# SixChat and IRP

## A Declaration of Independence from NAT

*Lawrence E. Hughes*
*Co-founder and CTO, Sixscape Communications Pte. Ltd.*

## The Current Situation

Everyone working with networks today is aware that we have effectively run out of public IPv4 addresses, even with emergency measures (Network Address Translation and *private internets*). Four of the five regional NICs have either already ceased normal allocation of new IPv4 addresses, or will have by the early 2015. The actual or estimated dates for end of normal allocation are:

| | |
|---|---|
| APNIC (Asia/Pacific) | 15 April 2011 |
| RIPE (EU / ME) | 14 September 2012 |
| LACNIC (Latin America) | 10 June 2014 |
| ARIN (North America) | 4 March 2015 |
| AfriNic (Africa) | 19 June 2019 |

In reality, the world began running out of IPv4 public addresses in the mid 1990's. The IETF began working on a replacement technology (IPng, later renamed IPv6) but realized it could not be fully deployed before the expected end of IPv4 address allocation at the then current rate (estimated at the time to be by the year 2000). They regretfully also did some *bad engineering* (forgive me Father, for I have committed *kludge*). This came in two forms: Network Address Translation (NAT44) and private internets (RFC 1918). It may have been a necessary evil, but it *was* an evil.

We splintered the then monolithic IPv4 Internet into millions of *private internets*, each with its own private address space (as per RFC 1918, "Address Allocation for Private Internets", Feb 1996). We then created a gateway to allow these private internets to hide behind some of the remaining precious public IPv4 addresses. This was NAT44. This allowed one-way outgoing connections (two way *traffic*, but only outgoing *connections* – once an outgoing connection was made, traffic could go in both directions). This split the IPv4 Internet into two communities: first class netizens connected to the stub of the old IPv4 monolithic Internet, with public addresses, and second class netizens living in private internet barrios.

Any first class netizen could make connections to, or accept connections from, any other first class netizen. They could also accept connections from second class netizens, via NAT gateways. Second class netizens could make connection to any first or second class netizen. They could accept connections from any other second class netizen *in their barrio*, but not from any first class netizen, or any second class netizen from any other barrio.

We created *Client / Server* architecture for network applications to get around these limitations.

We created a limited number of powerful centralized *servers* on the public IPv4 Internet, whose addresses were published globally via public DNS. These servers were pretty much static (their addresses rarely changed). Public DNS servers only really needed to publish the name and addresses of these few servers. It was OK that an address change might take 24-48 hours to propagate globally. If this was a problem, the TTL (Time To Live) value could be reduced to an hour previous to the address change, and propagation would be much quicker (but still not instantaneous).

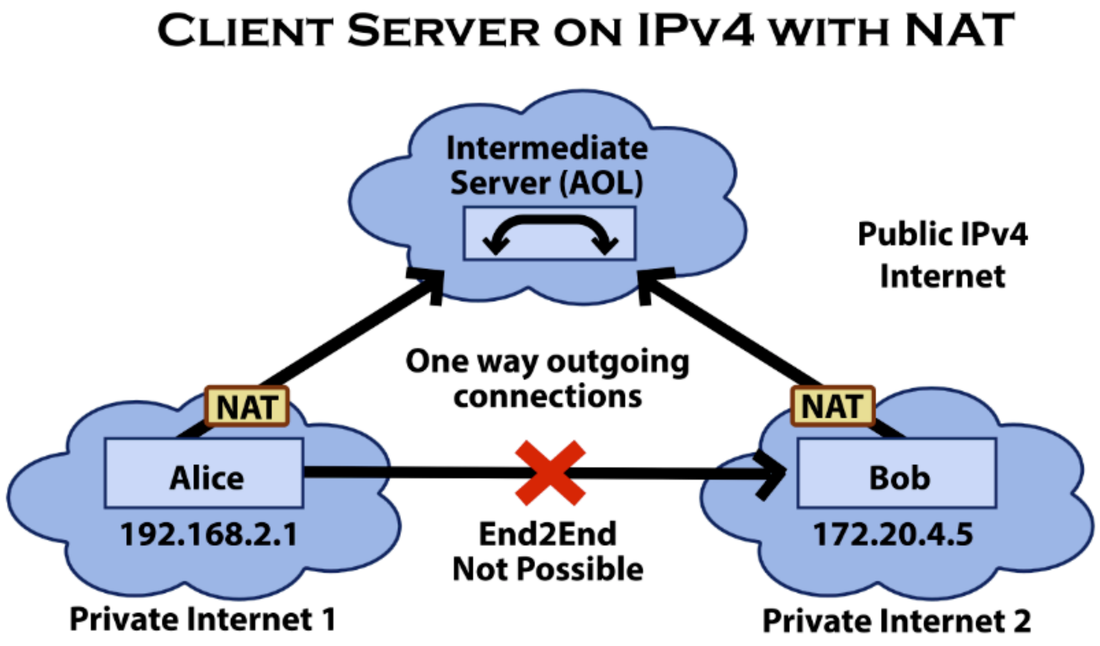
For a while, companies were able to get blocks of 256 public addresses, and deploy servers in their own little piece of the global IPv4 Internet. Even those are now becoming hard to come by, so more and more, servers must live in the networks of Telcos or ISPs, who still have some precious IPv4 public addresses. Even there, they've had to share single IPv4 addresses with multiple servers through use of nonstandard port numbers (like web over port 8080) or other tricks.

