DESIGN AND IMPLEMENTATION OF A BORDER ROUTER FOR SEAMLESS PEER-TO-PEER (P2P) UDP COMMUNICATION USING IPv4+4 ADDRESSES

Cihan TOPAL

Master of Science Thesis

Computer Engineering Program

June – 2008

This thesis is supported by The Scientific and Technological Research Council of Turkey (TUBİTAK). Grant No: 107E166

JÜRİ VE ENSTİTÜ ONAYI

Cihan Topal'ın **"IPv4+4 Adresleri ile Dikişsiz UDP İletişimi Sağlayan Yazılım Tabanlı bir P2P Dönüştürücü Tasarım ve gerçeklemesi"** başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki, Yüksek Lisans Tezi 11.06.2008 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	**İmza**
Üye (Tez Danışmanı)	**:Yard. Doç.Dr. CÜNEYT AKINLAR**	**……………,**
Üye	**: Yard. Doç.Dr. HAKAN ŞENEL**	**……………..**
Üye	**: Yard. Doç.Dr. EMİN GERMEN**	**……………..**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ………………… tarih ve ………………… sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ABSTRACT

Master of Science Thesis

DESIGN AND IMPLEMENTATION OF A BORDER ROUTER FOR SEAMLESS PEER-TO-PEER (P2P) UDP COMMUNICATION USING IPv4+4 ADDRESSES

Cihan TOPAL

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Assist. Prof. Dr. Cüneyt AKINLAR
2008, 42 pages

With the proliferation of P2P applications such as Voice over IP (VoIP), online games etc., there is an increasing demand for secure seamless peer-to-peer (P2P) UDP communication. Unfortunately, the current structure of the Internet, with hosts behind Network Address Translation (NAT) boxes, causes well-known problems for P2P applications. There are several proposals, e.g., STUN, UPnP, MIDCOM, TURN among others, to enable P2P UDP communication for nodes behind NAT boxes, but each technique offers a partial solution that works in special limited cases and fails in others.

Alternatively, several NAT-extended architectures have been proposed, e.g., IPv4+4, TRIAD, IPNL, RSIP, to restore end-to-end addressing and connectivity, however, they all require a complete and simultaneous upgrade of all existing network infrastructure, which simply is not feasible.

In this thesis, it is presented and implemented an extended Internet architecture that tries to offer a complete solution with IPv4+4 addresses to the secure seamless P2P UDP communication problem. The source files (lsrr_nat.c, nat.h, lsrr.h) of the implemented software are presented in the CD attached to the back cover.

Keywords: P2P Communication, Network Address Translation, IPv4+4.

ÖZET

Yüksek Lisans Tezi

IPV4+4 ADRESLERİ İLE DİKİŞSİZ UDP İLETİŞİMİ SAĞLAYAN YAZILIM TABANLI BİR P2P DÖNÜŞTÜRÜCÜ TASARIM VE GERÇEKLEMESİ

Cihan TOPAL

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Cüneyt AKINLAR
2008, 42 sayfa

VoIP, çevrimiçi oyunlar, vb. gibi P2P (uçtan uca) uygulamaların sayısının çoğalmasıyla beraber, güvenli dikişsiz P2P UDP iletişimine artan bir talep bulunmaktadır. Ancak çoğunluğu NAT (Ağ Adres Dönüşümü) cihazlarının arkasında bulunan kullanıcılardan oluşan mevcut Internet yapısı P2P uygulamalar için ciddi problemler teşkil etmektedir. NAT cihazı arkasındaki kullanıcılar için P2P UDP iletişimini sağlamaya yönelik STUN, UpnP, MIDCOM, TURN vb. gibi önerilen sistemler olmasına rağmen; bu sistemler belirli kısıtlamalar altında çalışabilmekte ve kısmi çözümler sunabilmektedirler.

Bunlarla beraber IPv4+4, TRIAD, IPNL, RSIP gibi bazı NAT tabanlı mimariler de uçtan-uca adresleme ve iletişimi sağlamak adına önerilmiş, ancak mevcut ağ yapısının tamamının eşzamanlı yükseltilmesini gerektirdiklerinden dolayı makul bir çözüm olamamışlardır.

Bu tez çalışmasında IPv4+4 adreslerini kullanarak güvenli, dikişsiz uçtan-uca UDP iletişimi problemini çözebilecek bir Internet mimarisi önerilmiş ve gerçeklenmiştir. Ayrıca gerçeklenen yazılımın kaynak kodları (lsrr_nat.c, nat.h, lsrr.h) tezin arka kapağında bulunan kompakt diskte sunulmaktadır.

Anahtar Kelimeler: P2P (Uçtan Uca) İletişim, Ağ Adres Dönüşümü, IPv4+4.

ACKNOWLEDGEMENTS

"I hear and I forget. I see and I remember. I do and I understand."
Confucius (551 BCE - 479 BCE)

Dear Sir,

You made me *hear*,
You made me *see*,
You made me *do*…

I appreciate for all the things that you made me *understand*…

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

IPv4 [1] architecture is well entrenched with Network Address Translation [4, 5] boxes. Although a NAT-based Internet is suitable for the client/server type of communication, e.g., HTTP, where the server is on the public Internet and the client is on a private address realm, i.e., the client has a non-routable private IPv4 address [3], this architecture creates well-known problems for Peer-to-Peer (P2P) communication. With the widespread use of P2P UDP [2] applications such as VoIP [24], online games etc., there is an increasing demand for secure seamless P2P UDP communication, but the existing NAT-based IPv4 architecture is a real hindrance for ubiquitous deployment. The problem arises from the fact that nodes behind NAT boxes do not have globally routable IPv4 addresses, making P2P communication impossible. What's required for seamless P2P communication is a globally-routable IP address for each node in the Internet.

IPv6 [7], when deployed, would provide each node on the Internet with a globally routable IPv6 address, which would enable end-to-end connectivity necessary for P2P communication. But the industry has been slow in transitioning to IPv6 as it requires a complete rehaul of the network infrastructure, i.e., routers and end-hosts. The difficulty in transition to IPv6 also stems from the fact that IPv6 addresses are not backward-compatible.

To enable P2P UDP communication in the presence of NATs, hole punching techniques have been proposed exemplified by STUN [8, 9]. The general idea with these techniques is to let a node behind a NAT box talk to a server on the public Internet and learn (global IP address, port) pair assigned to its (private IP address, local port) pair for a certain session, and then disclose this information to the peer for direct communication. Although such port prediction and hole punching methods enable P2P communication in certain limited cases, they are not reliable and fail to work for all NAT types.

There are other initiatives exemplified by UPnP [10] and MIDCOM [11] that propose on-demand port allocation for a session by apriori negotiations with the border router. Within these frameworks, a node talks to the border router or an agent before initiating a P2P session and allocates the required ports necessary for the communication. The node then discloses the border router IP address and the allocated ports to the peer for P2P communication. The problem with these proposals is that they require a simultaneous upgrade of all existing border routers, which makes the transition difficult. Additionally, by opening up a port to all incoming traffic, UPnP reduces the level of security provided by the current NAT boxes [19].

Other alternatives, exemplified by TURN [13], propose putting an intermediate node (a relay agent) in the path of the communication, which would terminate the session for a peer and act like a proxy. Although this would work for all NAT types, it requires the deployment of such relay agents in the global Internet, which is not only difficult but is also against the nature of P2P communication.

Alternatively, several NAT-extended architectures have been proposed, exemplified by IPv4+4 [14], TRIAD [16], IPNL [17], RSIP [18], to restore end-to-end addressing and connectivity. If deployed, these architectures would enable P2P communication, but they all require a complete rehaul of the existing network infrastructure including all border routers, end-hosts and even existing client/server applications, which simply is not feasible.

In this thesis it is presented a new Internet architecture that tries to offer a complete solution to the secure seamless P2P UDP communication problem. The framework is based on the use of IPv4+4 addresses [14] and the standard IPv4 Loose Source Record Route (LSRR) option [21]. An IPv4+4 address is a backwards-compatible globally unique network address for a node behind a NAT box, and is formed by concatenating the 32-bit globally routable IPv4 address of the border router with the 32-bit private IPv4 address of an internal node. The proposal which is presented in this thesis requires no changes whatsoever to end-hosts and Internet routers. The only requirement is a simple upgrade of border routers with a new LSRR-based packet forwarding algorithm for the P2P UDP traffic (described in section 3). The upgraded border router performs traditional NAPT (Network Address Port Translation) forwarding for the client/server communication, and the proposed LSRR-based forwarding for the P2P UDP traffic. It is first described how P2P UDP communication works in the presence of new upgraded border routers. Then it is shown how it is possible to enable P2P UDP communication even in the presence of a legacy NAT box on the path of the communication. This is very important as it lays the ground for a simple transition to the proposed Internet architecture.

2. PROBLEM DEFINITION AND RELATED WORK

IPv4 Internet is structured around NAT and firewall boxes. These devices are located at the edge of almost all business and home networks. Often, NAT and firewall functionality is bundled into a single box, called a middlebox, a home (residential) gateway or a border router. In the rest of this thesis, it would be used these terms interchangeably and it is assumed that they have bundled firewall and NAT functionality.

