IMPROVING THE PERFORMANCE OF THE GNUTELLA NETWORK

by

André Dufour

B.A.Sc. University of Ottawa, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in the School
of
Engineering Science

SIMON FRASER UNIVERSITY

Summer 2006

APPROVAL

Name:	André Dufour
Degree:	Master of Applied Science
Title of thesis :	Improving the Performance of the Gnutella Network

Examining Committee: Dr. Rodney Vaughan, Chairman

____
Dr. Ljiljana Trajković
Professor, Engineering Science, SFU
Senior Supervisor

____
Dr. Joseph Peters
Professor, Computing Science, SFU
Supervisor

____
Dr. Mohamed Hefeeda
Assistant Professor, Computing Science, SFU
Examiner

Date Approved: ______________________________

Abstract

In this thesis, the behaviour of the Gnutella peer-to-peer (P2P) file sharing network is examined and a proposal is put forth to improve its performance. Gnutella's overlay topology is not well matched to the underlying physical network and the network therefore exhibits sub-optimal performance in terms of message latency. In order to evaluate this performance, we modified an existing Gnutella simulation framework developed for the ns-2 simulator to gather information about query and query hit propagation. The protocol implemented in the simulation was then modified to use the Vivaldi synthetic coordinate system in order to bias neighbour selection to favour nodes that are "close" in the Euclidean sense. Simulations showed that the modified Gnutella protocol yielded an improvement in both query and query hit propagation times.

Keywords

Computer networks – computer simulation, computer network architectures, peer-to-peer architecture (computer networks)

Acknowledgements

I would like to thank my senior supervisor, Dr. Ljiljana Trajković, and the members of the Communication Networks Laboratory for their support throughout this degree. I am also thankful for the help of Dr. Joseph Peters and Dr. Mohamed Hefeeda as members of my committee. I would like to extend a special thanks to Dr. Jason Carey of the University of Alberta for his thorough review and insightful comments on my thesis. Sincere thanks are also due to the organizations that have funded me throughout my studies: the Natural Science and Engineering Research Council (NSERC), the Advanced Systems Institute of British Columbia, Simon Fraser University, Agilent Technologies, and Nova-Tech Engineering Inc. Finally, I am eternally grateful to my parents for their support and encouragement throughout all my studies.

Contents

List of Tables

List of Figures

List of Symbols

c_c : a Vivaldi tuning parameter controlling the magnitude of the response to each new sample

c_e : a Vivaldi tuning parameter controlling the weighting of new samples in error calculations

δ : Vivaldi timestep

d_j : Degree of node j

e^2 : Squared error on a Vivaldi sample

E^2 : Squared error for the entire Vivaldi system

e_i : Local Vivaldi error

e_s : Relative Vivaldi sample error

F_{ij} : Force node j exerts on node i

R : Sampled round trip time

w : Vivaldi sample weight

List of Abbreviations

AS: Autonomous system

BGP: Border gateway protocol

DNS: Domain name system

GUID: Globally unique identifier

HTTP: Hypertext transfer protocol

ICMP: Internet control message protocol

IP: Internet protocol, version 4

LAN: Local area network

NCS: Network coordinate system

ns-2: Network simulator 2

NSA: Neighbour selection algorithm

P2P: Peer-to-peer

PDNS: Parallel distributed network simulator

QoE: Quality of experience

QRP: Query routing protocol (in Gnutella)

RTT: Round trip time

TCP: Transmission control protocol

TTL: Time to live (field in a Gnutella messages)

UMASS: University of Massachusetts (refers to a model for peer behaviour)

Chapter 1

Introduction

With the advent of peer-to-peer (P2P) networks, the landscape of data communications has been radically altered. Since the popularization of the technology through the Napster file sharing network [1], P2P has grown to be the leading source of traffic in the Internet [2]. Although file sharing is still the most popular application, P2P technology has found niches in distributed processing [3], [4], online chatting [5], [6], and gaming [7], [8], for example. Furthermore, because P2P networks rely on a distributed overlay network, they exhibit a high degree of tolerance to random node failure: an alternate path or location can usually be found for the desired resource.

Gnutella is one of the most popular P2P file sharing protocols. It is an open standard protocol implemented by many vendors, such as Limewire and Bearshare. In addition to sharing content, thousands of nodes on the Gnutella network collaborate to forward control messages, such as queries, through the Gnutella overlay topology. This topology is formed as nodes learn the addresses of other nodes from a bootstrap server and from the nodes they are already connected to. This process does not take into consideration the underlying physical topology and, as such, can lead to inefficient network utilization [9]. Nodes are as likely to connect to distant nodes as to close ones, which results in longer message latency.

1.1 Objectives

In this thesis, a modification to the Gnutella protocol is proposed in order to more closely align the overlay topology with the underlying physical topology. The proposal uses the Vivaldi coordinate system [10] to assign synthetic coordinates to each participating node in the Gnutella network. The Euclidean distance in this coordinate system may be used to predict the round trip time between two nodes. With these modifications, every time a node sends a message, it includes its Vivaldi coordinates in addition to the message payload. Thus, the receiving node, knowing its own coordinates, can estimate the round trip time to the sending node. Nodes therefore have the means to decide whether to accept connection requests based on node proximity.

In order to evaluate the performance of the proposed protocol modification, we extended an existing network simulator called GnutellaSim [11] to create a new simulator: *Gnutaldi*. This simulator models the Gnutella version 0.6 network. We rewrote significant portions of the code in order to implement statistics gathering logic and enhance the performance of the simulator. We also redesigned the message generation and parsing classes in order to make the code faster and more maintainable. The redesigned message classes included the proposed implementation of Vivaldi coordinates. The augmented protocol agent classes included routines for maintaining the coordinates at each node and inserting them into messages.

The BRITE topology generator [12] was used to create a physical network topology and evaluate the speed with which queries were satisfied. For each scenario, the performance of the network with and without the proposed enhancements is compared.

1.2 Organization of the Thesis

This thesis is based on peer-reviewed research accepted for publication [13]. Chapter 2 presents background information on themes relevant to this research. In Chapter 3, details are provided on the operation of the Vivaldi coordinate system. The following chapter introduces the proposed modifications to the Gnutella protocol are introduced. In Chapter 5 the architecture of the simulator is outlined. Next, in Chapter 6 the topic of synthetic network topology generation is discussed. Simulation results appear in Chapter 7. Finally, the conclusions of this work appear in Chapter 8.

