Published in IET Communications
Received on 28th January 2008
Revised on 5th August 2008
doi: 10.1049/iet-com.2008.0364

ISSN 1751-8628

Enabling peer-to-peer communication for hosts in private address realms using IPv4 LSRR option and IPv4+4 addresses

C. Topal   C. Akinlar
Computer Engineering Department, Anadolu University, Eskisehir, Turkey
E-mail: cakinlar@anadolu.edu.tr

Abstract: Enabling peer-to-peer (P2P) communication for hosts behind network address translation (NAT) boxes is an important and difficult problem. Existing proposals, for example, UPnP, MIDCOM, TURN, STUN, STUNT, P2PNAT, NATBlaster among others, offer only partial, limited and non-deterministic solutions. A framework that offers a complete solution to the P2P communication problem is presented. The proposed framework is based on the use of IPv4+4 addresses and the standard IPv4 Loose Source Record Route (LSRR) option and requires no changes whatsoever to end-host protocol stacks and Internet routers. The only requirement is a simple upgrade of border routers with a new LSRR-based packet-forwarding algorithm for the P2P traffic. The implementation of a Linux-based border router that runs the proposed forwarding algorithm is detailed, and how P2P applications can benefit from this framework is described.

1 Introduction

The current IPv4 [1] Internet architecture with hosts behind network address translation (NAT) [2, 3] boxes is suitable for the client/server type of communication, for example, HTTP, where the server is on the public Internet and the client is on a private address realm [4], but it creates well-known problems for important peer-to-peer (P2P) applications such as VoIP [5, 6] and file sharing [7]. The problem arises from the fact that nodes behind NAT boxes do not have globally routable IPv4 addresses, which makes them unreachable from the Internet.

IPv6 [8], when deployed, would provide each node on the Internet with a globally routable IPv6 address, which would restore the end-to-end connectivity necessary for P2P communication. But the industry has been slow in transitioning to IPv6 as it requires a complete upgrade of the network infrastructure, that is, routers and end-hosts. Ou [9] and Woodyatt [10] discuss the difficulties in transitining to IPv6 and argue that it may take a long time for some parts of the IPv4 Internet to switch to IPv6.

Fig. 1 shows a snapshot of the current Internet architecture. We have two private realms A and B, for example, two home or small office home office (SOHO) networks, connected by the public Internet. X and Y are hosts in private realms A and B, respectively. They have non-routable private IPv4 addresses [4], X and Y respectively. A and B are border routers, shortly BR, 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.

To establish communication for hosts behind NAT boxes, for example, between X and Y in Fig. 1, existing applications use proxies in the Internet, for example, host S in Fig. 1. In this model, both hosts send packets to the proxy, which relays the traffic to the communicating pair. Clearly, this model solves the problem but very inefficiently: All traffic must go through the proxy, which not only puts extra load on the Internet, but also makes the proxy a central point of congestion and failure. Ideally, we want the hosts to exchange packets directly without the help of such proxies, called P2P communication.

To enable P2P communication for nodes behind NAT boxes, hole punching techniques have been proposed exemplified by STUN [11, 12], STUNT [13, 14],