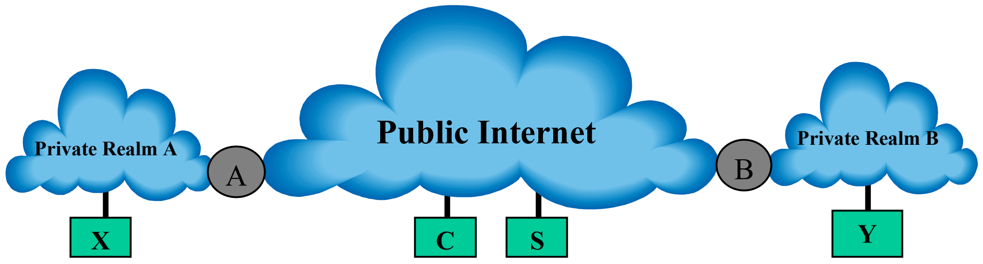


Figure 2.1: Two private realms, A and B, connected by the public Internet. A and B are middleboxes, a.k.a., home (residential) gateways, border routers. They perform Network Address Port Translation (NAPT), and act as Firewalls for their private realm

Figure 2.1 shows a snapshot of the current Internet architecture. There are two private realms A and B, e.g., two home or SOHO networks, connected by the public Internet. X and Y are hosts on private realms A and B respectively. They have non-routable private IPv4 addresses [3], X and Y respectively. A and B are border routers that perform Network Address Port Translation (NAPT) and also act as Firewalls for their private realms. C and S are hosts on the public Internet, and have unique, routable IPv4 addresses.

The above NAT-based IPv4 architecture is suitable for the client/server type of communication, e.g., HTTP, where the server is on the public Internet and the client is on a private address realm. Since the communication is initiated by the private host, e.g., X, the border gateway A dynamically allocates a port for the session in the NAT table and opens the port for incoming/outgoing traffic in the firewall. This allows only solicited traffic belonging to the session to enter the private realm. Unsolicited incoming traffic, that is, traffic initiated from outside the private realm, is not allowed inside thus providing security.

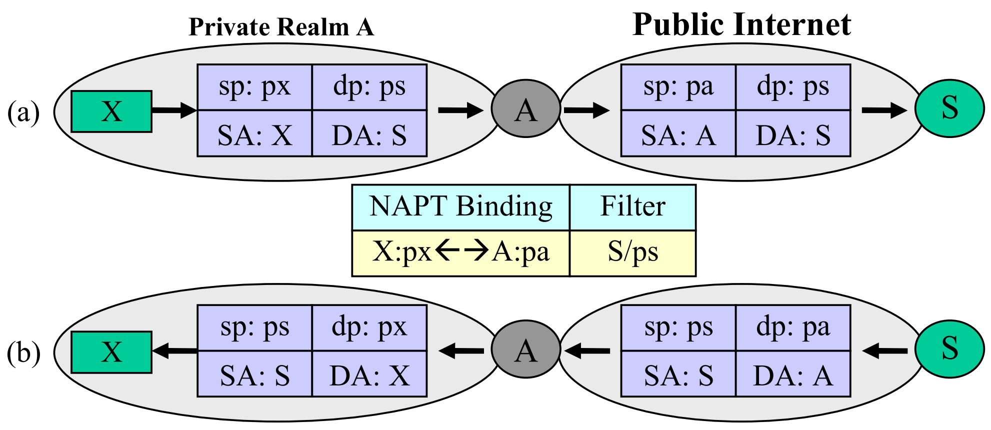


Figure 2.2: NAPT/Firewall functionality performed by the border gateway A, when X talks with S

Figure 2.2 illustrates how NAPT and firewall work together at the border gateway A for an example session between X and S. X initiates the session by sending a packet with source port sp=px, destination port dp=ds, source IPv4 address SA=X and destination IPv4 address DA=S. When border router A receives this packet, it allocates port pa for the session, creates a NAPT binding and a filter in the NAT table (X:px<->A:pa, Filter: S/ps). The filter means that incoming traffic to A:pa is allowed to pass inside only if it is coming from S:ps. This is called a symmetric NAT binding [6] and is very restrictive. Border router then sets the source address to A, SA=A, the source port to pa, sp=pa, and forwards the packet to the public Internet. This packet will make it to S through regular IPv4 forwarding in the Internet.

When the server replies, its packet will have the following parameters: <sp=ps, SA=S, dp=pa, DA=A>. The border router A will receive this packet, consult its NAT table and locate the NAPT entry for the session. The entry filter allows this packet to pass inside since it is coming from S:ps. So the border router changes the destination address to X, DA=X, destination port to px, dp=px, and forwards the packet to X.

NAT box permits client/server type communication when the server is on the public Internet and the client is behind a NAT box. For this scenario, bidirectional P2P communication between X and S is still possible, albeit a little messy. But when both peers are behind a NAT box, enabling P2P communication is a daunting task. Consider the case where X and Y want to exchange bidirectional P2P UDP traffic using ports px and py respectively. Initially, X and Y exchange their communication endpoint parameters, e.g., X:px and Y:py, using a separate control protocol such as SIP, H.323, etc. The obvious problem is that it is not possible to deliver any traffic to X:px from the Internet. So if X is to tell Y to send packets to X:px, no traffic will arrive at X. Similarly, no traffic can be delivered to Y:py from the Internet.

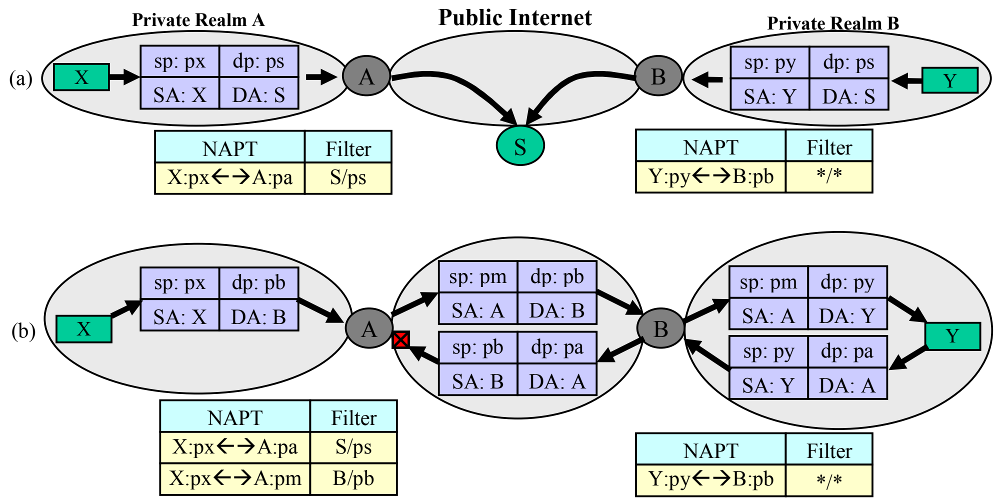


Figure 2.3: **(a)** X and Y talk to the STUN server S and learn their NAT bindings A:pa and B:pb respectively. We assumed that A is a symmetric NAT, B is a cone NAT
(b) X and Y exchanging packets: The packet from X is passed inside by the cone NAT B. The packet from Y is blocked and dropped by the symmetric NAT A

Simple Traversal of UDP Through Network Address Translators (STUN) [8], is a hole punching technique intended to solve the P2P UDP communication problem. The idea is illustrated in Figure 2.3. Before revealing its endpoint parameters X:px to Y, X talks to a STUN server S on the public Internet and learns (public IP address, port) pair assigned to its endpoint X:px. In the example, the border router assigns A:pa to X:px as seen in A's NAT table. X then informs Y to send UDP traffic to A:pa. Similarly, Y talks to a STUN server on the public Internet and learns (public IP address, port) pair assigned to its endpoint Y:py. In the example, B:pb is assigned to Y:py. Y then informs X to send UDP traffic to B:pb. The NAT tables at A and B reject that A is a symmetric NAT (Filter: S/ps indicates that only traffic sent from S:ps will be allowed inside) and B is a cone NAT [6] (Filter: */* indicates that all traffic sent to B:pb will be passed inside regardless of the source IP and source port number). In Figure 2.3(b), X and Y exchange packets. X sends a packet to <B:pb>. As the packet passes through the border router, a new NAT binding (X:px<->A:pm, Filter: B/pb) is created. This is due to the fact that symmetric NATs create a binding based on the source IP address and source port number as well as the destination IP address and destination port number. Since the destination IP address of the new packet is different from that of the STUN server, the symmetric NAT A creates a new mapping. This packet will make it to border router B and will be passes inside since B is a cone NAT. Looking at the packet from Y to X, we see that the packet is destined to <A:pa>. After passing through the border router B, its headers will have <sp=pb, SA=B, dp=da, DA=A>. When this packet makes it to A, it will locate the NAT entry for dp=da. The filter for this NAT entry passes a packet inside only is SA=S and sp=ps, which do not match the packet's source IP address and source port number. So the packet is dropped. To sump up the discussion, hole punching techniques exemplified by STUN work only if all border routers on the path of the communication are cone NATs. It is known that most border routers today are symmetric NATs [19], which make hole punching techniques to fail in certain cases. It should also be noted that cone NATs are susceptible to port scan attacks, which create additional security risks [19]. So making all border routers cone NATs is not an acceptable solution to the P2P UDP communication problem due to a reduced level of security.

