

INVESTIGATION OF ETHERYATRI'S COMPATIBILITY WITH IPV6

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Srinivasa Sankar Nemani

Certificate of Approval:

Dean Hendrix
Associate Professor
Computer Science and
Software Engineering

Homer Carlisle, Chair
Associate Professor
Computer Science and
Software Engineering

Gerry Dozier
Associate Professor
Computer Science and
Software Engineering

Stephen L. McFarland
Dean
Graduate School

INVESTIGATION OF ETHERYATRI'S COMPATIBILITY WITH IPV6

Srinivasa Sankar Nemani

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
August 7, 2006

INVESTIGATION OF ETHERYATRI'S COMPATIBILITY WITH IPV6

Srinivasa Sankar Nemani

Permission is granted to Auburn University to make copies of this thesis at its discretion, upon request of individuals or institutions and at their expense. The author reserves all publication rights.

Signature of Author

Date of Graduation

VITA

Srinivasa Sankar Nemani, son of Venkata Narayana Sarma and Nagamani Nemani was born 1975, in Vizianagaram, Andhra Pradesh, India. He graduated from Andhra University in 1996, with a Bachelor of Engineering degree in Mechanical Engineering. He joined Graduate School, Auburn University in 1996.

THESIS ABSTRACT

INVESTIGATION OF ETHERYATRI'S COMPATIBILITY WITH IPV6

Srinivasa Sankar Nemani

Master of Science, August 7, 2006
(Bachelors of Engineering, Andhra University, India, 1996)

89 Typed Pages

Directed by Homer Carlisle

Internet Protocol version six (IPv6) enjoys only a scant existence today. Many universities, companies and other organizations are building protocol stacks, hardware and applications to support IPv6, awaiting its full-fledged emergence into consumers' lives. This thesis is such an effort to add a .NET based mobile agent system to the list of application frameworks that support IPv6. An IPv6 enabled mobile agent framework would help researchers and enthusiasts in their pursuit for a killer application that could spur IPv6's growth. Mobile agent community could also benefit in a similar way from this. This would get mobile agent technology into more hands, there by improving the chance of finding a killer application, mobile agent community is waiting for. EtherYatri.NET is one such mobile agent system built using Microsoft .NET Framework. This thesis investigates EtherYatri.NET's support for IPv6 and adds the support where needed. Microsoft .NET Framework SDK 2.0, Visual Studio 2005 Beta2 and EtherYatri.NET v0.5.7 are used for this thesis.

ACKNOWLEDGEMENTS

The author would like to express his special thanks to his advisor, Dr. Homer Carlisle, Associate Professor, Department of Computer Science and Software Engineering for the guidance and support provided during his research work. The author would also like to thank his committee members Dr. Dean Hendrix & Dr. Gerry Dozier for their help with his thesis. The author would like to thank Dr. Hari Narayan, other professors and staff members of the Department of Computer Science and Software Engineering for providing inspiration and contributing to the author's understanding of various topics of Computer Science.

The author acknowledges Dr. P. K. Raju and Dr. Mrinal Takhur, professors in the Department of Mechanical Engineering for their support through out his stay at Auburn.

Style manual or journal used: Guide to preparation and Submission of Theses and
Dissertations, Graduate School, Auburn University

Computer software used: MS Office 2003

TABLE OF CONTENTS

LIST OF FIGURES	x
LIST OF SOURCE CODE SAMPLES	xi
CHAPTER 1: INTRODUCTION TO IPV6	1
1.1 Network protocols	1
1.1.1 The OSI model	3
1.1.2 The TCP/IP model	6
1.1.2.1 Internet Protocol V4	8
1.1.2.2 Internet Protocol V6	13
CHAPTER 2: INTRODUCTION TO ETHERYATRI.NET	23
2.1 Software Agents	23
2.2 Mobile Agents, a closer look	25
2.2.1 Advantages of Mobile Agents	26
2.2.2 Mobile Agent Frameworks	28
2.2.3 .NET Framework for Mobile Agents	29
2.2.4 EtherYatri.NET	33
CHAPTER 3: LITERATURE REVIEW	34
3.1 Agent Technologies	34
3.1.1 Non-Intelligent Agents	34
3.1.2 Intelligent Agents	35
3.1.3 Mobile Agents	37
3.2 Internet Protocol v6	45

CHAPTER 4: ENABLING IPV6 IN ETHERYATRI.NET	48
4.1 The Research Environment	48
4.1.1 Setting up an IPv6 test environment	48
4.1.2 Setting up EtherYatri.NET development environment	57
4.3 Enabling IPv6 in EtherYatri.NET	62
CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH POSSIBILITIES	67
5.1 Conclusions	67
5.2 Potential future research	69
BIBLIOGRAPHY	72

LIST OF FIGURES

Figure 1.1 7 layers in the OSI model	4
Figure 1.2 Classification of IP addresses	9
Figure 1.3 Number of hosts in IP classes	10
Figure 1.4 Make up of an IP packet	12
Figure 1.5 Global unicast address	17
Figure 1.6 Local unicast address	17
Figure 1.7 Site local address	18
Figure 1.8 Multicast address	18
Figure 1.9 Make up of an IPv6 header	20
Figure 1.10 IPv6 Extension headers	22
Figure 2.1 Communication between a client and a server	25
Figure 2.2 Communication between mobile agents	26
Figure 3.1 IP v6 support in various operating systems	47
Figure 4.1 Windows IP Configuration on host1 before installing IPv6	49
Figure 4.2 Installing IPv6 protocol on Windows Server 2003	49
Figure 4.3 Windows IP Configuration on host1 after installing IPv6	50
Figure 4.4 Building the source files using the .NET Framework 2.0 Redistributable ..	56
Figure 4.5 the chat client and server communicating over IPv6	57
Figure 4.6 Two instances of WinAH running on ports 8000 and 8001 respectively ...	59
Figure 4.7 creating the HelloWorldDemo agent at port 8000	60
Figure 4.8 sending the agent to another host using the Send Agent dialog	60
Figure 4.9 Message showing that agent has arrived at the destination	61
Figure 4.10 Exception - when IPv6 address was used in the destination URL	63
Figure 4.11 Exception - after a valid URL format was used for the destination URL .	64

LIST OF SOURCE CODE SAMPLES

Source code 4.1 The ConcurrentIO class from Chat.IO.cs	52
Source code 4.2 the Chat.Server class from Chat.Server.cs	53
Source code 4.3 the Chat.Client class from Chat.Client.cs	55
Source code 4.4 using the bindTo property of the channels to bind to IPv6 addresses	65
Source code 4.5 using the bindTo property of the channels to bind to IPv6 addresses	65

CHAPTER 1

INTRODUCTION TO IPV6

Just as humans have gone beyond a solo existence, the days of isolated information appliances are quickly fading away as wireless networks, the Internet, the World Wide Web (WWW), broadband connectivity and intranets spread across the information technology landscape. A variety of devices including house hold appliances, home and personal entertainment systems, personal communication devices and computers are providing information to the consumers to help improve their quality of life. Examples of such devices include alarm clocks, refrigerators, microwave ovens, televisions, set-top boxes, cell phones, PDAs, camcorders, MP3 players and laptops. The advent of these information appliances is increasing the demand for connectivity infrastructure more sophisticated than that currently exists in the world of computer networks. Future computer networks must support a huge number of devices, enable secure communications and support always-on devices. Computer network infrastructure consists of both hardware and software. Advancements to network infrastructure need to be made in both hardware and software to support the plethora of informational devices.

1.1 Network protocols

Protocol stacks are one of the building blocks of network software. Abstraction of lower level systems is a key ingredient in the recipe to develop extensible information systems. The same applies to the development of computer networks as well. Network

software hides hardware complexities and low-level communications details providing a high-level abstraction for applications. And so, most application programs rely on network software to communicate instead of interacting with the network hardware directly [1].

Software that runs network hardware made by different vendors must agree on a set of rules to be used while communicating. These sets of rules are known as protocols. “An agreement that specifies the format and meaning of messages computers exchange is known as a communication protocol” [1].

There are several aspects to network communication. Communicating with hardware, addressing and handling application specific rules are few of those. Network protocols would have been very complex, had they addressed all the aspects of network communication with one set of rules. For that reason, designers have chosen to break the network protocol into pieces. These pieces most commonly are modeled as layers. In the layering model, a protocol suite can be designed by specifying a protocol for each of the layers. Protocols are put to practical use through standards. ISO (International Organization for Standardization), ANSI (American National Standards Institute), ITIC (Information Technology Industry Council), IEEE (Institute of Electrical and Electronic Engineers), EIA/TIA (Electronic Industries Alliance/Telecommunications Industry Association), ITU-T (International Telecommunication Union - Telecommunication Standardization Sector) and ETSI (European Telecommunications Standards Institute) are a few of the international organizations that are responsible for managing the standards development process [2]. ISO’s OSI model and TCP/IP model are the two most known layering networking models.

1.1.1 The OSI model

The Open Systems Interconnect Reference Model's (also known as The OSI Model) original goal was to provide basis for designing a universal protocol suite. Even though the suite did not enjoy a widespread and exact implementation, the model became valuable for education and development of other models. The model defines a set of layers and a number of concepts that make understanding networks easier.

In the late 1970s, two projects, one administered by the International Organization for Standardization (ISO), and the other by the International Telegraph and Telephone Consultative Committee (CCITT), each developed networking models that were very similar. Later in 1983, these two models merged to form The Basic Reference Model for Open Systems Interconnection (The OSI Model). In 1984 ISO published it as ISO 7498 and CCITT published it as X.200 [3].

The OSI model which is also known as 7-layer model, defines 7 layers namely Physical, Data link, Network, Transport, Session, Presentation and Application; Physical being the lowest and Application being the highest layer.

Application
Presentation
Session
Transport
Network
Data Link
Physical

Figure 1.1 7 layers in the OSI model

- **Physical layer**

Layer1: The physical layer carries digital data using electrical and physical signals. The specifications in physical layer also define voltages & layouts, how connections are established & terminated and how resources are shared. DSL, ISDN, Fast Ethernet and ATM are examples of physical layer components.

- **Data link layer**

Layer2: The Data link layer provides an abstraction for the physical layer. Physical addresses that are hard-coded into the network cards are used for addressing and connectivity is available only among locally attached nodes. This layer provides the services such as error detection and correction to the physical layer. Breaking network packets into frames of data, controlling access to the media are done at this layer. Ethernet, HDLC, ADCCP, Aloha and LLC are examples of Data link layer protocols.

- **Network layer**

Layer3: The Network layer provides addressing, routing, segmentation & de-segmentation functions for transferring data. The addressing scheme is logical and hierarchical. IP and IPv6 are examples of Network layer protocols.

- **Transport layer**

Layer4: Transport layer controls reliability, state and connection for a communication session. By using port numbers, transport layer can uniquely identify different applications on a single node on the network. TCP and UDP are examples of transport layer protocols.

- **Session layer**

Layer5: The Session layer provides for initiation, termination and restart of dialogues between application processes. Setting up and closing down TCP/IP sessions is an example of functionality provided at this layer.

- **Presentation layer**

Layer6: The Presentation layer abstracts the issues related to data representation from the application layer. MIME encoding, encryption and similar manipulation are done at this layer. Converting an ASCII encoded text into a Unicode is an example of work done at this layer.

- **Application layer**

Layer7: This layer is the highest layer of the OSI model and interfaces directly to the application processes. Telnet, FTP, HTTP are examples of Application layer protocols. [4].

Real world implementations of network protocol suites do not reflect the OSI model. The failure of the OSI model in this context is attributed to bad timing, bad technology, bad implementations and bad politics. The competing TCP/IP protocols were already in widespread use by research universities by the time OSI protocols appeared. OSI was too complex, some layers are empty (ex: session and presentation) while some layers are overfull (ex: physical and data link). OSI model has some functions such as addressing, flow control and error control redundantly defined in different layers. Due to its complexity, the initial implementations were huge, unwieldy and slow while some competing implementations were quite good and free. OSI model protocols did not address issues specific to internetworking. Involvement of governmental organizations did not help much either [5].

1.1.2 The TCP/IP model

TCP stands for Transmissions Control Protocol and IP stands for the Internet Protocol. Research work for TCP/IP was started in the late 1960s and early 1970s funded by the Advanced Research Projects Agency (ARPA), the research wing of the US Department of Defense (DoD). BBN, an ARPA hired firm developed a research network called ARPANET that became operational first in 1972. DoD, after building MILNET (Military Installation in US) and MINET (Military Installation in Europe), to encourage the wide adoption of TCP/IP, funded BBN and University of California, Berkley to implement TCP/IP in Berkley version of UNIX. TCP/IP has been the foundation for the Internet.

ARPA's goal for TCP/IP was to develop a network that is completely decentralized and fully redundant so that few non-functioning computers could not bring

the whole network down. The network should be flexible enough to support time-sensitive applications such as voice.

The TCP/IP model consists of 4 layers namely Network Interface, Internet, Transport and Application. The network interface layer is the equivalent of OSI model's physical and data link layer while the application layer combines the aspects of session, presentation and application layers from OSI model. The Internet and transport layers operate at the network and transport layers of the OSI model. With all these goals and reasons mentioned earlier, TCP/IP stood as a pragmatic approach to network models. TCP/IP is the most widely used protocol suite in computer networks today. Even though TCP/IP bears the name TCP, in the transport layer it is not limited to TCP and supports other protocols such as UDP. The transport layer usually defines ports to uniquely address various applications running on a host. For example, HTTP applications listen on TCP port 80.

TCP/IP's internet layer uses a number of protocols including IP, ARP and ICMP. IP stands for Internet Protocol and is responsible for addressing and routing packets. ARP stands for Address Resolution Protocol and is used to match hardware addresses to the IP address. ICMP stands for Internet Control Management Protocol and is used to report errors and send messages about the delivery of the packets [6].

1.1.2.1 Internet Protocol V4

In both TCP/IP and the OSI models, network layer is responsible for addressing and thus dictates the number of devices that can be hosted on a network. The network layer protocol used in the TCP/IP model is IP (Internet Protocol). The IP standard specifies that each host is assigned a unique 32-bit number known as the host's Internet

Protocol address or IP address where a host is any machine that has a two-way access to other machines on the network [7].

An internetwork is a network of networks. To make routing efficient, the IP address is divided into two parts. The first part (prefix) identifies a network on the internetwork while the second part (suffix) identifies a host inside a network. The number of bits assigned to each of the two parts is not the same across networks. If the sizes were the same, the number hosts per network would be constant and impractical. For instance, a small organization may want a network of its own but may not have as many hosts as a bigger organization. Consequently, there are more small organizations than large organizations that have a large number of hosts per network. To accommodate varying sizes of networks, the IP address was originally divided into 5 classes namely A, B, C, D and E. These classes not only have different number of bits assigned to the prefix and suffix but also have different address spaces. For instance class A has a smaller prefix size than its suffix size. So class A can accommodate fewer networks but a large number of hosts per network. At the same time class A is allocated a bigger address space making the overall number hosts in the class larger than that of other classes [1].