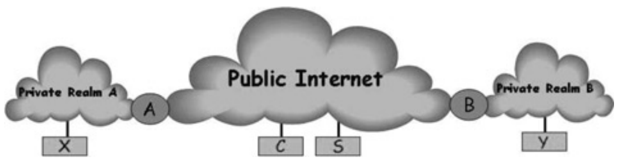

Figure 1 *Two private realms, A and B, connected by the public Internet*

A and B are border routers that perform NAPT and firewall functionality

P2PNAT [15, 16] and NATBlaster [17]. The general idea with these techniques is to let a node behind a NAT box, for example, X, talk to a server S on the public Internet and learn (public IP address:port) pair, for example, A:pa, assigned to its (private IP address:local port) pair, for example, X:px, for a certain session and then disclose this information to the peer, for example, Y, for direct communication. Y employs the same prediction procedure and sends its predicted endpoint parameters, for example, B:pb, to X over an out-of-band channel such as SIP [5, 18]. X would then send the ensuing traffic to B:pb, and Y would send to A:pa for direct P2P communication. Although this port prediction may work if both BRs are cone NATs, it fails when one of the BRs is a symmetric NAT [19]. A symmetric NAT creates a NAT binding based on the source IP address, shortly SA, and source port number, shortly sp, as well as the destination IP address, shortly DA, and destination port number, shortly dp, and is very restrictive [19]. It is known that most BRs today are symmetric NATs [20], which makes hole punching techniques fail in many cases. It should also be noted that cone NATs are susceptible to port scan attacks, which create additional security risks [20, 21]. So making all BRs cone NATs is not an acceptable solution for the P2P communication problem because of a reduced level of security.

For P2P TCP [22] communication, more complex techniques are employed: After both peers initiate a TCP connection and the outgoing SYN packet from a host establishes the necessary NAT state at the host's NAT box, the peers use such techniques as port prediction, low TTL-valued packets or coordination by a third-party server (in the form of spoofed IP packets or sequence number notification by an out-of-band protocol) to establish the TCP connection [13]. Although these techniques help set up the TCP connection in certain limited cases, they are very complex and unreliable.

There are other initiatives exemplified by UPnP [23] and MIDCOM [24, 25] that propose on-demand port allocation for a session by a priori negotiations with the BR. Within these frameworks, a node talks to BR or an agent before initiating a P2P session and allocates the required ports necessary for the communication. The node then discloses the BR 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 [20].

Other alternatives, exemplified by TURN [26], 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.

Additionally, several NAT-extended architectures have been proposed, exemplified by IPv4+4 [27], TRIAD [28], IPNL [29], RSIP [30], 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 BRs, end-hosts and even existing client/server applications, which simply is not feasible.

In this paper, we present a framework based on the use of IPv4+4 addresses [27] and the standard IPv4 Loose Source Record Route (LSRR) option [31] that offers a complete solution to the P2P communication problem. Our proposal requires no changes whatsoever to end-hosts and Internet routers. The only requirement is a simple upgrade of BRs with a new LSRR-based packet-forwarding algorithm for the P2P traffic (described in Section 2.3). The upgraded BR performs NAPT forwarding for the traditional client/server traffic and the proposed LSRR-based forwarding for the P2P traffic. We describe how P2P communication works in the presence of upgraded BRs. We then detail our implementation of a Linux-based BR that runs the proposed LSRR-based forwarding algorithm in Section 3, and describe how P2P applications can benefit from our framework for seamless, secure P2P communication.

2 Proposed framework: IPv4+4 addresses and LSRR-based forwarding

An IPv4+4 address is a backward-compatible, globally unique network address for a node behind a NAT box, and

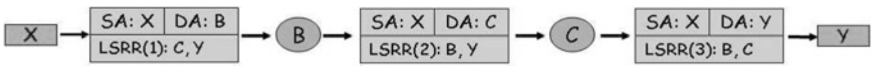

Figure 2 *X sending a packet to Y with IPv4 LSRR option*

The packet stops at B and C before arriving at Y is formed by concatenating the 32-bit globally routable IPv4 address of the BR with the 32-bit private IPv4 address of an internal node [27]. In a sense, the globally unique IPv4 address of the BR identifies the private realm and the private IPv4 address of the host is like an extension number inside the realm. With this convention, IPv4+4 address of X in Fig. 1 is A.X and that of Y is B.Y. Given that each node in a private realm can uniquely be identified by a 64-bit IPv4+4 address, how can we make use of these addresses to solve the P2P communication problem? Turanyi *et al.* [32] propose a solution by defining a new protocol and header structure, and requiring the upgrade of all BRs, end-hosts and even all client/server applications to the new protocol, which simply is not feasible. What we need is a way to make use of IPv4+4 addresses while requiring minimal changes to the existing network infrastructure.

In this section, we show how this can be done with the IPv4 LSRR option [31]. We first describe how LSRR works and then demonstrate how P2P communication can be achieved if BRs employ a modified version of the LSRR forwarding/filtering algorithm. To benefit from our framework, P2P applications must send IPv4 packets with the LSRR option. As transmission of IPv4 packets with LSRR option is part of all major operating system protocol stacks, for example, Windows, Linux, Solaris, our proposal demands no changes whatsoever to end-host protocol stacks or IPv4 routers.