Another approach advocated by Microsoft is the Universal Plug and Play (UPnP) [10]. The idea is the following: Before initiating a session, a UPnP-enabled node inside the private realm talks to the UPnP-enabled border router and opens up a port for communication. For example, before starting a session at endpoint X:px, X talks to its UPnP-enabled border router A and opens a port, say

A:pa. Similarly, Y talks to its UPnP-enabled border router B and opens a port, say B:pb. They then reveal their endpoint parameters A:pa and B:pb to each other through a control protocol, e.g., SIP. Once the ports are open for communication, all traffic arriving at A:pa will be passed to X:px, and all traffic arriving at B:pb will be passed to Y:py. In a sense, UPnP-enabled border routers behave similar to existing cone NATs. There are two drawbacks with this approach:

(1) All border routers and end-hosts must be UPnP-enabled. It is unreasonable to expect all border routers to support UPnP simultaneously.

(2) Since an UPnP-enabled border router behaves like a cone NAT, there is a reduced level of security. Specifically, after a port is opened up for communication, all traffic will be passed inside regardless of where it is coming from, which could easily be exploited.

IEFT MIDCOM architecture [11] proposes a solution similar to UPnP. The idea is to have intelligent agents that would talk to MIDCOM-enabled border routers using a MIDCOM protocol [12] to establish global communication parameters for the session. The goal is to separate out the application intelligence from the border router into agents and make the border router responsible for only NAPT and maybe firewall functionality. This enables new applications to be deployed without requiring a border router upgrade. Although MIDCOM approach addresses the security issue present in UPnP and cone NATs, it still requires all border routers on the path of the communication to be MIDCOM-enabled, which is a big hindrance to its deployment. That is, there is no clear path for transitioning to MIDCOM architecture.

There are other approaches such as TURN [13] that propose inserting a relay agent in the public Internet on the path of the communication. Thus all traffic will pass through the relay agent, which would enable communication similar to a proxy. This approach not only requires the deployment of such relay agents in the Internet, but also requires all end-nodes to be TURN-enabled. It is also against the nature of P2P communication.

3. AN EXTENDED INTERNET ARCHITECTURE: IPV4+4 ADDRESSES AND LSRR-BASED FORWARDING

Although a 32-bit private IPv4 address is not globally unique, it is possible to obtain a globally unique 64-bit network address by concatenating it with the 32-bit globally unique IPv4 address of the border router [14]. In a sense, the globally unique IPv4 address of the border router identifies the private realm and the private IPv4 address of the host is like an extension number inside the realm. With this convention, the globally unique IPv4+4 address of node X in Figure 2.1 is A.X, and IPv4+4 address of node Y is B.Y. Given that each node in a private realm can uniquely be identified with a 64-bit IPv4+4 address, how can we make use of these addresses to solve the P2P UDP communication problem? Authors in [15] propose a solution by defining a new protocol and header structure, and requiring the upgrade of all border routers, end-hosts and even all client/server applications to the new protocol, which is not very feasible. There is need for a way to make use of IPv4+4 addresses while requiring minimal changes to the existing network infrastructure.

In this section it is shown how this can be done with the standard IPv4 Loose Source Record Route (LSRR) option [21]. It is first described how LSRR works and then show that if border routers employ a modified version of the LSRR forwarding/filtering algorithm for the P2P UDP traffic, secure seamless P2P UDP communication can be achieved. For the proposal which is presented in this thesis to work, P2P applications are required to make use of IPv4+4 addresses and send UDP packets by using the IPv4 LSRR option. Since transmission/reception of UDP packets with the IPv4 LSRR option is part of all major operating system protocol stacks, e.g., Windows, Linux, Solaris, the proposal demands no changes whatsoever to end-host protocol stacks or IPv4 routers.

3.1 IPv4 Loose Source Record Route (LSRR) Option

IPv4 Loose Source Record Route (LSRR) option has been defined in RFC 791 [1]. LSRR allows the sender of an IPv4 packet to specify a list of nodes (IPv4 routers) that the packet must pass through on its way to the destination, and to record the route information. Although this type of explicit source-based routing information is not necessary for the correct forwarding of a packet, the following benefits are listed in [21]:

(1) To potentially specify a shorter path by the source,

(2) To avoid certain networks for performance or security reasons,

(3) To test and monitor certain IPv4 routers.

LSRR is implemented as an option included in the IPv4 header and specifies a list of IPv4 addresses where the packet must make stops on its way to the destination. The destination address of the initial packet contains the IPv4 address of the first hop node. At each stop, the address pointed to by the option pointer is taken from the list and placed in the destination address field, and that element of the list is replaced by the IPv4 address of that stop [21].

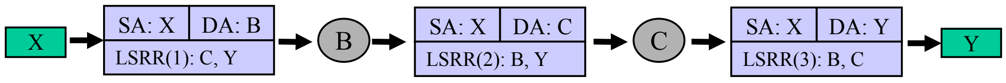


Figure 3.1: Packet Transmission from X to Y with IPv4 LSRR option: The packet makes stops at B and C before arriving at Y

Figure 3.1 shows an example packet flow using the LSRR option: X sends a packet to Y, but wishes the packet to make stops at B and C before arriving at Y. This is achieved by specifying a source route by the LSRR option as follows: When the packet leaves X, the IPv4 source address (SA) is set to X's IPv4 address and the destination IPv4 address (DA) is set to the next hop node B's IPv4 address. X also specifies the path that the packet must follow in the network after stopping at B. The LSRR option indicates that the packet needs to go to C and then to Y. LSSR option pointer (specified in parenthesis) indicates where the packet should be sent at the next hop B. At the beginning this is set to 1[1], meaning that the next hop after B is the first IPv4 address on the list, i.e., C. Since DA=B, the packet will be delivered to B. When B receives the packet, it looks at the LSRR option and realizes that the option is not exhausted yet. So it swaps DA, i.e., B, with the IPv4 address indicated by the LSSR pointer, i.e., C. B also increments the LSRR pointer to point to the next IPv4 address on the list. Notice that SA is not changed. The new packet with SA=X and DA=C will be delivered to C. C also notices that the LSRR option is not exhausted, so it performs LSRR processing similar to B: C swaps the DA with the IPv4 address pointed to by LSRR, increments the LSRR pointer, and forwards the packet to Y. The new packet has SA=X and DA=Y and will be delivered to Y. Y realizes that LSRR processing is done and that it is the last stop on the source route. Y also learns the route back to X from the LSRR values, which contains the path in the reverse direction.

3.2 P2P UDP Communication with Modified LSRR-based Border Routers

In this section it is shown that if border routers employ a modified version of the LSRR forwarding/filtering algorithm for the P2P UDP traffic, secure seamless P2P UDP communication can be achieved. It is assumed that P2P applications use IPv4+4 addresses and send/receive UDP packets with the IPv4 LSRR option as described below. It is noted that with the proposed algorithm, the border router will be as secure as the existing symmetric NAPT/firewall boxes. It is not illustrated regular client/server communication through our border router as that would use traditional NAPT translation.

Assume that a node X with IPv4+4 address A.X wishes to establish a bidirectional P2P UDP communication with another node Y with IPv4+4 address B.Y. Further assume that X wishes to use UDP port px and Y wishes to use UDP port py for this communication. It will be demonstrated the P2P session key as

[1] In a real IPv4 implementation LSRR option pointer contains the byte offset of the next IPv4 address on the list from the beginning of the option, and would initially be equal to 4. At each stop it will be incremented by 4.

<A.X:px, B.Y:py>. How X and Y learn the IP addresses of their border routers, and how they agree on the P2P UDP communication parameters is out of the scope of this work. But it is noted that existing P2P applications, e.g., VoIP using SIP [24, 25], online gaming etc., use a separate control path (protocol) to establish these parameters before the actual P2P UDP communication begins.

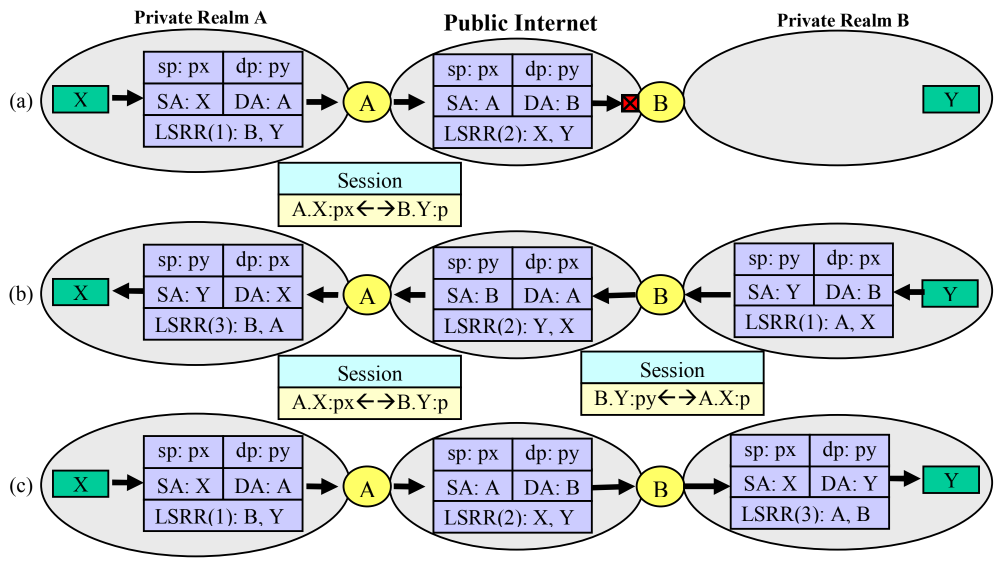


Figure 3.2: Using modified LSRR forwarding and IPv4+4 addresses for P2P UDP communication:
(a) Packet transmission from A.X:px to B.Y:py,
(b) Packet transmission from B.Y:py to A.X:px,
(c) Packet transmission from A.X:px to B.Y:py

Figure 3.2 depicts the packet flow between A.X:px and B.Y:py. It is assumed that after X and Y agree on the P2P UDP communication parameters, X is the first node to send a UDP packet to Y. Figure 3.2(a) depicts the packet flow from X to Y: When the packet leaves X, it has sp=px, dp=py, SA=X, DA=A, LSRR(1): B, Y. Note that DA equals the IPv4 address of X's border router A. X also specifies in LSRR that A must forward the packet to B, which must forward the packet to Y, the packet's final destination. When A receives the packet, it first tries to locate the P2P session in its session table. Note that it is required the border router to maintain a session table for P2P UDP traffic, in addition to the traditional NAT table for the client/server traffic. Since this is the first outgoing packet belonging to the session, the border router creates an entry in the session table. Thus a session is created only by the solicited (initiated from inside) outgoing traffic.

