Implementing IPv4+4 Addressing Architecture with IPv4 LSRR Option for Seamless Peer-to-Peer (P2P) Communication

Cihan Topal and Cuneyt Akinlar

Computer Engineering Department, Anadolu University, Eskisehir, Turkey
cihant@anadolu.edu.tr, cakinlar@anadolu.edu.tr

Abstract. IPv4 architecture is well entrenched with Network Address Translation (NAT) boxes, which cause well-known problems for Peer-to-Peer (P2P) applications. IPv6 would enable end-to-end connectivity when deployed, but the industry has been slow in transitioning to IPv6. IPv4+4 has been suggested as an alternative NAT-extended addressing architecture, where the idea is to assign 64-bit end-to-end globally unique addresses for nodes on private address realms by concatenating the 32-bit globally routable IPv4 address of the realm (border) gateway with the 32-bit private IPv4 addresses of the nodes. While IPv4+4 addressing proposal is neat, existing IPv4+4 implementations require changes to all border gateways and end-hosts, which hinders its deployment. In this paper we show how the IPv4+4 addressing architecture can be implemented by using a modified version of the standard IPv4 Loose Source Record Route (LSRR) option. Our proposal requires no changes to existing IPv4 infrastructure (assuming all IPv4-compliant nodes implement LSRR as required by RFC 791), thus enabling seamless end-to-end communication for P2P applications. We demonstrate packet forwarding with the 64-bit IPv4+4 addresses, and illustrate how the widely-used P2P voice over IP protocol, the Session Initiation Protocol, can make use of our proposal for seamless end-to-end communication.

1 Introduction

While Network Address Translation (NAT) [1,2] has been proposed as a stop-gap measure to slow down IPv4 [13] address depletion, NAT boxes have been widely adapted and deployed all over the IPv4 infrastructure. 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 [15], this architecture creates well-known problems for Peer-to-Peer (P2P) communication.

To enable P2P communication in the presence of NATs, hole punching techniques have been proposed exemplified by STUN [8], STUNT [9,10,11]. The general idea with these proposals 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 cases, they are not reliable and fail to work for all NAT types. Furthermore NATs exhibit different behavior for each transport layer protocol, i.e., UDP [16], TCP [17], SCTP [18], further complicating the matters for P2P communication.

IPv6, 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. Alternatively, several NAT-extended architectures have been proposed to restore end-to-end addressing and connectivity [4,5,6,7]. IPv4+4 is one such addressing architecture, which proposes assigning each node a IPv4-compatible 64-bit globally unique network address formed by concatenating the 32-bit globally routable IPv4 address of the realm (border) gateway with the 32-bit private IPv4 address of the node [4]. Although this addressing proposal is very neat, the existing IPv4+4 implementation [5] requires changes to all realm gateways and end-hosts, and is not feasible in the short term.

In this paper we dwell on the IPv4+4 addressing architecture, and show how it can be implemented with a modified version of the IPv4 Loose Source Record Route (LSRR) option [13,14]. RFC 791 states that all IPv4-compliant nodes *must* implement LSRR [13]. Thus our proposal allows immediate deployment of the IPv4+4 addressing architecture without requiring any changes to existing IPv4 infrastructure, i.e., no changes to IPv4 routers, NAT devices, and hosts, assuming they already implement LSRR. We expect the emerging P2P applications such as voice over IP, interactive gaming, file sharing etc., to benefit from our IPv4+4 implementation proposal for seamless end-to-end connectivity without the need for complex, non-deterministic port prediction techniques. We do not advocate porting the existing client-server applications to IPv4+4 as they already work through NATs.