2.1 IPv4 LSRR option

IPv4 LSRR option, defined in RFC 791 [1], 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. DA 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 swapped with DA of the packet [31].

Fig. 2 shows an example packet flow using the LSRR option: X sends a packet to Y and wishes the packet to make stops at B and C on the path to the destination. This is achieved using the LSRR option as follows: When the packet leaves X, SA is set to X, SA = X, and DA is set to the first hop node B's IPv4 address, DA = B. LSRR(1): C, Y indicates that after B the packet must go to C and then to Y. LSSR option pointer (specified in parentheses) indicates where the packet should be sent by the next hop. At the beginning this is set to 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. When B receives the packet, it realises that LSRR option is not exhausted yet. So it swaps DA = B, with the IPv4 address pointed to by the LSSR pointer, that is, C. B then increments the LSRR pointer. Notice that SA is not changed. The new packet with SA = X and DA = C will be delivered to C. C performs LSRR processing similar to B: It swaps DA = C with the IPv4 address pointed to by LSRR pointer, that is, Y, and increments the LSRR pointer. The new packet has SA = X and DA = Y and will be delivered to Y. Y realises that LSRR option has exhausted 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.

2.2 P2P communication with modified LSRR-based border routers

Assume that X having IPv4+4 address A.X wishes to establish bidirectional P2P communication with Y having IPv4+4 address B.Y. Further assume that X wishes to use

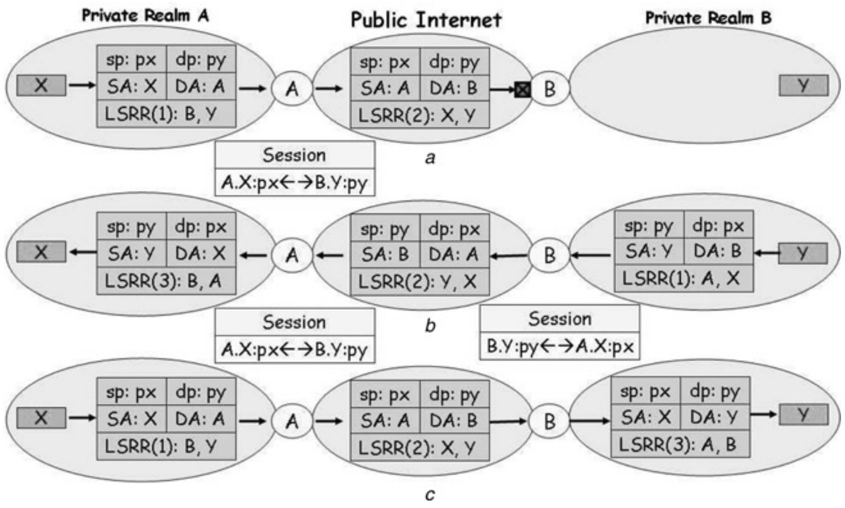

Figure 3 *Using modified LSRR forwarding and IPv4+4 addresses for P2P UDP communication*

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

port px and Y wishes to use port py for this communication. We designate the P2P session key as ⟨A.X:px, B.Y:py⟩. How X and Y learn the IP addresses of their BRs, and how they agree on the P2P communication parameters, is out of the scope of this paper. Existing P2P applications, for example, VoIP using SIP [5, 33] and file sharing [7], use a separate control path (protocol) to establish these parameters before the actual P2P communication begins.

Below we first consider P2P UDP [34] packet exchange between hosts X and Y. We then consider P2P TCP connection establishement and ensuing packet exchange between the same two hosts.

Fig. 3 depicts UDP packet flow between A.X:px and B.Y:py. We assume that after X and Y agree on the communication parameters, X is the first node to send a UDP packet to Y. Fig. 3*a* depicts 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 BR, 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 this packet, it tries to locate the P2P session in its session table. Our BR A maintains a session table for P2P traffic, in addition to a NAT table for the traditional client/server traffic. As this is the first outgoing packet belonging to the session, A creates an entry in the session table. Thus a session is created only by solicited (initiated from inside) outgoing traffic.

