remainder of the route creation packet (beginning with the `bckKeyAlg` field). The symmetric key is ultimately derived from the `preKeySeed` field.

pad0 Six bytes set to the value 6.

routeCrypt A variable length encrypted buffer. The algorithm used is the one specified in `symKeyAlg` above. The key is derived as follows: the value of encrand chosen above and the destination AIP's public ElGamal encryption key (g, p, y) are used to calculate the 128-byte value $y^{encrand} \ mod \ p$. The first 2 bytes of this 128-byte value are discarded. The next 24 bytes are passed to the key generation routine described in section 4.1.4 in [8]. The next 8 bytes are used as an initialization vector (IV) for the algorithm. When decrypted, the buffer contains a subfield, described in section A.1.2.

A.1.2 Route crypt

The Route crypt field is defined in table 5.

```
struct route_crypt {
    u8 bckSymAlg;
    u8 fwdSymAlg;
```

Table 3: Route create message

rand0 . . . (8 bytes) . . .			
vers	type	pri	spec
spec		ACI	
routes_1 . . . (456 bytes) . . . ⋮ routes_N			
nymAuth . . . (146 bytes) . . .			
ate . . . (8 bytes) . . .			
lekv			

Table 4: Route information

preKeySeed . . . (128 bytes) . . .		
symKeyAlg	encrKeyVers	pad0 . . .
. . . pad0		
routeCrypt . . .		

Table 5: Route crypt

bckSymAlg	fwdSymAlg	flags	hostEnt . . .
hostEnt . . . (65 bytes) . . .			
oldIP . . . (16 bytes) . . .			
oldPort		oldACI	
oldPreKeySeed . . . (128 bytes) . . .			
numHomeIPAddr	pad0		
homeIPAddrs_1 . . . (96 bytes) . . . ⋮ homeIPAddrs_N			
expTime			
pad1			

```
    u8 flags;
    struct entity hostEnt;
    struct in6_addr oldIP;
    u16 oldPort;
    u16 oldACI;
    u8 oldPreKeySeed[128];
    u8 numHomeIPAddr;
    u8 pad0[3];
    struct home_addr homeIPAddrs[numHomeIPAddr];
    u32 expTime;
    u8 pad1[7];
};
```

bckSymAlg A 1 byte value indicating the symmetric algorithm used by the entry AIP to encrypt data flowing back to the client. The symmetric key used by the AIP for encryption (and by the client for decryption) is ultimately derived from the `preKeySeed` field. It is generated from bytes 34 to 57 (inclusive, starting from 0) of the 128-byte value $y^{encrand} \ mod \ p$ calculated above.

fwdSymAlg A 1 byte value indicating the symmetric algorithm used by the entry AIP to decrypt data originating from the client. The symmetric key used by the AIP for decryption (and by the client for encryption) is ultimately derived from the `preKeySeed` field.

flags A 1 byte field containing the following flags:

If bit 0 is set this is an exit AIP and the `hostEnt` field will contain zeros. Otherwise, if this bit is not set, then this is an intermediate AIP and the `hostEnt` must be valid.

If bit 1 is set then this is an switching AIP and the `hostEnt` field will contain zero. The `oldIP`, `oldACI` and `oldPreKeySeed` fields must be valid so that the route binding can be updated.

hostEnt A 65 bytes entity identifier specifying the next AIP in the route, or 65 bytes of 0 (zero) if this is the last hop or if it is a switching AIP. The type is set to 1, see section A.1.5 below.

oldIP A 16 byte value indicating the next lower entity (client or AIP) IP address that together with the `oldACI` and `oldPort` identifies which route to update.

oldPort A 2 byte field indicating the port of the old next lower entity that, together with the `oldIP` and `oldACI`, identifies which route to update.

oldACI This 2 byte field specifies the old ACI value used by the client when communication with the switching AIP. This value together with the `oldIP` and `oldPort` uniquely identifies the previously created route to update.

oldPreKeySeed A 128 byte value that contains the `preKeySeed` from the old registration with the switching AIP. By submitting the correct old "shared secret" the client authenticates itself to the switching AIP. If this value, after key calculation in the switching AIP, gives the correct key then the switching AIP knows that the request is valid.

numHomeIPAddr This 1 byte value indicates the number of valid home IP addresses that follow.

A future implementation must choose whether to use a variable length of the `homeIPAddrs` field below, or to use a fixed size. In case of using a fixed size, the `numHomeIPAddr` value must indicate how many IP addresses are actually valid. The field `numHomeIPAddr` and `numPorts` must always contain a correct value of 0 or greater.

If this field is 0 then there is no homeIPAddrs to register in the exit AIP. If this field indicates supplied `homeIPAddrs` then the exit AIP should register these IP addresses as belonging to the client.