Chapter 2

Background

This chapter provides background information on themes relevant to this thesis. The first subsection discusses the fundamentals of P2P communication. Subsection 2.2 provides details about the Gnutella P2P network. An introduction to network simulation tools and the *ns-2* simulator [14] is given in Subsection 2.3. Finally, Subsection 2.4 presents the GnutellaSim simulation package, which was used in this research.

2.1 P2P Networks

In the last several years, P2P (P2P) networks have emerged as a new model for computer communication. Radically departing from the traditional client-server paradigm, P2P gives each network participant significant autonomy and an enhanced role in the fundamental operation of the network. P2P has achieved remarkable popularity in the short time it has been in use and has found applications in file sharing, distributed data processing, and online gaming, for example. P2P networks are also related to ad-hoc and sensor networks because they employ a decentralized, distributed mode of communication. P2P communication has been a strong disruptive force in networking and, as such, is an important research topic.

This subsection presents important P2P properties and applications.

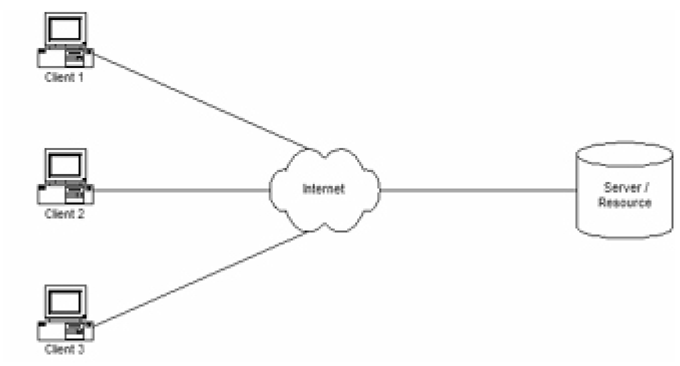


Figure 2.1: Client-Server networking paradigm.

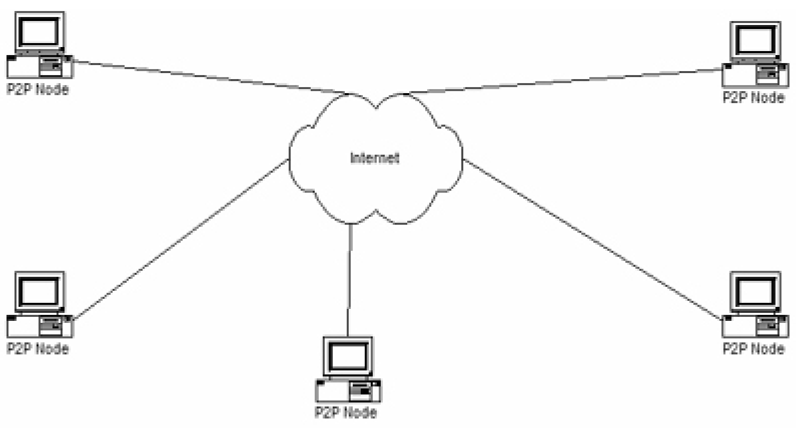


Figure 2.2: P2P networking paradigm.

2.1.1 P2P Network Properties

A fundamental premise of P2P networks is to allow nodes at the edge of the network to collaborate together in a decentralized fashion. P2P networking causes systems to peer together and form a network where there is no concept of a client or server: nodes both provide and consume resources. P2P nodes also act as routers in the P2P topology, forwarding traffic destined for other peers through the network.

A traditional client-server configuration is shown in Fig. 2.1. With this model, the server holds the content or resource of interest to the clients. If the server fails or if its communication link ceases to function, all clients are deprived of the server's services.

P2P networks, conversely, do not have a central point of failure. Resources and services are distributed amongst the peers participating in the network. The flat

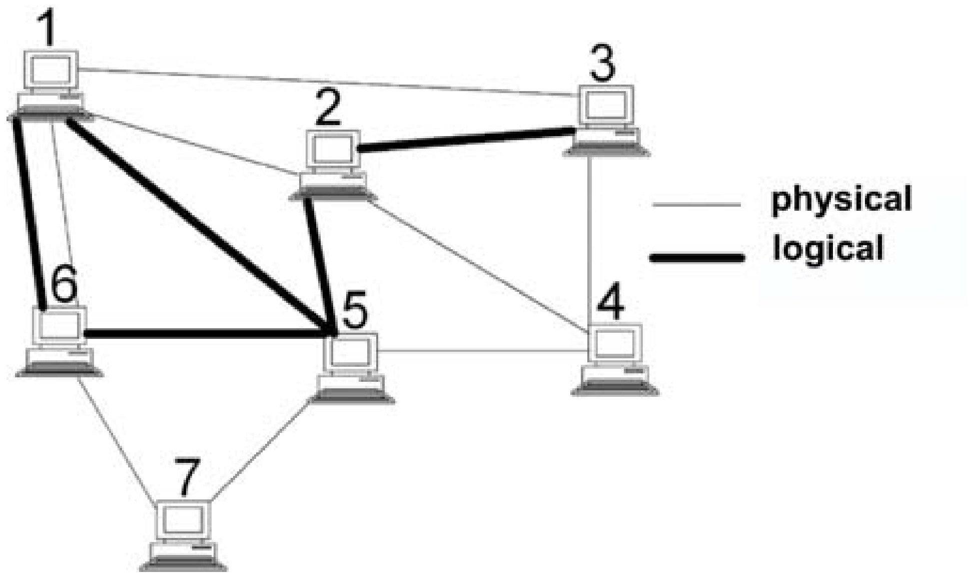


Figure 2.3: Physical and logical topologies in P2P communication.

hierarchy through which P2P nodes relate is shown in Fig. 2.2. This hierarchy is in sharp contrast to the client-server paradigm represented in Fig. 2.1.

P2P networks are formed as an application-layer overlay, superimposed on the existing physical infrastructure of routers and links. This overlay, sometimes called the *logical topology*, is the conduit for all message exchanged between participants in the network. Peers associate by forming *neighbour* relationships. Each peer is only aware of its neighbours and only communicates directly with them. The application-layer messages between neighbours are routed through the physical topology by network-layer devices (routers). An example of relationship between physical and logical topologies is shown in Fig. 2.3. In some cases, logical links follow physical links, as in the connection between nodes 1 and 6. In other cases, nodes that are neighbours in the application layer overlay, such as 1 and 5, are separated by numerous hops in the physical topology.