The packet now goes through LSRR processing. According to the regular LSRR processing, border router A swaps the IP address pointed to by the LSRR pointer and the destination IP address. So the packet header becomes: sp=px, dp=py, SA=X, DA=B, and LSRR option becomes: LSRR (2): A, Y. Notice that standard LSRR processing does not change the source IP address X. So if the border router were to send this modified packet to the Internet, it will most likely be dropped by the first ISP router due to ingress filtering [20]. According to RFC 1918 [3], realm routers must perform packet filtering and no packet having a private source or destination address should be sent out to the public Internet. This means that it is not possible to send the packet with SA=X. Therefore, we propose a modified LSRR processing algorithm at our border router: The border router moves the first address in LSRR list, B, to DA, puts the source address, X, in its place in LSRR list, and sets SA=A for the outgoing packet. So when the packet leaves A, its header contains: sp=px, dp=py, SA=A, DA=B, and LSRR option becomes: LSRR (2): X, Y. Notice that the source and destination port numbers are not changed.

When B receives this packet, it consults its P2P session table to locate the session. Since B is not aware of the session yet (recall that border router B will learn about the session after a packet is sent by Y to X), the packet is simply dropped. This way, unsolicited incoming traffic will simply be dropped similar to existing NAT boxes. Traffic for a session will be passed inside only if the border router knows about that session. Just like this packet is dropped by B, all subsequent packets from A.X:px to B.Y:py will be dropped until Y sends a packet to A.X:px.

Figure 3.2(b) illustrates Y sending a packet to A.X:px. When packet leaves Y, its header contains: sp=py, dp=px, SA=Y, DA=B, LSRR(1): A, X. When B receives this packet, it tries to locate P2P session and fails. Since this is the first outgoing packet belonging to the session, a new session is created in the session table. B then performs modified LSRR option processing similar to border router

A described above. So when the packet leaves B, its header contains: sp=py, dp=px, SA=B, DA=A, LSRR (2): Y, X.

When A receives this packet, it consults its session table and locates the session. So the packet is passed inside the private realm after LSRR processing. When the packet leaves A towards host X, its header contains: sp=py, dp=px, SA=B, DA=X, LSRR (3): Y, A. This packet will make it to X. Host X will realize the LSRR option has been exhausted and deliver the packet to the application listening to port px. The P2P application will not only get the packet, but also learn the reverse path to Y from the LSRR option.

In Figure 3.2(c) it is illustrated X sending another packet to B.Y:py. This time the packet is passed inside private realm B because B now knows about the session.

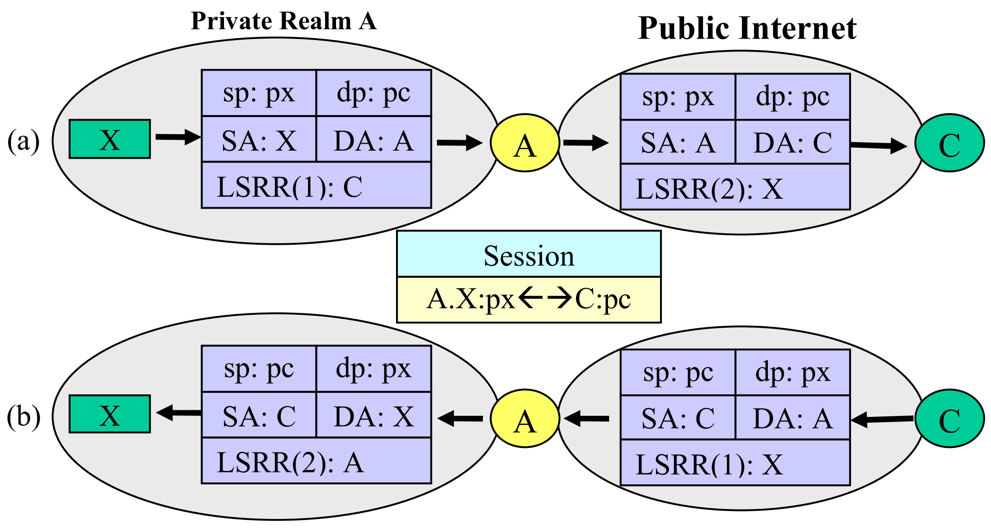


Figure 3.3: P2P UDP Communication between host X on private realm A and host C on the public Internet. **(a)** outcoming traffic, **(b)** incoming traffic.

P2P UDP communication using LSRR-based packet forwarding can also be used when one of the peers is in a private realm and the other is on the public Internet. Figure 3.3 illustrates this scenario, where host A.X in private realm A is talking to host C on the public Internet. Clearly, this P2P communication is very similar to X talking to Y. Instead, fewer addresses are inserted into the LSRR list. But otherwise, the LSRR processing at the border router A is the same as before.

The above illustrations should make it clear that if both border routers keep a P2P session table and employ the modified LSRR processing and filtering algorithm described above, secure seamless P2P UDP communication can always be achieved. Although the proposal in this thesis demands the upgrade of all existing border routers, it requires no changes to the end-host protocol stacks or routers. All major operating system protocol stacks allow transmission and reception of UDP packets with LSRR option using the existing sockets API. So, to benefit from the proposed framework, a P2P UDP application needs to learn the IPv4 address of its border router and it can start sending/receiving LSRR-based UDP packets for P2P communication. It is also noted that the proposed border routers are as secure as the existing symmetric NAT boxes:

(1) A new session is created only by solicited traffic, i.e., traffic originating from inside the realm;

(2) An incoming packet is passed inside only if all session parameters match an existing session.

3.3 Border Router Packet Forwarding Algorithm

In the previous sections the packet forwarding/filtering algorithm is described employed by our border router for each type of traffic. In this section, the algorithm in a more formal manner using a pseudocode will be presented. It is possible to divide the algorithm into two cases:

(1) When a packet is received from an internal host, that is, when BR (Border Router) receives a packet from one of its LAN interfaces (Figure 3.3),

(2) When a packet is received from the Internet, that is, when BR receives a packet from its WAN interface (Figure 3.4).

Both algorithms are divided into 3 parts, labeled I, II and III.

In part I, it is described the handling of packets that do not contain the LSRR option. These packets belong to client/server type communication where the client is in the private address realm and the server is in the Internet. When BR receives such a packet, it tries to locate the NAT entry for the session (step I.1). If the packet is received from a LAN interface and the session is not found, then this

```
Symbols used in the algorithm:
SA: Source IP address of the packet
DA: Destination IP address of the packet
sp: Source port of the packet
dp: Destination port of the packet
A: WAN IP address of the border router (BR)

BR receives a packet P from the LAN interface:
I. ProcessPacket(P[SA=X, DA=S, sp=px, dp=ps])
    I.1. <A:pa> = LocateNATEntry(<X:px, S:ps>)
    I.2. if (entry not found) then
        I.2.1. <A:pa> = CreateNATEntry(<X:px, S:ps>)
    I.3. Set SA=A, sp=pa

II. ProcessPacket(P[SA=X, DA=A, sp=px, dp=pc, LSRR(1): C])
    II.1. LocateSession(<A.X:px, C:pc>)
    II.2. if (session not found) then
        II.2.1. CreateSession(<A.X:px, C:pc>)
    II.3. Set SA=A, DA=C, LSRR(2): X

III. ProcessPacket(P[SA=X, DA=A, sp=px, dp=py, LSRR(1): B, Y])
    III.1. LocateSession(<A.X:px, B.Y:py>)
    III.2. if (session not found) then
        III.2.1. CreateSession(<A.X:px, B.Y:py>)
    III.3. Set SA=A, DA=B, LSRR(2): X, Y
```

Figure 3.4 Algorithm for processing packets received from a LAN interface

is the first packet belonging to the session. So BR creates a NAT entry in the NAT table (step I.2.1), and changes packet's SA and source port (step I.3). If the packet is received from the WAN interface and the session is not found, then the packet is simply dropped (step I.2). If the NAT entry is found for an incoming packet, then the packet's DA and destination port are changed, and the packet is forwarded inside (step I.3).