The packet now goes through LSRR processing. According to regular LSRR processing, A swaps the IP address pointed to by the LSRR pointer and DA. 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 SA = X. So if A were to send this modified packet to the Internet, it would be dropped at the first ISP router due to ingress filtering [35]. According to RFC 1918 [4], BRs must perform packet filtering and no packet having a private SA or DA should be sent out to the public Internet. This means that we cannot send the packet with SA = X. Therefore we propose a modified LSRR processing algorithm at our BR: A moves the first address in LSRR list, B, to DA, puts SA = 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 sp and dp are not changed.

When B receives this packet, it consults its P2P session table. As B is not aware of the session yet (notice that B will learn about the session after a packet is sent from Y to X), the packet is simply dropped. Thus, unsolicited incoming traffic is simply dropped at our BR similar to existing NAT boxes. Traffic for a session is passed inside only if the BR knows about that session.

Fig. 3*b* illustrates Y sending a packet to A.X:px. When the packet leaves Y, its header contains: sp = py, dp = px, SA = Y, DA = B, LSRR(1): A, X. B receives this packet and creates a new entry in its session table because this is the first packet belonging to the session. B then performs the proposed LSRR processing similar to A, and sends the packet out with headers: sp = py, dp = px, SA = B, DA = A, LSRR(2): Y, X.

When A receives this packet, it locates the session, employs our LSRR processing algorithm and sends the packet to X with the following packet header: sp = py, dp = px, SA = Y, DA = X, LSRR(3): B, A. Host X receives this packet and realises that the LSRR option has exhausted. The packet is then delivered to the application listening to port px. P2P application not only gets the packet, but also learns the reverse path to Y from the LSRR option.

In Fig. 3*c* we illustrate 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.

P2P TCP communication is more difficult than P2P UDP communication: As TCP is a connection-oriented protocol, a connection must be established first before the packet flows begins. Fig. 4 depicts how a TCP connection between X and Y can be established. We have both X and Y as active participants, that is, clients, and BRs assist in connection setup. After the exchange of the session parameters via a seperate control protocol, we assume that X is the first node to send out the TCP SYN packet. This is illustrated in Fig. 4*a*, where the packet from X reaches A, goes through our LSRR processing and then reaches B. By now A is aware of the TCP session ⟨A.X:px, B.Y:py⟩, but B does not know anything about the session yet. So B simply drops the packet. When the TCP SYN packet from Y reaches B (Fig. 4*b*), B creates an entry in its session table and forwards the packet to A. At this point, both A and B know about the session. When A receives this TCP SYN packet, it drops the packet and behaves like a TCP server sending out SYN+ACK packets to both X and Y (Fig. 4*c*). It is easy to see that A can do this because it knows the sequence numbers used by both X and Y. These SYN+ACK messages will make it to X and Y. The TCP layers at X and Y will send out ACK packets to complete the 3-way handshake. In Fig. 4*d*, we depict how the ACK packet from X is processed at BRs A and B before arriving at Y. ACK packet from Y to X will be subject to similar processing.

It is possible that the initial SYN packets from X and Y cross A and B at the same time. In this case, B generates SYN+ACK when the SYN packet from X reaches B, and A generates SYN+ACK when the SYN packet from Y reaches A. Even if A and B forward these SYN+ACK packets generated by the other side, they will simply be