A variety of devices may join a P2P network. The most common example is home computers connected to the Internet. Internet-enabled mobile phones may also participate in P2P exchanges. Because these devices at the edge of the network are not always in use, they are not always connected to the network. Transient node presence and the associated network variability are important characteristics of P2P communication. This property further sets P2P networks apart from the traditional client-server paradigm, where servers are reliable network entities that may always be contacted at the same address. While the steadfastness of the server approach may be an attractive feature, it leads to an architecture with a single point of failure. If a server malfunctions or its link to the network fails, the services provided by the server will be disrupted for all users. The distributed nature of P2P networks, conversely, guards against this type of failure scenario: there is no single point of failure. Content or services may be provided by multiple nodes at different locations in the network. The robustness of P2P networks has provided a haven for users interested in illicit activities such as the illegal distribution of copyrighted material. Since removing any single entity does not disable the network, it has proven very difficult for copyright owners to halt the undesired distribution of their works.

P2P file-sharing is of great concern to many network operators because of the large amount of traffic involved. Some Internet service providers have found that, at times, 90% of the traffic transmitted over their network is due to P2P applications [2]. This traffic is taxing network resources and making them less available for services that generate revenue for service providers, such as long-distance voice traffic or virtual private networks (VPNs) provisioned for customers. Furthermore, much of the P2P traffic is neither originating nor terminating in the particular provider's network: it is only being routed through a P2P node residing in that network on its way to its final destination. While policies may prevent this type of behavior for layer-3 routing by not advertising routes through the provider's autonomous system (AS) to other ASs via Border Gateway Protocol — BGP, routers are not aware of application-layer routing decisions made in the P2P network and, therefore, cannot intervene. Measurements in the Gnutella P2P network have shown that less than 5% of Gnutella connections link nodes that are in the same AS [9]. Thus, P2P traffic often crosses AS boundaries, which is more costly than intra-AS traffic from the service providers' point of view. Because of the providers' need to control P2P traffic, many network equipment manufacturers, such as Cisco/P-Cube [15] and Caspian [16], are developing devices that may identify P2P traffic flows and apply more stringent policies.

2.1.2 Applications of P2P Networks

While there any many applications of P2P technology, one of the most common is undoubtedly file-sharing. The popularity and notoriety of the Napster [1] file sharing application was largely responsible for making P2P a household word. Started in 1999, the Napster network aimed to help Internet users exchange digital music files [17]. It relied on a central server for processing queries, but the actual file exchange was done on a P2P basis, without the files ever passing through the server. Because of concerns surrounding illegal music trading on the Napster network, the Recording Industry Association of America filed a lawsuit against Napster, charging that the company had engaged in tributary copyright infringement [17]. Further to the lawsuit, Napster suspended operations for a time; nevertheless the era of P2P networks was launched.

There are many examples of P2P technology being used for file-sharing. The Gnutella network [18] is an open-standard file-sharing community used by applications such as Limewire [19] and Bearshare [20]. Unlike Napster, it does not rely on a central server for query processing. Queries are forwarded through a logical overlay network consisting of Gnutella peers. These peers act not only as clients and servers, but also as forwarding agents for Gnutella control traffic. The actual downloading of files is done by direct communication between the peers involved.

BitTorrent [21], another popular file-swapping program, may download segments of the desired content from multiple peers at the same time. It is also innovative in that BitTorrent peers penalize nodes that do not share sufficient content by reducing their download rate; so called "file leaches" are, therefore, less successful. To further encourage sharing, users' BitTorrent clients offer the received parts of partially downloaded files for download by other peers: peers do not have to have the entire file to share parts of it.

Chord [22] is a P2P file sharing network that addresses the issue of efficiently locating the node or nodes that store content by employing a distributed hashing algorithm. Chord may resolve lookups by sending only $O(\log N)$ messages. Each Chord node maintains a routing database for other nodes that grows logarithmically with the size of the network. Although Chord is not widely used, its efficient lookup mechanism is very promising.

The Freenet network [23] protects the anonymity of those sharing and downloading content using its clients. The stated intent of its designers is to allow users to publish and download content without fear of censorship [23]. Version 0.7 features a scalable “darknet”. Darknets are file sharing networks where nodes only connect to trusted nodes. Since human relationships (and consequently trust) create small-world networks [23], [24], Freenet may respect trust and still find a short path between two peers participating in the network [23].

Many terabytes of data are shared on these file-sharing networks and they account for an appreciable amount of the traffic on Internet, as discussed in Subsection 2.1.1.

In addition to file sharing, P2P has found applications in online gaming. Researchers have developed a P2P version of *Xiangqi* (Chinese chess) [7] [8], for example, which allow users to interact with other users at the edge of the Internet without employing a central server.

Online chatting systems such as MSN Messenger [5] and ICQ [6] are further applications of P2P technology. While they do rely on central servers, they link resources (in this case people) at the edge of the network, which is the essence of P2P. Similarly, in keeping with this definition of P2P, applications that use the aggregate processing power of computers throughout the Internet are examples of P2P technology. Folding@Home [3] is an innovative P2P solution that uses the distributed processing power of thousands of computers to analyze complex protein folding and aggregation problems. SETI@Home [4] is a similar P2P system that uses the processing cycles to analyze radio signals.

There are also generic P2P frameworks, not attached to any particular application. Microsoft Research’s Pastry [25] routing and location substrate forms a P2P overlay network and is the foundation for P2P applications such as the Scribe group communication system [26] and the SplitStream content distrubution system [27]. Sun Microsystem’s JXTA framework [28] is a set of open protocols that allow P2P communication between devices. It has been used for a range of applications, including chatting, gaming and file-sharing.

In short, P2P communication has found many applications in today’s networks. It is not yet pervasive, but it has certainly been and continues to be a strong disruptive force in computer communication.

2.2 The Gnutella Network

The project described in this thesis involves improving the performance of the Gnutella P2P file-sharing network [29]. Accordingly, it seems appropriate to discuss some of the key characteristics of Gnutella. In subsection 2.2.1 the syntax and semantics of the Gnutella protocol are discussed. In subsection 2.2.2 details about the topology of the Gnutella network are provided, including a fundamental flaw in the way nodes associate to form the overlay.