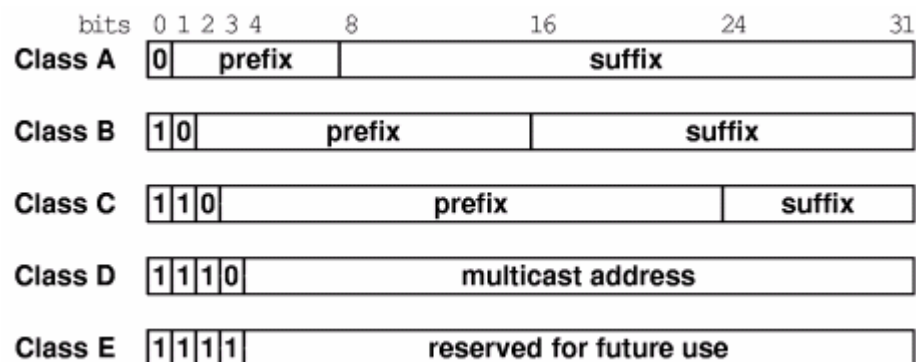


Figure 1.2 Classification of IP addresses

The first four bits of an address determine which class the address belongs to and thus determine the network address and the host address parts of the IP address. Although IP addresses are represented in binary format in network software, a more user-friendly notation called dotted decimal notation is used in the user interfaces to the software. The dotted notation shows the decimal value of each byte (8 bits) of the 4 bytes delimited by a period (.). Ex: a 32-bit binary number 10000001 00110100 00000110 00000000 is shown in dotted notation as 129.52.6.0. In dotted decimal notation, the first octet values for Class A addresses range from 0-127, Class B addresses range from 128-191, Class C addresses range from 192-223, Class D addresses range from 224-239 and Class E addresses range from 240-255. Class A addresses are usually registered by huge multinational companies with large networks [8]. GE, IBM, AT&T Bell Laboratories and Xerox have Class A addresses [9]. Auburn University, University of Washington, Purdue, and MIT have Class B addresses. Usually small businesses and small universities have Class C addresses. Sometimes small universities and businesses get Class A&B IP addresses through a registration authority which had leased some Class A&B address space. Class D addresses are reserved for multicasts and Class E addresses are reserved for experimental purposes.

Address Class	Bits In Prefix	Maximum Number of Networks	Bits In Suffix	Maximum Number Of Hosts Per Network
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

Figure 1.3 Number of hosts in IP classes

Since addresses were assigned in one of the three classes shown above, a lot of addresses were being wasted. For example, if a business wanted 100 addresses, it would

be assigned the smallest class i.e., C which would still waste 156 addresses. Also as the internet grew, the class based addressing has made the routing tables in the backbone routers long and less efficient. To overcome these problems, Classless Inter Domain Routing (CIDR) has been implemented in early 1990s. Instead of being limited to network identifiers of 8, 16 or 24 bits, CIDR currently uses prefixes anywhere from 13 to 27 bits. Thus, blocks of addresses can be assigned to networks as small as 32 hosts or to those with over 500,000 hosts. This allows for address assignments that much more closely fit an organization's specific needs. A CIDR address includes the standard 32-bit IP address and also information on how many bits are used for the network prefix. For example, in the CIDR address 206.13.01.48/25, the "/25" indicates the first 25 bits are used to identify the unique network leaving the remaining bits to identify the specific host. [10].

There are some IP addresses that are reserved for special purposes [1].

- **Network Address**

An IP address that has all host bits set to 0 refers to the entire network that is identified by the network bits of the IP address. Such an address is called Network Address. Since the Network Address refers to an entire network and not a single host, it should never appear as the destination address in a packet [1].

- **Directed Broadcast Address**

An IP address that has all host bits set to 1 is a Directed Broadcast Address for the network that is identified by the network bits of the IP address. Packets sent to this address will be delivered to all hosts on the destination network.

- **Limited Broadcast Address**

An IP address that has all bits (both network and host) set to 1 is a Limited Broadcast Address. This address is used to send packets to all the hosts that are on the same local physical network as the sending host.

- **This Computer Address**

An IP address that has all bits (both network and host) bits set to 0 refers to the host that it is running on. This computer address is different from a Loopback Address. For example a host that is configured to request a dynamic IP at startup using DHCP (Dynamic Host Configuration Protocol), wouldn't know its own IP (since it has not requested one yet) and neither would it know the DHCP server's IP address. In that case the packets sent contain a source IP address of 0.0.0.0 (This Computer Address) and a destination IP address of 255.255.255.255 (Limited Broadcast Address).

- **Loopback Address**

Any IP address with a prefix of 127 is a Loopback Address. The rest of the bits do not matter. So 127.0.0.1, 127.1.23.47, 127.0.0.0, 127.255.255.255 all are valid examples of a Loopback Addresses. Network application developers use loopback addresses to test their applications. The applications that communicate through the protocol stack could run on the same computer. When one application sends data to another data travels down the protocol stack to the IP software which forwards it back up through the protocol stack to the second program. These packets however do not go through the physical layer.

The data in IP layer is sent as packets/datagrams and each packet has a header area and data area. IP allows the size of datagrams to be determined by the applications, thus making IP adaptable to many applications.

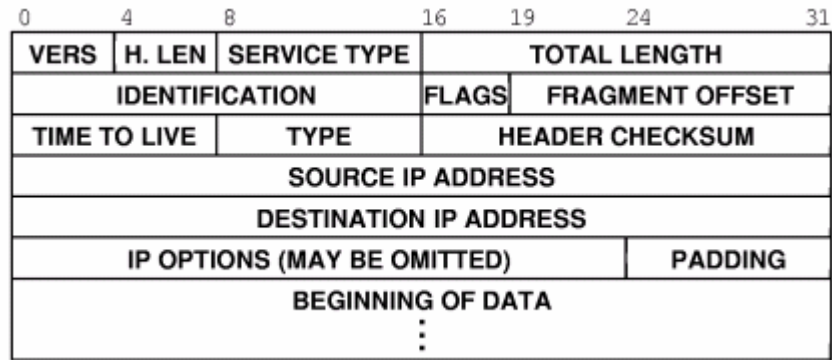


Figure 1.4 Make up of an IP packet

The IP datagram begins with 4-bit protocol version number and a 4-bit header length that specifies the number of 32-bit quantities in the header. The SERVICE TYPE field contains a value to indicate sender's preference between a route with minimal delay and a route with maximum throughput. The TOTAL LENGTH field contains a 16-bit integer that specifies the total number of octets in the datagram including the header and the data. A sender places a unique identification number in the IDENTIFICATION field of each outgoing datagram. The identification number is used in reassembling fragmented datagrams. The FLAGS field indicates whether a datagram is fragmented. The FRAGMENT OFFSET field tells a receiver how to order fragments within a given datagram. The TIME TO LIVE field contains the number of hops the datagram can travel before being discarded. This is used to prevent a datagram from traveling forever around a path that contains a loop that could be caused by a software malfunction. The TYPE field identifies the higher layer protocol carried in the datagram [11]. The HEADER CHECKSUM field ensures that bits of the header are not changed in transit. After that come the source IP address and the destination IP address. If IP OPTIONS field size does

not add up to a 32 bit multiple, PADDING field is used to contain zero bits to make the size a 32 bit multiple.

1.1.2.2 Internet Protocol V6

Since its inception in the 80's, IPv4 has not changed substantially, but its usage by consumers and their demands have changed. To meet the increasing demands, both the network hardware and software had to work around IPv4. Examples include local interpretations of Type of Service field. The following are some of IPv4's limitations:

- IPv4 addresses have become relatively scarce. Many organizations today use some sort of Network Address Translation (NAT) services to provide multiple private IP address that share one public IP address. Often ports have to be mapped from the public IP address to one of the private IP addresses to accept incoming connections into the network. That means only one of the hosts on the private network can offer a specific kind of service. For example there can be only one web server on the private network that can accept incoming connections on the well known HTTP port 80. Application developers that develop network applications have to spend time and resources planning for their applications to work with computers on a private network.

- Due to the way IPv4 addressing works, routers have to maintain large routing tables.

- Configuring IP addresses and automating the configuration is complex and could be simplified.

- IPv4 did not provide for encryption at the IP level. Even the IPsec standard is optional and encryption is a lot of times implemented at the application layer. (Ex: SSL).

- IPv4 also did not provide for Quality of Service and real-time support.

Real-time delivery of audio and video require efficient routing [12].

The initial work on the next version of IP was referred to as IP – The Next Generation (IPng). Since the version number 5 was used for an experimental protocol known as ST, IPng was given the version number 6 and consequently became what is now known as IPv6.

IPv6 brings with it, the successful design features from IPv4. For example, IPv6 is connectionless, provides a way to avoid infinite travel of a packet and retains most of the general features from IPv4 that are essential for network layer. At the same time, IPv6 is designed from ground up keeping in mind the future demands of the Internet.

There are 5 main groups of features that are new in IPv6 [1].

- **Address Size**

The most obvious improvement in IPv6 is the size of the address. Address size in IPv6 is 128 bits as opposed to 32 bits in IPv4. This allows for 2^{128} or 340,282,366,920,938,463,463,374,607,431,768,211,456 (or 3.4×10^{38}) unique addresses. Even to assign a unique IP address to every device that needs one, this number is too big. The reason behind having such large an address space is not only to connect a large number of devices but also to encourage hierarchical addressing and routing that reflects the Internet topology making routing more efficient. Initially only a small portion of this address space is available for use by hosts. Conserving public addresses is no longer going to be the reason to have hosts on a private LAN using NATs.

- **Header Format**

By introducing extension headers and not providing backward compatibility with IPv4 header format, IPv6's header format is made more efficient for processing by routers and other network hardware and software. The IPv6 header is a fixed size (40 bytes) and non-essential, optional fields are specified in the extension headers.

- **Extension Headers**

IPv6 supports extension headers. Extension headers bring extensibility to the IPv6 protocol. When new features are required, the elements that are needed by the feature can be placed in an extension header. Routers can still transmit the datagram, while network hardware and software that support the new feature can interpret the extension header.

- **Support for audio and video**

New fields are defined in IPv6 to help routers make smart choices when a high-quality/low-cost path is required. Support for Quality of Service is built in.

- **Automatic address configuration**

IPv6 hosts can configure addresses themselves both in a DHCP (Dynamic Host Configuration Protocol)'s presence and in its absence. Hosts on a link can automatically obtain an IP address for that link called link-local addresses making ad hoc connections easy. These addresses can also be derived from prefixes advertised by local routers.

- **Built-in security**

IPSec is required in IPv6. Since it is part of the protocol, there is no need for proprietary implementations and consequently interoperability is easier to achieve.

IPv6 addresses are divided along 16-bit boundaries that are represented in hexadecimal values and are separated by colons (:). This notation is known as colon-hexadecimal notation.

For example: 105.220.136.100.255.255.0.0.0.0.18.128.12.10.255.255 is an IPv6 address in dotted decimal notation which in binary format would look like:

01101001 11011100 10001000 01100100 11111111 11111111 00000000 00000000
00000000 00000000 00010010 10000000 00001100 00001010 11111111 11111111

The same in colon-hexadecimal notation would be:

69DC:8864:FFFF:0000:0000:1280:0C0A:FFFF which can further be simplified by using leading zero suppression which removes leading zeros in each block but retaining at least one digit in the block. After leading zero suppression, the address looks like:

69DC:8864:FFFF:0:0:1280:C0A:FFFF which can then be simplified using zero compression which replaces sequence of zeros with two colons (::). With zero compression the address becomes 69DC:8864:FFFF::1280:C0A:FFFF. Zero compression can only be used on one sequence of zeros per address [1].

These techniques are important because many addresses in the IPv6 address space are expected to have strings of zeros. Especially the IPv4 addresses that are mapped to IPv6 to help with the transition will contain 96 zero bits in the higher order bits.

In IPv6 the prefix (the network id part) of the address is specified using the CIDR notation: The address followed by a / and the size of the prefix.

There are a variety of IPv6 addresses [13]:

- **Global Unicast Addresses**

A global unicast address is routable and reachable on the public IPv6 network. A global unicast address has the three high-order bits set to 001. So the address prefix for the global unicast addresses is 2000::/3. It is followed by a 45-bit Global Routing Prefix. The three fixed bits and the 45-bits together are referred to as a site prefix. The site prefix is followed by a 16-bit subnet ID which is followed by a 64-bit interface ID that identifies an interface on a subnet.

001	Global Routing Prefix	Subnet ID	Interface ID
3 bits	45-bits	16-bits	64-bits
Public Topology (48-bits)		Site Topology	Interface Identifier

Figure 1.5 Global unicast address

Public topology includes public internet service providers while site topology includes the local network services to a site. Network cards are identified by interface identifiers [14].

- **Local Unicast Addresses**

Link-Local and Site-Local are the two types of local-use unicast addresses. Following is the format for Link-Local addresses:

111111010	0	Interface ID
10 bits	54-bits	64-bits

Figure 1.6 Local unicast address

The prefix for Link-local addresses is always FE80::/64. Link-Local addresses are valid only on a single link. Example uses include automatic address configuration and neighbor discovery.

Following is the format for Site-Local addresses:

1111111011	Subnet ID	Interface ID
10 bits	54-bits	64-bits

Figure 1.7 Site local address

Site-local addresses are designed to be used for addressing inside of a site without the need for a global prefix. Site-local addresses have been deprecated by RFC 3879 due to their ambiguity and fuzzy definition of sites [15] and are still mentioned to be deprecated in the internet draft on IPv6 addressing [16].

- **Multicast Addresses**

An IPv6 multicast address identifies a group of interfaces typically on different nodes. All multicast IPv6 addresses start with a binary 11111111. There are permanently assigned, reserved, pre-defined and solicited multicast addresses. Multicast addresses have the following format:

11111111	Flags	Scope	Group ID
8 bits	4-bits	4-bits	112-bits

Figure 1.8 Multicast address

- **Anycast Addresses**

An IPv6 anycast address is an address that is used to identify any of the interfaces that are assigned the same address. A packet sent to an anycast address is routed to the

“nearest” host with that address depending on the distance calculated by the routing protocols.

Anycast addresses are allocated from the same address space as the unicast addresses and share the same syntax and format. When a unicast address is assigned to more than one interface, it turns into an anycast address and the nodes to which the address is assigned must be explicitly configured as having an anycast address [16]. For example a subnet router anycast address could be used to provide a more efficient service by providing multiple routers that can route packets for a subnet.

- **Required Addresses**

A host and a router are required to identify themselves with their Link-Local Addresses, any other configured Unicast and Anycast Addresses, the loopback address and all multicast addresses that resolve to their interfaces.

- **Special Addresses**

The address 0:0:0:0:0:0:0:0, which can also be written as :: in zero compressed format, is known as the unspecified address. It is not a valid address to be assigned to any node or to be routed by an IPv6 router. This address is used to indicate the absence of an address and is used in situations such as automatic address configuration where the initializing host does not know the address to be specified in the Source Address field.

The unicast address 0:0:0:0:0:0:0:1, which can also be written as ::1 in zero compressed format is known as the loopback address and can be used by a node to send an IPv6 packet to itself. It is a link-local unicast address assigned to a virtual interface and is not assigned to any physical interface. Packets that are sent outside of a single node never contain this as the source address or the destination address, routers never

forward a packet with this address as the destination address and packets received on a physical interface are dropped if they contain this address as the destination address [16].

- **IPv6 addresses with embedded IPv4 addresses**

IPv4 mapped IPv6 addresses are used for interoperability with IPv4 applications. The IPv6 address `::FFFF:x.y.z.w` represents the IPv4 address `x.y.z.w`. With an IPv4 mapped IPv6 address, IPv6 only clients and servers can interoperate with IPv4 clients and servers.