The rest of the paper is organized as follows: In section 2, we describe the IPv4+4 addressing architecture. Then in section 3 we describe the details of the IPv4+4 implementation given in [5], which proposes a new header structure and requires changes to realm (border) gateways and all end-hosts for deployment. Then in section in section 4, we describe how IPv4+4 can be realized using a modified version of the standard IPv4 Loose Source Record Route (LSRR) option. The advantage of our proposal is that it uses the existing IPv4 infrastructure and requires no changes whatsoever to IPv4 routers, realm gateways or end-hosts. Our proposal further allows communication over all transport layer protocols, e.g., UDP [16], TCP [17] or SCTP [18], and is not specific to any of them. Finally in section 5, we show how the emerging P2P application, voice over IP using the Session Initiation Protocol (SIP) [20], can make use of the IPv4+4 addressing and LSRR-based implementation to provide seamless end-to-end communication. We conclude the paper in section 6.

2 IPv4+4 Addressing Architecture

The basic idea in IPv4+4 is provide the nodes in a private realm, i.e., nodes having private IPv4 addresses, with an end-to-end, globally unique network address. This is achieved by concatenating the public IPv4 address of the realm gateway with the private IPv4 address of the node [4].

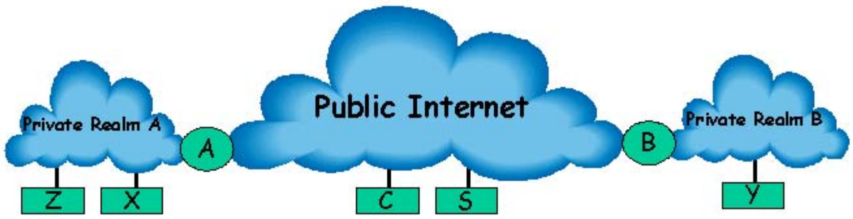

Fig. 1. Example network with 2 private realms connected to the public Internet through realm (border) gateways A and B

Figure 1 shows an example network with 2 private realms connected to the public Internet through realms gateways with public IPv4 addresses A and B. Nodes within the private realms, e.g., X and Y, have non-unique private IPv4 addresses. In the IPv4+4 architecture, node X would have the globally unique IPv4+4 address A.X and node Y would have the globally unique IPv4+4 address B.Y [4]. Nodes in the public Internet already having a globally unique IPv4 address, e.g., node C, simply use 0.0.0.0 as their public IPv4+4 part. That is, node C's IPv4+4 address is 0.C.

3 Related Work - An IPv4+4 Implementation

In this section we talk about an IPv4+4 implementation given in [5]. We detail the proposed header structure, the packet forwarding algorithm and the deployment requirements of the proposed implementation.

3.1 Header Format

The header format for IPv4+4 is shown in Figure 2. The first 20 bytes of the header comprise the regular IPv4 header. The next 12 bytes specify the IPv4+4 extension header. Conceptually this packet structure can be viewed as an IPv4 packet encapsulated within another IPv4 packet. This is what makes IPv4+4 backward compatible. That is, legacy IPv4 routers simply look at the first 20 bytes of the header, which is exactly the same as the IPv4 header, and forward the packet properly using the destination IPv4 address. To legacy routers and hosts IPv4+4 extension header looks like a new transport layer protocol with 12 bytes of header. However IPv4+4 hosts view the full IPv4+4 header as a new network protocol with 64-bit addresses [5].

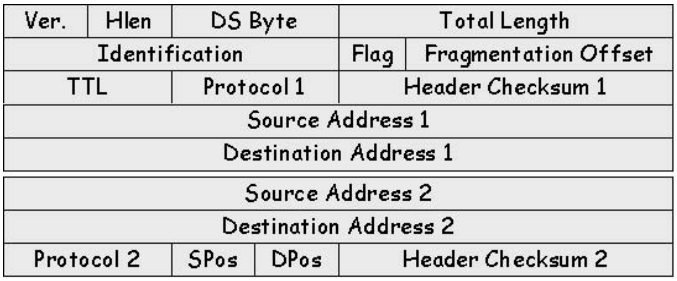

Fig. 2. IPv4+4 header structure

Protocol1 field in IPv4 header indicates IPv4+4 encapsulation, and is set to 233. *Source Address 1* and *2* fields together specify source IPv4+4 address, and *Destination Address 1* and *2* fields together specify the destination IPv4+4 address. *SPos* and *DPos* fields specify which portion of the IPv4+4 address is in the IPv4 header part, i.e., the private IPv4 portion or the public IPv4 portion. *Protocol2* field specifies the transport layer protocol, that is, UDP, TCP or SCTP among others.