Home users used to get at least *one* public IPv4 address (or several if they were willing to pay a lot more), but if they had more than one device, had to create a tiny private internet, hidden behind their one public address. This still allowed incoming connections to one server for each possible port via *Port Translation* in firewalls, and tunneled IPv6 over IPv4 using 6in4 tunneling.

As we get further beyond the end of IPv4 allocation in some regions, more and more, home users get *no* public IPv4 addresses. In the Philippines today, there are no longer any residential accounts with even a single public address. They often get one *private address* behind CGN (Carrier Grade NAT), and their nodes must live behind a *second* NAT44 gateway, making their nodes *third class netizens*. Their nodes can still make outgoing connections via the two gateways to servers on the public IPv4 network, but this has broken even more protocols and complicated things like tunneling IPv6 into home networks over IPv4.

If I chat with you via AOL Instant Messenger (AIM), my node must make a one-way outgoing connection via my NAT gateway to a public server at AOL, and your node must make a one-way outgoing connection via your NAT gateway to that same public server. AIM then relays messages between the two one-way connections, going through their intermediary server. All communications between me and you must work this way on the current Internet because of NAT. Skype *appears* to be direct but this is accomplished with NAT Traversal, which depends on having an external node that helps "smuggle" data into internal nodes that make an outgoing connection to the external STUN server. If I am talking with my son in another room of my house, and I send him a file, the speed is limited by our uplink to ISP, even though there is Gigabit speed between us. If the house link to the Internet goes down, I can no longer chat with him via Skype. It's really the same old model, just disguised with NAT Traversal.

Alice, in private internet 1 can make outgoing connections to the intermediary server at AOL which is on the public IPv4 Internet. Bob, in private internet 2, can also make an outgoing connection to the AOL intermediary server. The server at AOL shuttles chat messages between Alice and Bob back and forth between these two one-way outgoing connections. Alice cannot connect directly to Bob, or vice versa. End2End direct is not possible, unless Alice and Bob are in the same private internet (which means there is no NAT gateway between them).



**The Bright Future with IPv6**

So what is the alternative to being relegated into lower and lower levels of netizenship, with more and more NAT gateways between you and the public IPv4 Internet? The *real* solution to the depletion of IPv4 public addresses is and always has been IPv6 – the successor protocol to IPv4. Phones evolve smoothly from one generation to the next. The Internet has already moved from the 1G version (ARPANET) to the 2G version (IPv4), way back in 1983. It is time for the 3G Internet (IPv6).

1G Internet was based on the Host/Host protocol, which had 8 bit addresses (max of 256 nodes). Amazingly this lasted from 1969 to late 1982.

2G Internet (IPv4) began in 1983, was severely brain damaged in the mid 1990s' and is still with us in rather crippled form. The public 2G Internet could accommodate maybe 2 billion nodes (many of the 4.3 billion theoretical were used for things like broadcast and experimental; others were simply wasted in inefficient allocation schemes).