- **IPv6 Header:**

4-bits	4-bits	4-bits	4-bits	4-bits	4-bits	4-bits	4-bits
Version	Traffic Class		Flow Label				
Payload Length				Next Header		Hop Limit	
Source Address							
Destination Address							

Figure 1.9 Make up of an IPv6 header

- **Version:** This 4-bit long field indicates the version of IP and is set to 6 in an IPv6 header.
- **Traffic Class:** The size of this field is 8-bits. It is used to specify an IPv6 packet's class or priority. This field is similar to IPv4's Type of Service field in its functionality.
- **Flow Label:** The 20-bit Flow Label field in the IPv6 header is used to specify the flow to which the IPv6 packet belongs to. There can be zero or more flows between a source and a destination. By default, the value for this field is zero which

indicates that the packet belongs to the default flow. Flows are used for quality of service required by communications such as audio and video. Intermediate routers provide special handling for packets with flow labels.

- Payload Length: It is a 16-bit unsigned integer that indicates the length of the IPv6 payload following this header in bytes. The length includes the extension headers. If the payload is bigger than that can be defined with 16-bits (65,535 bytes), the Jumbo Payload option is used in the Hop-by-Hop Options extension header.

- Next Header: These 8-bits indicate the type of header immediately following the IPv6 header. Usually indicates the next extension header or the upper level protocol such as TCP, UDP etc. When indicating an upper layer protocol above the Internet layer, the same values used in the IPv4 Protocol field are used here. There are several extension headers, each with a header format. The extension headers have the following Next Header values:

Next	Header
0	Hop-by-Hop Options Header
6	TCP
17	UDP
41	Encapsulated IPv6 Header
43	Routing Header
44	Fragment Header
46	Resource Reservation Protocol
50	Encapsulating Security Payload
51	Authentication Header
58	ICMPv6
59	No next header
60	Destination Options Header

Figure 1.10 IPv6 Extension headers

- Hop Limit: The size of this field is 8 bits. The Hop Limit is similar to the IPv4 TTL field and is used to avoid infinite travel of packets due to loops.
- Source Address: 128-bit address of the originating host.
- Destination Address: 128-bit address of the intended recipient of the packet (possibly not the ultimate recipient, if a Routing extension header is present).

CHAPTER 2

INTRODUCTION TO ETHERYATRI.NET

2.1 Software Agents

As computer networks grow and enabling technologies mature, modern software developers must fit the systems they build into a complex information grid consisting of servers and clients connected by a plethora of local- and wide-area networks. One of the most promising ideas for both unleashing the power of distributed systems and reducing their complexity is agent technology, especially "intelligent" agents. It is difficult to find a concise definition for agent that is in-line with most researchers' and developers' idea of what an agent should be. The definition given King, J.A. in AI Expert states, "An intelligent agent is considered to be a computer surrogate for a person or process that fulfills a stated need or activity. The surrogate entity provides decision-making capabilities that are similar to the described intentions of a human. This surrogate can be given enough of the persona of a user or the gist of a process to perform a clearly defined or delimited task. An intelligent agent can operate within the confines of a general or precisely represented need and within the boundaries of a given information space" [17]. A more expansive definition of Intelligent Agents given by IBM defines, "Intelligent Agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires. Intelligent agents

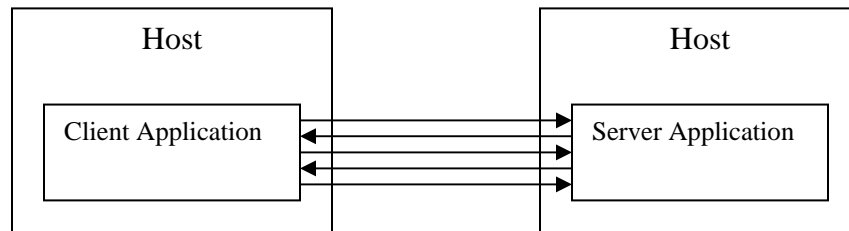
can further be described in terms of a space defined by these two dimensions of agency and intelligence” [18]. These and other definitions of agents indicate that agents should possess several or all of the following characteristics [18], [19]:

- Autonomy
- Authority
- Mobility
- Interaction with the system
- Asynchrony
- User representation
- Collaboration
- Adaptation, flexibility
- Reasoning and learning
- Independence
- Persistence
- Goal oriented
- Active/proactive

Agents are usually small in size; their collaboration with other agents is what makes applications. Mobile agents replace the old paradigm of "*bringing the data to the computation*" by a new one that says "*move the computation to the data*". Mobile agents have the ability to transport themselves from one host to another host that is reachable over a network. This ability allows mobile agent to move to the host at which the data or objects that the agent want to interact or collaborate with reside.

2.2 Mobile Agents, a closer look

Mobile agents provide advantages over the traditional client/server architectures. The following figure illustrates the network behavior of a typical client/server application.

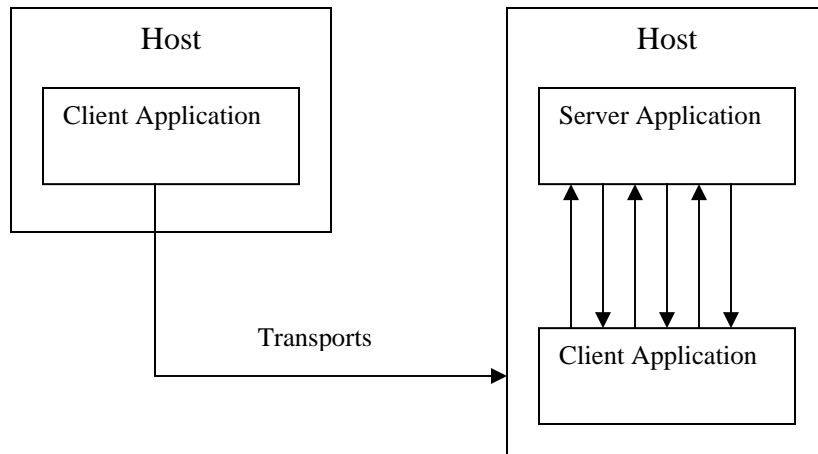


In a typical client/server scenario, client and server applications communicate via requests and responses causing several roundtrips across the network

Figure 2.1 Communication between a client and a server

A client/server application typically consists of two components: a client component and a server component. The client and server components are usually on separate machines and they communicate over a network. When the client needs to access data or other resources on the server, it sends a request to the server over the network. The server in turn sends a response to the request. This request/response communication occurs many a time in the traditional client/server architectures, such as Remote Procedure Calls (RPC).

As opposed to the traditional client/server architecture, the mobile agent architecture does not depend solely on the Remote Procedure Call. The following figure illustrates the network behavior of mobile agent architecture.



In a mobile agent architecture, the client moves to the server and the requests/responses take place on the server's host avoiding too many round trips across the network

Figure 2.2 Communication between mobile agents

Just like in the client/server architecture, there is a client component and a server component. When the client needs to access data or other resources that are on the server's host, it migrates to the server's host and makes its requests to the server within the host as opposed to across the network. When the client finishes its transaction with the server component, it moves back to its host. For this process to take place the mobile agent architecture has to provide means to transport *state* and *code*. State consists of the properties of the object before it starts its journey and is used to reconstruct the object when it arrives at the destination. Code includes the behavior that the object exhibits upon arrival at the destination [20].

2.2.1 Advantages of Mobile Agents

Mobile agent architectures have myriad advantages to it in the network computing world. Danny B. Lange and Mitsuru Oshima, mention the following as seven good reasons to using mobile agents [20]:

- They reduce the network load
- They over come network latency
- They encapsulate protocols
- They execute asynchronously and autonomously
- They adapt dynamically
- They are naturally heterogeneous
- They are robust and fault-tolerant

Mobile agents reduce the network load by packaging conversations and dispatching to the destination host and in turn reduce the flow of raw data in the network. As mobile agents can be dispatched from a central controller to act locally and directly execute the controller's directions, they facilitate real-time systems such as robots by reducing latencies. Since mobile agents move to the host to communicate with the components, it is easier to abstract protocols, thus making it easy for protocols to evolve. After being dispatched, the mobile agent can continue to work independent of its originating process and host even with the originating host disconnected. Mobile agent paradigm lends itself to let programmers, program adaptability into the components. Mobile agents by nature need to be heterogeneous for them to move to a variety of host systems. Since mobile agent systems need to be independent, they tend to be fault-tolerant and recover themselves in an unfavorable situation.

These characteristics allow mobile agents to be used to build applications in a variety of domains including Electronic Commerce, Personal assistance, Secure brokering, Distributed information retrieval, Telecommunication network services, Workflow applications and group ware, Monitoring and notification, Information dissemination and Parallel processing [21].

2.2.2 Mobile Agent Frameworks

Mobile agents demand certain features from the technologies that are used to implement the infrastructure. These features include:

- An RPC (Remote Procedure Call) or RMI (Remote Method Invocation) mechanism for hosts to communicate with each other and for the agents to notify (call methods on) other agents on remote hosts.
- An object transfer capability for messages to carry objects back and forth. This capability helps a host to send an agent to another host by allowing the agent object to be passed as a parameter to a message.
- A dynamic type loader that can load types into the remote host's memory to rebuild the agent that is transferred.
- Platform independence, to let agents that are created on one platform to be run on a remote host on a different platform.
- Security on the host to protect the host's local resources from a malicious agent.

Besides the above mentioned requirements, there are some other features that make the agent frameworks more useful to the developers and users. These include:

- A simple, robust and scalable programming environment preferably with rapid application development capabilities that include a vast set of APIs (Application Programming Interfaces)
- Platform independent GUI (Graphical User Interface) support including Drag and Drop features for the users to be able to interact with the agents in a consistent way independent of the operating system they are running the agent system on.
- Secure protocol support for the communication channels that carry agents and their data from one host to another.
- Multi-threading support to isolate agents from each other for enhanced security.
- An object oriented development environment that makes it easy to develop reusable components and behavior repositories to make agent development faster.
- Ability to work through firewall restrictions in corporate environments.

2.2.3 .NET Framework for Mobile Agents

The .NET Framework consists of two main parts: the common language runtime (CLR) and a vast class library. The .NET Framework provides support for several of the features needed to implement a mobile agent framework.

- The .NET Remoting provides tools and APIs for .NET applications to invoke methods on and pass parameters to objects that are hosted in a separate process usually on a remote machine. Using the .NET Remoting API, one can develop the communications layer required for the mobility of the agents and for agents to communicate with agent hosts and other agents residing in remote agent hosts. The .NET Framework also provides support for developing and consuming web services and APIs

for socket programming. Mobile agent frameworks that cannot be built using the .NET Remoting could use web services or low-level socket communication.

- The .NET Framework provides several ways to serialize objects. Objects can be serialized using binary serialization, XML serialization, SOAP serialization and custom serialization. Binary, XML and SOAP can further be customized by use of attributes and other mechanisms. Object serialization allows for objects that are in memory to be converted into bytes. Since agents are typically modeled as objects, they need to be serialized to be transmitted to remote hosts.

- The .NET Framework has built in support to dynamically download assemblies and load them into memory. This capability can be used by agent hosts to download the necessary assemblies while receiving an agent from another agent host.

- CLR acts as a virtual machine by abstracting the underlying platform from applications built on the .NET Framework. These applications are referred to as managed applications. It is possible for managed applications that are built on one platform to run on a different platform as long as the target platform has a CLR available for it. Mobile agent frameworks built using the .NET Framework implicitly are platform independent and allow agents to be platform independent as well.

- Code Access Security is an integral part of the CLR and provides services that can be utilized to verify the identity of an assembly against tampering. The .NET Framework Class Library contains several classes that can be used for authenticating, authorization and cryptography. These classes along with Code Access Security can provide the necessary security infrastructure for developing mobile agent frameworks as

security is a critical requirement in an environment where agents can arrive from remote computers and perform tasks on a local computer.

The .NET Framework has several other features mentioned below, that make it attractive for both mobile agent framework developers and mobile agent application developers.

- The .NET Framework provides support for multiple-programming languages by allowing language compilers to compile code into Intermediate Language (IL) that can be understood by the CLR. And the .NET Framework SDK comes with two programming languages namely C# and VB.NET. Both C# and VB.NET are rich with object oriented features such as encapsulation, inheritance, selective polymorphism, operator overloading, event driven programming using delegates, enhanced loops and switch statements, generics etc. C# offers a syntax that is familiar to Java, C and C++ programmers while VB.NET offers a syntax that is familiar to Visual Basic developers. The .NET Framework provides features such as automatic memory management, runtime type checking and a huge library of classes. These features make .NET a simple, robust and scalable programming environment.

- The .NET Framework Class Library includes several classes to support encryption and decryption of data. This can be used to encrypt communication between agent hosts and agents.

- The .NET Framework allows for creating AppDomains that can load agent assemblies and can be isolated from the agent host AppDomain. Through its support for inter AppDomain and inter process communication using .NET Remoting and other

features, the .NET Framework lets developers provide enhanced security to agent hosts and agents from malicious agents.

- There are various Integrated Development Environments (IDEs) that are available for developers to build .NET applications. Microsoft Visual Studio .NET is such an IDE that integrates web and windows development and through its plug-in architecture supports multiple languages in the same IDE. Project and code templates can be used to implement repositories of agent behaviors and promote re-usability. These Rapid Application Development features both in the languages and in the IDE make the .NET Framework and Visual Studio .NET attractive for mobile agent developers. Through its support for multiple languages, it welcomes even more programmers to the community.

- ASP.NET, an integral part of the .NET Framework provides support for web applications and web services thus opening the possibility for developing a mobile agent framework that can use Hyper Text Transfer Protocol (HTTP) and Extensible Markup Language (XML) to overcome firewall restrictions.

The .NET Framework is also the foundation for many technologies that Microsoft is releasing in the near future. These include:

- Windows Presentation Foundation code named Avalon consists of a declarative programming language called Extensible Application Markup Language (XAML) and a library of classes that can utilize the rich UI experience provided by the framework. XAML should encourage agent developers to build nice graphical user interfaces that would improve the usability of agent hosts and agent based applications.

- Windows Communications Foundation code named Indigo provides the communications infrastructure necessary for building future mobile agent frameworks. Indigo has features such as built-in security, transaction management, service-oriented programming model, web services architecture and pluggable communication channels. These features allow for the development of a mobile agent framework that can overcome firewall restrictions, is secure, collaborate with other mobile agent frameworks, is adaptive to newer communication protocols such as IPv6 and potentially drive towards a mobile agent protocol that can be used by heterogeneous systems.

2.2.4 EtherYatri.NET

EtherYatri.NET is one of the few mobile agent frameworks developed using the .NET Framework. EtherYatri.NET was initially developed in 2003 by Siddharth Uppal of National Institute of Technology, Warangal, India as part of his undergraduate project. The toolkit is further developed and enhanced by its open source community [21], [22]. EtherYatri.NET toolkit includes Windows Agent Host (WinAH) that hosts mobile agents. It also comes with Visual Studio item templates that make it easy to develop mobile agents. EtherYatri.NET framework uses .NET features such as object serialization, remoting, object-oriented design etc. With the introduction of EtherYatri.NET, mobile agent community is now equipped with yet another framework to build mobile agent applications with. Providing IPv6 support for EtherYatri.NET encourages both the IPv6 and mobile agent communities to help find applications for both technologies. This thesis investigates IPv6 support in EtherYatri.NET.