All outbound connections should have the registered home IP addresses as source IP, set by the exit AIP. If more than one IP address is registered, then the implementation should select one of them as source IP.

If inbound connections are to be received in the exit AIP, then a list of port numbers have to be supplied and the `numPorts` field should be greater than 0. See section A.1.3 below.

pad0 3 bytes of padding, each set to the value 3.

homeIPAddrs This subfield stores the IP addresses and ports to use. See section A.1.3 below.

expTime A 4 byte unsigned value, specifying the expiration time of this route. The time is the number of seconds from the UNIX epoch.

pad1 6 bytes of padding, each set to the value 6.

A.1.3 Home addresses

The Home addresses field is defined in table 6.

```
struct home_addr {
    struct in6_addr homeIPAddr;
    u8 numPorts;
    u8 pad0[3];
    u16 homePorts[numPorts];
    u16 localPorts[numPorts];
};
```

homeIPAddr This 16 byte field specifies the home IP address that the exit AIP should affiliate with this route/client. Which ports the exit AIP accepts inbound connections to, with this IP, is determined by the ports given below.

numPorts This 1 byte value indicates the number of valid ports that follow. If inbound connections are to be accepted this field must be greater than zero and the ports to listen on must be supplied.

A future implementation must choose whether to use a variable length of the `homePorts` and

Table 6: Home Addresses

homeIPAddr . . . (16 bytes) . . .	
numPorts	pad0
homePorts_1 . . .	. . . homePorts_N
localPorts_1 . . .	. . . localPorts_N

`localPorts` fields below, or to use a fixed size. In case of using a fixed size, the `numPorts` value must indicate how many ports are actually valid.

homePorts Zero or more 2 byte fields, each indicating a port that the exit AIP should listen on.

localPorts Zero or more 2 byte fields, each indicating a local port that the client listens on. The number of local ports must be the same as the number of home ports above. The home port fields and the local port fields provide a mapping from exit AIP port numbers to local port numbers. In most cases they are identical.

pad0 3 bytes of padding, each set to the value 3.

A.1.4 NYM Authentication

The NYM Authentication field is defined in table 7.

```
struct nym_auth {
    u8 serialNum[8];
    struct entity nymEnt[65];
    u8 nymSig[65];
    u8 pad0[6];
};
```

serialNum 8 bytes of random data. The value is treated as a serial number and used to prevent replay attacks over the route lifetime, as specified by the `expTime` field. (Note that this requires approximately synchronized clocks among the routers and the client.) This field is not used by the AIPs.

nymEnt A 65 byte entity identifier specifying the nym that is creating the route. Possible values for the type subfield are 12 (anonymous) and 6 (Nym). See below.

NymSig A 65 byte DSA signature over the cleartext of the last route as well as the `serialNum` and `nymEnt` subfields. Note that, for the purpose of the signature only, the `preKeySeed` field is replaced by the key seed material generated from it. Thus, the signature covers data that is known only to the client and to the last hop. The 65 bytes of the signature are: two 29 byte bignums, representing the r and s parameters; 1 byte specifying the signature key version; 2 bytes specifying the size of the signature key, in bits (ie 1024 or 2048), followed by a four byte value giving the size of the data covered by the signature. All multi-byte fields are stored in network byte order (big-endian).

pad0 6 bytes of padding, all set to the value 6.

A.1.5 Entity

The entity field is defined in table 8.

```
struct entity {
    u8 type;
    u8 name[64];
};
```

type The type of the entity signing the packet.

name The name of the entity signing the packet.

A.2 ROUTE_DATA

The data packet is identical with the version 2 data packet, see table 9. The `type` field should contain the value 1, ie this is a data packet. The format of the message is described below.

The portion of the data packet from `rand1` to `mac`, inclusive, is multiply encrypted with keys known to each AIP along the route. At each AIP one layer of encryption is removed or added. This does not change the size of the packet.

```
struct route_data {
    u8 rand0[8];
    u8 vers;
    u8 type;
    u8 spec[4];
    u8 ACI;
    u8 rand1[8];
    u16 plen;
    u8 payload[];
    u8 mac[10];
    u8 ate[8];
    u8 lekv;
};
```

rand0 8 bytes of random data. It is changed at each AIP to make the data packet, once link encrypted, appear different between each router. This is to prevent known plaintext attacks.

vers A 1 byte unsigned number denoting the packet format version number.

type A 1 byte unsigned number denoting the type of the packet (ie route data, route creation, route tear down).

spec 4 bytes of packet type specific data. This field is currently unused and must be set to the value 4.

Table 7: NYM Authentication

serialNum . . . (8 bytes) . . .	
nymEnt . . . (65 bytes) . . .	
nymSig . . . (65 bytes) . . .	
. . .	pad0 . . .
. . . pad0	

Table 8: Entity

type	name . . .
. . . (64 bytes) . . .	