In part II, it is describe the handling of packets that contain an LSRR option with just one address in the option body, which is, there is just one BR between the communicating peers. These packets belong to either (1) client/server type communication where the client is in the global Internet and the server is on the private realm, e.g., connection request from C to A.X in Figure 2.1, or (2) P2P communication where one peer is on a private realm and the other peer is in the global Internet, e.g., P2P communication between A.X and C in Figure 2.1. When BR receives such a packet, it tries to locate the session in the session table step

```
Symbols used in the algorithm:
SA: Source IP address of the packet
DA: Destination IP address of the packet
sp: Source port of the packet
dp: Destination port of the packet
A: WAN IP address of the border router (BR)

BR receives a packet P from the WAN interface:
I. ProcessPacket(P[SA=C, DA=A, sp=pc, dp=pa])
I.1. <X:px> = LocateNATEntry(<C:pc, A:pa>)
I.2. if (entry not found) then Drop the packet and exit
I.3. Set DA=X, dp=px

II. ProcessPacket(P[SA=C, DA=A, sp=pc, dp=px, LSRR(1): X])
II.1. LocateSession(<A.X:px, C:pc>)
II.2. if (session not found) then Drop the packet and exit
II.3. Set DA=X, LSRR(2): A

III. ProcessPacket(P[SA=B, DA=A, sp=py, dp=px, LSRR(2): Y, X])
III.1. LocateSession(<A.X:px, B.Y:py>)
III.2. if (session not found) then Drop the packet and exit
III.3. Set SA=Y, DA=X, LSRR(3): B, A
```

Figure 3.5 Algorithm for processing packets received from the WAN interface

II.1). If the packet is received from a LAN interface and the session is not found, then this is the first packet belonging to the session. So BR creates an entry in the session table (step II.2.1), and changes packet's SA, DA and the LSRR option (step II.3). If the packet is received from the WAN interface and the session is not found, then the packet is simply dropped (step II.2). If the session is found for an incoming packet, then the packet's DA and the LSRR option are changed, and the packet is forwarded inside (step II.3).

In part III, it is described the handling of packets that contain an LSRR option with two addresses in the option body, that is, there are two BRs between the communicating peers. These packets belong to either (1) client/server type communication where the client is on one private realm and the server is on a different private realm, e.g., connection request from B.Y to A.X in Figure 2.1, or (2) P2P communication where a peer is on one private realm and the other peer is on a different private realm, e.g. P2P communication between A.X and B.Y in Figure 2.1. When BR receives such a packet, it tries to locate the session in the session table (step III.1). If the packet is received from a LAN interface and the session is not found, then this is the first packet belonging to the session. So BR creates an entry in the session table (step III.2.1), and changes packet's SA, DA and the LSRR option (step III.3). If the packet is received from the WAN interface and the session is not found, then the packet is simply dropped (step III.2). If the session is found for an incoming packet, then the packet's SA, DA and the LSRR option are changed, and the packet is forwarded inside (step III.3).

It is possible to see from steps I.2, II.2 and III.2 in Figure 3.5 that when BR receives a packet from the WAN interface and the session that the packet belongs to is not found in the BR's NAT or session tables, then the packet is simply dropped. These steps provide the necessary security in the sense that only solicited incoming traffic is let inside the private realm. All unsolicited traffic is simply dropped by BR.

3.4 P2P Communication in the Presence of a Legacy Border Router

It has been shown in the previous section that secure seamless P2P UDP communication is possible if both border routers on the path of the communication implement the proposed LSRR-based packet processing/filtering algorithm. The problem is, it is unreasonable to expect a simultaneous upgrade of all border routers. If the Internet is to transition to the proposed LSRR-based IPv4+4 framework, it must be a continuous process. This is why so many of the proposals including IPv4+4 [14], TRIAD [16], IPNL [17], MIDCOM [11], UPnP [10], among others, have not been adapted as they require a simultaneous upgrade of all border routers.

Fortunately, it is possible for the proposal of this thesis to work even in the presence of a legacy NAT box on the path of the communication.

Figure 3.6 shows an example P2P UDP session between A.X:px and B.Y:py when A is a new upgraded border router that implements the proposed LSRR-based packet processing/filtering algorithm, but B is a legacy border router that performs only the traditional symmetric NAPT and firewall services. We assume that before the communication starts, X and Y learn the IP address of their border routers A and B and then talk to their border routers to see if they implement LSRR-based forwarding or not. This second step can be achieved by designing a simple query/reply discovery protocol and having the upgraded border router implement it. Since a legacy border router will not implement this new discovery protocol, a host can easily determine the type of border router it is behind. In our example, X discovers that A is a new upgraded border router and Y discovers that B is a legacy NAT box.

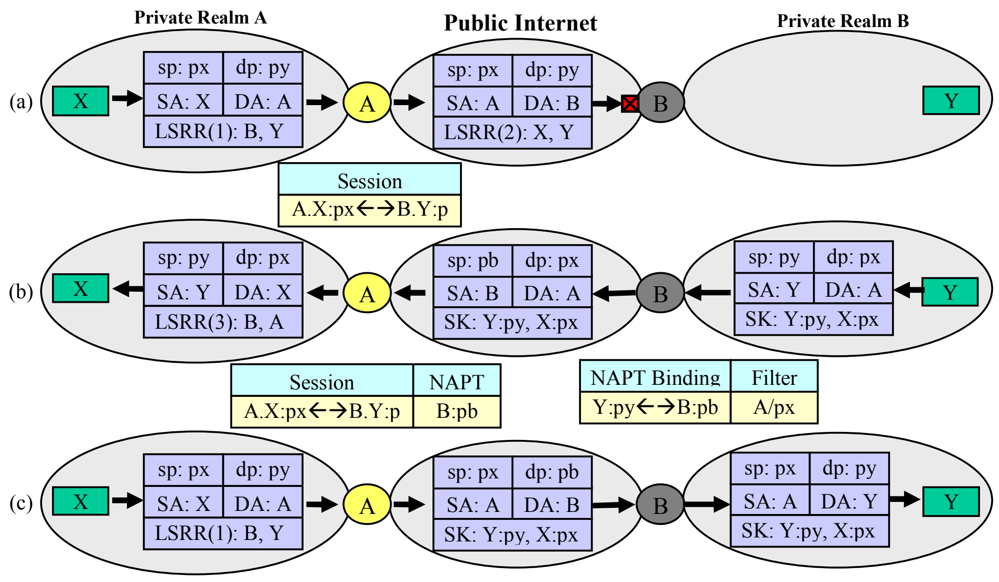


Figure 3.6: P2P UDP communication between A.X:px and B.Y:py when border router B is a legacy NAPT device

After discovery of the border router capabilities, packet transmission commences. This is illustrated in Figure 3.6. Since border router A supports LSRR-based forwarding, X follows the regular LSRR-based forwarding algorithm, sending the packet to A and specifying intermediate hops B, Y in the LSRR option. A will manipulate this packet using the proposed LSRR processing algorithm and forward it to B, where it will be dropped. Recall that B is a legacy NAT, and a legacy NAT will drop all unsolicited incoming traffic. This is illustrated in Figure 3.6(a).

When Y sends a packet to X, it cannot use LSRR-based forwarding. If it did, the packet will be dropped by B. It is known that IPv4 LSRR packets are considered to be a security threat [22] and dropped by existing NAT boxes. Even operating systems such as Windows and Linux disable transmission/reception of packets with LSRR option by default, which must be enabled before sending such packets. Since it is not possible to use IPv4 LSRR option to specify the session key, we need an alternative way to insert the session key inside the IPv4 header options area. Recall that the session key for the communication is (A.X:px <-> B.Y:py). Since the IPv4 header of the packet passing through the legacy NAT box B will have SA=B, DA=A, it is clear that the source and destination private IP addresses, X and Y, of the session key must somehow be inserted into the packet. Furthermore, since the NAT box changes the source port py to a new value, say pb, the source port py needs to be part of the created session key. With these in mind, it can be proposed that the sent session key consists of the source and destination port numbers (px, py) and the source and destination private IP addresses (X, Y). This is illustrated in Figure 3.6(b), where Y creates a session key, (SK: Y:py, X:px), and inserts it into the IPv4 options area. Y then sends the packet directly to border router A instead of B. Recall that had B implemented LSRR-based forwarding, the packet would have been sent to B. When the packet goes through B, it creates a NAPT binding for the session (Y:py<->B:pb), changes the source IP address to SA=B and the source port to sp=pb. When this packet reaches A, A sees the session key, locates the session in its session table, and also realizes that B is a legacy NAT. A then updates its session table to reject B's NAPT binding, changes the session key in the packet to an LSRR option and sends the packet to X. Having learnt that B is a legacy NAT, A also manipulates the packets traveling to Y by inserting the session key (SK: X:px, Y:py) and also changes the destination port to dp=pb so that the packet is passed inside by B. This is depicted in Figure 3.6(c). Thus, even if B is a symmetric NAT, the packet will be passed inside to Y since it is coming from A:px.

The only question left to answer in the above discussion is how to carry the session key in an IP packet. It is possible to define a new IP option, but this has two problems:

(1) It is known that Internet routers drop IP packets that contain unknown IP options [15],

(2) Existing operating system sockets API does allow setting IP options field to arbitrary values.

At Windows, Linux and Solaris, and the only IPv4 options available for uses are the LSRR and the Timestamp option. Since it cannot be used the LSRR option as the existing NAT boxes drop packets containing the LSRR option 2, using the Timestamp option [26] to carry the session key reveals as the most convenient solution. Details of how the timestamp option is used in our prototype implementation are given in section 5.

4. SEAMLESS P2P COMMUNICATION WITH THE SESSION INITIATION PROTOCOL USING IPv4+4 ADDRESSES

The Session Initiation Protocol (SIP) [24] is the emerging Voice over IP (VoIP) signaling protocol. It is used to establish, change and terminate multimedia sessions between two or more peers on an IP network. In this section our goal is to demonstrate how P2P communication would work for voice transmission between two IPv4+4 addressed nodes. It is assumed that the existing realm gateways, i.e., the NAT boxes on the private realm borders, support the IPv4 LSRR option.