2.2.1 The Gnutella Protocol

Gnutella is a very popular distributed file sharing protocol. The number of nodes participating in the network was estimated at about 50,000 in 2001 [9], when P2P was still a nascent technolgoy. The current widely deployed version is 0.6 [29], which is a two-tiered hierarchy of peers, termed *servents*, collaborating to share files and forward protocol traffic through the network. Gnutella is an open standard and, as such, lends itself well to study and simulation in academic circles.

Unlike the Napster network [1], which employed a central server to mediate communication between peers, Gnutella is a distributed network. When first connecting to the network, new nodes (known as *servents*) contact a “bootstrap” server to obtain the addresses of a few connected peers. However, further communication is handled through the P2P overlay without relying on servers. Once the new node has the addresses of existing nodes, it attempts to connect to them by sending Gnutella *connect* messages [18]. Two connected nodes are called *neighbours*. Several connection attempts may be necessary because nodes may not be willing to accept new connections or may have left the network. Once at least one connection has been established, the new node begins sending periodic Gnutella *ping* messages. These probes — not to be mistaken for Internet Control Message (ICMP) *ping* messages — are used to search for other nodes willing to accept new connections. They are sent by the new node to all its neighbours, which, in turn, *flood* them to all their neighbours. This recursive flooding continues until the ping’s time-to-live (TTL) field, which is decremented at each hop, reaches zero. Along the way, any node receiving a ping and willing to accept new connections responds with a *pong* message. The pong back-propagates to the originator of the ping, which may decide to attempt a connection with the pong’s sender.

In order to locate content shared in the Gnutella network, nodes must send *query* messages. Queries contain the search criteria (e.g., a file name) and are flooded similarly to ping messages. When a node receives a query that matches a resource it shares, it responds with a *query hit* message, which is back-propagated to the query originator. The originator may then decide to download the file from the node that sent the query hit. This is done directly, through an HTTP-like protocol [18], and the traffic does not pass through the P2P overlay network. Incidentally, this download activity represents a large proportion of the traffic on Internet [2].

Gnutella messages are carried over a reliable TCP (transmission control protocol) transport [29]. The default TCP port is 6346, although servents may negotiate a different port. The initial connection messages are sent in clear-text form, in a format somewhat similar to HTTP. Figure 2.4[29] shows a sample connection transaction. *Servent A* is attempting to establish a Gnutella connection with *Servent B*. After opening a TCP connection, *A* sends a Gnutella CONNECT message, indicating its version (0.6), the Gnutella application (BearShare, version 1.0) and the protocol options it supports (pong-caching and GGEP). Servent *B* responds with an OK message and a list of its supported extensions. The final OK message from servent *A* concludes the three-way handshake and establishes the Gnutella connection. The *Private-Data* headers encapsulate vendor-specific information.

Table 2.1: Header structure of Gnutella binary messages.

Octets	Description
0-15	Message globally unique identifier.
16	Payload Type
17	TTL (Time To Live)
18	Hops
19-22	Payload Length

After the connection has been established servents exchange binary messages. The header structure for these messages is shown in Table 2.1 [29].

The *message globally unique identifier* (GUID) is used to avoid forwarding the same message twice: peers will drop messages with identifiers they have encountered before. The *payload type* field identifies the type of message. The following values are applicable [29]:

```
Servent A                      Servent B

GNUTELLA CONNECT/0.6
User-Agent: BearShare/1.0
Pong-Caching: 0.1
GGEP: 0.5

                               GNUTELLA/0.6 200 OK
                               User-Agent: BearShare/1.0
                               Pong-Caching: 0.1
                               GGEP: 0.5
                               Private-Data: 5ef89a

GNUTELLA/0.6 200 OK
Private-Data: a04fce
```

Figure 2.4: Gnutella connection exchange.

- 0x00 = Ping
- 0x01 = Pong
- 0x02 = Bye
- 0x40 = Push
- 0x80 = Query
- 0x81 = Query Hit

The TTL field is a mechanism for limiting the scope of messages. It is initialized to a positive value, normally 7 [9]. Each time a message is forwarded by a servent, the TTL field is decremented. A servent will not forward a message with a TTL of zero. TTL fields are used widely in networking protocols such as IP. The *hops* count stored in octet 18 of the header keeps track of how many times a Gnutella message has been forwarded. It is initialized to zero and incremented by every servent that floods the message to its neighbours. The payload length field indicates the length of the message following the header.

Other than optional extensions, ping messages do not contain a payload. They are simply probes to find servents willing to accept new connections.

Table 2.2: Gnutella pong message structure.

Octets	Description
0-1	Port number on which the peer will accept connections.
2-5	IP Address of the responding peer.
6-9	Number of shared files.
10-13	Number of kilobytes shared.
14-	Optional protocol extension.

Pong messages are a reply to pings. They indicate that a servent is willing to accept a connection and they provide information about the responding peer. Table 2.2 shows the pong message structure [29].

Bye messages are an optional indication that a peer wishes to terminate a Gnutella connection. They bear no payload and are always sent with a TTL of 1, so that they are not accidentally propagated beyond the intended target [29].

Table 2.3: Gnutella push message structure.

Octets	Description
0-15	Servent identifier for the target of the push message.
16-19	The index identifying the desired content.
20-23	The IP address of the requesting peer.
24-25	The TCP port the requesting peer is listening on.
26-	Optional protocol extension.

Push messages are a means of downloading files from servents that are unable to accept incoming connections [29]. Nodes that are protected by a firewall, for example, would fall into this category. The push message instructs the receiver to open a connection to the specified peer and transfer the indicated content. The structure of push messages is shown in Table 2.3.

Table 2.4: Gnutella query message structure.

Octets	Description
0-1	The minimum download speed required (in kbps).
2-	NUL-terminated search criteria string.
Others...	Optional protocol extension.

Query message are the means to locate content in the Gnutella network. Their structure is shown in Table 2.4 [29].

Table 2.5: Gnutella query hit message structure.

Octets	Description
0	Number of matching files.
1-2	The TCP port to use for download requests.
3-6	The IP address of the responding peer.
7-10	The speed (in kbps) of the responding peer.
11-	The matches, presented as shown in table 2.6 [29].

Table 2.6: Gnutella query hit result set structure.

Octets	Description
0-3	Index assigned by the responding that identifies the file.
4-7	Size of the file in bytes.
8-	The null-terminated file name string.
Others...	Optional protocol extensions.

Query hits are sent in response to query messages. They indicate that the requested content is available at the originator of the query hit. The structure of query hit messages is shown in Table 2.5 [29].