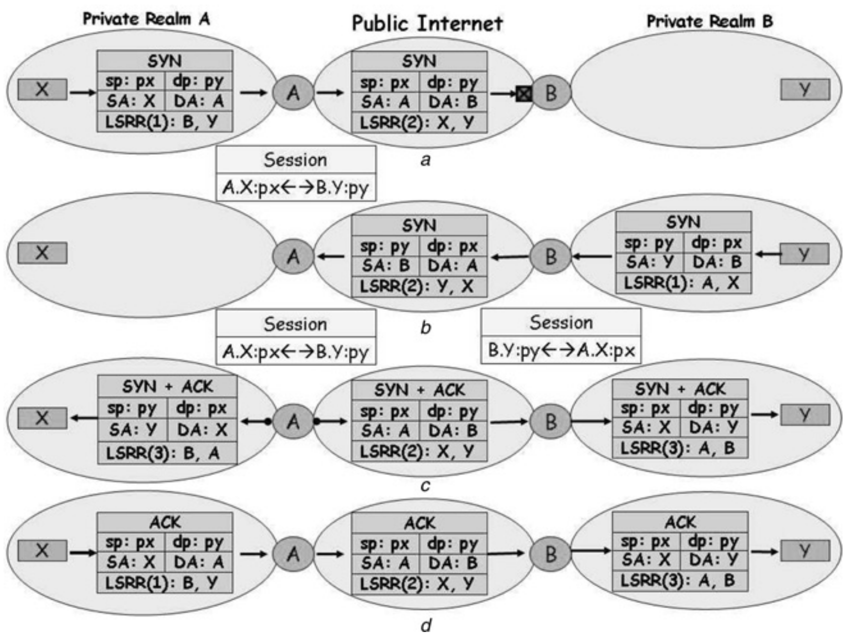

Figure 4 *Using modified LSRR forwarding and IPv4+4 addresses for P2P TCP communication*

X and Y are both active participants, that is, clients
a Initial SYN packet from A.X:px to B.Y:py
b Initial SYN packet from B.Y:py to A.X:px
c BR A generating SYN+ACK packets
d ACK message from A.X:px to B.Y:py

duplicate packets and dropped by X and Y, causing no harm to the connection setup.

After the connection setup, further packet exchange between X and Y is no different than UDP packets. BRs A and B simply apply our LSRR processing algorithm on the packets, and the packets seamlessly make it to both X and Y (refer to Fig. 4*d*). Finally, connection termination, that is, FIN messages, would be processed in the same manner except that BRs A and B would delete the session from their session tables after the connection is closed. Notice that this is no different than NAPT extry deletion algorithm employed by the existing NAT boxes.

2.3 Border router packet forwarding algorithm

In this section, we present our BR packet processing algorithm in a more formal manner using a pseudocode. We divide the algorithm into two cases: (1) When a packet is received from an internal host, that is, when BR receives a packet from one of its LAN interfaces (Fig. 5) and (2) when a packet is received from the Internet, that is, when BR receives a packet from its WAN interface (Fig. 6). The first algorithm consists of two parts, labelled I and II, and the second algorithm consists of three parts, labelled I, II and III.

Part I describes the handling of packets not containing the LSRR option. These packets belong to traditional client/server communication, where the client is in the private 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 is the first packet belonging to the session. So BR creates a NAT entry in the NAT table (step I.2.1) and changes the packet's SA and sp (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 dp are changed, and the packet is forwarded inside (step I.3).

