P2PSIP Working Group	E. Cooper	
Internet-Draft	A. Johnston	
Intended status: Standards Track	P. Matthews	
Expires: August 28, 2008	Avaya	
	February 25, 2008	

TOC

An ID/Locator Architecture for P2PSIP
draft-matthews-p2psip-id-loc-01

Status of this Memo

Abstract

This document describes an architecture where peers in an peer-to-peer overlay use special IP addresses to identify other peers. Two of the advantages of this approach are that (a) most existing applications can run in an overlay without needing any changes and (b) peer mobility and NAT traversal are handled in a way that is transparent to most applications.

Table of Contents

---

1. Introduction

TOC

This document describes a scheme whereby the applications running on a peer can use a special IP addresses, called "LSIs" (Locally Significant Identifiers), to identify other peers in the peer-to-peer overlay, rather than using real IP addresses or peer IDs. Using these LSIs brings the following advantages:

- An LSI is unique, unlike the real IP address of most peers (which is often a private IP address);
- An LSI can be used in the Socket API without change, unlike 160-bit peer IDs;
- Applications using LSIs do not have to worry about NAT traversal, mobility, or multi-homing, since these are handled by a helper application.

The scheme effectively turns the overlay into a VPN. Like other VPNs, it can be implemented so that most applications are unaware that they are using the VPN. Only applications that want to take advantage of the special properties of the overlay need to be aware.
Though not discussed further in the document, this scheme can be trivially extended to handle clients as well.
This scheme is not a Peer Protocol in itself. Rather, it is an enhancement to a Peer Protocol.
This approach can be compared with the approach taken by many of the other proposals in P2PSIP (e.g., RELOAD, ASP, P2PP, and XPP/PCAN). In these proposals, peers are identified with bitstrings that do not look like addresses, forcing applications that want to run in an overlay to use a new (as yet unspecified) API, rather than the existing Socket API. Furthermore, though these proposals handle NAT traversal for the Peer protocol, they do not handle NAT traversal for applications, forcing each application to invent its own ICE variation. None of these proposals currently consider mobility at all. All of this means that any application that wants to run in an overlay requires significant modification.
This scheme grew out of the authors' previous efforts to adapt HIP to peer-to-peer overlays. More details on the relationship of this work to HIP is given in Section 10 (Relationship to HIP).

---

2. Overview/Example

TOC

This section gives an overview of how the scheme works. It is non-normative.
This overview is in the form of an extended example and assume a particular implementation approach. While not fully general, experience has shown that this is a good way to explain the concepts.
Consider a peer-to-peer overlay. This overlay is assigned a domain name by the peer that created it; say it is "example.com". This overlay has a number of peers, of which there are three of interest, called "venus", "earth", and "mars". Each peer in the overlay is assigned a domain name underneith the "example.com" domain; for example "mars.example.com". The domain names of peers are NOT stored in DNS. Instead, each peer stores a mapping between its domain name and its peer ID in the overlay's Distributed Database.
The machines Venus and Mars are using popular commercial operating systems. To allow them to join the overlay, a user named Wilma has installed some peer-to-peer software. This software has two parts. One part an implementation of the Peer Protocol with some ID-LOC extensions, the other part is a TAP device driver http://en.wikipedia.org/wiki/TUN/TAP. This is shown in the following figure.