3.2 Packet Forwarding

Packet forwarding between two IPv4+4 nodes is achieved as follows: Assume node X in realm A wishes to send a packet to node Y in realm B (refer to Figure 1). The source IPv4+4 address will be A.X and the destination IPv4+4 address will be B.Y[1].

Figure 3(a) shows the address fields in IPv4+4 header as the packet moves from X to Y. When the packet leaves X, IPv4 source address (SA1), contains X's private IPv4 address, and the destination IPv4 address (DA1) contains the public IPv4 address of realm B, where Y is located. IPv4+4 extended source address (SA2) contains the public IPv4 address of realm A, where X is located, and IPv4+4 extended destination address (DA2) contains the private IPv4 address of Y. IPv4 routers in realm A (if any) will look at the IPv4 header, and simply forward the packet up the realm gateway A. The realm gateway is an integral part of the IPv4+4 architecture and has the job of swapping SA1 and SA2 for outgoing packets [5]. Thus after the packet moves past realm gateway A, SA1=A and SA2=X. DA1 and DA2 are untouched. Notice that for all routers in the public Internet, which look at the IPv4 header, this is a packet coming from realm gateway A and going to realm gateway B. Using regular IPv4 forwarding the packet will be delivered to B. The realm gateway B is responsible for swapping DA1 and DA2 for incoming packets [5]. SA1 and SA2 are untouched. After the

[1] How X learns Y's IPv4+4 address is beyond the scope of this paper. In [5] authors suggest augmenting the DNS architecture for this purpose. Interested reader may refer to that paper.

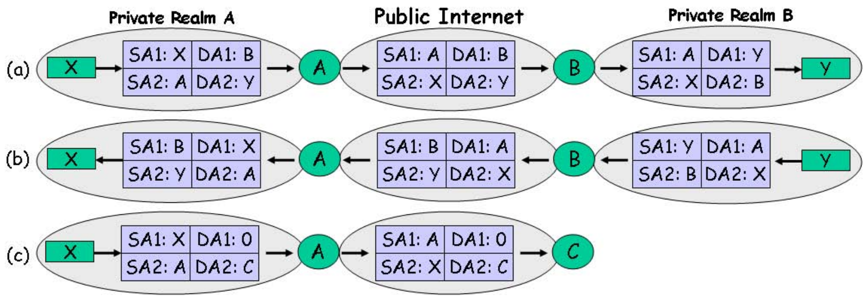

Fig. 3. Packet forwarding in the IPv4+4 architecture: (a) Packet transmission from X to Y, (b) Packet transmission from Y to X, (c) Packet transmission from X to C

packet moves past B, DA1=Y and DA2=B. Thus IPv4 routers in realm B (if any) will correctly deliver the packet to Y.

Figure 3(b) shows the packet flow in the reverse direction, from Y to X. Notice that similar translations occur as the packet moves past the realm gateways. Finally, Figure 3(c) shows a packet transmission to a host with a globally unique IPv4 address. Notice that the public IPv4+4 part of such hosts are set to 0, and the packet is delivered by regular IPv4 forwarding.

3.3 Deployment Requirements and Implementation Details

Deployment of and transition to IPv4+4 requires a stepwise upgrade of NATs and hosts. The first step in a site's transition starts with an upgrade of the NAT device to a IPv4+4 realm gateway. Recall from the previous section that a realm gateway is an integral part of the IPv4+4 architecture, and is responsible for the following tasks: (1) the swapping of addresses in IPv4+4 header, (2) the conversion of ICMPv4 message headers, (3) the participation in routing and filtering of private addresses [5]. So without an upgrade of the NAT to a realm gateway, no host inside a private realm can communicate with an external IPv4+4 host. After the NAT is upgraded with the realm gateway functionality, hosts inside the private realm can start upgrading to IPv4+4.