CHAPTER 3

LITERATURE REVIEW

Many research hours have been spent in both mobile agents and IPv6. The result is various mobile agent frameworks in Java & .NET and quite a few applications and frameworks that are IPv6 enabled.

3.1 Agent Technologies

Existing work in the area of Agent Technologies can be categorized into three categories:

- Non-Intelligent Agents
- Intelligent Agents
- Mobile Agents

3.1.1 Non-Intelligent Agents

Lot of existing software applications come into this category. Typically any program that has characteristics such as asynchronous execution, scheduling etc., is an agent. Anti-Virus programs can monitor currently running applications asynchronously for any malicious activity and/or can be scheduled to run in the absence of the user to scan for viruses. System tools under windows operating systems when configured, search for unused temporary files and delete them to save disk space on the computer. More advanced applications such as online stock brokerage systems have program logic that can buy or sell stocks in the absence of the user when certain criteria are met. Some

online job search websites such as Monster.com and Dice.com have search agents that can be configured to notify the users through various communication channels such as e-mail etc., when jobs that match users' interests are available.

Above mentioned applications are only a few of the many applications that are currently helping users to perform tasks on behalf of the users. Although seldom accepted by researchers as Software Agents due to their lack of intelligence, some of the above mentioned applications are already known to the masses as Agents. Some other names that are in use for such programs include knowbots (short for knowledge-based robots), softbots (short for software robot), taskbots (short for task-based robots), userbots (short for user robots), personal agents, autonomous agents and personal assistants. These applications are acting asynchronously on behalf of the users. They are intelligent to the extent of the logic users build into them. But they are not intelligent enough to change their own logic or learn from their past.

3.1.2 Intelligent Agents

There are numerous software applications that come into this category. These applications often possess Artificial Intelligence capabilities and thus are autonomous. Existing Intelligent Agent projects include:

- **NARVAL Project - Intelligent Personal Assistant**

NARVAL an acronym for Network Assistant Reasoning with a Validating Agent Language is an open-sourced framework that includes a language, an interpreter and an Integrated Development Environment for setting up intelligent personal assistants (IPA). Using NARVAL one can setup a Personal Assistant that can run on one's machine or on

a remote server and can be communicated with via various communication mechanisms such as email etc [23].

- **CLIPS**

CLIPS stands for C Language Integrated Production System. It provides a development environment for creating expert systems based on rules or objects [24]. CLIPS uses the Rete algorithm to process rules efficiently [25].

- **JESS**

JESS stands for Java Expert System Shell. JESS a rule engine and scripting environment developed using Sun's Java™ language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA [26]. JESS's design was inspired by the CLIPS system and added capabilities such as backwards chaining and the ability to “reason”. Using Jess, one can build Java applets, applications and scripts that have the capacity to work as intelligent agents using pre-defined knowledge in the form of declarative rules. Like CLIPS, Jess uses the Rete algorithm to process rules efficiently [25].

- **LISA**

LISA stands for Lisp-based Intelligent Software Agents. LISA is implemented in the Common Lisp Object System (CLOS). LISA is a production-rule system influenced by CLIPS and JESS and uses Rete algorithm [25]. LISA does not impose special contract restrictions such as inheriting from a specific class on Java objects and thus makes it possible to add reasoning functionality to existing CLOS applications. Since LISA is an extension to Common Lisp, all the features of Lisp are available [27].

- **IDOL Agents from Autonomy**

Intelligent Data Operating Layer (IDOL) is a digital content processing system developed by the Autonomy Corporation in San Francisco; CA. IDOL offers several features that enable organizations to effectively use digital content. Personalized Agents is one such feature. IDOL's Personalized Agents can be configured to alert and present users with customized reports by monitoring a variety of content repositories including news feeds, chat-streams, intranet and internet content [28].

3.1.3 Mobile Agents

Agents, due to their representative nature, have an inherent desire to network with systems including other agents to perform a task delegated by the owner in the most optimal way possible. As computer networks became more common, the demand for distributed applications increased tremendously. And many frameworks that enabled agents to use resources and data on remote systems were developed expanding the reach of a personal intelligent agent far away from the user's computer. Many of these frameworks were built with support for inter-agent communication while some had support for migration of the agents themselves. Following are some of the frameworks and enabling technologies:

- **KQML**

KQML stands for Knowledge Query and Manipulation Language [29]. KQML, developed by the external interfaces working group of the DRAPA Knowledge Sharing Effort, is a language for communication between agents. KQML is intended to be a high-level language to be used by knowledge-based systems to share knowledge at run time.

KQML provides a universal communication language that allows for interaction and interoperability with other software agents.

- **JASPER**

JASPER stands for Joint Access to Stored Pages with Easy Retrieval [30]. JASPER is an intelligent agent system that facilitates agents which on behalf of the user can retrieve information from the internet based on the user's input, learn from user's behavior and communicate with other agents allowing users to quickly and effectively share relevant information.

- **JAT**

JAT stands for Java Agent Template [31]. JAT is developed using the Java language and provides a template for building agents which communicate with other agents over the Internet. JAT abstracts the low level messaging and communication functionality so that agent authors can focus on application design. JAT agents are very close to being mobile agents but do not have built-in functionality to migrate from one host to another. Agents communicate with each other using KQML messages and could use Java's Remote Method Invocation (RMI) to migrate to another host via an agent. JAT agents could be Java applications or Applets.

While distributed agent frameworks and enabling technologies evolved, the underlying Remote Procedure Call (RPC) mechanisms also became more sophisticated and virtual machines were built. Some of the popular RPC mechanisms that let programs running on heterogeneous systems interact are Common Object Request Broker Architecture (CORBA) [32], Java Remote Method Invocation (RMI) [33], Microsoft Distributed Component Object Model (DCOM), .NET Remoting [34] and Web Services

[35], [36]. And the virtual machines supported the popular “Write Once Run Anywhere” paradigm. With these technologies in place, building frameworks that enabled software programs to communicate with and send objects to programs running on remote systems became more feasible.

Now, many mobile agent frameworks are available that support migration of agents. These include:

- **AgentSpace**

AgentSpace [37] is a mobile agent system built using Voyager [38] for communication layer and Java as the development environment. AgentSpace system consists of three components: a server, a client and an API. The server hosts and provides necessary services for the agents. The AgentSpace client is a loosely coupled system of applets that enables management of the agents and the servers remotely. AgentSpace API is used to develop agents that adhere to the contracts defined by the framework [39]. AgentSpace offers some additional features that usual Java based mobile agent frameworks don't offer. These include a database to store agents, a sophisticated middleware offered via Voyager that brings a richer communication mechanism than Java RMI and allows agent authors to specify a custom callback method using Java reflection, thus allowing for a more elegant code. Voyager is an Object Request Broker (ORB) from Recursion Software, Inc. Voyager simultaneously supports both CORBA and RMI and has support for Web Services through the integration of SOAP and WSDL.

- **Aglets SDK**

Aglet SDK is a mobile agent software development kit developed at IBM's Tokyo research laboratories [40]. Aglet stands for light weight agent. IBM's Aglets SDK is built

on Java RMI and ships with Tahiti, an Aglet server. Aglets framework uses Agent Transfer Protocol (ATP) as the underlying protocol to transport agents. One can develop Aglets using the SDK and host them in Tahiti. Tahiti has a simple to use user interface to manage the Agent Host. Since Aglets SDK is built on top of JDK, one can use all the libraries that can be used from any Java program. Besides the support for mobility, Aglets SDK also supports persistence of the agents.

- **Ajanta**

Ajanta is a mobile agent platform [41] built in Java. Agents can be deployed and run in the context of Agent Servers and can migrate from one server to another one. Applications built using Ajanta include "Active Monitoring of Network Systems using Mobile Agents" [42].

- **ARA**

Agents for Remote Action (ARA), is a Tcl-based mobile agent framework from the University of Kaiserslautern [43]. Agents are executed as independent processes so that they can be isolated from potentially malicious agents. Agents run in the core that is a single application process on top of the operating system. ARA also supports agents that are developed in different programming languages through interpreters. Mobile agents are usually run in an interpreter for reasons of portability and security.

- **Concordia**

Mitsubishi's Concordia [44] offers a full-featured middleware infrastructure for the development and management of network-efficient mobile agent applications. Similar to other Java Mobile Agent Frameworks, Concordia's agents are hosted in agent servers. In Concordia system, agents migrate through Conduit Servers. One of the important

differences between Concordia and other agent frameworks is the ability to have different entry points for the agent at different stops during its itinerary.

- **D'Agents**

D'Agents, formerly known as Agent Tcl is an academic project from Dartmouth [45] and is based mainly on Tool Control Language (TCL). Security is one of the main research areas and is implemented with Safe-TCL, a restricted environment for TCL. Unlike many mobile agent environments the D'Agents since 2.0 has support for multiple languages.

- **FarGo**

FarGo [46] is an environment for portable applications. The programming model extends Java, yet remains as close as possible to the regular Java programming model. FarGo has the capability to move an entire application among hosts while the application is still running. The layout can be programmed separately from the application's logic either within the application, or externally using a scripting language. Unlike in the other mobile agent environments, in FarGo complex applications which may include clients and servers take the place of agents. So rather than moving an agent FarGo has the ability to move an entire multi-tier application among hosts. FarGo is a research project, conducted at the Technion - Israel Institute of Technology, Department of Electrical Engineering.

- **Tryllian's Agent Development Kit**

Agent Development Kit (ADK) [47] is one of the few Mobile Agent Frameworks available commercially. ADK is built in Java, Extensible Markup Language (XML) [48] and JXTA [49]. The peer-to-peer communication between agents is achieved through a

JXTA-based distributed architecture with XML message-based communication that supports both FIPA [50] and SOAP. ADK environment comprises of Habitats, Rooms and Agents. Habitat is where agents, rooms, and messaging are hosted. A habitat is a collection of rooms that share a common code base and a Java Virtual Machine (JVM). A habitat also provides an Agent Runtime Environment (ARE), i.e., a JVM that provides various services. Examples of services include the agent execution model, inter platform communication, inter habitat agent travel, room & agent persistence and security model. A room is a container that holds agents and their resources. A room is the travel destination for mobile agents and the rooms typically provide agents with a registry service, where agents can check in and out when they enter or leave the room. This is also where agents can advertise their properties and capabilities, in order for other agents to find them. Each room has a unique address that agents use during their travels. An agent has the ability to move itself to another location, but the location needs to have a habitat installed. Besides these components, as a commercial development kit, ADK comes with some pre-built agent behaviors to shorten development times.

- **Gypsy**

The Gypsy project on mobile agents is a mobile agent development environment developed by the Distributed Systems Group of the Information Systems Institute at the Technical University of Vienna. Gypsy has Agent Servers, Communicators, Places and Agents themselves. Agents move from one Place that is hosted in a Gypsy Server to another Place hosted on another Gypsy Server through Communicators [51]. Gypsy has the capability to transport agents by email through its Email Agent Communicator.

Gypsy also implements the agents as Java Beans [52] which enables these agents to be managed in a Graphical User Interface environment with features such as drag and drop.

- **TACOMA**

TACOMA (Tromsø And Cornell Moving Agents) [53] project is a collaboration among the University of Tromsø, Norway, Cornell University and University of California, San Diego. Originally based on UNIX and Tcl-TCP (Transmission Control Protocol), the latest version of TACOMA is written in C and supports agents that are written in various programming languages such as C, Tcl/Tk, PERL, Python and Scheme. TACOMA has also been ported to Windows NT, Windows CE and Palm OS.

The .NET Framework allows mobile agent frameworks to provide multiple programming language support to agent authors. There are rapid development tools available from Microsoft to speed up the process of developing agent frameworks as well as agents. Several mobile agent frameworks are built using the .NET Framework.

- **MAPNET**

MAPNET [54] is a .NET based mobile agent platform. Its basic design is based on the Mobile Agent System Interoperability Facilities (MASIF) [55] specification. The MAPNET system is developed at the Technical University of Varna, Bulgaria and includes an agent server that hosts mobile agents. It relies heavily on .NET features such as threading, remoting, serialization and security. They've also implemented a global service for registering agents and agent service using remoting. Future work directions include strengthening security by enforcing agents' permissions, and extending the class-loading service.

- **EtherYatri.NET**

EtherYatri.NET [21] was originally developed as a research project at National Institute of Technology in Warangal, India. It later has been improved and maintained by the open source community. EtherYatri.NET is a .NET based mobile agent system. It includes WinAH (Windows Agent Host) that hosts mobile agents and leverages Visual Studio.NET's rapid application development capabilities by providing agent templates and using declarative programming model at places. EtherYatri is also used at Monash University in Australia as part of their mobile agent coursework. Current improvements are being made to the framework by the open source community [22].

Some other mobile agent technologies are JATLite from Stanford University, Jumping Beans from Aramira Corporation, Knowbot System Software from Corporation for National Research Initiatives, Mobile Agent Platform from Universita' di Catania, Mobile Code Toolkit from Carleton University, MONADS from the University of Helsinki, Plangent from Toshiba Corporation and SOMA: Secure and Open Mobile Agent from University of Bologna. FIPA, JAFMAS [56] and MARS [57] are standards and frameworks that facilitate interoperation of heterogeneous agent systems.

David Wallace Croft's "Intelligent Software Agents: Definitions and Applications" [58] and "Software Agents Fundamentals" from BTextact Technologies [59] have definitions for many kinds of agents that include Software Agents, Intelligent Software Agents, Mobile Agents, Distributed Agents, Multiple Agents, Collaborative Agents, Social Agents, Interface Agents, Reactive Agents and Hybrid Agents. They also have useful discussions about what agents are and what they are not.

3.2 Internet Protocol v6

A lot of research work is going on to define IPv6 protocol, addressing and other pieces. At the same time much work has been done by universities, corporations and open-source communities to get IPv6 implementations into as many hands as possible. The result is that now many operating systems ship with one or more implementations that support IPv6.

Shortage of IP addresses, technical limitations of IPv4 were among several reasons that prompted IETF in 1992, to start the initial discussions about a new internet protocol. The initial proposals used several names including “IPv7” for the new protocol. A common name, IP: Next Generation (IPng) was introduced in the initial Requests for Comments (RFC) to generally identify the eventual next generation of IP. A comparison of proposals submitted in May 1993 [60] and a white paper solicitation in December 1993 [61] are examples of such RFCs. After two years of discussions, IPv6 was chosen as the final IPng proposal and starting with a specification for IPv6 in December 1995 [62], many proposals have been made covering various aspects of IPv6 and rendering some previous proposals obsolete [63]. Specifications in several areas of IPv6 have matured over many revisions while other areas of IPv6 are still being worked on. There have been new proposals made as late as 2005 including one on IPv6 scoped address architecture in March 2005 [64].

Microsoft Research made available an IPv6 implementation in 1998. Two years later, in March 2000 a technology preview for Windows 2000 systems was released. When Windows XP was released in October 2001, it included an IPv6 stack and necessary components for developers to enable IPv6 in their applications. The first

production stack and components were shipped with Microsoft Windows Server 2003 family of operating systems in March 2003. The Advanced Networking Pack for Windows XP that was released in July 2003 included an IPv6 Internet Connection Firewall and a Teredo client. With the release of Windows XP Service Pack 2 in August 2004, Microsoft replaced the IPv6 Internet Connection Firewall by adding support for IPv6 in their regular Windows Firewall. Today both Windows Server 2003 family and Windows XP ship with various implementations of IPv6 including a dual stack and IPv6 over IPv4 tunneling protocols such as Teredo, Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) [65].