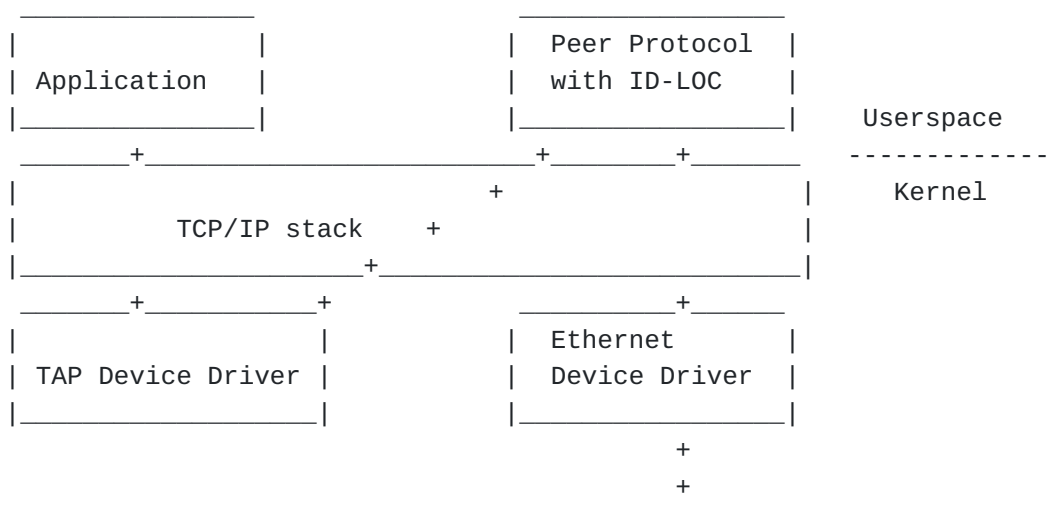


Figure 1

The "+" signs show the typical path of an application data packet traveling to/from a remote peer. Packets sent by the application pass down through the kernel's TCP/IP stack. Packets satisfying certain criteria are intercepted by the TAP driver and passed to the Peer Protocol, which modifies them before sending them back down through the kernel's TCP/IP stack and out through the Ethernet device driver. In the reverse direction, incoming packets arrive at the Ethernet device driver and pass up through the TCP/IP stack and are delivered to the Peer Protocol. There they are modified and then passed to the TAP driver which reinjects them into the bottom of the TCP/IP stack. They then pass up through the TCP/IP stack and are delivered to the application.

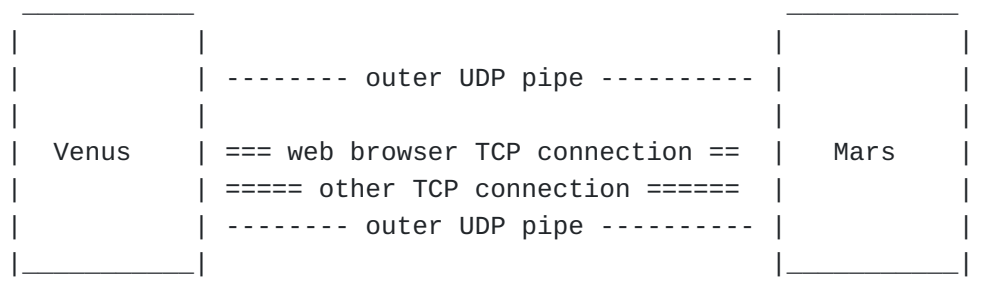
Wilma wishes to view a website on the machine Mars. To do this, she opens a popular web brower and enters "http://mars.example.com" into the address bar. This causes the web browser to do gethostbyname() on "mars.example.com", which in turn causes a DNS query packet to be formed and sent down the TCP/IP stack. It is important to note that this web browser has not been modified in any way, and thus has no knowledge that it is operating in a peer-to-peer overlay.

The DNS query packet is intercepted by the TAP driver, which passes it to the Peer Protocol process. The Peer Protocol notices that the domain name is in the "example.com" overlay which Venus is currently a member of. So the Peer Protocol does a Distributed Database query for "mars.example.com" and gets back the 160-bit peer ID of Mars.

The Peer Protocol process stores the peer ID of Mars and assigns it an LSI (call it Y). The Peer Protocol process then creates a DNS response packet indicating that "mars.example.com" maps to Y. This packet is

passed to the TAP driver, which injects it into the bottom of the TCP/IP stack.
The result is that the Wilma's web browser gets back the LSI "Y" as the address of Mars.
Wilma's web browser then issues a connect() call to create a TCP connection to "Y". This causes the TCP/IP stack to send a SYN packet with destination "Y". This packet is intercepted by the TAP driver and passed to the Peer Protocol process.
The Peer Protocol stores the TCP SYN while it sets up a UDP connection between Venus and Mars. This UDP connection is established using the connection establishment procedures of the peer protocol and uses ICE to traverse any NATs between Venus and Mars. This UDP connection is then uses as a "pipe" to carry all traffic between Venus and Mars encapsulated inside it.

> This approach is known as the "Outer UDP encapsulation". An alternative approach, known as the "Null encapsulation" is described in the normative text below.