Consider the network shown in Figure 2.1. Assume that Alice logged in X and Bob logged in Y are two SIP users located behind NAT boxes. They only know about their private IPv4 address, X and Y respectively, and the public IPv4 address of the SIP server, S. Assume that they are not even aware of the public IPv4 addresses of their realm gateways. The first thing that a SIP client performs during startup is to register its current location with the SIP server. In Alice's case, she would send a registration request similar to the following (it is only shown the relevant fields of the SIP message. SIP message examples are taken from [28]):

```
REGISTER sip:X SIP/2.0
From: sip:alice@X
To: sip:alice@X
Contact: sip:alice@X
```

The registration packet is usually sent over UDP and would have <SA=X, DA=S>. Notice that this packet would be subject to NAT processing at the border gateway A, and have its source address changed. So the packet would have <SA=A, SA=S> on the public Internet. When S receives the registration request, it realizes that Alice is logged into a machine with private IPv4 address X (induced from Contact: sip:alice@X) located behind the NAT box A (induced from SA of the packet). With our proposal, the SIP server S would send the following reply back:

```
SIP/2.0 200 OK
From: sip: alice@X
To: sip: alice@X
Contact: sip: alice@A.X
```

Notice that “Contact:” header in the reply contains the full 64-bit IPv4+4 address A.X of node X. Thus Alice would learn that she is behind a NAT box with global IPv4 address A. Bob would go through a similar registration procedure and learn that he is behind a NAT box with global IPv4 address B. That is, Bob's IPv4+4 address is B.Y. The SIP server S also learns both Alice and Bob's IPv4+4 contact addresses as A.X and B.Y.

When Alice wishes to establish a VoIP session with Bob, she would send an Invite request via S. With our proposal, Alice would specify her IPv4+4 address for media exchange instead of her private IPv4 address. A sample Invite message with IPv4+4 address is shown below:

```
INVITE sip:bob SIP/2.0.
From: sip:alice
To: sip:bob
Content-Type: application/sdp

v=0
o=alice 2890844526 2890844526 IN IP4+4 A.X
s=-
c=IN IP4+4 A.X
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Notice that Alice puts her IPv4+4 address A.X within the media session description by SDP [25]. Specifically, Alice indicates that she expects the media in PCM format sent to IPv4+4 address A.X:49172.

When Bob receives the invitation request, he would reply with a message similar to the following, where Bob specifies his IPv4+4 address B.Y:3456 for media exchange:

```
SIP/2.0 200 OK
From: sip:alice
To: sip:bob
Contact: sip:bob@A.Y
Content-Type: application/sdp

v=0
o=bob 2890844527 2890844527 IN IP4+4 B.Y
s=-
c=IN IP4+4 B.Y
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Alice would finally send an ACK message, and the session would be established. The media exchange is now P2P between A.X:49172 and B.Y:3456, and can be achieved by IPv4 LSRR as described in section 3.2.

It is important to note that IPv4+4 addresses would be used only if the user is logged into a host behind a NAT box. If Alice were to login at a host with a globally routable IPv4 address, e.g., node C in Figure 2.1, she would register the contact address sip:alice@C. The SIP server S would realize that Alice's host is not behind a NAT box, and so further communication with Alice would use IPv4 address C, and not an IPv4+4 address. Thus IPv4+4 addresses would only be needed for hosts behind NAT boxes.

5. FRAMEWORK TESTBED AND IMPLEMENTATION DETAILS

To test the proposed framework, testbed is set up depicted in Figure 2.1 in one of the laboratories of the Anadolu University, Department of the Computer Engineering as seen in the Figure 5.1. There are two private realms A and B connected over the Internet. There are one host X in realm A and one host Y in realm B and one host C on the Internet. All end-hosts are PCs running Windows. Border routers A and B are emulated by two PCs, each having 2 interfaces, one attached to the private realm and one attached to the Internet. Both border routers run Linux.

Figure 5.1. Laboratory where the testbed is established and implementations take place.

5.1 Details of Implemented Driver

After the establishment of the development environment, it has been designed a Linux driver that implements the border router functionality. That driver, developed as a Linux kernel loadable module for 2.6 kernels, implements NAPT functionality for the traditional client/server traffic and the proposed

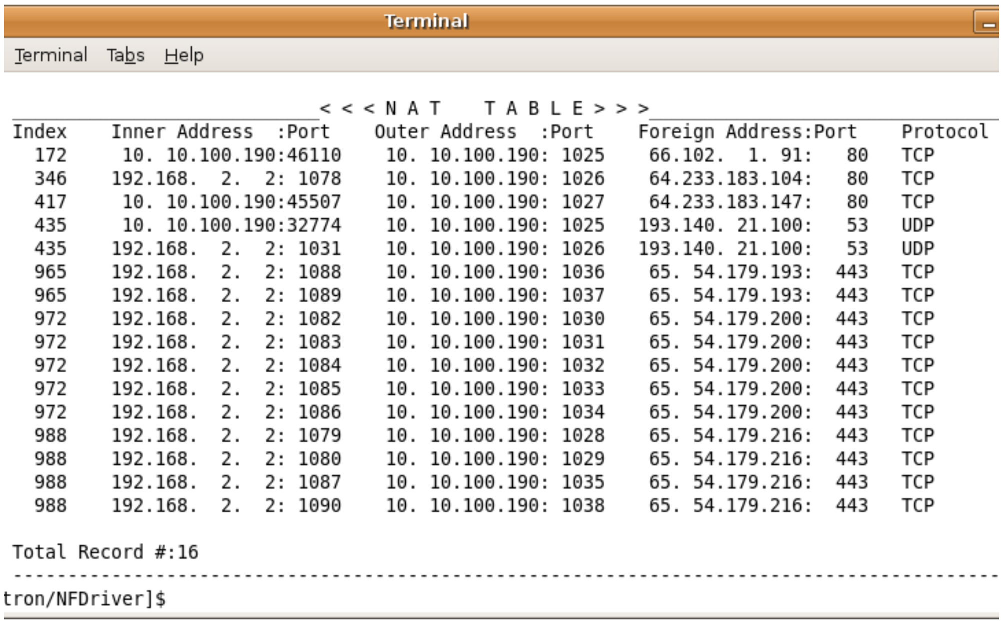
LSRR-based packet filtering/forwarding algorithm described in section 3 for the P2P UDP traffic. The driver consists of about 1000 lines of C code in three individual source and header files - lsrr_nat.c, nat.h, lsrr.h, (see App. 1) -, and uses the Linux Netfilter architecture [23]. Netfilter is a set of hooks in the Linux kernel protocol stack that allows callback functions to be registered. They are attached two hooks at the border router WAN interface, one for incoming packets and one for outgoing packets. When a packet arrives from the WAN; Netfilter gives the packet to the implemented driver, which processes the packet before letting it move up the protocol stack. Similarly, before a packet is sent down the WAN interface, Netfilter calls our driver, which processes the packet.

5.1.1 NAPT Implementation

At the first step of the development of the driver, legacy NAPT software (see App. 1), is implemented for a Linux 2.6 computer with two Ethernet interfaces. Thus, it is provided the use of a computer like a standard SOHO gateway which implements standard address and port translation operations with a compatibility of UDP and TCP protocols. The implemented driver can be inserted as a module into the Linux OS kernel and it can capture the all packets of incoming/outgoing network traffic, also it can either perform the necessary manipulations on the packets or drop them. Also, it records the translation information of all network sessions on a table and it stores this table to the Linux's kernel log. Thus, any user can view the content of this NAPT table (see Figure 5.2).

5.1.2 Modified LSRR Algorithm Implementation

After the NAPT implementation is done, the driver is extended to be able to perform the LSRR based packet forwarding algorithm (see App. 1) for UDP protocol. Processing of UDP packets containing IPv4 LSRR option follows the rules described in section 3.2: For an outgoing packet, the session table is searched. If this is the first outgoing packet belonging to the session, an entry is created in the session table.