Other operating systems such as Macintosh, UNIX, Linux and several routers provide IPv6 implementations. Macintosh OS X, also known as Jaguar supports IPv6 since version 10.2. The KAME project and INRIA provide free implementations of IPv6 for BSD Unix. Following is a table that shows IPv6 support in various flavors of UNIX.

UNIX OS	Earliest version with built-in support	Earliest version supporting KAME	Earliest version supporting INRI
IBM AIX	4.3		
BSDI 3.1	4.0	3.1	
COMPAQ Tru64	4.0D		
Free BSD	4.0	2.2.8	2.2.5
IRIX	6.5.19		
LINUX	Kernel 2.2		
Net BSD	1.5	1.4.2	1.3.3
Open BSD	2.7	2.7	
Solaris	8		
HP-UX	11i		

Figure 3.1 IP v6 support in various operating systems

Several router vendors such as Cisco, Hitachi, Nortel Networks, Juniper Networks, 6wind, IJ and Yamaha offer hardware that support IPv6 [66].

With support available from OS and hardware vendors, many researchers and enthusiasts started porting several applications to run on IPv6. These applications span a across areas such as mail, multi-media, DNS and Remote Access [67]. Both Java and the .NET Framework have support for enabling IPv6 in the applications. There is ongoing research work in the area of building IPv6 enabled distributed applications based on Java & the .NET Framework. The topic of thesis and a unicast discovery protocol implemented using IPv6 and JINI [68] are examples of such work.

CHAPTER 4

ENABLING IPV6 IN ETHERYATRI.NET

This thesis investigates the IPv6 capabilities of the EtherYatri.NET mobile agent framework. An IPv6 test environment is setup using virtual machines. The EtherYatri.NET framework's open source project is used to understand its architecture and the framework is evaluated against the test IPv6 network. A list of items to be altered in the framework for it to work with IPv6 protocol is formed. These changes are made in the open source project and the project is re-evaluated against the IPv6 test network.

4.1 The Research Environment

The environment used for this investigation consists of the following components:

- Virtual PC virtual machine software
- Windows Server 2003 operating system with built-in IPv6 stack
- Visual Studio 2005 development environment
- .NET Framework SDK 2.0
- EtherYatri.NET mobile agent toolkit

4.1.1 Setting up an IPv6 test environment

Two virtual computers named host1 and host2 are created using the Virtual PC software and Windows Server 2003. A local only network is setup between host1 and host2. On a local only virtual network, virtual machines can see each other and are

isolated from the host machine's network. Other kinds of networks supported by the Virtual PC are used to transfer files to and from the host machine.

By default, IPv6 is disabled in Windows Server 2003. The ipconfig command can be used to see the IP configuration on a windows machine.

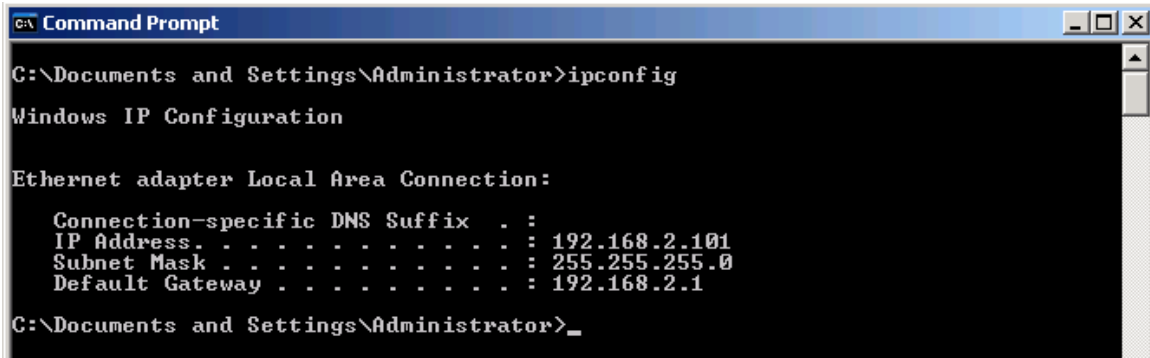


Figure 4.1 Windows IP Configuration on host1 before installing IPv6

IPv6 is enabled on both the computers using the install protocol feature in the network adapter properties dialog.

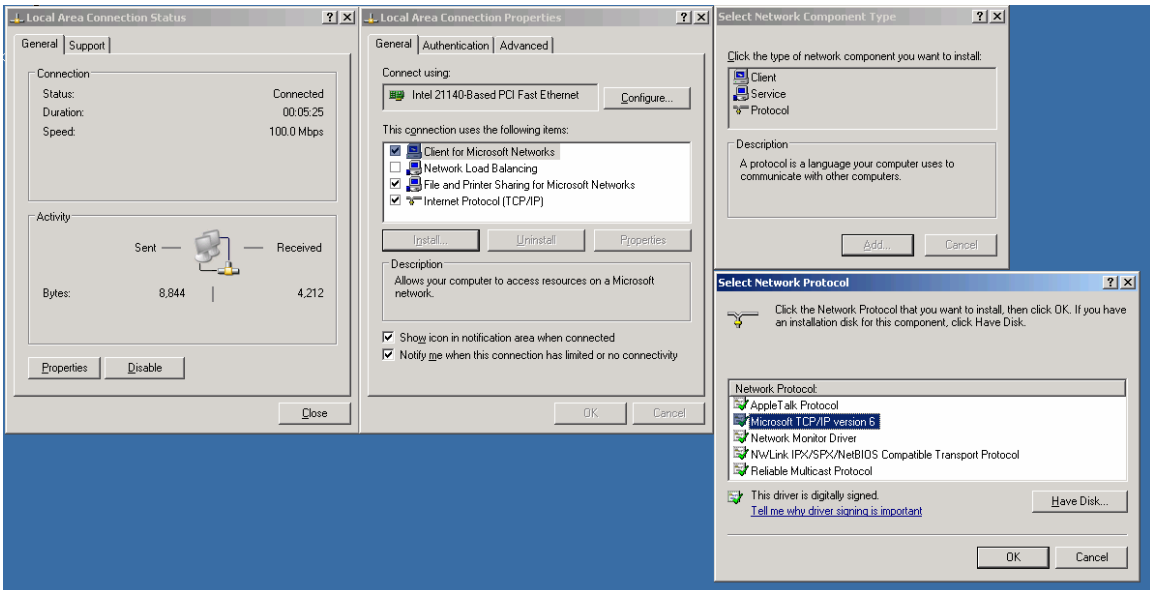
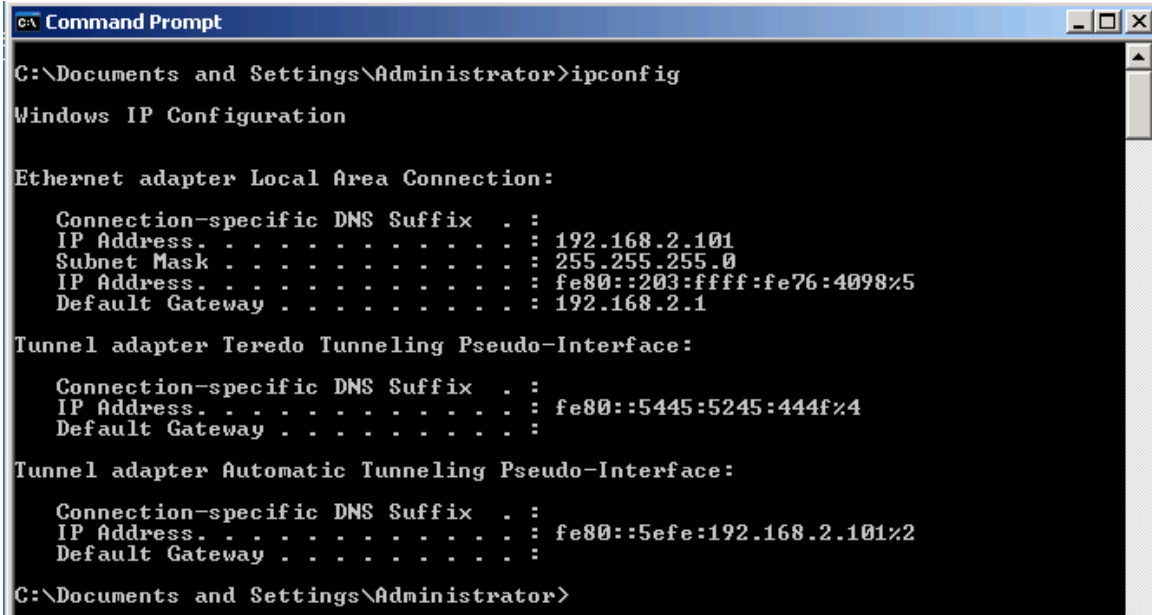


Figure 4.2 Installing IPv6 protocol on Windows Server 2003

Running the ipconfig command after installing the IPv6 protocol shows the IPv6 address, Teredo Tunneling address and Automatic Tunneling address. Installing IPv6 protocol automatically creates the Teredo and Automatic Tunneling interfaces.



```
c:\ Command Prompt
C:\Documents and Settings\Administrator>ipconfig
Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . .               : 192.168.2.101
    Subnet Mask . . . . .             : 255.255.255.0
    IP Address. . . . .               : fe80::203:ffff:fe76:4098%5
    Default Gateway . . . . .         : 192.168.2.1

Tunnel adapter Teredo Tunneling Pseudo-Interface:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . .               : fe80::5445:5245:444f%4
    Default Gateway . . . . .         :

Tunnel adapter Automatic Tunneling Pseudo-Interface:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . .               : fe80::5efe:192.168.2.101%2
    Default Gateway . . . . .         :

C:\Documents and Settings\Administrator>
```

Figure 4.3 Windows IP Configuration on host1 after installing IPv6

To test the IPv6 setup and the IPv6 capabilities of the .NET Framework, a simple client-server program is written using C# and System.Net classes of the .NET Framework. This program consists of three files namely Chat.IO.cs, Chat.Server.cs and Chat.Client.cs.

The Chat.IO.cs file contains a multi-threaded Chat.IO.ConcurrentIO class that simultaneously sends local console input to the remote stream and prints text from the remote stream on the local console. This class is used by both Server and Client classes for IO.

```

using System;
using System.IO;
using System.Threading;

namespace Chat.IO
{
    public class ConcurrentIO
    {
        private Stream mRemoteStream;

        public ConcurrentIO(Stream remoteStream)
        {
            this.mRemoteStream = remoteStream;
        }

        private ManualResetEvent mLocalInputWaitHandle = new ManualResetEvent(false);
        private ManualResetEvent mRemoteInputWaitHandle = new ManualResetEvent(false);
        public void Start()
        {
            Thread remoteInputThread, localInputThread;
            Console.WriteLine("Enter ^Z to end.");

            localInputThread = new Thread(new ParameterizedThreadStart(this.ReadWrite));
            localInputThread.IsBackground = true;
            localInputThread.Start(false);

            remoteInputThread = new Thread(new ParameterizedThreadStart(this.ReadWrite));
            remoteInputThread.IsBackground = true;
            remoteInputThread.Start(true);

            int index = EventWaitHandle.WaitAny(new WaitHandle[]
                { this.mLocalInputWaitHandle, this.mRemoteInputWaitHandle });
            throw new ApplicationException((index == 0 ? "Local" : "Remote")
                + " input has ended.");
        }

        private void ReadWrite(object useRemoteStreamAsInput)
        {
            TextReader reader = null;
            TextWriter writer = null;
            ManualResetEvent waitHandleToBeSet = null;
            string line = null;

            try
            {
                if((bool)useRemoteStreamAsInput)
                {
                    reader = new StreamReader(this.mRemoteStream);
                    writer = Console.Out;
                    waitHandleToBeSet = this.mRemoteInputWaitHandle;
                }
                else
                {
                    reader = Console.In;
                    writer = new StreamWriter(this.mRemoteStream);
                }
            }
        }
    }
}

```

```

        waitHandleToBeSet = this.mLocalInputWaitHandle;
    }

    while ((line = reader.ReadLine()) != null)
    {
        writer.WriteLine(line);
        writer.Flush();
    }
}
catch(IOException ioex){}
finally
{
    waitHandleToBeSet.Set();
}
}
}
}

```

Source code 4.1 The ConcurrentIO class from Chat.IO.cs

The Chat.Server.cs file contains a Chat.Server class that opens a socket and listens for incoming connections on an IP address and a port number that can be passed as command line arguments. It accepts a single connection, prints out the connection end point information and hands over the remote stream to an instance of the ConcurrentIO class for IO processing.

```

using System;
using System.Net;
using System.Net.Sockets;

namespace Chat
{
    class Server
    {
        static void Main(string[] args)
        {
            Socket connection=null;
            try
            {
                if (args.Length != 2)
                {
                    Console.WriteLine("Usage: Chat.Server <ipaddress> <port>");
                    return;
                }
                TcpListener server = new TcpListener
                    (IPAddress.Parse(args[0]), int.Parse(args[1]));
                server.Start();
                connection = server.AcceptSocket();

                Console.WriteLine("Local:");
                Console.WriteLine("\tIP Address:\t\t" +
                    ((IPEndPoint)(connection.LocalEndPoint)).Address.ToString());
                Console.WriteLine("\tIP Address Family:\t" +
                    ((IPEndPoint)(connection.LocalEndPoint)).AddressFamily);
                Console.WriteLine("\tTCP Port:\t\t" + ((IPEndPoint)(connection.LocalEndPoint)).Port);
                Console.WriteLine("Remote:");
                Console.WriteLine("\tIP Address:\t\t" +
                    ((IPEndPoint)(connection.RemoteEndPoint)).Address.ToString());
                Console.WriteLine("\tIP Address Family:\t" +
                    ((IPEndPoint)(connection.RemoteEndPoint)).AddressFamily);
                Console.WriteLine("\tTCP Port:\t\t" + ((IPEndPoint)(connection.RemoteEndPoint)).Port);

                Chat.IO.ConcurrentIO io =
                    new Chat.IO.ConcurrentIO(new NetworkStream(connection));
                io.Start();
            }
            catch(Exception ex)
            {
                if(ex.Message.StartsWith("Local"))
                    connection.Close();
                Console.WriteLine(ex.Message);
            }
        }
    }
}