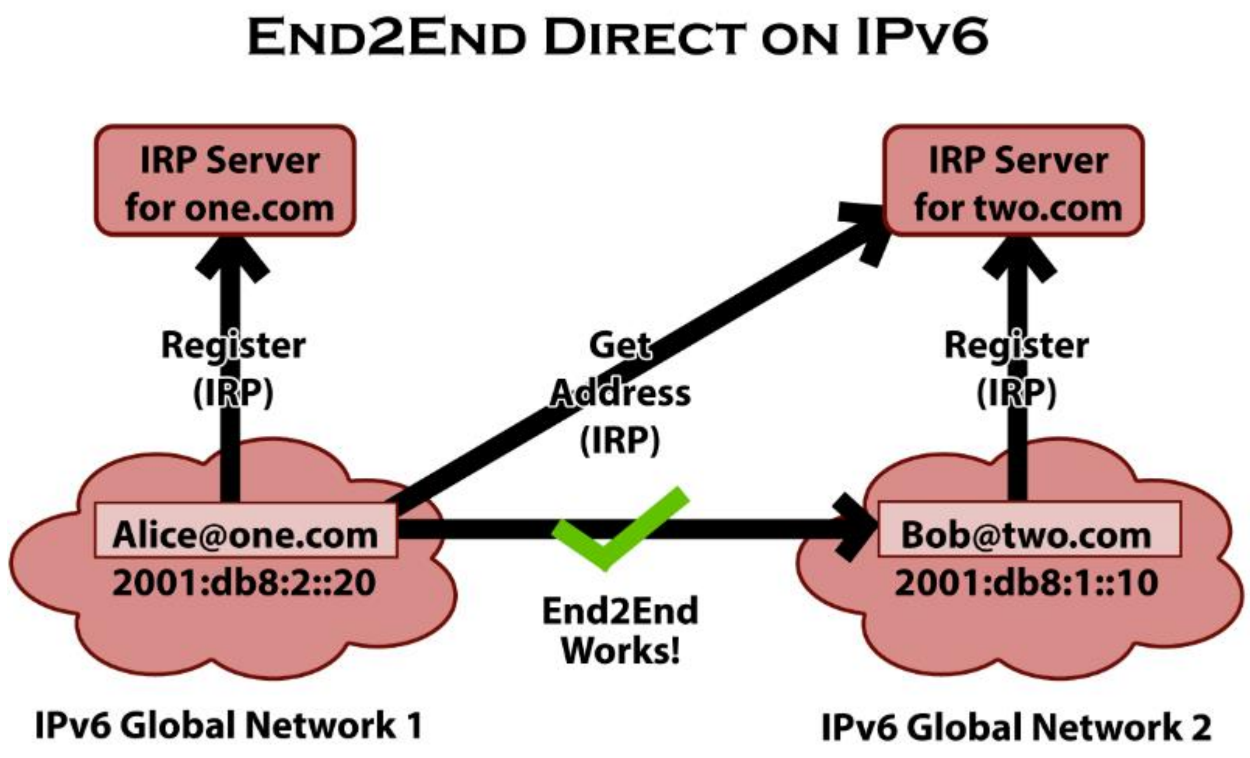
3G Internet was specified originally in late 1995, and is now mature and being deployed worldwide. It has $2^{125}$ public addresses marked for allocation (2000::/3 block, 4.25E+37 addresses, enough for every human alive today to get about 5,000 "/48" blocks – each of which is really sufficient enough for the entire world for hundreds of years if managed properly).

So there are plenty of public addresses available with IPv6. There is no conceivable excuse for deploying NAT66. The only reason to deploy NAT is to extend the lifetime of your address space. NAT provides *no* additional security. So what are the implications of essentially unlimited public addresses and NO NAT?

We have "restored the monolithic global Internet". OK. So far so good. What specifically can we do now, that we can't in an increasingly splintered IPv4 world?