To program on top of IPv4+4, a new socket API with the new address space can be defined similar to IPv6 socket API. But in [5], authors report how the regular IPv4 sockets API can be used by the bump-in-the stack method [19]. That is, they use the address block 1.0.0.0/8 and assign an IPv4 address from within this range to each IPv4+4 address that the host has communication with. From the application's perspective the application is talking to an IPv4 host, but the translation layer changes between the IPv4 address and the IPv4+4 address. Regardless, using IPv4+4 at a host still requires changes to the host's protocol stack.

Although IPv4+4 header is backward compatible to IPv4, which allows legacy IPv4 routers forward IPv4+4 packets without any changes, transition to IPv4+4 still requires the upgrade of all end hosts and private realm border gateways, i.e., NAT devices. Given prevalent NAT device deployment without IPv4+4 support, it is unrealistic to expect customers to throw away their NAT devices and start buying IPv4+4-compliant border gateways. Additionally, upgrade of all end hosts to IPv4+4 is a daunting task in itself.

4 A Modified Loose Source Record Route (LSRR)-Based Implementation of the IPv4+4 Addressing Architecture

IPv4+4 addressing architecture is a very neat idea. But implementing IPv4+4 by defining a new protocol and header structure, and requiring the upgrade of realm gateways and end-hosts to the new protocol is a serious limiting factor for the transition. What we need is a way to implement IPv4+4 with minimal changes to the existing network components. In this section we show how this can be done with the IPv4 Loose Source Record Route (LSRR) option. We first detail how LSRR works, and then show how a modified version of LSRR can be used to implement IPv4+4. Finally in section 5, we put everything together with a demonstration of the proposed usage of IPv4+4 addressing in the famous voice over IP protocol, the Session Initiation Protocol.

4.1 IPv4 Loose Source Record Route (LSRR) Option

IPv4 Loose Source Record Route (LSRR) option has been defined in RFC 791 [13]. 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 the packet, the following benefits are listed in [14]: (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 [14].

Figure 4 shows an example packet flow using the LSRR option: X wishes to send a packet to Y, but wants 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 should 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

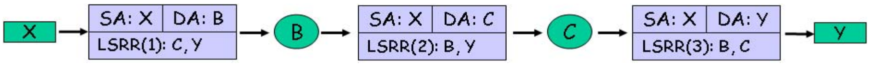

Fig. 4. Packet Transmission with IPv4 Loose Source Record Route (LSRR) option: X sends a packet to Y. The packet makes stops at B and C before arriving at Y.

to 1[2], 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 onwards. 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.

4.2 Modified LSRR to Implement IPv4+4 Addressing Architecture

In this section we show how a modified version of the IPv4 LSRR can be used to implement IPv4+4 addressing architecture. As discussed in section 2, the implementation of IPv4+4 given in [5] requires a lot of changes to the network infrastructure and is not feasible. To start using IPv4+4 addresses, a transition strategy that requires minimal or no changes to the IPv4 infrastructure is needed.

Consider the network in figure 1, and assume that X with IPv4+4 address A.X wishes to send a packet to Y with IPv4+4 address B.Y (We will demonstrate how X learns its and Y's IPv4+4 address in the context of voice over IP (VoIP) in section 5). Figure 5(a) depicts the packet flow with LSRR: X sends out a packet with IPv4 source address SA=X, and IPv4 destination address DA=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 puts the address pointed to by LSRR pointer, the first address, into the destination address of the packet and records its own IPv4 address in the LSRR address list. So when the packet leaves A, SA=X and DA=B. When the packet is delivered to B, B performs a similar LSRR processing. Thus after the packet leaves B, SA=X and DA=Y with LSRR list containing A and B. When Y receives the packet it can realize that the packet is coming from A.X, and if a packet needs to be send back it can follow the reverse LSRR path.