```

Source code 4.2 the Chat.Server class from Chat.Server.cs

The Chat.Client.cs file contains a Chat.Client class that opens a socket connection to a server listening on an IP address and a port number that can be passed as command line arguments. Once the connection is established, it prints out the connection end point information and hands over the remote stream to an instance of the ConcurrentIO class for IO processing.

```
using System;
using System.Net;
using System.Net.Sockets;

namespace Chat
{
    class Client
    {
        static void Main(string[] args)
        {
            Socket connection = null;
            IPAddress serverIP = null;
            try
            {
                if (args.Length != 2)
                {
                    Console.WriteLine("Usage: Chat.Client <ipaddress> <port>");
                    return;
                }
                serverIP = IPAddress.Parse(args[0]);
                connection = new Socket(serverIP.AddressFamily,
                    SocketType.Stream, ProtocolType.Tcp);
                connection.Connect(serverIP, int.Parse(args[1]));
                Console.WriteLine("Local:");
                Console.WriteLine("\tIP Address:\t\t" +
                    ((IPEndPoint)(connection.LocalEndPoint)).Address.ToString());
                Console.WriteLine("\tIP Address Family:\t" +
                    ((IPEndPoint)(connection.LocalEndPoint)).AddressFamily);
                Console.WriteLine("\tTCP Port:\t\t" + ((IPEndPoint)(connection.LocalEndPoint)).Port);
                Console.WriteLine("Remote:");
                Console.WriteLine("\tIP Address:\t\t" +
                    ((IPEndPoint)(connection.RemoteEndPoint)).Address.ToString());
                Console.WriteLine("\tIP Address Family:\t" +
                    ((IPEndPoint)(connection.RemoteEndPoint)).AddressFamily);
                Console.WriteLine("\tTCP Port:\t\t" + ((IPEndPoint)(connection.RemoteEndPoint)).Port);

                Chat.IO.ConcurrentIO io =
                    new Chat.IO.ConcurrentIO(new NetworkStream(connection));
                io.Start();
            }
        }
    }
}
```

```

        catch (Exception ex)
        {
            if (ex.Message.StartsWith("Local"))
                connection.Close();
            Console.WriteLine(ex.Message);
        }
    }
}

```

Source code 4.3 the Chat.Client class from Chat.Client.cs

The Microsoft .NET Framework Version 2.0 Redistributable Package (x86) is available for free on Microsoft Developer Network downloads site and includes the compilers and libraries needed to build and run the chat client and server programs. By default the redistributable package installs the files under a Microsoft.NET\Framework\v2.0.50727 folder under the windows folder. This folder is referred to as the .NET Framework folder in the rest of this document. A folder by the name chat is created to store the files created for the chat program. Chat.IO.cs, Chat.Server.cs and Chat.Client.cs files are saved in this folder. First the Chat.IO.cs compiled as a dynamic linked library (dll) using the csc compiler that resides in the .NET Framework folder. Then Chat.Server.cs and Chat.Client.cs files are compiled into two separate executables (exes) by referencing the Chat.IO.dll.

```

C:\ Command Prompt
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>cd \windows\Microsoft.NET\Framework\v2.0.50727

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>md chat

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>cd chat

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat>dir
Volume in drive C has no label.
Volume Serial Number is 841D-439C

Directory of C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat

11/18/2005  06:26 PM    <DIR>          .
11/18/2005  06:26 PM    <DIR>          ..
11/17/2005  07:04 PM                1,873 Chat.Client.cs
11/15/2005  07:45 PM                2,421 Chat.IO.cs
11/17/2005  07:04 PM                1,795 Chat.Server.cs
               3 File(s)                6,089 bytes
               2 Dir(s)      14,459,314,176 bytes free

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat>..\csc /t:library Chat.IO.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

Chat.IO.cs(64,13): warning CS0168: The variable 'ioex' is declared but never
used

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat>..\csc /r:Chat.IO.dll Chat.Ser
ver.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat>..\csc /r:Chat.IO.dll Chat.Cl
ient.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat>dir
Volume in drive C has no label.
Volume Serial Number is 841D-439C

Directory of C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat

11/18/2005  06:27 PM    <DIR>          .
11/18/2005  06:27 PM    <DIR>          ..
11/17/2005  07:04 PM                1,873 Chat.Client.cs
11/18/2005  06:27 PM                4,608 Chat.Client.exe
11/15/2005  07:45 PM                2,421 Chat.IO.cs
11/18/2005  06:27 PM                4,096 Chat.IO.dll
11/17/2005  07:04 PM                1,795 Chat.Server.cs
11/18/2005  06:27 PM                4,608 Chat.Server.exe
               6 File(s)                19,401 bytes
               2 Dir(s)      14,459,293,696 bytes free

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\chat>

```

Figure 4.4 Building the source files using the .NET Framework 2.0 Redistributable

The same process is repeated in both host1 and host2. Chat.Server is run on host1 using host1's IPv6 address and port 1234. Chat.Client is run on host2 by passing host1's IPv6 address and port 1234. Messages that are printed out by the client and the server

indicate that an IPv6 connection has been established between the two. Some test messages are exchanged between the two and a stream terminator input is used to terminate the input on the client which then closes the connections both on the server and the client.

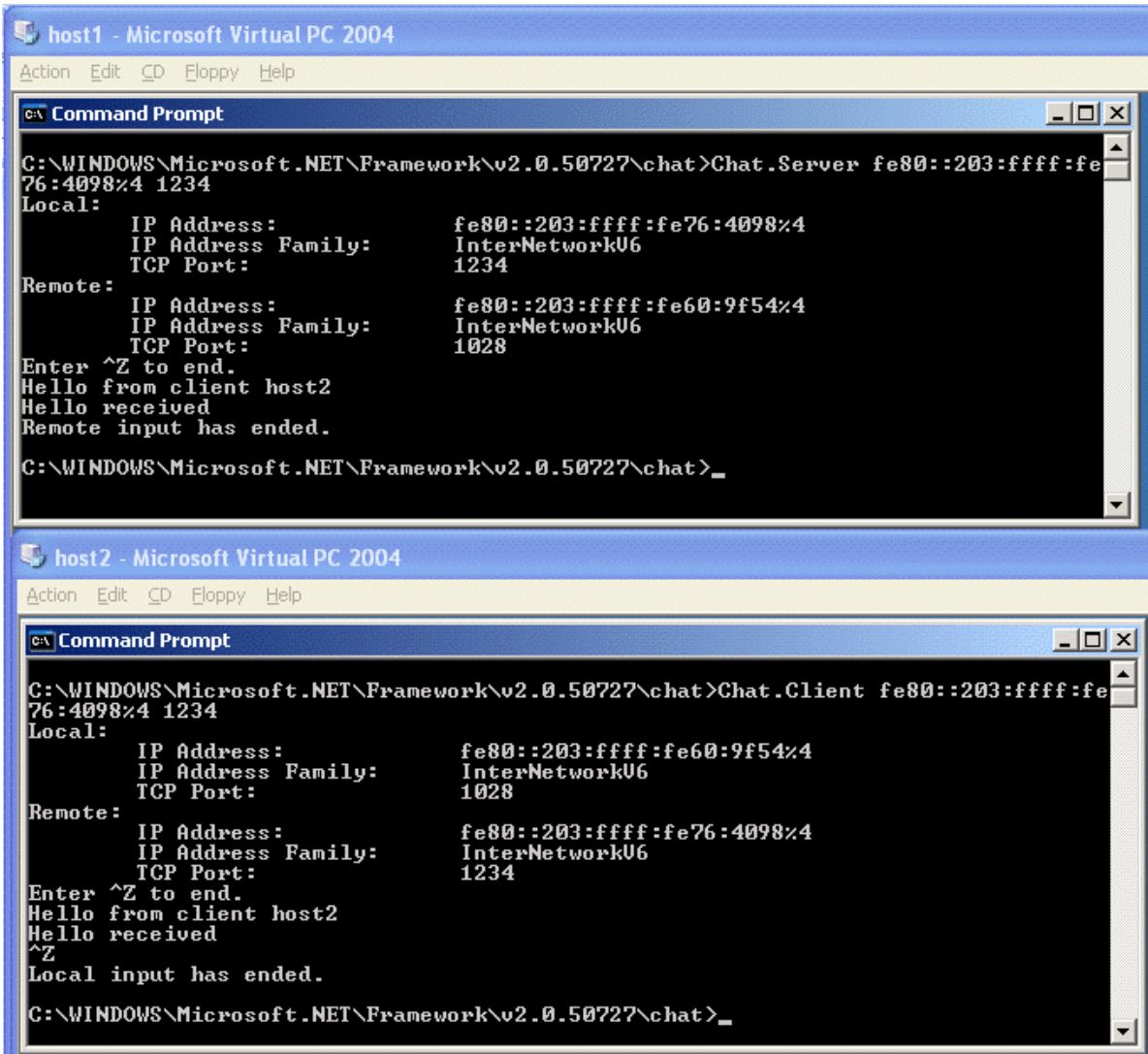


Figure 4.5 The chat client and server communicating over IPv6 between host1 and host2

4.1.2 Setting up EtherYatri.NET development environment

EtherYatri.NET is an open source project and the source code is available to download from GotDotNet workspace [22]. The latest version of the code available has

Visual Studio projects included to organize the source code into multiple dll projects. To take advantage of the new features, especially the IPv6 capabilities of the .NET Framework 2.0, the EtherYatri.NET project files were converted to Visual Studio 2005.

The HelloWorld Demo agent that comes with EtherYatri.NET was used for initial testing of the projects after converting them to the .NET Framework 2.0. Two instances of WinAH (EtherYatri.NET Windows Agent Host) were run one on port 8000 and the other on port 8001 both using the HTTP protocol and on the same machine. Each instance of the agent host can be used to create agents, host them and send them to other agent hosts. The two instances of the agent hosts listen on URLs <http://sankarsdell:8000/AgentHost>, <http://sankarsdell:8001/AgentHost> respectively where sankarsdell was the machine's name the hosts were running on. Using the NewAgent function, a new instance of the HelloWorldDemo agent was created at port 8000. When the agent was successfully created it was shown in a grid in the Active Agents tab. The using the SendAgent function, that agent was sent to the agent host running on port 8001. When the agent was successfully sent to the destination, it was removed from the source agent host's Active Agents tab and a message appeared on the destination agent host's output window to show that the agent had arrived.