The answer is simple. Any IPv6 node can connect directly to any other IPv6 node, anywhere in the world (subject to port blocks in firewalls between the nodes). This is true today *within* an IPv4 private internet. It is definitely *not* true in the global IPv4 InterNAT.  If my node has only a private address, there is no way anyone (not in my private internet) can connect directly to me. If I have a spare IPv4 public address, I can use 1:1 (bidirectional NAT) to map connections to that address (assigned to the outside NIC of my firewall) to and from my internal node. This works for most things, like email and web, but still breaks IPsec and other protocols, like VoIP. I can also map a single port from one public IPv4 address on my NAT gateway to a single inside node (e.g. an internal mail server). Same deal – works with some protocols but not all. If I have two mail servers, or two web servers, I can't use both of them (at least not using standard port numbers). You can only map a given port (e.g. 80) onto multiple internal nodes.

If I have an internal node with a global IPv6 address, and I open port 80 through my border firewall, then any external IPv6 node can connect directly to my web server over IPv6. There is no NAT gateway that prevents incoming connections. The entire global IPv6 Internet is one monolithic address space, just like the early IPv4 Internet, before we shattered it.



Now Alice (who has a global IPv6 address in her /64 block) and Bob (who has a global IPv6 address in his /64 block) can connect directly. Either node can initiate the connection and the other node can accept it (assuming there is no firewall between them blocking incoming or outgoing connections on that port. The diagram shows Alice connecting directly to Bob, but Bob could also connect directly to Alice.

The question is, how does Alice discover Bob's current global IPv6 address in order to connect to him? He might have various IPv6 addresses over the day – one at work, one at his favorite coffee shop, one at home, etc. With servers, you resolve the server address using global DNS. You ask your local DNS server to resolve the address, and it either already knows it, or finds it from other DNS servers. Somewhere there is an authoritative DNS server that knows the address of that server. This works OK for fairly static nodes. What about highly mobile nodes that might change their IPv6 address frequently? Also, there is no simple way to insure only you can register your current address with DNS – it lacks individualized authentication.  There is a simple authentication scheme called TSIG, but everyone would have to use the same key, which means anyone can update anyone's address. This is not adequate.

Also, most people only register a small number of servers in public DNS. If they deploy DNS for internal nodes, that is accessible only to other internal nodes. It would be a lot of work to publish the IPv6 addresses of all internal nodes in DNS, and keep them up to date. And even if you did, there would be a long delay after one of them changed before everyone could get the new address. DNS was designed for the IPv4 with NAT world, for relatively static servers where it works great. It is not so great for direct End2End connections between highly mobile nodes.

What we need is an address registry on which someone can easily update their address, with individual authentication, so only *they* can update their address. This implies a user directory and good authentication. Username and password can work for this, but cryptographic authentication with an X.509 client cert is much better. That way no username/password database is required on the Address Registry Server, and no password is sent over the Internet. As long as you have a user directory, you could publish other information in it, including their name, their email address, or other identifying information. We have created the Identity Registry Protocol to perform these functions. Our Domain Identity Registry (DIR) server implements this protocol. The user specifies their IRP UserID (looks like an email address) with authentication so that only they could update their own address.

If you integrate a PKI into the address registry, both Alice and Bob could obtain X.509 client certificates and use them for mutual authentication and secure symmetric session key exchange. Once they share a symmetric session key, all traffic between them could be encrypted.

**Bob and Alice Register Their Identities with Their Respective DIR Servers**

Anytime Bob starts any IRP-enabled application (or if his IPv6 address changes), his application updates his current IPv6 address on his DIR server. Earlier he requested and downloaded an X.509 client certificate. This involves generating a public/private keypair, and submitting the public key (along with his username and other information) to his DIR server as a PKCS #10 Certificate Signing Request (CSR). The DIR admin generates a digital cert from his CSR and Bob downloads that to his application. Optionally he can generate a PKCS #12 package (containing his digital certificate and private key, encrypted with a passphrase known only to him). This can be safely uploaded to his DIR server for key backup and recovery, or for transferring his key material to another computer. Alice has likewise obtained an X.509 client certificate, using her Domain Identity Registry.