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

Figure 5 *Algorithm for processing packets received from a LAN interface*

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

III. ProcessPacket(P[SYN, 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. Send P[SYN+ACK, SA=Y, DA=X, sp=py, dp=px, LSRR(3): B, A]
    III.4. Send P[SYN+ACK, SA=A, DA=B, sp=px, dp=py, LSRR(2): X, Y]
```

Figure 6 *Algorithm for processing packets received from the WAN interface*

Part II describes the handling of packets that contain an LSRR option with two addresses in the option body. These packets belong to P2P communication where one peer is on a private realm and the other peer is on a different private realm, for example, P2P communication between A.X and B.Y in Fig. 1. When BR receives such a packet, it tries to locate the session in the session table (step 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 SA, DA and the LSRR option are changed, and the packet is forwarded inside (step II.3).

We see from steps I.2 and II.2 in Fig. 6 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.

Step III in Fig. 6 describes the special case of BR-assisted TCP connection establishment. When BR receives a SYN packet, it first locates the session. If the session is not found, BR simply drops the packet. If the session is found and the connection is not established yet, BR generates two SYN+ACK packets (steps III.3 and III.4), one for each peer, necessary for the peers to complete the 3-way handshake. After the peers receive these SYN+ACK packets, they would send out ACK packets to complete the connection setup (refer to Fig. 4). Further packets belonging to the session would be processed by step II in both algorithms. So, after the TCP connection is established, both P2P TCP and UDP packets are processed in the same manner.

3 Implementation details and concluding remarks

To implement and test the proposed framework, we set up the network depicted in Fig. 1. We have two private realms A and B connected by the Internet. There is one host X in realm A and one host Y in realm B, both of which are PCs running Windows.

BRs A and B are emulated by PCs or router boards with at least two interfaces, one attached to the Internet and the other(s) attached to the private realm. We used two different router boards: PC Engines ALIX.2C3 [36] and Pronghorn SBC-250 [37]. ALIX.2C3 is a router board with 500 MHz AMD processor and three Ethernet interfaces. Pronghorn SBC-250 is a router board with an Intel IXP425 processor and two Ethernet interfaces. Both of these router boards run the latest 2.6.x Linux kernel [38].

We have designed a Linux driver that implements BR functionality. Our driver, developed as a Linux-kernel loadable module for 2.6 kernels, implements the packet-forwarding algorithms in Figs. 5 and 6. The driver consists of about 1500 lines of C code and uses the Linux Netfilter architecture [39]. Netfilter is a set of hooks in the Linux kernel protocol stack that allows callback functions to be registered. We attach two hooks at BR WAN interface, one for incoming packets and one for outgoing packets. When a packet arrives from the WAN, Netfilter gives the packet to our 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.

Our implementation of the proposed framework demonstrates that if BRs keep a P2P session table and employ our LSRR processing/filtering algorithm, secure seamless P2P communication can easily be achieved. Although our proposal demands the upgrade of all existing BRs, we require no changes to end-host protocol stacks or IPv4 routers. All major operating systems that we tested, for example, Windows, Linux, Solaris, have the necessary functionality and the API to send/receive IPv4 packets having the LSRR option using the existing sockets API [40]. So our implementation requires no changes whatsoever to host TCP/IP protocol stacks. To benefit from our framework, a P2P application must learn the IPv4 address of its BR and send IPv4 packets using the LSRR option. We also note that our BRs are as secure as the existing 'symmetric' NAT boxes [19]: (1) A new session is created only by solicited traffic, that is, traffic originating from inside the realm and (2) an incoming packet is passed inside only if all session parameters match an existing session.

We have shown that an LSRR-based Internet using IPv4+4 addresses seamlessly enables P2P communication. So, we believe that an LSRR-based Internet using IPv4+4 addresses is the way to go in the evolution to the next-generation Internet until IPv6 is fully deployed.

4 Acknowledgment

This work was partially supported by Turkish Science and Technology Research Institute (TUBITAK) grant 107E166.