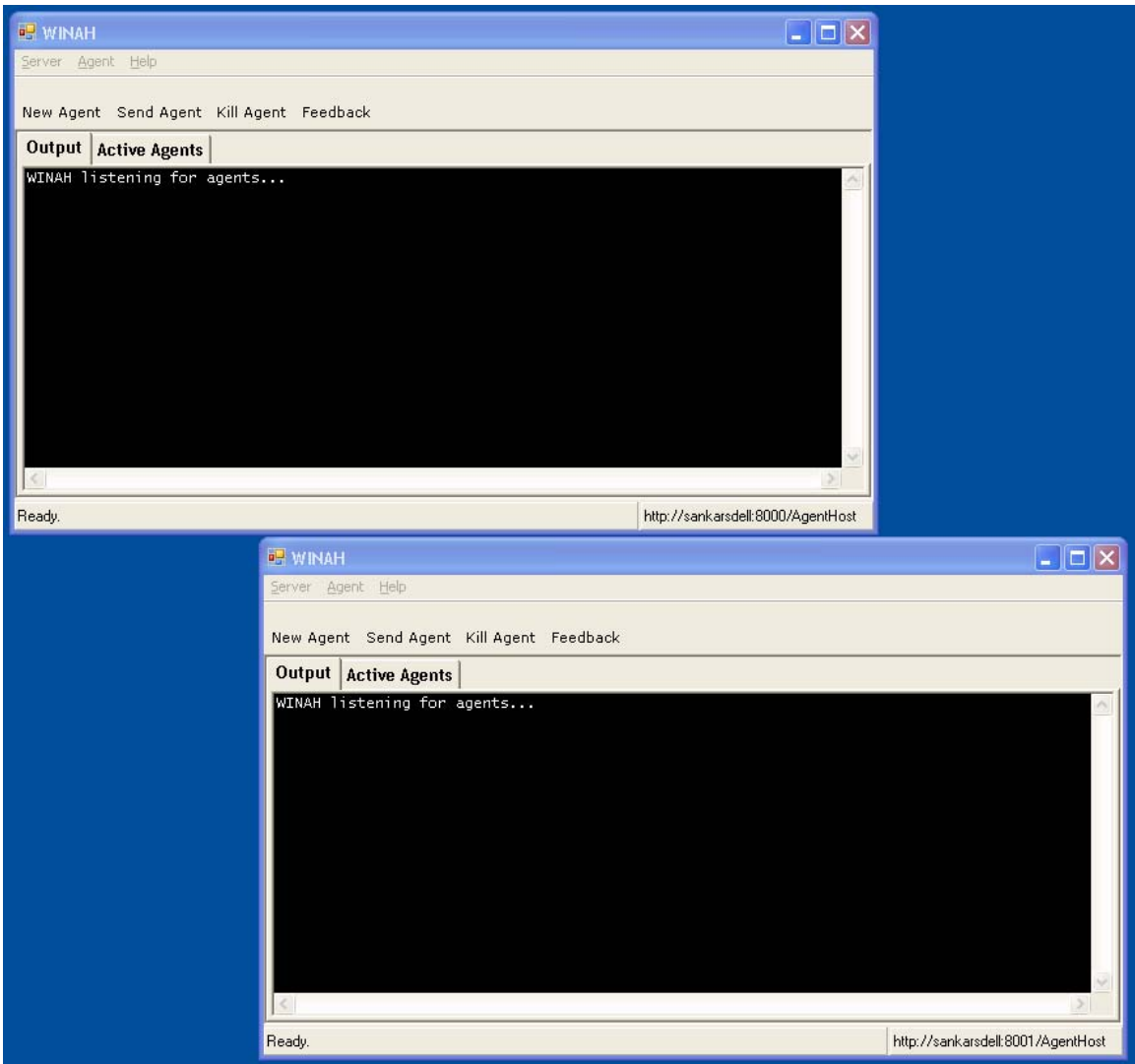


Figure 4.6 Two instances of WinAH running on ports 8000 and 8001 respectively

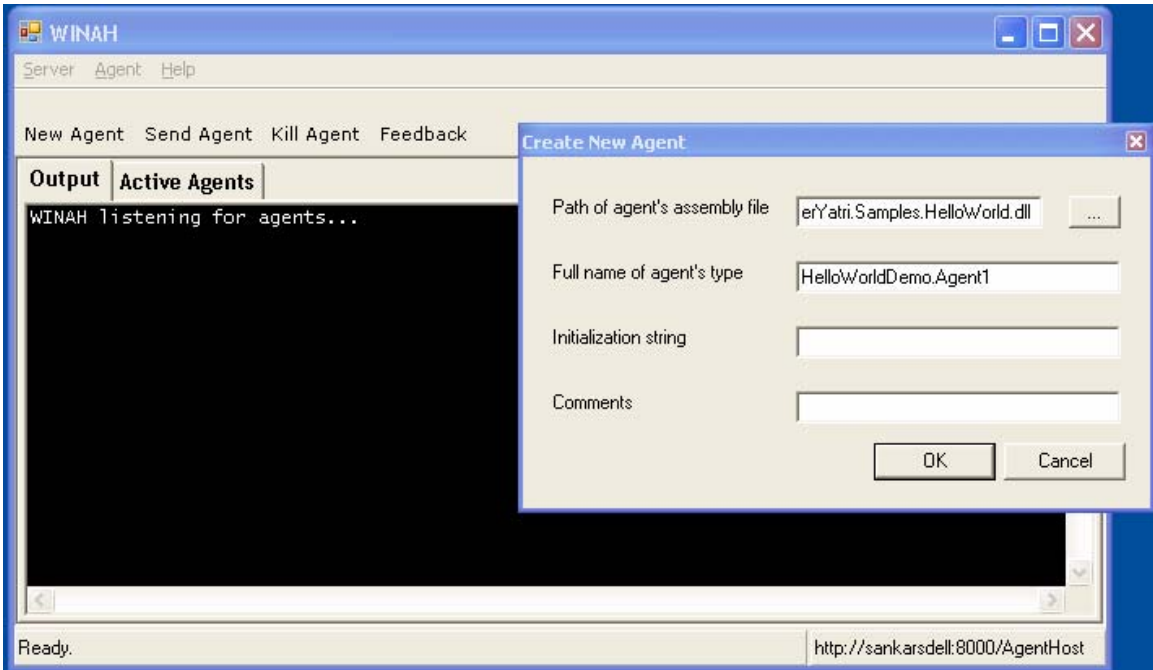


Figure 4.7 Creating the HelloWorldDemo agent at port 8000

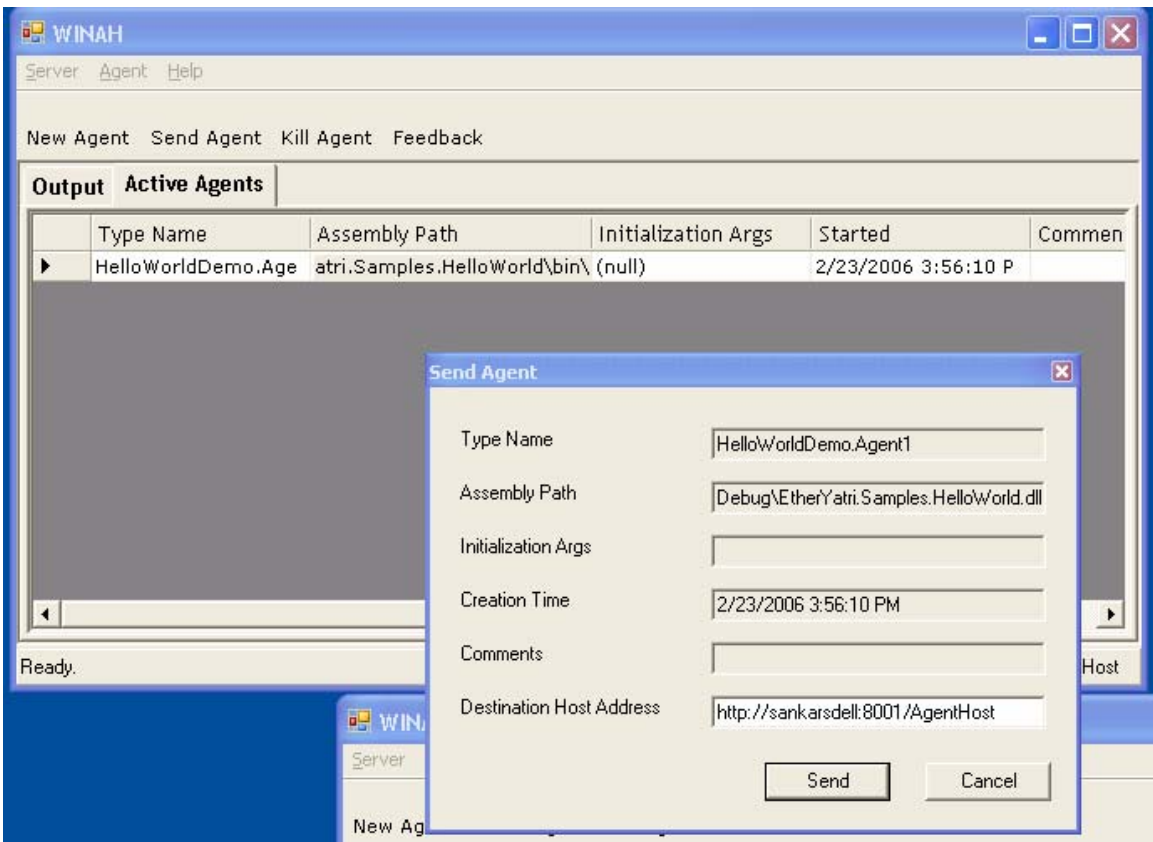


Figure 4.8 Sending the agent to another host using the Send Agent dialog

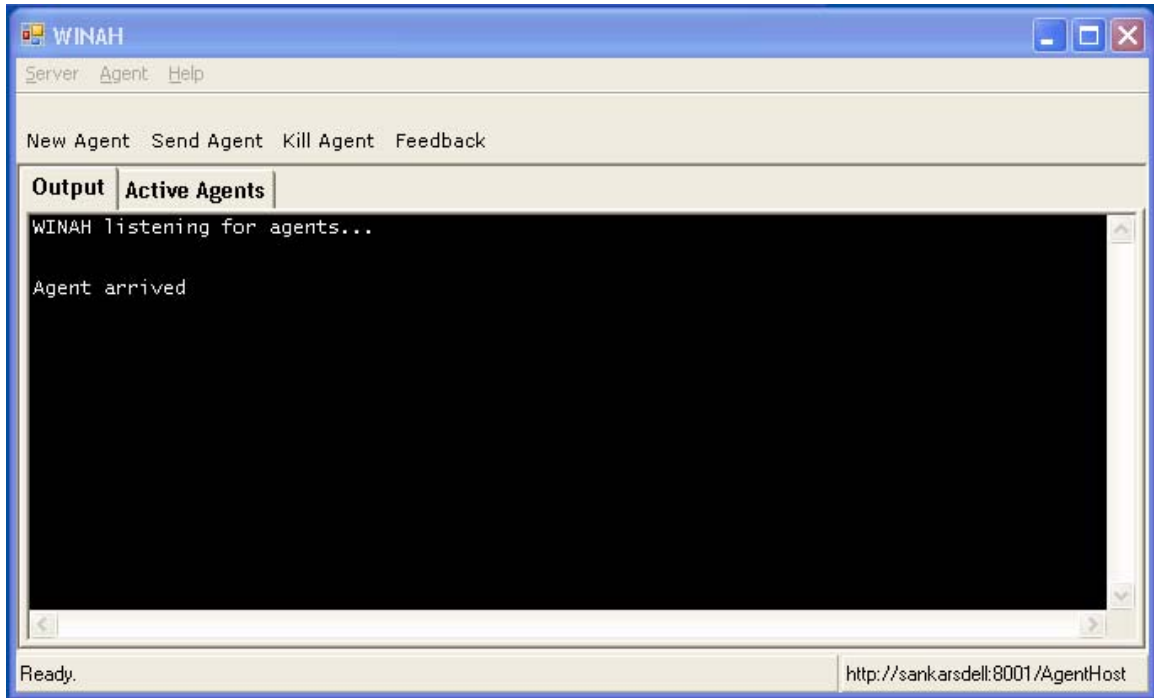


Figure 4.9 Message showing that agent has arrived at the destination

Understanding how EtherYatri.NET works

EtherYatri.NET uses .NET Remoting for communication between agent hosts. Agents are created using dynamic invocation features such as .NET Reflection. Once agents are created, they are given unique identifiers and their references are held by the agent host. To move an agent from one agent host to another, EtherYatri.NET simply invokes the Execute method on the remote agent host object and passes the serialized agent object and the bytes from the assembly that defines the agent class to the method. For an agent host to invoke an Execute method on a remote agent host, the remote agent host registers itself as a remotable object with the Remoting infrastructure. Remoting channels are created and registered to facilitate the communication. Once agents are sent to the remote hosts, the sending agent or agent host can invoke a method on the remote agent through the receiving agent host. This is done by using another remote method

called `InvokeAgentMethodOnRemoteHost`. The unique identifier of the agent, method name and the serialized parameters are passed to this method and the remote host uses .NET Reflection to find and invoke the method. There are several areas where EtherYatri.NET could be improved to take advantage of the .NET framework features but this thesis only addresses the IPv6 aspect of the potential improvements.

4.2 Enabling IPv6 in EtherYatri.NET

First EtherYatri.NET was tested on IPv6 to see if the framework already worked with IPv6. The `HelloWorldDemo` agent was used for this experiment as well. Two agent hosts were run on ports 8000 and 8001 listening on HTTP. Once the agent was created, in the send agent dialog, an IPv6 address was used instead of machine name or IPv4 address. When the send button was clicked, EtherYatri.NET displayed an exception message indicating that the send operation failed. The error message displayed indicated that the URL was being parsed as an IPv4 URL and failing as the IPv6 address contained colon (:) characters that were used to separate the address from the port number in a URL.

```

System.UriFormatException occurred
  Message="Invalid URI: A port was expected because of there is a colon (':') present but the port could not
be parsed."
  Source="mscorlib"
  StackTrace:
    Server stack trace:
      at System.Uri.CreateThis(String uri, Boolean dontEscape, UriKind uriKind)
      at System.Net.WebRequest.Create(String requestUriString)
      at System.Runtime.Remoting.Channels.Http.HttpClientTransportSink.SetupWebRequest(IMessage
msg, ITransportHeaders headers)
      at System.Runtime.Remoting.Channels.Http.HttpClientTransportSink.ProcessAndSend(IMessage msg,
ITransportHeaders headers, Stream inputStream)
      at System.Runtime.Remoting.Channels.Http.HttpClientTransportSink.ProcessMessage(IMessage msg,
ITransportHeaders requestHeaders, Stream requestStream, ITransportHeaders& responseHeaders, Stream&
responseStream)
      at System.Runtime.Remoting.Channels.SoapClientFormatterSink.SyncProcessMessage(IMessage
msg)
    Exception rethrown at [0]:
      at System.Runtime.Remoting.Proxies.RealProxy.HandleReturnMessage(IMessage reqMsg, IMessage
retMsg)
      at System.Runtime.Remoting.Proxies.RealProxy.PrivateInvoke(MessageData& msgData, Int32 type)
      at EtherYatri.IAgentHost.Execute(Byte[] agentAssemblyBytes, Byte[] agentState, String
agentTypeName, AgentId agentId)
      at EtherYatri.AgentHost.MoveAgent(String destinationAddress, AgentId agentId)

```

Figure 4.10 Exception thrown when IPv6 address was used in the destination URL

RFC 2732 addresses the cause for the above exception. According to the RFC, when literal IPv6 addresses are used as part of URLs, the IP address needs to be specified within square-brackets. Ex: `http://[::]:8000/AgentHost` [69]. When the above test was run using the square-brackets in the URL, the `UriFormatException` did not occur.

Attempting to send an agent with the correct URL format resulted in a `WebException`., indicating that the source agent host was not able to connect to the destination agent host. This was caused due to the fact that `EtherYatri.NET` was not listening on the IPv6 address used in the URL.

```

System.Net.WebException occurred
  Message="Unable to connect to the remote server"
  Source="mscorlib"
  StackTrace:
    Server stack trace:
      at System.Net.HttpWebRequest.GetRequestStream()
      at System.Runtime.Remoting.Channels.Http.HttpClientTransportSink.ProcessAndSend(IMessage msg,
ITransportHeaders headers, Stream inputStream)
      at System.Runtime.Remoting.Channels.Http.HttpClientTransportSink.ProcessMessage(IMessage msg,
ITransportHeaders requestHeaders, Stream requestStream, ITransportHeaders& responseHeaders, Stream&
responseStream)
      at System.Runtime.Remoting.Channels.SoapClientFormatterSink.SyncProcessMessage(IMessage
msg)
    Exception rethrown at [0]:
      at System.Runtime.Remoting.Proxies.RealProxy.HandleReturnMessage(IMessage reqMsg, IMessage
retMsg)
      at System.Runtime.Remoting.Proxies.RealProxy.PrivateInvoke(MessageData& msgData, Int32 type)
      at EtherYatri.IAgentHost.Execute(Byte[] agentAssemblyBytes, Byte[] agentState, String
agentTypeName, AgentId agentId)
      at EtherYatri.AgentHost.MoveAgent(String destinationAddress, AgentId agentId)

```

Figure 4.11 Exception thrown after a valid URL format was used for the destination URL

EtherYatri.NET used HTTP and TCP Channels of .NET Remoting infrastructure for the agent hosts to accept incoming requests. These channels when registered with the .NET Remoting, by default, were binding to all known IPv4 addresses on the host machine and not binding to any IPv6 addresses. Connections to URLs with the machine name, localhost, the loop-back IP address and the configured IPv4 addresses were successful. And connections to URLs with IPv6 address were not successful. To fix this, the code was modified to create HTTP and TCP channels with the “bindTo” property set to “::”. Setting that property makes the channels listen for incoming requests on all IPv6 addresses configured on the host.

```

Hashtable channelProps = new Hashtable();
channelProps.Add("port", AgentHost.portNumber);
channelProps.Add("bindTo", ":::");
Hashtable sinkProps = new Hashtable();

// Expose AgentHost using http or tcp as specified by the user
if (protocol.ToLower().CompareTo("http") == 0)
{
    AgentHost.channel = new System.Runtime.Remoting.Channels.Http.HttpChannel(channelProps,
        null, null);
}
else if (protocol.ToLower().CompareTo("tcp") == 0)
{
    AgentHost.channel = new System.Runtime.Remoting.Channels.Tcp.TcpChannel(channelProps,
        null, null);
}

```

Source code 4.4 Using the bindTo property of the channels to bind to IPv6 addresses

After setting the bindTo property of the channels, an agent was successfully sent from one agent host to another agent host using an IPv6 address in the URL.

```

// Expose AgentHost using http or tcp as specified by the user
Hashtable ipv6ChannelProps = new Hashtable();
ipv6ChannelProps.Add("port", AgentHost.portNumber);
ipv6ChannelProps.Add("name", "ipv6channel");
ipv6ChannelProps.Add("bindTo", ":::");

Hashtable ipv4ChannelProps = new Hashtable();
ipv4ChannelProps.Add("port", AgentHost.portNumber);

// Expose AgentHost using http or tcp as specified by the user
if (protocol.ToLower().CompareTo("http") == 0)
{
    AgentHost.channels.Add(new
System.Runtime.Remoting.Channels.Http.HttpChannel(ipv6ChannelProps, null, null));
    AgentHost.channels.Add(new
System.Runtime.Remoting.Channels.Http.HttpChannel(ipv4ChannelProps, null, null));
}
else if (protocol.ToLower().CompareTo("tcp") == 0)
{
    AgentHost.channels.Add(new
System.Runtime.Remoting.Channels.Tcp.TcpChannel(ipv6ChannelProps, null, null));
    AgentHost.channels.Add(new
System.Runtime.Remoting.Channels.Tcp.TcpChannel(ipv4ChannelProps, null, null));
}

```

Source code 4.5 Using different channels for IPv4 and IPv6 addresses

Setting the bindTo property of the channels to bind to IPv6 addresses, made the agent hosts listen only on the IPv6 addresses and connections to IPv4 addresses stopped

working. To fix this, two channels with different bindTo properties were created and registered so that connections to IPv4 addresses were received by the IPv4 channel and connections to IPv6 addresses were received by the IPv6 channel. When the channels were created, by default they were assigned a name that was same as the protocol used (EX: HTTP, TCP etc.). Creating two different channels for the same protocol, one for IPv4 and one for IPv6 failed with an error saying that a channel with the name already existed. So the name property of the channel was used to explicitly assign a name for the channels to make it work. With these changes, EtherYatri.NET started communication on IPv6 as well as IPv4.

CHAPTER 5

CONCLUSIONS AND FUTURE RESEARCH POSSIBILITIES

5.1 Conclusions

IPv6 was enabled in EtherYatri.NET by taking advantage of the built-in support for IPv6 in the .NET socket libraries. Other changes have been made to the EtherYatri.NET framework. These changes include making the channels configurable and displaying the network information about the agents.