Alice's Application Connects to Bob's Application

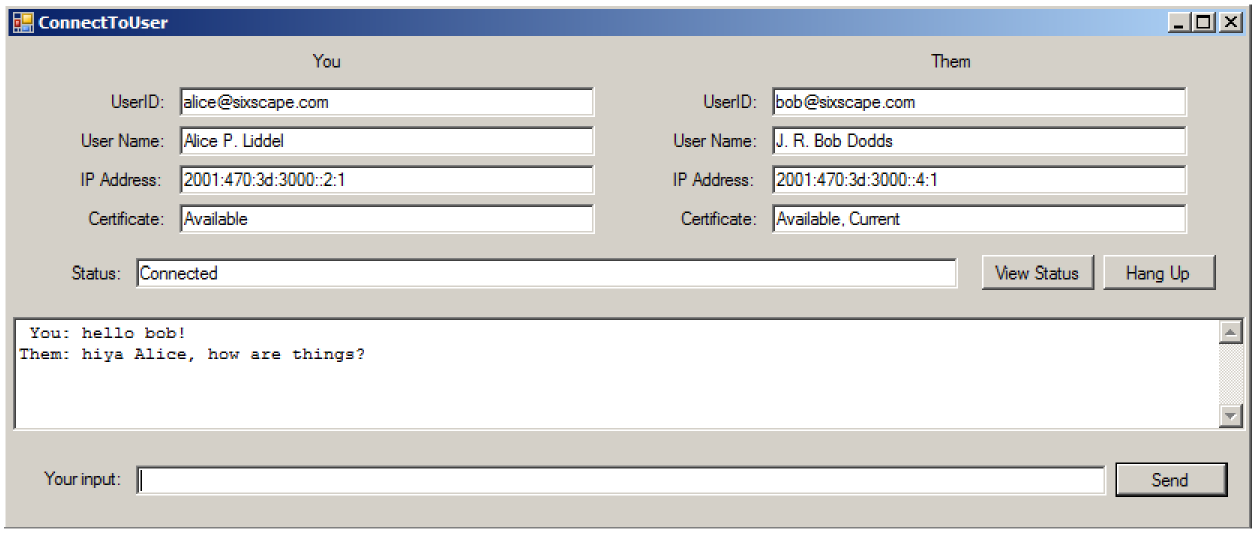
When Alice wants to connect to Bob, her node will try to use the last IPv6 address she has stored for him. If his application does not respond, she asks his DIR server for his current address, using IRP. Assuming he is online, her application connects to his, and they do a security handshake.

The handshake goes like this

1. Alice sends Bob a **Capabilities_Offer** message, which lists the capabilities her application has, such as Certificate based authentication; Certificate based key exchange; Diffie-Hellman key exchange; Secure synchronous chat, etc.
2. Bob sends Alice a **Capabilties_Accept** message that lists the items from her offer that his application also supports. This negotiates the common capabilities supported by both.
3. Alice sends Bob a **CryptoSuite_Offer** message, which lists the cryptographic algorithms she supports (e.g. RSA1024, RSA2048, AES128CBC, AES256CBC, SHA1, SHA256, etc).

4. Bob sends Alice a **CryptoSuite_Accept** message, which lists the items from her offer that he also supports. This negotiates the common cryptographic algorithms supported by both.
5. Alice sends Bob a **User_Info** message containing her IRP UserID, her name, and her client digital certificate. The UserID and Name must match the fields in her digital cert. Bob validates Alice's digital certificate, by following the certification path from her cert to a trusted root cert, as well as checking for certificate expiration or revocation.
6. Bob creates a random string of 16 characters and encrypts it with the public key from Alice's cert. He sends the encrypted string as a challenge to Alice in a **Crypto_Challenge** message.
7. Alice decrypts the challenge string with her private key and sends the result to Bob in a **Challenge_Response** message. Bob checks the returned string and if it matches the random string he created, this strongly authenticates Alice. Only she has the private key needed to respond correctly to his challenge.
8. Bob sends Alice a **User_Info** message containing his IRP UserID, his name, and his client digital certificate. The UserID and Name must match the fields in his digital cert. Alice validates Bob's digital certificate, by following the certification path from her cert to a trusted root cert, as well as checking for certificate expiration or revocation.
9. Alice creates a random string of 16 characters and encrypts it with the public key from Bob's cert. She sends the encrypted string as a challenge to Bob in a **Crypto_Challenge** message.
10. Bob decrypts the challenge string with his private key and sends the result to Alice in a **Challenge_Response** message. Alice checks the returned string and if it matches the random string he created, this strongly authenticates Bob. Only he has the private key needed to respond correctly to her challenge.
11. Alice generates a random symmetric session key for the strongest common symmetric key algorithm and encrypts it with Bob's public key. She sends this to Bob in a **Key_Exchange** message.
12. Bob receives Alice's **Key_Exchange** messge and decrypts the encrypted session key with his private key. This securely exchanged symmetric session key can now be used to encrypt traffic being sent by either party, and decrypt it at the other end.