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

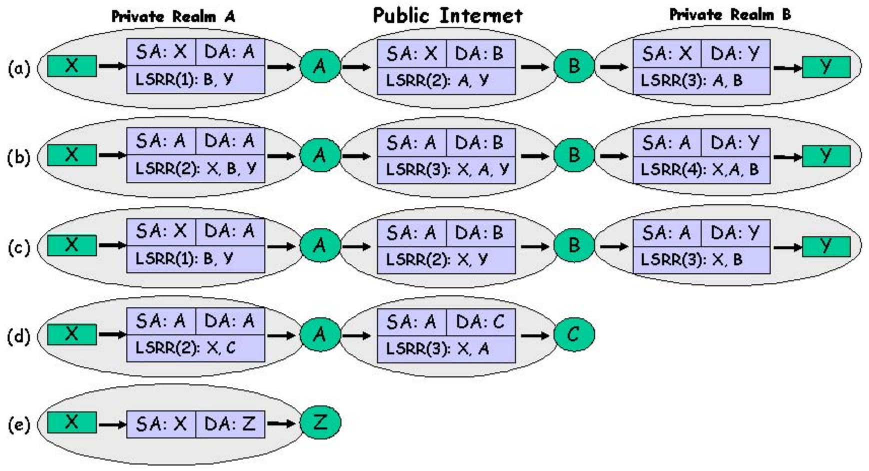

Fig. 5. Using modified LSRR to implement IPv4+4: (a) Packet transmission from X to Y with pure LSRR, (b) Packet transmission from X to Y with modified LSRR, (c) Packet transmission from X to Y where the realm gateway performs special LSRR processing, (d) Packet transmission from X to C, (e) Packet transmission between two hosts, X and Z, in the same private realm (refer to Figure 1)

The above method almost works with one flaw: Notice that due to the nature of LSRR processing, the source IPv4 address is not changed anywhere in the network. This means that when the packet leaves private realm router A and is in the public Internet its SA=X, which is a private IPv4 address. According to RFC 1918 [15], realm routers must perform packet filtering and no packet having a private source or destination address should be sent out to the public Internet. It is also well known that most ISPs perform ingress filtering for incoming packets [12]. So even if A is made to send out the packet with a private source address, the packet is most likely to be dropped by the first ISP router.

For the proposed method to work, all packets leaving private realm A must have a source IPv4 address equal to A. There are two ways to achieve this: (1) The packet leaves the sending host with SA=A, (2) The realm gateway performs special treatment to outgoing packets with the LSRR option. It changes not only the destination address but also the source address before the packet is sent out to the public Internet.

Figure 5(b) illustrates method (1): X sends out a packet with SA=A, DA=A, LSRR(2)=X, B, Y. The LSRR pointer points to the second address, B. In a sense this looks like a packet that has originated from A with SA=A, DA=X, LSRR(1)= A, B, Y, and X just performed regular LSRR processing on the packet changing the packet headers to SA=A, DA=A, LSRR(2)=X, B, Y. The packet now makes it to A, which makes DA=B and LSRR(3)=X, A, Y before sending out the packet to the public Internet. The packet will make one more stop at B, and be finally delivered to Y. With this method, no changes to realm gateways are required as long as they support LSRR processing. But the host must be able to send out packets with SA=A.

Figure 5(c) illustrates method (2): X sends out a packet with SA=X, DA=A, LSRR(1)=B, Y. When A receives this packet, it performs special treatment instead of regular LSRR processing: A 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 the outgoing packet will have SA=A, DA=B and LSRR(2)=X, Y. The rest of the packet delivery involves regular LSRR processing by B and Y. Although this method is ideal, it requires all existing border routers to be upgraded, and therefore it is not feasible at this time.

Figure 5(d) shows X talking to C, which has a globally routable address. As illustrated, this communication is very similar to X talking to Y. Instead less addresses need to be put into LSRR list.

Finally, figure 5(e) shows X talking to Z, which is on the same private network. Clearly the hosts can communicate over standard IPv4 without the need for source based routing.