Table 9: Route data

rand0 . . . (8 bytes) . . .		
vers	type	spec
spec		ACI
rand1 . . .		
. . . rand1 . . .		
plen		payload . . .
. . . (variable length) . . .		
mac . . .		
. . . mac . . .		
. . . mac		ate . . .
. . . ate . . .		
. . . ate	lekv	

ACI A 2 byte unsigned number indicating (along with the packet's source address and port) to which AIP the packet should be forwarded.

rand1 8 bytes of random data. This is to prevent known plaintext attacks against the payload.

plen A 2 byte unsigned value giving the length of the `payload` field. This is the length of the actual payload, not including any extra space that may be necessary to pad it to the appropriate length.

payload This field typically contains an IP packet with a an upper bound on its size. Any trailing, unused portion of this field will be filled with a fixed value.

mac The first 10 bytes of a 20 byte SHA-1 message authentication code over the `rand1`, `plen` and entire payload fields (including padding). The associated key is taken from the 128 byte shared secret passed in the last route field of the route creation packet.

ate 8 bytes, each containing the number 8. This allows us to identify and reject noise packets (cover traffic) after link decryption.

lekv A 1 byte unsigned value specifying the version of the link encryption key used to link encrypt the packet. This is necessary to allow packets to be decrypted while a new link encryption key is being negotiated. In practice, this value is either 0 or 1.

A.3 ROUTE_CREATE_ACK

```
struct route_ack {
    u8 rand0[8];
    u8 vers;
    u8 type;
    u8 spec[4];
    u8 ACI;
    u8 rand1[8];
    u16 plen;
    u8 payload[];
    u8 mac[10];
    u8 intermediateACI[N];
    u8 ate[8];
    u8 lekv;
};
```

Our modified version of the acknowledgement message contains the ACIs that our route has in the different AIPs along the path.

The fields `rand1`, `plen`, `payload` and `mac` will be multiply encrypted with each AIPs secret key. This works the same way as the version 2 route acknowledgement message, except that each AIP on the path from exit to entry AIP adds its own ACI, padded with zeros to 8 bytes, to the encrypted part of the message. Finally the AIP encrypts the previously encrypted message part plus the added ACI value. The size is thus changed.

First, the exit AIP puts these values in the payload:

```
u16 specACI;
u8 numHomeIPAddr;
struct home_addr homeIPAddrs;
```

The `numHomeIPAddr` and `homeIPAddrs` fields are constructed in the same way as in the ROUTE_CREATE message, see section A.1.3. If an home IP address or a port number is not listed in the acknowledgement, that means that this IP or port could not be registered in the exit AIP. The reason is not supplied but the message could easily be extended to include a status field after each port number. This status field should in that case contain "OK" or error code.

The size of the payload field is the only variable size field in the packet. Its size must be chosen to ensure that the size of `rand1` through `mac`, inclusive, is a multiple of 8 bytes.

The number of `intermediateACI` fields are also variable and depends on the number of AIPs used in the route. The number of fields will be equal to the number of AIPs minus one since the ACI value in the exit AIP is already provided in the payload (specACI).

The portion of the ROUTE_CREATE_ACK message from `rand1` to `mac` and zero or more `intermediateACI` fields, is concatenated with the ACI value used in the current AIP. This concatenation is then encrypted with the key shared with the client. The message size is thus changed. This process is repeated in each AIP along the route.

When the client removes each layer of the telescope encryption it has to remove the appended ACI value.

All fields in the message are identical to the fields in the route data message defined above (section A.2), with the exception of the following:

type A 1 byte unsigned number denoting the type of the packet. Should be set to 5, ie this is a route create acknowledgement.

intermediateACI A 8 byte field containing the ACI value used in the respective intermediate AIP. 1 byte ACI data padded to 8 bytes. All AIPs (including the entry AIP) should add its ACI value in a new `intermediateACI` field after previously added values. The size of the whole packet thus changes.

A.4 ROUTE_KEEPALIVE

With each route in the Freedom System there is an expiration timer associated. This has implications when creating a route for inbound connections since the client does not send any outbound traffic that reset these timers. When these timers expire the associated route will be destroyed. To avoid routes being destroyed when no ordinary outbound traffic is sent a keepalive message is sent by the client. The keepalive message is an ordinary data packet, with a special type 10, ie this is a route keepalive message.

This message is recognized by the AIPs. When receiving a keepalive message the associated timer is reset and the message is passed along the route. The exit AIP resets its timer and discards the data packet.