At this point, Alice and Bob have achieved mutual strong authentication and securely shared a symmetric session key. Alice sees a "Connect to User" window including the status of Bob's cert:

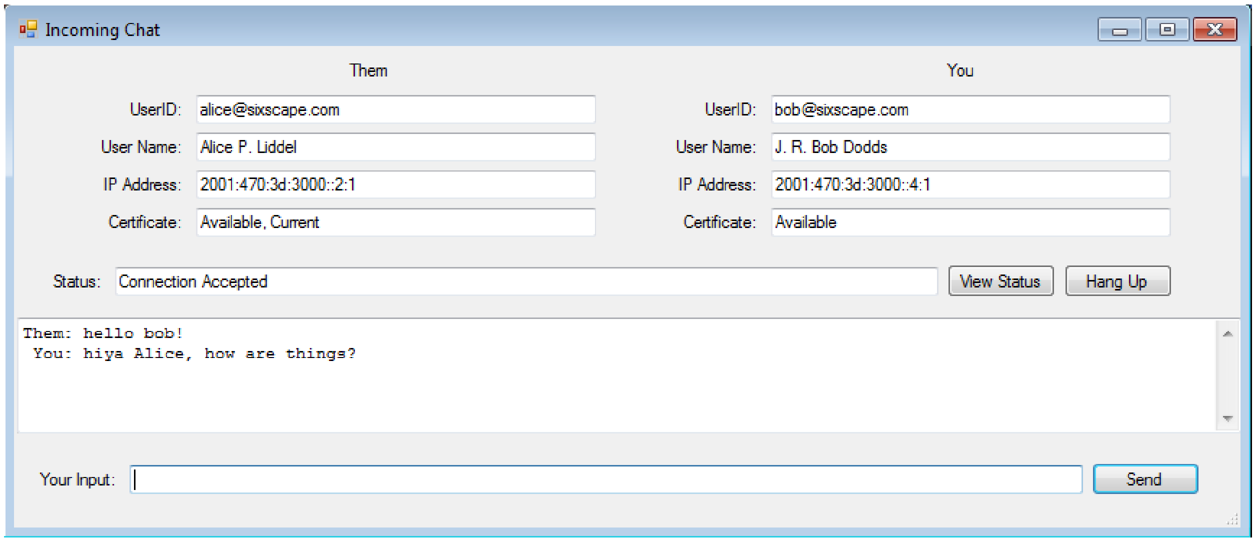


| You | | Them | |
|---|---|---|---|
| UserID: | alice@sixscape.com | UserID: | bob@sixscape.com |
| User Name: | Alice P. Liddel | User Name: | J. R. Bob Dodds |
| IP Address: | 2001:470:3d:3000::2:1 | IP Address: | 2001:470:3d:3000::4:1 |
| Certificate: | Available | Certificate: | Available, Current |

Status: Connected

```
 You: hello bob!
Them: hiya Alice, how are things?
```

Bob also sees an “Incoming Chat” window including the status of Alice’s cert.



| Them | | You | |
|---|---|---|---|
| UserID: | alice@sixscape.com | UserID: | bob@sixscape.com |
| User Name: | Alice P. Liddel | User Name: | J. R. Bob Dodds |
| IP Address: | 2001:470:3d:3000::2:1 | IP Address: | 2001:470:3d:3000::4:1 |
| Certificate: | Available, Current | Certificate: | Available |

Status: Connection Accepted

```
Them: hello bob!
 You: hiya Alice, how are things?
```

They can now chat back and forth securely.

This is very different from conventional Client/Server chat systems. There is no intermediary server, like there is with AIM or Skype. Alice has connected directly to Bob. There are a number of advantages of this:

- Higher availability – as long as Alice and Bob both have network connectivity between them, it doesn’t matter whether or not either or both of them have connectivity to the Internet (assuming Alice already has Bob’s current address – otherwise she would need access to his IRP server, but only to get his address).