Once this UDP pipe is established, the Peer Protocol process on Venus then modifies the TCP SYN so that it will travel inside the "UDP pipe" to the machine Mars. By doing this, the web browser and the web server do not need to run ICE or deal with peer IDs.
At Mars, the UDP header is removed and the TCP SYN is then passed to the TAP driver on Mars, which passes it up through the TCP/IP stack. Subsequent TCP packets between Venus and Mars are also encapsulated inside UDP and sent along the pipe.

---

3. Terminology

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.) [RFC2119].

Readers are expected to be familar with [I-D.ietf-p2psip-concepts] (Bryan, D., Matthews, P., Shim, E., Willis, D., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP," July 2008.) and the terms defined there.
This document defines the following terms:

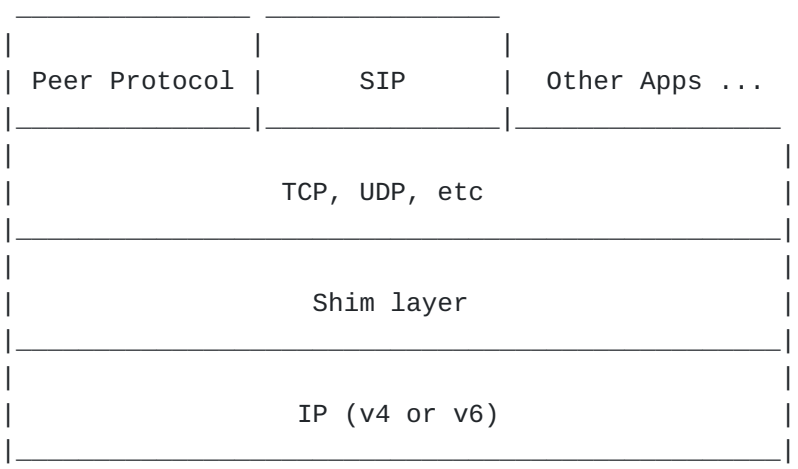
> **LSI:** An IP address uses to identify a peer in the overlay.
>
> **Outer UDP Encapsulation** An encapsulation scheme for packets travelling between two peers in the overlay that insert a UDP header and a demux header between the IP header and the existing transport header.
>
> **Null Encapsulation** An excapsulation scheme for packets travelling between two peers in the overlay that does not insert any extra headers, but instead modifies fields in the existing IP and transport headers.

---

4. Details

TOC

Figure X shows the conceptual relationship between the parts discussed in this section.



In this architecture, the Peer Protocol is responsible for creating the mapping between LSIs and real addresses, while the Shim layer is responsible for doing the translation on a packet-by-packet basis as well as adding any necessary encapsulation. More details on these roles can be found below.

TOC

4.1. LSI

An LSI is either:

- An IPv4 address selected from a range to be allocated by IANA (likely a /16), or

- An IPv6 address selected from a range to be determined (perhaps the ORCHID range [RFC4843] (Nikander, P., Laganier, J., and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)," April 2007.)).

An LSI has local significance only.
Applications can freely intermix LSIs with ordinary ("real") addresses. For example, an application can use LSIs to identify nodes in the overlay, and real addresses to identify nodes off the overlay.

4.2. Peer Protocol

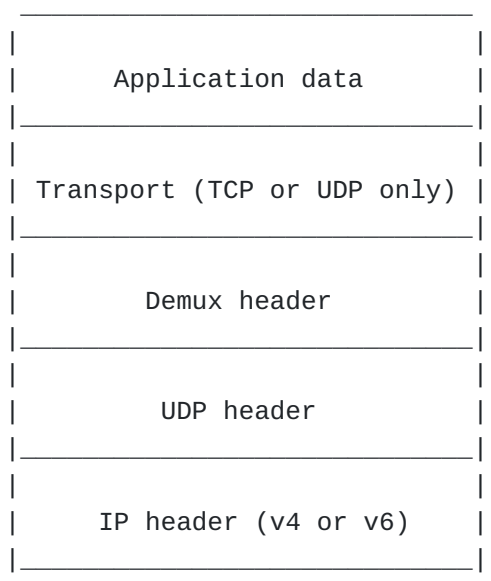
TOC

The job of the Peer Protocol in this scheme (in addition to its other duties of managing the overlay and implementing the Distributed Database [I-D.ietf-p2psip-concepts] (Bryan, D., Matthews, P., Shim, E., Willis, D., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP," July 2008.)) is to establish connections between peers and to manage the mappings between LSIs and real addresses. To do this, the Peer Protocol does an ICE exchange with the destination peer to negotiate a set of addresses and ports to use for the data traffic. The stimulus for doing this ICE exchange is an indication from the Shim layer saying that is has no set of real addresses to use for a given destination LSI (cf. an ARP cache miss). The Peer Protocol then does an ICE exchange with the destination peer, routing the Offer/Answer though other peers in the overlay. Once the exchange has completed, the Peer Protocol installs the appropriate mapping entry into the Shim layer.