Table 10: Route acknowledgement

rand0 ... (8 bytes) ...		
vers	type	spec
spec		ACI
rand1 ...		
... rand1 ...		
plen		payload ...
... (variable length) ...		
mac ...		
... mac		intermediateACI_1 ...
... intermediateACI_N		
ate ... ... (8 bytes) ...		
lekv		

Table 11: Advertisement

type	code	checksum
numAddr	AddrEntrySize	lifetime
AIPAddr_1		
prefLevel_1		
⋮		
AIPAddr_N		
prefLevel_N		

B AIP discovery

B.1 Advertisements

The AIP Advertisement message is defined in table 11.

```
struct router_adv {
    u8 type;
    u8 code;
    u16 checksum;
    u8 numAddr;
    u8 addrEntrySize;
    u16 lifetime;
}

struct router_adv_router {
    u32 AIPAddr;
    u32 prefLevel;
}
```

type always 9, ie ICMP Router Advertisement

code 0 The AIP handles common traffic, that is it acts as a router for IP datagrams not necessarily related to the Freedom System.

16 The AIP does not route common traffic.

checksum The 16-bit one's complement of the one's complement sum of the ICMP message, starting with the ICMP Type. For computing the checksum, the `checksum` field is set to 0.

numAddr The number of AIP addresses advertised in this message.

addrEntrySize The number of 32-bit words of information per each router address. This version of the protocol utilizes 2 32-bit words.

lifetime The maximum length of time that the advertisement is considered valid in the absence of further advertisements.

AIPAddr The ICMP Router Advertisement portion of the AIP Advertisement may contain one or more AIP addresses. Thus, an AIP may include one of its own addresses in the advertisement. An AIP may discourage use of this address as an entry AIP by setting the preference to a low value and by including the address of another AIP in the advertisement (with a correspondingly higher preference). Nevertheless, an entry AIP must route datagrams it receives from registered Freedom users, if the busy flag is not set.

prefLevel The preferability of each `AIPAddr` as an entry AIP address, relative to other AIP addresses advertised. A signed, twos-complement value; higher values mean more preferable.

The TTL for all AIP Advertisements must be set to 1.

The destination address must be set either to "all systems on this link" multicast address (224.0.0.1) or the "limited broadcast" address (255.255.255.255).

B.1.1 Flying Freedom Advertisement extension

The Flying Freedom advertisement extension follows directly after the ICMP Router Advertisement fields. It is used to indicate that an ICMP Router Advertisement is also an AIP Advertisement being sent by an AIP. See table 12.

```
struct advertisement_extension {
    u8 type;
    u8 length;
    u16 seq;
    u16 DHPort;
    u8 flags;
    u8 reserved;
    u32 bytePrice;
    u32 timePrice;
};
```

type No type number is currently assigned.

length 14 bytes.

seq The count of AIP Advertisement messages sent since the agent was initialized. See section 2.3.2 in [2].

DHPort The port number used for Diffie-Hellman key exchange, in network byte order.

flags **R** Registration required. Registration by supplying a valid NYM to this AIP is required (not in use)

B Busy. This AIP will not accept more connections from more Freedom clients, ie no more ROUTE_CREATE messages are accepted.

Y The byte price field is valid.

T The time price field is valid.

reserved Sent as zero; ignored on reception.

bytePrice The price per byte.

timePrice The price per time unit, eg seconds.

B.1.2 Prefix-lengths Extension

The Prefix-lengths extension may follow the Flying Freedom Advertisement Extension. It is used to indicate the number of bits of network prefix that applies to each AIP Address listed in the ICMP Router Advertisement portion of the AIP Advertisement. The Prefix-lengths extension is defined as in table 13.

type 19 (Prefix-lengths Extension)

length N, where N is the value of the `numAddr` field in the ICMP Router Advertisement portion of the Agent Advertisement.

prefixLength(s) The number of leading bits that define the network number part of the corresponding AIP Address listed in the ICMP Router Advertisement portion of the message. The prefix length for each AIP Address is encoded as a separate byte, in the order that the AIP Addresses are listed in the ICMP Router Advertisement portion of the message.

B.2 Solicitation

An AIP Solicitation is identical to an ICMP Router Solicitation with the further restriction that the IP TTL Field must be set to 1. See table 14.

```
struct agent_sol {
    u8 type;
    u8 code;
    u16 checksum;
    u32 reserved;
};
```

type 10, ie ICMP Router Solicitation

code 0

checksum The 16-bit one's complement of the one's complement sum of the ICMP message, starting with the ICMP Type. For computing the checksum, the checksum field is set to 0.

reserved Sent as 0; ignored on reception.

Table 12: Advertisement extension

type	length	seq	
DHPort		flags	reserved
bytePrice			
timePrice			

Table 13: Prefix-length extension

type	length	prefixLength[0]	...

Table 14: Solicitation

type	code	checksum
reserved		