- Higher performance – with Client/Server chat, the performance is limited to that of your connection to the Internet (usually your uplink speed). Sending a file via Skype can take a very long time, even if the recipient is sitting next to you. With Direct End2End if both parties are in the same LAN, files are sent at full LAN speed (e.g. 1 Gbit).

- Better security – there is no intermediary server where the traffic is in plaintext. With Skype, it may be secure from Alice to Skype and from Skype to Bob, but it is in plaintext at Skype. All Skype traffic goes through Skype servers, where it can be intercepted and even blocked or changed. With End2End Direct there is no intermediary server. It is never in plaintext during transmission. Also if both parties are in the same LAN, traffic never leaves that LAN. Traffic is more decentralized, hence more difficult for others to intercept or block. In addition to certificate based secure key exchange, if the two nodes connect directly, they can exchange a key securely using Diffie-Hellman Key Exchange, even if neither has a digital certificate. This can provide privacy, but not authentication.

- Better scalability – there is no intermediary server to become a bottleneck. So long as the network bandwidth between parties is sufficient, any number of communicating pairs can connect.

In the IPv4 InterNAT, we are forced to use Client/Server, if traffic transits a NAT gateway. This works, but End2End Direct would be better. In IPv4 this works only within a LAN with a single address space (no NAT gateways are crossed). In the IPv6 Internet, Client/Server does still work, but we also have the option of End2End Direct now.

**Application Design for End2End Direct**

With Client/Server all applications are either *client model*, or *server model*.

A *client model* application typically makes one outgoing connection to a single server. With web, it might disconnect from one server and reconnect to another, but at it only makes outgoing connections, and usually only one at a time. This is very simple to implement.

A *server model* application is much more difficult to implement. It must listen for incoming connections, possibly over both IPv4 and IPv6. It typically spawns a new thread or process for each incoming connection. It may be able to handle hundreds or even thousands of simultaneous incoming connections. Some servers may also make outgoing connections, as with SMTP Email – if a server accepts an incoming message for someone with an account on another server, it can relay that message to the correct email server via an outgoing connection.

End2End Direct applications require a somewhat different design. Every application must be able to make outgoing connections (like a client) *and* accept incoming connections – simultaneously. It may be able to make and accept multiple connections, simultaneously. This is much more difficult to implement than a client, and is more like a server that runs on you node (rather than a server node). It normally has a GUI interface, while most traditional server applications don't. This architecture is called a *User Agent*.

VoIP is inherently End2End, and well suited to implementation in End2End Direct applications. It is possible to implement VoIP as Client/Server but this complicates the overall system design. Unfortunately, with IPv4 this is the only viable design. VoIP will be far easier to use on IPv6, but only if vendors implement it as End2End Direct.

**SixChat – A Secure End2End Direct Application**

SixChat is the first Secure End2End Direct application created by Sixscape Communications. It requires a Domain Identity Registry server for each participating domain. Users register with their DIR server, and obtain an X.509 client digital certificate.

Client digital certs can be requested from each SixChat application, and signed on the DIR server (distributed PKI model), or they can be created on the DIR Server and downloaded onto each SixChat User Agent as PKCS #12 files or distributed in USB hardware tokens (centralized PKI model).

Currently SixChat features End2End Direct synchronous chat (both parties present during the connection). We will soon be adding End2Endd Direct S/MIME mail. This will add an S/MIME composition window and an InBox. No mail server is required – messages are sent directly from sender to recipient. No SMTP or IMAP required. We will also provide End2End secure file transfer, with a user interface similar to "Norton Commander" (as used in WinSCP).

We will also provide an intermediary server for asynchronous chat, mail and file transfer. This is a temporary message store for times that the recipient is not online. You will be able to send messages to the intermediary message store, and the next time the recipient goes online the messages will be delivered to their User Agent.

SixChat and the Intermediary message store use the SixChat protocol, which was reviewed by IANA (like IRP) and granted exclusive use of port 4605. This is an XML based protocol which is highly extensible. It can accommodate chat, mail, file transfer, voice and probably other kinds of traffic, with appropriate extensions.