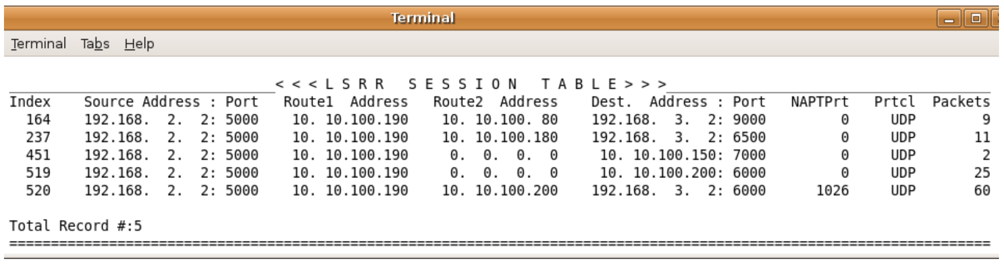


```
                          < < < N A T    T A B L E > > >
Index    Inner Address  :Port    Outer Address  :Port    Foreign Address:Port    Protocol
  172     10. 10.100.190:46110    10. 10.100.190: 1025    66.102.  1. 91:   80    TCP
  346    192.168.  2.  2: 1078    10. 10.100.190: 1026    64.233.183.104:   80    TCP
  417     10. 10.100.190:45507    10. 10.100.190: 1027    64.233.183.147:   80    TCP
  435     10. 10.100.190:32774    10. 10.100.190: 1025   193.140. 21.100:   53    UDP
  435    192.168.  2.  2: 1031    10. 10.100.190: 1026   193.140. 21.100:   53    UDP
  965    192.168.  2.  2: 1088    10. 10.100.190: 1036    65. 54.179.193:  443    TCP
  965    192.168.  2.  2: 1089    10. 10.100.190: 1037    65. 54.179.193:  443    TCP
  972    192.168.  2.  2: 1082    10. 10.100.190: 1030    65. 54.179.200:  443    TCP
  972    192.168.  2.  2: 1083    10. 10.100.190: 1031    65. 54.179.200:  443    TCP
  972    192.168.  2.  2: 1084    10. 10.100.190: 1032    65. 54.179.200:  443    TCP
  972    192.168.  2.  2: 1085    10. 10.100.190: 1033    65. 54.179.200:  443    TCP
  972    192.168.  2.  2: 1086    10. 10.100.190: 1034    65. 54.179.200:  443    TCP
  988    192.168.  2.  2: 1079    10. 10.100.190: 1028    65. 54.179.216:  443    TCP
  988    192.168.  2.  2: 1080    10. 10.100.190: 1029    65. 54.179.216:  443    TCP
  988    192.168.  2.  2: 1087    10. 10.100.190: 1035    65. 54.179.216:  443    TCP
  988    192.168.  2.  2: 1090    10. 10.100.190: 1038    65. 54.179.216:  443    TCP

Total Record #:16
-------------------------------------------------------------------------------------------
tron/NFDriver]$
```

Figure 5.2. NAT Table stored into Linux kernel log

The necessary information about LSRR sessions is stored another table (see Figure 5.3) which is also stored to the Linux kernel log and visible for the users. After the entry of the session is created in the table, the packet is processed using the modified LSRR processing algorithm and the packet is sent to the Internet. When a packet is received, the session is searched in the session table. If the session is not found, then the packet is simply dropped. Thus the designed border router does not pass any packets that do not belong to a known session, which provides security. Clearly the border router allows only solicited incoming traffic similar to current NAPT boxes. So the communication is as secure as the existing NAPT boxes.



```
                                  <<<LSRR  SESSION  TABLE>>>
Index     Source Address : Port   Route1  Address    Route2  Address     Dest.  Address : Port    NAPTPrt    Prtcl  Packets
  164     192.168.  2.  2: 5000    10. 10.100.190     10. 10.100. 80     192.168.  3.  2: 9000          0      UDP        9
  237     192.168.  2.  2: 5000    10. 10.100.190     10. 10.100.180     192.168.  3.  2: 6500          0      UDP       11
  451     192.168.  2.  2: 5000    10. 10.100.190      0.  0.  0.  0      10. 10.100.150: 7000          0      UDP        2
  519     192.168.  2.  2: 5000    10. 10.100.190      0.  0.  0.  0      10. 10.100.200: 6000          0      UDP       25
  520     192.168.  2.  2: 5000    10. 10.100.190     10. 10.100.200     192.168.  3.  2: 6000       1026      UDP       60

Total Record #:5
=====================================================================================================================
```

Figure 5.3. LSRR Session table stored into Linux kernel log

It is described in section 3.4 that P2P UDP communication is possible even if one of the border routers on the path of the communication is a legacy NAT. In this case, the host behind the legacy NAT box does not use the LSRR option to send the packet, but rather puts a session key into the IPv4 options area. Since it is not possible to use the LSRR option (it has been observed that legacy NAT boxes drop packets containing LSRR option), and the only other IPV4 option implemented by major operating systems is the IPv4 timestamp option, the use of the IPv4 timestamp option [26] to store the session key remains as the most appropriate solution.

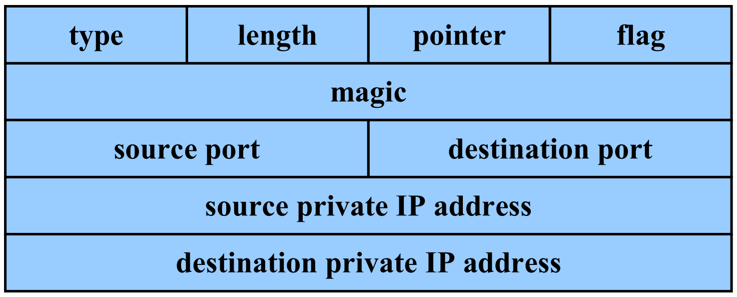


Figure 5.4: How the session key is stored in an IPv4 Timestamp option

Figure 5.4 depicts how the session key, e.g., A.X:px<->B.Y:py, is stored in IPv4 timestamp option. The first 4 bytes are the option type = 68, length = 20 (total option length), ptr = 5 (offset of the first IP address in the list) and flag = 3 (meaning that a router must set the timestamp only if the IP address pointed to by the option pointer matches its IP address). The bytes 4-8 are set to a special magic value "v4+4" to designate that this timestamp is not a regular IPv4 timestamp option, but rather contains our P2P session key. Since it is very unlikely that the IP address of a router will be equal to "v4+4", the timestamp option would reach the destination border router without being modified. The source and destination ports, e.g., px and py, follow the magic number. Finally, the private source and private destination IP addresses of the session key complete the option. Note that it is not stored the border router IP addresses in the timestamp option since they are already found in the IPv4 header of the packet (refer to Figure 3.4(b)).

5.2 Details of the Implemented Border Router Hardware

After the designed Linux network driver is implemented for a computer with two Ethernet interface, it is investigated for an embedded microprocessor board to deploy the driver on it. Then two convenient embedded microprocessor boards are selected. Both of them are able to run Linux and have two or more Ethernet interfaces.

5.2.1 Pronghorn SBC-250 Microprocessor Board

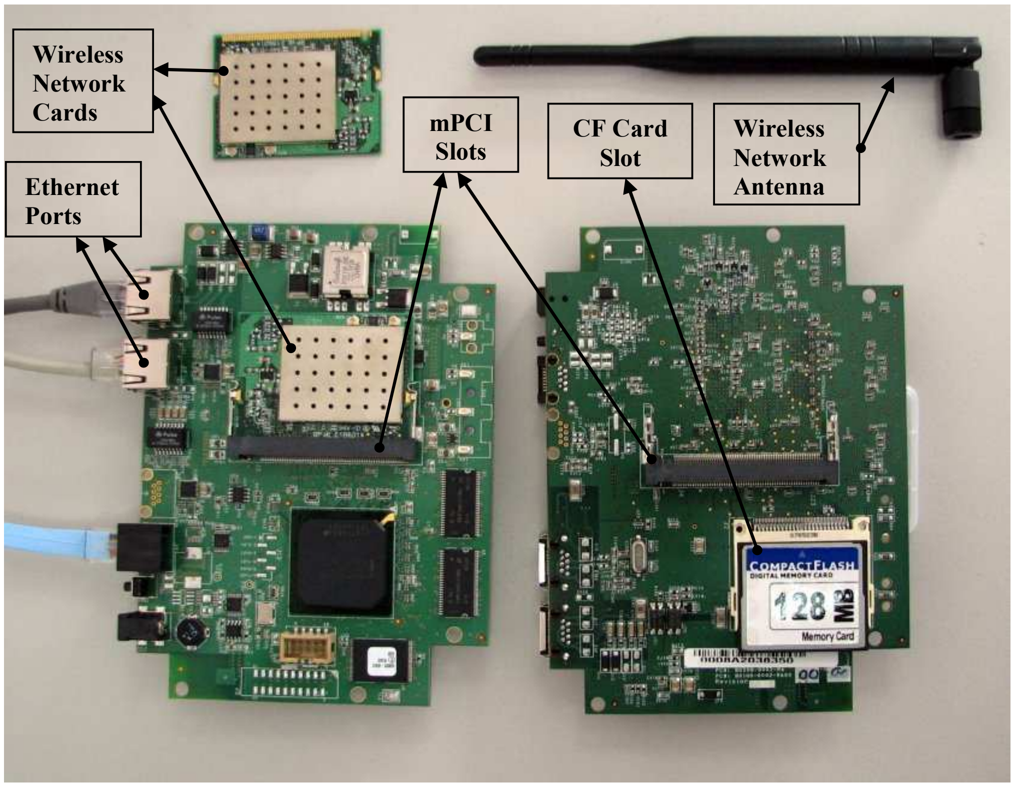


Figure 5.5. Front and back views of the Pronghorn SBC-250 with several peripherals

Pronghorn SBC-250 is a microprocessor board that has 533 MHz Intel IXP425 processor, 64 MB SDRAM, 16 MB Flash ROM and two 10/100 Ethernet interface on it. Also it has 1 CF card slot and 2 mPCI extension slots. It is possible to increase the number of Ethernet interfaces of this card by plugging wired or wireless cards to the mPCI slots. In the Figure 5.5 front and back views of Pronghorn SBC-250 can be seen with the several peripherals.

5.2.2 ALIX.2 Microprocessor Board

The other microprocessor board which is selected to implement the border router with the proposed LSRR-based seamless P2P communication algorithm is ALIX.2 (see Figure 5.6) with the AMD Geode LX800 500 MHz processor.