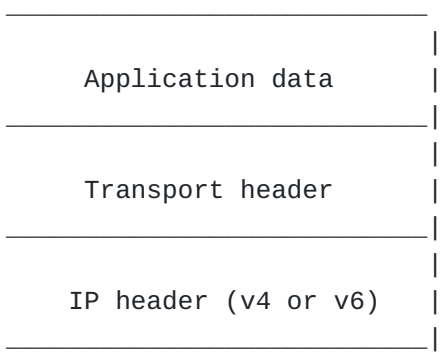
4.3. Shim Layer

TOC

The shim layer is a new layer introduced between the IP layer and the transport layer. It has two functions: translating LSIs to/from real addresses, and adding any necessary encapsulation.
There are two forms of encapsulation: null encapsulation and outer-UDP encapsulation.

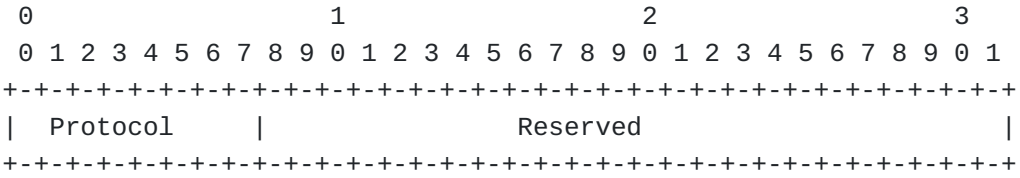


Outer-UDP Encapsulation



Null Encapsulation

The Demux header looks like:



Here the protocol field indicates which transport (or other) protocol follows, and uses the same codepoints as used for the ‘protocol’ field in the IPv4/IPv6 header.

The null encapsulation adds no extra bytes but simply translates LSIs to real addresses and modifies port numbers as necessary to traverse NATs. The null encapsulation is very similar to existing protocol stacks, but requires more work to set up and maintain because each connection requires its own set of ICE connectivity checks.

By contrast, the Outer-UDP encapsulation adds a UDP header plus a 4-byte demux header between the IP header and the transport header. The Outer-UDP encapsulation multiplexes all connections between two given nodes inside a single UDP "pipe". Because intervening NATs see only the outer UDP header, this encapsulation requires only one ICE exchange (to set up the outer pipe), regardless of how many connections there are inside the pipe.

The Outer-UDP encapsulation can be used with all transport protocols, while the null encapsulation can only be used with UDP and TCP.

To explain the mapping and encapsulations in more detail, consider a transport layer PDU is sent from X:x to Y:y, where X is the LSI of the local host, Y is the LSI of the remote host, and x and y are the port numbers allocated off of these identifiers. For both encapsulations,

the Peer Protocol will have used ICE to determine a corresponding set of real addresses and ports.
For the null encapsulation, each transport layer 5-tuple (transport protocol,X,x,Y,y) will have a corresponding set of real addresses and ports (X’,x’,Y’,y’). When sending, the port numbers x and y in the transport header are replaced with x’ and y’, and an IP header is added containing addresses X’ and Y’ is added. (TBD: Are the addresses in the transport layer pseudo-header also replaced?). The reverse replacement is done when receiving a PDU.
If either X or Y change their real address, then an ICE exchange is required to determine a new 5-tuple for each connection. For UDP, this new 5-tuple is simply used in place of the old.

> OPEN ISSUE: For TCP, this doesn’t work, since generating the new 5-tuple requires a new TCP handshake. This seems to imply that the TCP layer has to be aware of the change in address. So what do we do? Do we just say “don’t use null encapsulation for TCP if you want mobility to work”? Or do we figure out how to make this work?