5 References

[1] POSTEL J.: 'Internet protocol Darpa Internet program protocol specification', RFC 791, 1981

[2] TSIRTSIS G., SRISURESH P.: 'Network address translation – protocol translation (NAT-PT)', RFC 2766, 2000

[3] SRISURESH P., HOLDREGE M.: 'IP network address translator (NAT) terminology and considerations', RFC 2663, 2000

[4] REKHTER Y., MOSKOWITZ B., KARRENBERG D., DE GROOT G., LEAR E.: 'Address allocation for private internets', RFC 1918, 1996

[5] ROSENBERG J., SCHULZRINNE H., CAMARILLO G., JOHNSTON A., PETERSON J., SPARKS R., HANDLEY M., SCHOOLER E.: 'SIP: session initiation protocol', RFC 3261, 2002

[6] Wikipedia IP Phone page, http://en.wikipedia.org/wiki/SIP_Phone, accessed August 2008

[7] ABERER K., HAUSWIRTH M.: 'An overview on peer-to-peer information systems'. Workshop on Distributed Data and Structures (WDAS 2002), Paris, France, March 2002

[8] DEERING S., HINDEN R.: 'Internet protocol', Version 6 (IPv6) specification', RFC 2460, 1996

[9] OU G.: 'The truth about the IPv6 transition', http://blogs.zdnet.com/Ou/?p=367, accessed August 2008

[10] WOODYATT J.H.: 'IPv6 and NAT . . . again!', http://jhw.vox.com/library/post/ipv6-and-nat-again.html, accessed August 2008

[11] ROSENBERG J., WEINBERGER J., HUITEMA C., MAHY R.: 'STUN – simple traversal of user datagram protocol (UDP) through network address translators (NATs)', RFC 3489, 2003

[12] GUHA S., TAKEDA Y., FRANCIS P.: 'NUTSS: a SIP-based approach to UDP and TCP network connectivity'. SIGCOMM Workshops, Portland, OR, 2004

[13] GUHA S., FRANCIS P.: 'Characterization and measurement of TCP traversal through NATs and firewalls', Internet Measurement Conference, Berkeley, CA, 2005

[14] GUHA S.: 'STUNT – simple traversal of UDP through NATs and TCP too: work in progress', http://nutss.gforge.cis.cornell.edu/pub/draft-guha-STUNT-00.txt

[15] FORD B., SRISURESH P., KEGEL D.: 'Peer-to-peer communication across network address translators'. USENIX Annual Technical Conference, Anaheim, CA, 2005

[16] EPPINGER J.L.: 'TCP connections for P2P Apps: a software approach to solving the NAT problem'. Technical Report, Carnegie Mellon University, CMU-ISRI-05-104, Pittsburg, PA, 2005

[17] BIGGADIKE A., FERULLO D., WILSON G., PERRING A.: 'NATBLASTER: establishing TCP connections between hosts behind NATs'. ACM SIGCOMM ASIA Workshop, Beijing, China, 2005

[18] JOHNSTON A., DONOVAN S., SPARKS R., CUNNINGHAM C., SUMMERS K.: 'Session initiation protocol (SIP) basic call flow examples', RFC 3665, 2003

[19] HUSTON G.: 'Anatomy: a look inside network address translator', *Internet Protocol J*, 2004, **7**, (3), pp. 2–32

[20] Newport Networks Ltd: 'Solving the firewall and NAT traversal issues for multimedia over IP services', http://www.newport-networks.com, accessed August 2008

[21] MCNAB C.: 'Network security assesment' (O'Reilly Publishing, 2008, 2nd edn.)

[22] POSTEL J.: 'Transmission control protocol', RFC 793, 1981

[23] UPnP Forum, http://www.upnp.org, accessed August 2008

[24] SRISURESH P., KUTHAN J., ROSENBERG J., MOLITOR A., RAYHAN A.: 'Middlebox communication architecture and framework', RFC 3303, 2002

[25] STIEMERLING M., QUITTEK J., TAYLOR T.: 'Middlebox communication (MIDCOM) protocol semantics', RFC 3989-bis, 2007

[26] ROSENBERG J., MAHY R., HUITEMA C.: 'Traversal using relay NAT (TURN)', draft-rosenberg-midcom-turn-08.txt, 2005

[27] TURANYI Z., VALKO A.: 'IPv4+4'. 10th Int. Conf. Networking Protocols (ICNP 2002), 2002

[28] CHERITON D., GRITTER M.: 'TRIAD: a scalable deployable NAT-based Internet architecture'. Technical Report, 2000

[29] FRANCIS P., GUMMADI R.: 'IPNL: a NAT-extended Internet architecture'. ACM SIGCOMM, 2001

[30] BORELLA M., LO J., GRABELSKY D., MONTENEGRO G.: 'Realm specific IP: framework', RFC 3102, 2001

[31] 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, accessed August 2008

[32] TURANYI Z., VALKO A., CAMPBELL A.: 'Design, implementation and evaluation of IPv4+4'. ACM SIGCOMM, Stanford, CA, 1988, pp. 314–329

[33] HANDLEY M., JACOBSON V.: 'SDP: session description protocol', RFC 2327, 2004

[34] POSTEL J.: 'User datagram protocol', RFC 768, 1980

[35] FERGUSON P., SENIE D.: 'Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing', RFC 2827, 2000

[36] PC Enginees ALIX2C3, http://www.pcengines.ch/alix2c3.htm, accessed August 2008

[37] Pronghorn SBC-250, http://www.adiengineering.com, accessed August 2008

[38] Linux Homepage, http://www.linux.org, accessed August 2008

[39] Linux Netfilter Homepage, http://www.netfilter.org, accessed August 2008

[40] STEVENS W.R., FENNER B., RUDOFF A.M.: 'UNIX(r) network programming: the sockets networking API' (Addison-Wesley, 2003, 3rd edn.)