5 Seamless P2P Communication with the Session Initiation Protocol Using IPv4+4 Addresses

The Session Initiation Protocol (SIP) [20] 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. We assume 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 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 (we only show the relevant fields of the SIP message. SIP message examples are taken from [22]):

```
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 [21]. 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 4.2.

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 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.

6 Concluding Remarks

With the current NAT-based IPv4 Internet architecture nodes behind NAT boxes do not have globally unique end-to-end addresses. This creates problems for P2P applications, where the communication needs to be started from outside of the NAT box. To solve this global identification problem, IPv4+4 addressing architecture has been proposed, which suggests concatenating the global IPv4 address of the border gateway, i.e., the NAT box, with the private IPv4 address of an internal node to create a 64-bit globally unique network-layer address. With nodes on private realms having such 64-bit globally unique identifiers, the question arises as to how packet exchange will work in this addressing architecture. Although authors present one implementation method in [5], it requires changes to all end-hosts and border routers, and this is not feasible.

In this paper we presented a modified IPv4 loose source record route (LSRR)-based implementation of IPv4+4. Our proposal requires no changes to hosts, IPv4 routers or the border gateways, i.e., the NAT boxes, and enables seamless end-to-end P2P communication so long as all IPv4-compliant nodes implement the IPv4 LSRR option correctly, as required by RFC 791 [13].

Assuming proper LSRR support from the current NAT-boxes, we demonstrated how P2P would work in the context of VoIP with the SIP protocol. We are in the process of testing if the currently deployed NAT boxes implement the LSRR option correctly, if at all. If they do support the LSRR option, we believe that a LSRR-based IPv4+4 addressing is the way to go in the evolution to the next generation Internet.

References

1. Tsirtsis, G., Srisuresh, P.: Network address translation - protocol translation (NAT-PT). RFC 2766 (2000)
2. Srisuresh, P., Holdrege, M.: IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (1999)
3. Deering, S., Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (1996)
4. Turanyi, Z., Valko, A.: IPv4+4. In: 10th International Conference on Networking Protocols (ICNP 2002) (2002)
5. Turanyi, Z., Valko, A., Campbell, A.: Design, Implementation and Evaluation of IPv4+4. In: ACM SIGCOMM, Stanford, CA, pp. 314–329. ACM Press, New York (1988)
6. Francis, P., Gummadi, R.: IPNL: A NAT-Extended Internet Architecture. In: ACM SIGCOMM. ACM Press, New York (2001)
7. Cheriton, D., Gritter, M.: TRIAD: A Scalable Deployable NAT-based Internet Architecture. Technical Report (2000)
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., Takeday, Y., Francis, P.: Simple Traversal of UDP through NATs and TCP too (STUNT) `http://nutss.gforce.cis.cornell.edu/`
10. Guha, S., Takeda, Y., Francis, P.: NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity. In: SIGCOMM Workshops, Portland, OR (2004)
11. Biggadike, A., Ferullo, D., Wilson, G., Perrig, A.: NATBLASTER: Establishing TCP connections between hosts behind NATs. In: ACM SIGCOMM Asia Workshop, Beijing, China (2005)
12. Ferguson, P., Senie, D.: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (2000)
13. Postel, J.: Internet Protocol Darpa Internet Program Protocol Specification. RFC 791 (1981)
14. 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`
15. Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., Lear, E.: Address Allocation for Private Internets. RFC 1918 (1996)
16. User Datagram Protocol: RFC 768 (1980)
17. Transmission Control Protocol: RFC 793 (1981)
18. Stream Control Transmission Protocol: RFC 2960 (2000)
19. Tsuchiya, K., Higuchi, H., Atarashi, Y.: Dual Stack Hosts using the Bump-in-the-Stack Technique (BIS). RFC 2767 (2000)
20. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (2002)
21. Handley, M., Jacobson, V.: SDP: Session Description Protocol. RFC 2327 (2004)
22. Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Summers, K.: Session Initiation Protocol (SIP) Basic Call Flow Examples. RFC 3665 (2003)