For the outer-UDP encapsulation, there is a single 5-tuple (UDP,X’,x’,Y’,y’) for each (X,Y) pair. When sending, the transport header is not modified, instead a demux header and a outer UDP header is added. Ports x’ and y’ are inserted in the outer UDP header, and an IP header containing addresses X’ and Y’ is added.
Mobility is simpler with the Outer-UDP encapsulation. In this case, only a single ICE exchange is required, and the new 5-tuple is simply used in place of the old. There are no TCP concerns in this case, since the TCP header is never modified.

---

5. Domain Names

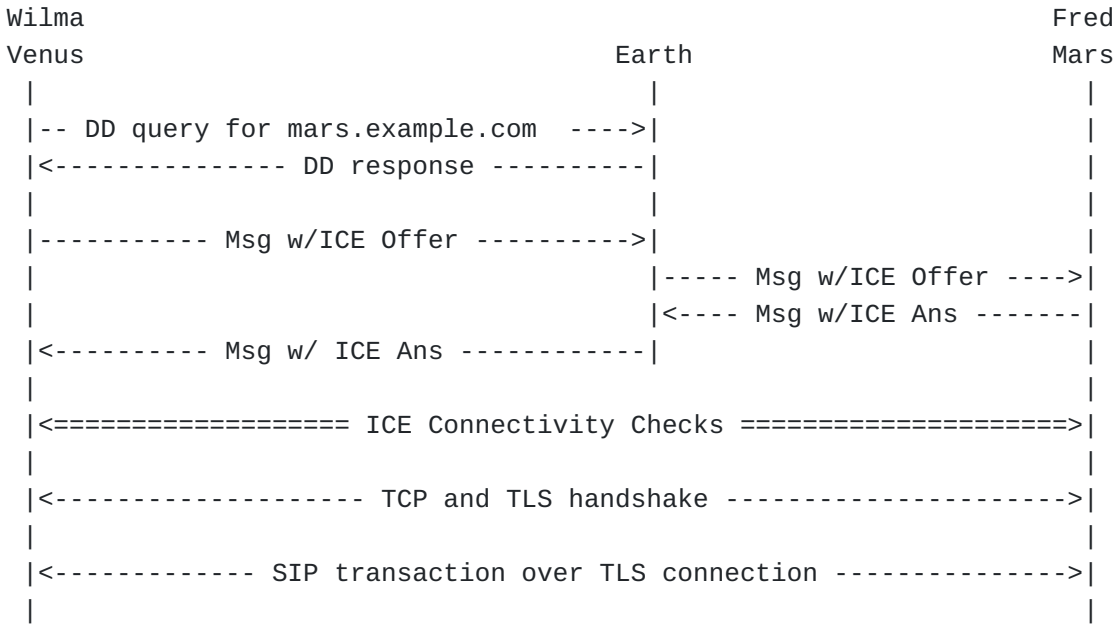
TOC

Each overlay is assigned a domain name by the peer that creates the overlay. This can be any domain name that the peer has authority over. Each peer is assigned a unique domain name underneith the overlay's domain name. This document does not specify how this assignment is done, but one option might be to use the peer's machine name as the label in front of the overlay domain name, and then use some scheme to break ties.
Each peer MUST store a mapping between its domain name and its peer ID in the Distributed Database. The peer's domain name MAY be stored in DNS as well.

TOC

6. Example

In this section, we show a SIP call between two UAs in an overlay. This example illustrates how this scheme allows applications to work in an overlay without being aware of that fact. The two SIP UAs in this example use standard client-server SIP to communicate, without needing any SIP extensions.
IMPORTANT NOTE: Without extensions to SIP, there is no way to do an AOR to contact URI lookup using the Distributed Database. So in this example, Wilma calls Fred by specifying Fred's machine name, using the domain name scheme described in the previous section. With this caveat, everything works with SIP as it is today.
The figure below shows the call flow for this example.



This example shows three machines, named "Venus", "Earth", and "Mars" which are part of a larger overlay named "example.com". Wilma is on Venus, and Fred is on Mars.
Wilma initiates the call by typing in "sips:fred@mars.example.com" into her UA. Wilma's UA does a gethostbyname() call to resolve "mars.example.com" and this is resolved by doing a Distributed Database lookup. In this example, it turns out that the corresponding resource record is stored on the machine "Earth". As a result, an LSI for the peer Mars is returned from the gethostbyname() call to Wilma's UA.