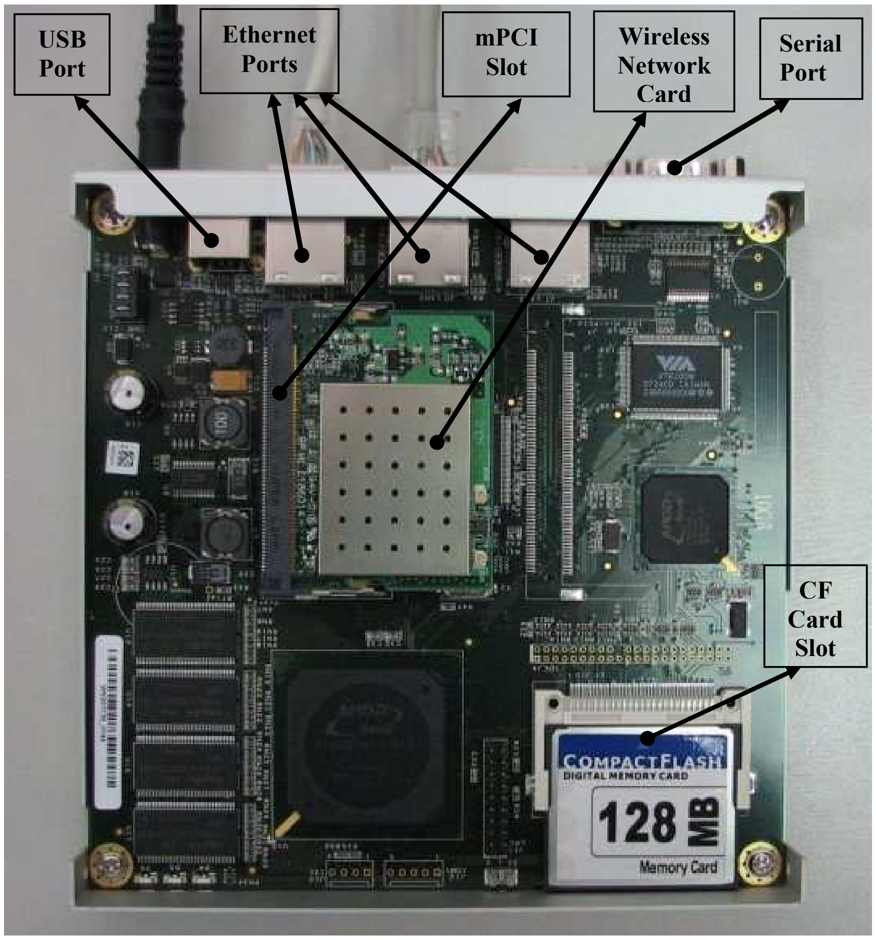


Figure 5.6. ALIX.2 embedded board with several peripherals

The other specifications of the ALIX.2 can be listed as the following:

- 256 KB cache (64K data + 64K instruction + 128K L2)
- 1 to 3 Ethernet channels (Via VT6105M, 10 / 100 Mbit/s)
- 1 or 2 miniPCI sockets for 802.11 wireless cards and other expansion
- 128 or 256 MB DDR SDRAM, 64 bit wide for high memory bandwidth
- 512 KB flash for PC Engines tinyBIOS
- CompactFlash + optional 44 pin IDE header for user's operating system and application
- 7 to 18V (absolute maximum) DC supply through DC jack or passive power over Ethernet
- 1 serial port (DB9 male)
- 2 USB 2.0 ports (optional)
- Header for LPC bus (use for flash recovery or I/O expansion)

Also there is a special aluminum cover for the board which is served by the producer. Thus, with the mounting of the board to its special cover; it becomes a standalone border router (see Figure 5.7).

Figure 5.7 ALIX.2 board covered with its cover

6. CONCLUDING REMARKS

With the Internet structured around NAT boxes to save IPv4 addresses, the internet is in dire need of a clean solution to the P2P UDP communication problem for hosts behind NAT boxes. The lack of a total solution to this important problem has been delaying the ubiquitous deployment of such important services as the Voice over IP using SIP, H.323 among others. There are a lot of partial solutions to the problem including STUN, UPnP, MIDCOM, TURN, ALGs. None of these solutions cover all possible scenarios as each solution works in certain cases but fails in others. IETF is in the process of standardizing a protocol called Interactive Connectivity Establishment (ICE) [27] that aims in testing the environment for available resources and choosing the best possible alternative before the communication begins. With so many partial solutions around, we are yet to come up with a proposal that solves the P2P UDP communication problem and lays the ground for a new Internet architecture with a plan for stepwise transition.

In this thesis it has been presented a new Internet architecture that tries to offer a complete solution to the secure seamless P2P UDP communication problem. The main component of the proposed framework is a new border router that not only performs the traditional NAPT forwarding for the client/server traffic, but also performs LSRR-based forwarding/filtering for the P2P UDP traffic. P2P applications that wish to make use of the new border router functionality are required to learn the IPv4 address of the border router (which can be achieved by a separate control protocol such as SIP), and send UDP packets with IPv4 LSRR option.

Transition to the proposed LSRR-based Internet using IPv4+4 addresses requires a stepwise evolution and is very much feasible. Firstly, no changes to end-host protocol stacks are required. Existing IPv4 protocol stacks of all major operating systems that have been tested, e.g., Windows, Linux, Solaris, has the necessary functionality and the API to send/receive UDP packets having the IPv4 LSRR option. Secondly, no changes to Internet routers are required as our proposal does not change the IPv4 header in any way. All routers see an IPv4 packet that carries a standard IPv4 LSRR option in the IP header. Thus all packets would be delivered to the appropriate border router or host depending on where the packet is destined to. The only requirement is a simple upgrade of border routers with the proposed LSRR-based packet forwarding. Fortunately, border routers need not be upgraded simultaneously. The communication is possible even if just one border router on the path of the communication implements the proposed algorithm. So one can upgrade his/her border router and immediately start enjoying the benefits of seamless P2P UDP communication. The other side of the communication can still be using the legacy NAPT box. Over time, everybody can be expected to upgrade their border routers and people would all have transitioned to an LSRR- based Internet using IPv4+4 addresses. We believe that our proposal is neat and an LSRR-based Internet using IPv4+4 addresses is the way to go in the evolution to the next generation Internet for secure seamless support of P2P UDP communication.

REFERENCES

[1] Postel, J.: Internet Protocol Darpa Internet Program Protocol Specification. RFC 791 (1981)

[2] User Datagram Protocol. RFC 768 (1980)

[3] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., Lear, E.: Address Allocation for Private Internets. RFC 1918 (1996)

[4] Tsirtsis, G., Srisuresh, P.: Network address translation - protocol translation (NAT-PT). RFC 2766 (2000)

[5] Srisuresh, P., Holdrege, M.: IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (1999)

[6] Huston, G.: Anatomy: A Look Inside Network Address Translator. The Internet Protocol Journal, http://www.cisco.com, **4**(3), (Sep 2004)

[7] Deering, S., Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (1996)

[8] Rosenber, J., Weinberger, J., Huitema, C., Mahy, R.: STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs). RFC 3489 (2003)

[9] Guha, S., Takeda, Y., Francis, P.: NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity. SIGCOMM Workshops, Portland, OR (2004)

[10] UPnP Forum: http://www.upnp.org

[11] Srisuresh, P., Kuthan, J., Rosenber, J., Molitor, A., Rayhan, A.: Middlebox Communication Architecture and Framework. RFC 3303 (2002)

[12] Stiemerling, M., Quittek. J., Taylor, T.: Middlebox Communication (MIDCOM) Protocol Semantics. RFC 3989-bis (2007)

[13] Rosenberg, J., Mahy, R., Huitema, C.: Traversal Using Relay NAT (TURN). draft-rosenberg-midcom-turn-08.txt (2005)

[14] Turanyi, Z., Valko, A.: IPv4+4. 10th International Conference on Networking Protocols (ICNP 2002) (2002)

[15] Turanyi, Z., Valko, A., Campbell, A.: Design, Implementation and Evaluation of IPv4+4. ACM SIG-COMM, Stanford, CA (1988) 314-329

[16] Cheriton, D., Gritter, M.: TRIAD: A Scalable Deployable NAT-based Internet Architecture. Technical Report (2000)

[17] Francis, P., Gummadi, R.: IPNL: A NAT-Extended Internet Architecture. ACM SIGCOMM (2001)

[18] Borella, M., Lo, J., Grabelsky, D., Montenegro, G.: Realm Specific IP: Framework. RFC 3102 (2001)

[19] Newport Networks Ltd: Solving the Firewall and NAT Traversal Issues for Multimedia over IP Services. http://www.newport-networks.com (2006)

[20] Ferguson, P., Senie, D.: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (2000)

[21] Postel, J., Reynolds, J.: Comments on the IP Source Route Option. http://www.mirrorservice.org/sites/ftp.isi.edu/in-notes/museum/ip-source-route-comments.txt (1987)

[22] McNab, C.: Network Security Assesment. O'Reilly Publishing (2008)

[23] Linux Netfilter Homepage. http://www.netfilter.org

[24] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (2002)

[25] Handley, M., Jacobson, V.: SDP: Session Description Protocol. RFC 2327 (2004)

[26] Su, Z.: A Specification of the Internet Protocol (IP) Timestamp Option. RFC 781 (1981)

[27] Rosenberg, J.: Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. Work in progress (2007)

[28] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Summers, K.: Session Initiation Protocol (SIP) Basic Call Flow Examples. RFC 3665 (2003)

Appendix 1: Implemented Software

A network driver is implemented as a Linux kernel module for Linux 2.6.22 version during the thesis work. Its source and header files (lsrr_nat.c, nat.h, lsrr.h) are available in the compact disc attached to the back cover.