The previous release of Gnutella, version 0.4 [18], was a flat hierarchy of servents where all peers were considered equal. Powerful computers with multi-megabit connections to the Internet were treated in the same way as piddling home computers with 56-kilobit dial-up connections. With the increase in Gnutella's popularity, and the concomitant increase in network traffic, underpowered computers with slow connections began to be overwhelmed with the task of forwarding Gnutella control messages. As a result, a two-tiered network was implemented in version 0.6 of the protocol, with two classes of peers: *ultrapeers* and *leaves.* Ultrapeers shield their attached leaves from P2P network traffic that is not relevant to them. Leaves only keep a few connections to ultrapeers open [29] and these ultrapeers only send queries to leaves they think may satisfy them. Leaves do not forward queries or any other Gnutella control traffic, which reduces the load on their limited resources. In order to determine which queries should be forwarded to leaves, ultrapeers network normally use the *query routing protocol* (QRP). Leaves construct a hash table containing entries for each of the words in the names of the resources they are sharing. This hash table is communicated to a leaf's ultrapeers. Ultrapeers will forward queries to leaves according to search criteria matches against this table [29].

According to the specification [29], ultrapeer election is based on the following criteria:

Firewall protection The peer must not be shielded by a firewall.

Operating system Certain operating systems have more scalable socket implementations than others [29]. These include Linux, Windows 2000/NT/XP, and Mac OS/X.

Bandwidth Eligible peers should have at least 15KB/s downstream capacity and 10KB/s upstream bandwidth.

Uptime, stability Peers should have been in the network for at least a few hours.

Sufficient memory and processing power Forwarding control traffic requires memory and CPU cycles.

The above rules are quite loosely specified, and it is the responsibility of each implementation to define the precise conditions under which a Gnutella client should become an ultrapeer. Since ultrapeer election is done in a distributed system, each node bears the responsibility for choosing to become an ultrapeer or not [29]. Ultrapeers may change roles to become leaves once again if the implementation deems it appropriate.

The Gnutella 0.6 network may interwork with legacy 0.4 servents. These servents establish neighbour relationships directly with ultrapeers, and behave in exactly the same way as if they were peering with 0.4 implementations. Ultrapeers essentially treat legacy peers as ultrapeers.

2.2.2 The Gnutella Network's Topology

The Gnutella network has been studied extensively in academia. Because it is an open standard, it lends itself particularly well to data collection and analysis. One especially comprehensive analysis was conducted at the University of Chicago [9]. By developing a *crawler* program, researchers were able to use the *ping* and *pong* messages exchanged in the Gnutella network to gather information about its topology. According to these measurements, made in 2001, the Gnutella network had about 50,000 nodes in its largest connected component. Furthermore, the crawler compiled a list of some 400,000 nodes that had been active at some point during the seven month study [9].

The Gnutella network was found to exhibit *power law*, or *scale-free*, properties. Power law distributions were studied extensively by Pareto [30] and are governed by:

$$f_v \propto g_v^k. \tag{2.1}$$

The power law states that an attribute f of vertex v in a network is governed by the attribute g of that vertex, raised to the constant negative power k. In the case of the Gnutella network, the node degree distribution obeys a power law [9]: the number of nodes in the network exhibiting a particular node degree diminishes according to a power law as the node degree increases. If D is a node degree and f_D is the number of nodes with D neighbours, the power law implies that

$$f_D = aD^k, \tag{2.2}$$

where a and k are constants. This is similar to one of the power laws observed by Faloutsos et al. [31], for the AS-level topology of the Internet. The World Wide Web graph, where the vertices are the web pages and the edges are the hyperlinks, also exhibits a power law distribution [32], [33].

Power law networks are common, even beyond the field of communication networks, and their properties have been well studied. Power laws have been observed in cellular participation in biochemical reactions, Hollywood actor collaboration, protein regulatory networks, technical paper co-authorship, and sexual contacts [24].

Power law networks are highly resilient in the face of random node failure [24]. It has indeed been shown that a large proportion of the nodes in a scale-free network may be removed without severely interrupting network connectivity. Conversely, these networks are highly susceptible to the selective removal of a small number of *hub* nodes. These are the rare nodes with a very high number of neighbours. The failure of these nodes may catastrophically disrupt the network.

Nodes in the Gnutella network are generally close to each other. It has been observed that 95% of node pairs are less than 7 hops apart [9]. Since the most common TTL value used in Gnutella messages is 7, this implies that almost all flooded messages reach almost all Gnutella nodes [9]. This traffic traffic is significant, not only in the Gnutella network, but in Internet as a whole [9]. Gnutella control traffic alone accounted for about 1.7% of the estimated total traffic on the United States core network in 2000 [9]. This does not even include file transfers, which consume orders of magnitude more bandwidth than control traffic.

Given the sheer quantity of traffic being exchanged on the network, it seems appropriate to investigate the efficiency of the Gnutella overlay. Efficiency, in this connection, refers to the efficient utilization of network resources during message exchange.

An example of how the Gnutella network's topology may lead to inefficient use of network resources is shown in Fig. 2.3. All communications in the Gnutella network rely purely on the topological information known at the application layer. Hence, nodes may only send messages directly to their logical neighbors. If node 1 needs to retrieve content stored in node 2, its messages must first pass through node 5. This represents only two hops in the logical topology. In the physical topology, this corresponds to at least 5 hops: either $\{1, 6, 7, 5, 4, 2\}$ or $\{1, 3, 4, 5, 4, 2\}$. The inefficiency is particularly egregious in the second path where the physical link between nodes 4 and 5 is traversed twice. If nodes 1 and 2 had elected to be neighbors in the logical topology, only a single physical hop would have been required and no links would have been traversed twice. This would have resulted in lower message latency and a more efficient use of network bandwidth.

It has been observed that less than 5% of Gnutella overlay links connect nodes that are in the same autonomous system (AS) [9]. Autonomous systems, being controlled by a single administrative authority, imply a certain notion of locality. Intra-AS communication is often faster than communicatin between ASs. It is also less expensive for network operators. Another potential indicator of locality is the domain name hierarchy [9]. The round trip time for communication between two hosts on the same domain (e.g., *sfu.ca*) is expected to be smaller than for hosts in distinct domains. Thus, if Gnutella hosts established neighbour relationships with hosts on the same domain, they would incur lower communication costs. Ripeanu and Foster analyzed network entropy to test if the Gnutella network contains a notion of hierarchy [9].