> NOTE: The Peer Protocol allocates an LSI and remembers that it maps to the machine named "mars.solar-system.p2p" which has the peer id learned from the response.

Wilma's UA then issues a connect() to this LSI. This causes TCP to send a SYN to this LSI. Since there is currently no direct connection

between Venus and Mars, the Shim layer finds no mapping for this LSI and thus generates an indication to the Peer Protocol.
The Peer Protocol layer on Venus now does an ICE offer/answer exchange with the Peer Protocol layer on Mars. The Offer is sent on the existing connection to Earth, which forwards it to Mars, and the Answer is returned in the same way. ICE connectivity checks are then done, and the result is a tuple of real addresses and ports for the connection.
If null encapsulation is used, then the TCP connection was established as part of the ICE connectivity checks. This new connection is used only for SIP signaling, and subsequent connections require a new offer/answer exchange.
But if Outer-UDP encapsulation is used, then all the ICE connectivity checks do is establish a UDP "pipe" between the two peers, and the TCP and TLS handshakes must still be done inside that pipe (as shown above). However, this UDP pipe can be used for all traffic between Venus and Mars, including subsequent RTP packets) without the need of subsequent offer/answer exchanges.

---

7. IANA Considerations

TOC

TBD.

---

8. Security Considerations

TOC

TBD.

---

9. Appendix: Discussion of Design Choices

TOC

This appendix discusses the thinking around some of the design choices made.

9.1. LSIs have Local Significance

TOC

In the design presented here, the LSIs presented to applications have local significance only. For IPv4, this seems to be the only reasonable choice, as it would be difficult to get an IPv4 block of addresses large enough to be of wider significance. However, for IPv6, a wider scope would be possible, and that option was considered. In particular, it would have been possible to use a globally scoped identifier, like

the HIT of HIP. At first blush, it seems that using a globally scoped identifier would allow an applications to send the identifier (embedded in protocol messages) to an application on other nodes and have that identifier make sense.
However, an examination of the details shows that there are problems with this approach. Say a node X has an indentifier for node Z (e.g., a HIT) and sends its to node Y. For Y to be able to use this identifier, it must know how to establish a connection with node Z. If node Y is in multiple overlays, then Y has no idea which overlay to search to find node Z. It is this difficulty that led us to the decision to make LSI have local significance only.

---

10. Relationship to HIP

TOC

The fundamental concept in this document, that of an identifier for a node which is distinct from the node's real addresses, has been adopted from HIP. In HIP, this identifier (known as a HIT [I-D.ietf-hip-base] (Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol," October 2007.)) is always an IPv6 identifier, and has global scope and cryptographic properties, making it computationally hard for an second node to steal a node's identity. (Current HIP implementations also implement an IPv4 identifier as a local identifier, but the properties of this IPv4 identifier are not currently specified anywhere).

---

11. References

TOC

11.1. Normative References

TOC

[I-D.ietf-mmusic-ice]	Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," draft-ietf-mmusic-ice-19 (work in progress), October 2007 (TXT).
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

11.2. Informative References

TOC

[I-D.ietf-hip-base]	Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, “Host Identity Protocol,” draft-ietf-hip-base-10 (work in progress), October 2007 (TXT).
[I-D.ietf-p2psip-concepts]	Bryan, D., Matthews, P., Shim, E., Willis, D., and S. Dawkins, “Concepts and Terminology for Peer to Peer SIP,” draft-ietf-p2psip-concepts-02 (work in progress), July 2008 (TXT).
[RFC4843]	Nikander, P., Laganier, J., and F. Dupont, “An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID),” RFC 4843, April 2007 (TXT).

Authors' Addresses

TOC

	Eric Cooper
	Avaya
	1135 Innovation Drive
	Ottawa, Ontario K2K 3G7
	Canada
Phone:	+1 613 592 4343 x228
Email:	ecooper@avaya.com
	Alan Johnston
	Avaya
	St. Louis, MO 63124
	USA
Email:	alan@sipstation.com
	Philip Matthews
	Avaya
	100 Innovation Drive
	Ottawa, Ontario K2K 3G7
	Canada
Phone:	+1 613 592 4343 x224
Email:	philip_matthews@magma.ca

Full Copyright Statement

TOC

Intellectual Property