IPv6 is an emerging protocol that is being supported by many operating systems. Although the protocol may be subject to future changes, the specifications have gone through many updates and are now well defined. Many implementations of IPv6 exist today and more sophisticated and integrated implementations are still being introduced (EX: Microsoft has introduced a dual layer implementation in Windows Vista as opposed to a dual stack architecture in Windows XP). Domain Name Service for IPv6 is still not available in many implementations.

The process of enabling IPv6 protocol in network applications depends on the application code, the communication mechanism it uses and the built-in support the operating system and the network libraries have for IPv6. Yet, there usually are some steps in the process that are applicable for many network applications. Following are some of the steps that may apply to other network applications as well:

1. Identify and investigate the APIs used by the application for network communications to make sure that the APIs support IPv6. Most of the network APIs today have built-in support for IPv6. If the APIs used by the application do not support IPv6, one may need to change the application so that it uses a network API that supports IPv6. It is a good design practice to abstract the communications layer so that future enhancements to communication mechanisms can be easily incorporated. EtherYatri.NET uses .NET Remoting as the communication mechanism. .NET Remoting uses .NET sockets that have built-in support for IPv6.

2. Study the APIs to understand how to use IPv6 in the application. This usually involves writing simple network client and server to communicate via IPv6. A chat client and server were written in C# to understand how IPv6 could be used in .NET sockets.

3. Identify the end points in the application code where the underlying communication mechanism is directly used. A socket bind, a .NET Remoting channel registration are examples of direct calls into the communication infrastructure from the application. Once these end points are identified, changes may need to be made to the end points' code to enable IPv6 communication. Abstracting direct calls to the communications APIs by means of application code helps in this step. EtherYatri.NET has two different places where it registers .NET Remoting channels using the .NET API for Remoting directly. Changes had to be made in both the places to register an IPv6 channel as well.

4. Enumerate the several of forms of input that the application has to accept host identifiers (URIs, IP addresses, host names etc.) to be used while initiating the end points. These may include user interfaces, persistent storage (configuration files, databases,

registry etc.), hard coded values in the code and input received during network communications. Application code sometimes alters the input before it uses the input to initiate an end point. For each of the enumerated forms of input, code needs to be examined to make sure any alterations to the input do not create a problem in establishing IPv6 based communication.

5. Applications sometimes verify the input they receive. An IPv4 based network application may validate IP addresses input that the application receives to be in the IPv4 dotted notation (a.b.c.d) and may stop the application code from establishing network communication. For each of the various forms of input, the verification code may need to be updated to accommodate IPv6 address formats as well.

6. Test the application on IPv6. Setting up an IPv6 environment using virtualization software seems to be convenient for experiments. This may include examining the established connections to make sure they are using IPv6. This is essential especially for the operating systems that do not support IPv6 only networking. In a mixed environment with the current support in the communications APIs, it is possible that some settings are required to ensure IPv6 communication.

Automated tools are not currently available to enable IPv6 in network applications. The number of APIs available for network communications, the number of ways in which network applications can make use of these APIs make it hard for any one tool to provide such capability. Network APIs that have built-in support for IPv6 minimize the changes that are needed in the application code. A well defined software design will definitely help in enabling IPv6 or any other communication enhancements.

5.2 Potential future research

With new IPv6 implementations, more sophisticated distributed frameworks such as Windows Communication Foundation, Internet 2 and mobile roaming high-speed internet, there are more opportunities to enhance and extend the concept of moving code and also study & augment the capabilities of IPv6. Following are some of the potential research projects that have been identified while working on this thesis:

- A mobile agent framework based on Windows Communication Foundation that could potentially include features such as:

- Support for scripting based agents (moving scripts)
- Communications among hosts behind firewalls
- Support for pluggable communication protocols
- Tools and libraries that make it easy to create agents
- Support for transactions (distributed and stand-alone)
- Take advantage of new UI technologies such as Asynchronous Javascript And XML (AJAX) and Windows Presentation Foundation.

- Study DNS support in various IPv6 implementations

- Make necessary changes to EtherYatri.NET to work with IPv6 DNS

- Research application of mobile code in areas such as multi-player network gaming and other forms of digital communication and entertainment.

- Research the impact of IPv6 on private IPs.

- Study the need for a web based network translation service that could help hosts behind firewalls listen on public ports. Several file and song sharing applications currently provide identity for hosts outside their private network. For example if a host

inside a firewall or a private local LAN with a private IP address wants to listen for mobile agents, currently the router or firewall has to open the port and forward requests to the host. IPv6 address space could solve the problem or a service like the one being proposed could help hosts get a virtual IPv6 address that can forward the requests to the host.

These and many other projects could help researchers, companies and other interested parties find new applications for both mobile code and IPv6 protocol.

BIBLIOGRAPHY

- [1] Douglas E. Comer: *Computer Networks and Internets*
- [2] Charles M. Kozierok: *International Networking Standards Organizations*, The TCP/IP Guide,
http://www.tcpipguide.com/free/t_InternationalNetworkingStandardsOrganizations.htm.
- [3] Charles M. Kozierok: *History of the OSI Reference Model*, The TCP/IP Guide,
http://www.tcpipguide.com/free/t_HistoryoftheOSIReferenceModel.htm.
- [4] Wikipedia: *Description of Layers*, OSI Model,
http://en.wikipedia.org/wiki/OSI_model.
- [5] Andrew S. Tanenbaum: *A Critique of the OSI Model and Protocols*, Computer Networks, Section 1.4.4
- [6] Whatis.com: *Development of TCP/IP*, Understanding TCP/IP,
http://whatis.techtarget.com/definition/0,,sid9_gci989915,00.html.
- [7] Whatis.com: *Host*, http://whatis.techtarget.com/definition/0,,sid9_gci212254,00.html.
- [8] Howstuffworks.com: *What is an IP address?*
<http://computer.howstuffworks.com/question549.htm>.
- [9] Internet Assigned Numbers Authority: *Internet Protocol V4 Address Space*,
<http://www.iana.org/assignments/ipv4-address-space>.
- [10] Pacific Bell Internet: *Classless Inter-Domain Routing (CIDR) Overview*, CIDR Info
<http://public.pacbell.net/dedicated/cidr.html>.

- [11] Internet Assigner Numbers Authority: *Protocol Numbers*,
<http://www.iana.org/assignments/protocol-numbers>.
- [12] Microsoft Corporation: *Introduction to IP Version 6*, Microsoft Windows Server 2003 White Paper, published September 2003, updated March 2004
<http://download.microsoft.com/download/5/2/5/525343cc-7ba4-4e3b-a96a-c7a040d98d2d/IPv6.doc>.
- [13] Hinden, R., Deering, S.: *RFC 3513 - Internet Protocol Version 6 (IPv6) Addressing Architecture*, Network Working Group <http://www.faqs.org/rfcs/rfc3513.html>.
- [14] Hinden, R., O'Dell, M., Deering, S.: *RFC 2374 - An IPv6 Aggregatable Global Unicast Address Format*, Network Working Group
<http://www.faqs.org/rfcs/rfc2374.html>.
- [15] Huitema, C., Carpenter, B.: *RFC 3879 - Deprecating Site Local Addresses*, Network Working Group <http://www.faqs.org/rfcs/rfc3879.html>.
- [16] Hinden, R., Deering, S.: *IP Version 6 Addressing Architecture*,
<http://www.ietf.org/internet-drafts/draft-ietf-ipv6-addr-arch-v4-04.txt>.
- [17] King, J. A: *Intelligent Agents: Bringing Good Things to Life*, AI Expert, February, 1995, pp. 17-19.
- [18] Atkinson, B., et al: *IBM Intelligent Agents*, presented at Unicom Seminar on Agent Software, London, UK, May 25, 1995.
- [19] Todd Sundsted: *Agents on the move*, Java World, July 1998
<http://www.javaworld.com/javaworld/jw-07-1998/jw-07-howto.html>.
- [20] Danny B. Lange, Mitsuru Oshima: *Programming and Deploying Java Mobile Agents with Aglets*

- [21] Siddharth Uppal, *EtherYatri.NET*
<http://www.geocities.com/siddharthuppal/EtherYatri.htm>.
- [22] *EtherYatri workspace*,
<http://www.gotdotnet.com/Workspaces/Workspace.aspx?id=4d8ceb9e-b334-4d96-8b95-8d3480cabc71>.
- [23] *Narval*, <http://www.logilab.org/projects/narval>.
- [24] *CLIPS: A Tool for Building Expert Systems*, <http://www.ghg.net/clips/CLIPS.html>.
- [25] Charles L. Forgy: *Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem*, *Artificial Intelligence* 19(1982), 17-37.
- [26] *Jess, The Rule Engine for The Java Platform*, <http://herzberg.ca.sandia.gov/jess/>.
- [27] *The Lisa Project*, <http://lisa.sourceforge.net/>.
- [28] *Personalized Agents from Autonomy*,
<http://www.autonomy.com/content/Products/IDOL/f/Agents.html>.
- [29] *UMBC KQML Web*, <http://www.cs.umbc.edu/kqml/>.
- [30] John Davies, Richard Weeks, and Mike Revett: *Jasper: Communicating Information Agents for WWW*, <http://www.w3.org/Conferences/WWW4/Papers/180/>.
- [31] *Java Agent Template*, <http://www-cdr.stanford.edu/ABE/JavaAgent.html>.
- [32] *The OMG's CORBA Website*, <http://www.corba.org/>.
- [33] *Java Remote Method Invocation - Distributed Computing for Java*,
<http://java.sun.com/marketing/collateral/javarmi.html>.
- [34] *Microsoft .NET Remoting: A Technical Overview*,
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp>.

- [35] *Simple Object Access Protocol (SOAP)*, <http://www.w3.org/TR/SOAP/>.
- [36] *Web Services Description Language (WSDL)*, <http://www.w3.org/TR/wsdl>.
- [37] *AgentSpace, a Next-Generation Mobile Agent System*,
<http://berlin.inesc.pt/agentspace/>.
- [38] *Voyager* <http://www.recursionsw.com/products/voyager/orbpro.asp>.
- [39] Alberto Silva, Jose Delgado: *AgentSpace as a Framework to Support Interlibrary Cooperation*, <http://berlin.inesc-id.pt/alb/static/papers/1998/crimea98.pdf>.
- [40] *About Aglets*, http://www.trl.ibm.com/aglets/about_e.htm.
- [41] *Ajanta Programmers' Guide*, <http://www.cs.umn.edu/Ajanta/papers/Guide/toc.html>.
- [42] Anand Tripathi, Tanvir Ahmed, Sumedh Pathak, Abhijit Pathak, Megan Carney, Murlidhar Koka, and Paul Dokas: *Active Monitoring of Network Systems using Mobile Agents*, To appear in proceedings of Networks 2002, a joint conference of ICWLHN 2002 and ICN 2002.
- [43] *The ARA Platform for Mobile Agents*, http://www.wagss.informatik.uni-kl.de/Projekte/Ara/index_e.html.
- [44] Mitsubishi Electric ITA, Horizon Systems Laboratory, 1432 Main Street, Waltham, MA 02154, USA: *Concordia: An Infrastructure for Collaborating Mobile Agents*,
<http://www.cis.upenn.edu/~bcpierce/courses/629/papers/Concordia-MobileAgentConf.html>.
- [45] *D'Agents: Mobile Agents at Dartmouth College*, <http://agent.cs.dartmouth.edu/>.
- [46] Ophir Holder, Israel Ben-Shaul, Hovav Gazit: *Dynamic layout of distributed applications in FarGo*, Proceedings of the 21st International Conference on Software Engineering, Los Angeles, California, United States, 1999, pp. 163-173.

- [47] *Tryllian's Agent Development Kit*,
<http://www.tryllian.com/technology/product1.html>.
- [48] *Extensible Markup Language (XML)*, <http://www.w3.org/XML/>.
- [49] *Project JXTA*, <http://www.sun.com/software/jxta/>.
- [50] *Foundation of Intelligent Physical Agents*, <http://www.fipa.org/>.
- [51] M. Jazayeri and W. Lugmayr: *Gypsy: A Component-based Mobile Agent System*,
Accepted at the 8th Euromicro Workshop on Parallel and Distributed Processing
(PDP2000) (Rhodos, Greece, January 19 -21, 2000).
- [52] *Java Beans*, <http://java.sun.com/products/javabeans/>.
- [53] Dag Johansen, Robbert van Renesse, and Fred B. Schneider: *An Introduction to the TACOMA Distributed System Version 1.0*, Technical Report 95-23, Department of
Computer Science, University of Tromsø, Norway, June 1995.
- [54] Dilyana Staneva, Denitsa Dobрева, The Technical University of Varna, Bulgaria:
MAPNET: A .NET Based Mobile Agent Platform. Presented at the International
Conference on Computer Systems and Technologies - CompSysTech'2004.
<http://ecet.ecs.ru.acad.bg/cst04/Docs/sII/213.pdf>,
<http://ecet.ecs.ru.acad.bg/cst04/Docs/sI/15.pdf>.
- [55] *MASIF Specification*. <http://www.omg.org>.
- [56] *JAFMAS, Java Based Framework for Multi-Agent Systems*,
<http://www.ececs.uc.edu/~abaker/JAFMAS/>.
- [57] *MARS, Mobile Agent Reactive Space*, <http://polaris.ing.unimo.it/MOON/MARS/>.

- [58] David Wallace Croft, Senior Intelligent Systems Engineer, Special Projects Division, Information Technology, Analytic Services, Inc.: *Intelligent Software Agents: Definitions and Applications*,
<http://www.alumni.caltech.edu/~croft/research/agent/definition/>.
- [59] *Software Agent Fundamentals* from BTextact Technologies,
<http://more.btexact.com/projects/agents.htm>.
- [60] Dixon, T.: *RFC 1454 - Comparison of Proposals for Next Version of IP*, Network Working Group <http://www.faqs.org/rfcs/rfc1454.html.old>.
- [61] Bradner, S., Mankin, A.: *RFC 1550 - IP: Next Generation (IPng) White Paper Solicitation*, Network Working Group <http://www.faqs.org/rfcs/rfc1550.html>.
- [62] Deering, S., Hinden, R.: *RFC 1883 - Internet Protocol, Version 6 (IPv6) Specification*, Network Working Group <http://www.faqs.org/rfcs/rfc1883.html>.
- [63] Hinden, R., Deering, S.: *RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification*, Network Working Group <http://www.faqs.org/rfcs/rfc2460.html>.
- [64] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., Zill, B.: *RFC 4007 - IPv6 Scoped Address Architecture*, Network Working Group
<http://www.faqs.org/rfcs/rfc4007.html>.
- [65] Microsoft's Objectives for IP Version 6,
<http://www.microsoft.com/windowsserver2003/techinfo/overview/ipv6.mspx>.
- [66] IPv6 Implementations, <http://www.ipv6.org/impl/index.html>.
- [67] IPv6 Enabled Applications, <http://www.ipv6.org/v6-apps.html>.
- [68] Momin, M.: *An implementation of unicast discovery protocol using IPV6 and JINI*, Thesis (M.S.) Advisor: Dr. Carlisle, H.--Auburn University, 2004.

[69] Hinden, R., Carpenter, B., Masinter, L.: *RFC 2732 - Format for Literal IPv6 Addresses in URL's*, Network Working Group <http://www.faqs.org/rfcs/rfc2732.html>.