They define the entropy of a set C of size $|C|$ as

$$E(C) = \sum_{i=1}^{n} \left(-p_i \log(p_i) - (1-p_i)\log(1-p_i)\right), \tag{2.3}$$

where p_i is the probability of randomly selecting a host with domain i and n is the number of distinct domain names [9]. The entropy of a network with $|C|$ nodes and k clusters is defined as

$$E(C_1, C_2, \ldots, C_k) = \sum_{i=1}^{k} \frac{|C_i|}{|C_1| + |C_2| + \ldots + |C_k|} \times E(C_i). \tag{2.4}$$

The entropy with clustering (2.4) around highly connected nodes in the Gnutella overlay was not lower than the entropy without clustering (2.3). Hence, Gnutella nodes cluster independently of the domain hierarchy [9].

The two experiments [9] show that the Gnutella overlay topology is not well matched to the underlying physical topology. As a result, Gnutella uses network resources inefficiently. Not only does this have an adverse effect on network utilization as a whole, but it also negatively impacts users because their messages take longer to circulate through the network, which delays their eventual download of the desired content.

It is clearly desirable to improve the way the Gnutella overlay uses the underlying physical infrastructure in order to achieve better network utilization and improve users' *quality of experience* (QoE).

2.3 Network Simulation

The complexity of real-world networks often precludes the use of closed-form mathematical models [34]. It is not feasible, for example, to find an equation that describes the behaviour of the Internet. The use of testbeds of realistic scale is also generally not possible, given the large number of systems involved. It would be prohibitively expensive to construct a 50,000-node network to model the Gnutella network, for instance. Thus, research must often rely on simulation in order to approximate the networks of interest.

Simulation presents a number of potential pitfalls. In particular, using an overly simplified simulation model may cause critical aspects of Internet behaviour to be overlooked [35]. Also, if many researchers rely on the same simulator, they risk being affected by the same software defects and underlying assumptions [35], which may lead to erroneous conclusions. Nevertheless, simulation does play an important role, when measurement and rigorous mathematical modelling is not possible.

In this subsection the simulation tools used in investigating the effects of the proposed enhancements to the Gnutella network are discussed.

2.3.1 Opnet

Opnet [36] is one of the most popular tools used in industry. It is a powerful simulator with a rich library of simulated network devices and support for user-defined state machines. Its flexibility is limited, however, because the user cannot access and modify the source code as with open-source simulation packages.

2.3.2 SSFNet

SSFNet [37] is a collection of open-source models for protocols and network elements. It is implemented in Java and utilizes SSF, the *Scalable Simulation Framework*. While the underlying framework has a high-performance C++ binding, the Java-based models raise important scalability concerns due to the inferior performance of Java in speed-sensitive applications.

2.3.3 Dedicated P2P Simulators

There are a number of tools dedicated to P2P network simulation. Among them is *p-sim* [38], which achieves high scalability by neglecting packet-level details. 3LS [39], the "three-layer simulator", on the other hand, is very much predicated on the importance of such details. SimP^2 [40] is another P2P simulator, which in intended to simulate simple file sharing networks. It does not, however, take into account the transient presence of nodes, which is a defining characteristic of P2P networks. There are many other simulators, but overall, no dedicated P2P network simulator has achieved wide acceptance in the research community.

2.3.4 ns-2

The *ns-2* network simulator [14] is one of the most popular discrete event simulators [41]. It evolved through the contributions of researchers at the Information Sciences Institute at the University of Southern California, and elsewhere. *ns-2* has received funding from DARPA and the National Science Foundation [14] and continues to grow as developers contribute to its open-source codebase.

Because of its extensive support for IP, TCP, mobile and routing protocols, *ns-2* has gained favour in the research community. Also, because the source code is freely available and may be modified, it is a versatile and flexible tool.

The realtime-critical portions of the *ns-2* engine, such as packet and event processing are written in the C++ programming language. Scripts that drive the tests are normally written in oTCL, which is the object-oriented version of the popular Tool Command Language (TCL). This combination of C++ and oTCL allows the user to benefit from the performance of a compiled language where needed and the flexibility of an interpreted language when appropriate.

Because they simulate packet-level exchanges, the processing and memory overhead of *ns-2* simulations is quite high. Consequently, *ns-2* is not particularly scalable. Simulations virtually grind to a halt with more than a few hundred nodes. A parallel version of ns known as PDNS [42] offers the possibility of distributing simulations on 8-16 workstations connected by a LAN.

ns-2 was employed in this research for several reasons. Although it introduces scalability issues, the level of detail supported by *ns-2* is important. It has been shown that the performance of P2P networks is highly sensitive to the details of the underlying physical network [9], [11], [35], [43]. For this reason, more scalable P2P simulators that sacrifice the packet-level information modelled by *ns-2* were ruled out. Also, the fact that it was possible to customize the source code to implement arbitrary statistics gathering where most convenient was a key feature. *ns-2* is well regarded in the research community and has a large base of users available for support; this weighed heavily in the selection process. Finally, the fact that there was a Gnutella simulation framework available for *ns-2* made the choice to use *ns-2* clear.

2.4 The GnutellaSim Simulation Package

GnutellaSim [44] is a packet-level simulator for the Gnutella network. It relies on a "scalable and extensible packet-level P2P [11]" simulation framework developed at Georgia Tech. This framework is in turn designed to run with the *ns-2* simulator, among others. The framework comprises a number of concrete classes that provide basic P2P packet forwarding and infrastructure services. It also defines a number of abstract classes, intended to be used as bases for particular P2P protocol implementations, such as GnutellaSim.

GnutellaSim extends *ns-2*'s existing TCP implementation by introducing several new features [11], including a socket-like interface, dynamic connection establishment of TCP sessions and real payload transfer. The software is structured in a three-layer architecture. The application layer is responsible for the users' behaviour profile, and the initiation of protocol messages such as queries. The protocol layer is concerned with protocol message semantics, network formation and message forwarding. The socket adaptation layer is a bridge between the socket-like interface provided by GnutellaSim's framework to the application and the underlying *ns-2* simulator.

GnutellaSim relies on the so-called UMASS model [45] to characterize peer behaviour. This model, as implemented in GnutellaSim, specifies the following parameters for peers:

- the average time they are offline
- the average time they are idle (not sending queries)
- the probability of going offline after a successful query
- whether they share content or not (freeloaders)
- the number of files they share

When a query arrives at a node, the probability of the node having the requested content, and generating a query hit, is conditioned by the number of files shared by that node. It is possible to define multiple classes of peers, with different parameter values, in the same simulation.

GnutellaSim implements the Gnutella 0.6 [29] protocol, with support for leaves, ultrapeers and legacy version 0.4 peers. The role of a node, however, is specified at configuration time and is static throughout the simulation.

GnutellaSim was used as the basis for evaluating the proposed modifications to the Gnutella protocol.

Chapter 3

The Vivaldi Coordinate System

In this chapter, the Vivaldi coordinate system is discussed. Vivaldi was used in the proposed modifications to improve the performance of the Gnutella protocol. The need for network coordinates in presented in Subsection 3.1. Several network coordinate systems are then introduced in Subsection 3.2. Finally, in Subsection 3.3, the operational details of the Vivaldi coordinate system are presented.

3.1 The Need for Network Coordinates to Improve Network Performance

As discussed in Subsection 2.2.2, the Gnutella network exhibits a mismatch between the physical and logical (overlay) topologies. This mismatch causes inefficient network utilization as peers establish *neighbour* relationships without considering the underlying physical network. It would be more efficient to bias the selection of neighbours to favour nodes that are physically close, thus aligning the P2P overlay with the physical topology. Since queries and query hits will propagate faster, the modifications should lead to an improved user QoE.

Round trip time (RTT) is a common measure of *closeness* in networks. RTT is the time it takes for a message to propagate from the sender to the receiver and to return to the sender. The RTT is of the order of milliseconds in wireline networks. It is reasonable to base the formation of the overlay on predicted RTT, choosing neighbors to which the RTT is low. *Network coordinate systems* provide a means to estimate inter-node RTT without the need for explicit measurement. They use regular protocol communication as a means to convey information and nodes are therefore continually updating their coordinates and distance estimates to their neighbours.

One simple alternative to using network coordinate systems would be to use ICMP echo (*ping*) messages to measure inter-node latency. It is not, however, feasible for every node in the Gnutella network to measure the RTT to every other node in the network when evaluating prospective neighbours. This would lead to $O(N^2)$ ICMP echo messages for an N-node network. Even if that were not prohibitively costly, there is no single resource that stores all the addresses of Gnutella nodes. Hence, it would not be possible to determine all the participants in the network in order to measure a round trip time to each one. Nevertheless, when a Gnutella node learns of a potential neighbor, it should be in position to decide whether it should peer with that node. The node receiving the connection request could simply *ping* the request originator before deciding whether to connect. It would thus obtain an instantaneous RTT measurement to the originator. This measurement could then be compared to the RTT to other neighbours and the closest ones could be selected. In order to ensure a valid comparison, all neighbour RTT values should be updated regularly, through polling. Such explicit measurements, especially when conducted on a regular basis, can be unattractive because their overhead cost often outweighs the benefit they yield [10].

Another potential approach would be to perform clustering based on one of the two locality indicators identified in Chapter 2: autonomous systems or domain names. Gnutella hosts are not aware of their AS. This information is only known by the routers propagating topology information through the network. For this reason, deliberately selecting neighbours within the same AS is not possible. Clustering based on domain names is possible, but would require a DNS lookup for every connection request, which could be onerous. With this method, it would be important to ensure that peers maintain a certain number of neighbors from outside their domain, otherwise the Gnutella network would become fragmented into islands corresponding to domain boundaries. Such a disconnected network would limit the horizon of searches and reduce the probability of locating desired content. Clustering based on domain names is not ruled out as a viable solution, but it is not explored in this research and remains an area for possible future investigation.

One final approach to be considered would be to use an RTT limit and only accept connections from nodes below a specific RTT value. By imposing a large enough minimum node degree n – that is by relaxing the maximum RTT requirement for the first n connections – the probability of obtaining a connected network is high [46]. The difficulty with this approach is choosing an appropriate RTT threshold. This could be an area for further research.

Other methods notwithstanding, network coordinate systems appear to be a most appropriate tool for biasing the formation of the P2P overlay to favour connections between nodes that are close together.

3.2 Network Coordinate Systems

Network coordinate systems (NCSs) assign coordinates to each node in the system and use the distance (Euclidean, Manhattan, Mahanalobis) to estimate the physical distance (RTT) between nodes [10]. Consider a system where coordinates are represented as 4-dimensional vectors. There are two nodes: N_1, with coordinates $(2,\ 3,\ 4,\ 7)$and N_2, with coordinates $(4,\ 3,\ 1,\ 2)$. Using the Euclidean distance function, the round trip time between the two nodes is estimated as:

$$\begin{aligned} RTT &= \sqrt{(2-4)^2+(3-3)^2+(4-1)^2+(7-2)^2} \\ &= \sqrt{38}. \end{aligned} \tag{3.1}$$

Many coordinate systems rely on fixed infrastructure nodes in order to calculate node coordinates. Global Network Positioning (GNP) [47], Network Positioning System (NPS) [48] and Lighthouse [49] use *landmark* or *beacon* nodes as reference points for coordinate calculation. Network participants derive their coordinates by measuring their RTT to landmarks. While these systems predict RTTs between nodes with some success, their reliance on fixed infrastructure nodes makes them incompatible with the P2P paradigm.

There are also coordinate systems that are fully decentralized, with no dependence on infrastructure nodes. The Practical Internet Coordinates (PIC) system [50] does not rely on fixed landmark nodes. Nevertheless, it is oversensitivity to changing network conditions, which may make it unsuitable in dynamic network conditions [10] such as P2P environments. Vivaldi [10] is another decentralized coordinate system, which is used by the Chord [22] P2P network's lookup algorithm. Vivaldi is presented in detail in Subsection 3.3.

3.3 Vivaldi Operation

Vivaldi [10] is a decentralized system used to assign synthetic coordinates to nodes participating in a network. It does not rely on fixed infrastructure nodes and, as such, may be be suitable for P2P networks. The Vivaldi coordinate system employs the Euclidean distance between two nodes' coordinates to estimate the RTT between them (3.1). This subsection presents the operational details of the Vivaldi algorithm.

3.3.1 Error Minimization

The error for a coordinate pair is defined as the difference between the predicted RTT (the coordinates) and the actual RTT [10]. The squared error function is

$$e = \left(R - \|x_i - x_j\|\right)^2, \tag{3.2}$$

where R is the actual RTT, and x_i and x_j are the two coordinates. If R_{ij} is the actual RTT value between nodes i and j, and x_i and x_j are defined as the coordinates of i and j, then the squared error E for the system is [10]

$$E = \sum_i \sum_j \left(R_{ij} - \|x_i - x_j\|\right)^2. \tag{3.3}$$

The Vivaldi algorithm employs the squared error function because it is analogous to spring relaxation in a physical spring-mass system [10]. These associations follow if a spring is placed between each pair of nodes for which latency measurements exist [10]:

Length of the spring: models the distance between the nodes given their current coordinates

Spring rest position: occurs when the coordinates predict the RTT with zero error.

Potential energy of the spring: the square of its displacement from its rest position. Models the error in the coordinate pair.

Potential energy of the system: the squared-error function (3.3). Minimizing this function gives optimal coordinates.

3.3.2 Spring Relaxation and Coordinate Adjustment

The Vivaldi algorithm models the movements of the nodes under the forces applied by the conceptual springs between them [10]. The algorithm seeks to minimize the potential energy of the spring system.

Let F_{ij} be the force that node j exerts on node i. Hooke's law [51] states that the force is proportional to the spring's displacement from its rest position, and in the opposite direction. Hence, the force is [10]

$$F_{ij} = (R_{ij} - \|x_i - x_j\|) \times u\,(x_i - x_j)\,, \tag{3.4}$$

where $(R_{ij} - |x_i - x_j|)$ is the magnitude of the spring displacement and $u\,(x_i - x_j)$ is a unit vector in the direction of the force [10] (pushing i along a line connecting it to j, either closer or farther). If nodes have identical coordinates, $u\,(x_i - x_j)$ is defined as a unit vector in an arbitrary direction [10].

Since the actual round trip time R_{ij} is not known, nodes adjust their coordinates in response to sampled RTT values learned from communication with other nodes. Based on these samples, nodes allow their coordinates to be "pushed" for a short time δ by the inter-node force (3.4) [10]. For a sample RTT r_{ij} between nodes i and j, node i will adjust its coordinates to:

$$x_i = x_i + \delta \times (r_{ij} - \|x_i - x_j\|) \times u\,(x_i - x_j)\,. \tag{3.5}$$

3.3.3 Vivaldi Algorithm

With every message sent, nodes participating in the Vivaldi process append three additional values: their coordinates, their estimate of the error on those coordinates, and a timestamp so that the receiver can calculate the RTT. A node receiving a message will apply the Vivaldi algorithm, which can be summarized in the following steps [10]:

1. *Calculate the credence to be given to the new sample*: If the sample bears a high error, the receiving node will in response adjust its coordinates only slightly.

Node movement is also conditioned by the local error. If a node has high local error, it will give more weight to reports from other nodes. The sample weight [10] is defined as the ratio of the local and sample errors:

$$w = \frac{e_i}{e_i + e_j}, \tag{3.6}$$

where e_i and e_j are the local and remote errors, respectively.

2. *Calculate the relative error of this sample*: Based on the RTT predicted by the coordinates $\|x_i - x_j\|$ and the measured RTT (rtt) [10]:

$$e_s = \frac{|\|x_i - x_j\| - RTT|}{RTT}. \tag{3.7}$$

3. *Update the local error*:

$$e_i = e_s \times c_e \times w + e_i \times (1 - c_e \times w), \tag{3.8}$$

where c_e is a tuning parameter.

4. *Update the local coordinates*: The amount to move the coordinates δ is a constant proportion c_c of the calculated sample weight (3.6). The recommended value $c_c = 0.25$ is based on empirical observation [10]. The suggested initial value for δ is 1. The coordinates are updated as [10]

$$\delta = c_c \times w \tag{3.9}$$

$$x_i = x_i + \delta \times (RTT - \|x_i - x_j\|) \times u(x_i - x_j). \tag{3.10}$$

The adaptive timestep δ employed by Vivaldi helps to achieve fast convergence and low oscillation by "believing" nodes with relatively low error more than nodes with high error [10].

3.3.4 Vivaldi Coordinate Types

After examining Euclidean and spherical coordinates, and evaluating their performance, the authors of the seminal Vivaldi paper [10] chose to employ Euclidean coordinates with height vectors. These height vectors model packet propagation time through the access link into the Internet's core. Height vectors redefine some of the usual vector operations as follows [10]:

1. *Difference* is defined as [10]:

$$[x, x_h] - [y, y_h] = [(x - y), x_h + y_h], \tag{3.11}$$

   where x and y are Euclidean coordinate vectors, and x_h and y_h are their height components. It is worth noting that whilst the Euclidean coordinates are subtracted as usual, the height components are added together in the difference operation. In Euclidean space, if a Vivaldi node is too close to nodes in opposite directions, the forces applied on it will cancel out [10]; with height vectors, the forces will push the node "up": its height component will increase even in the presence of equal and opposite forces.

2. *Magnitude* is defined as [10]:

$$\| [x, x_h] \| = \|x\| + x_h, \tag{3.12}$$

   where x_h is positive value.

3. *Scaling* is defined as [10]:

$$\alpha \times [x, x_h] = \|\alpha x, \alpha x_h\|, \tag{3.13}$$

   where α is a scalar.

Height vectors were found to predict the RTT more precisely than 2 and 3-dimensional Euclidean coordinates and spherical coordinates [10].

3.3.5 Vivaldi Accuracy

In a study involving 1,740 hosts on the Internet, the Vivaldi coordinate system, with two Euclidean dimensions and a height component was found to predict round trip time with a median relative error of 11% [10]. This result was achieved using Internet domain name servers as the nodes for which the RTT was to be predicted. Vivaldi's authors used the King [52] method in order to measure the actual RTT between each pair of nodes in the set. For example, to measure the RTT between A and B, the probing host first measures the RTT to A. It then requests that A resolve a domain served by B [10]. The difference between the two times is an estimate of the RTT between A and B. The RTT was continuously measured (100×10^6 times) over the course of a week, and compared with the results produced by the Vivaldi algorithm. The errors observed were as low as GNP [47], which uses fixed infrastructure nodes to help assign synthetic coordinates [10].

Chapter 4

Modifications to the Gnutella Protocol

In this chapter, modifications to the Gnutella protocol are proposed in order to implement and use the Vivaldi coordinate system.

It is worth noting that the nodes being considered when evaluating Vivaldi's accuracy (in Subsection 3.3.5) were stable DNS servers; P2P nodes are much more volatile. While DNS servers are virtually always available, P2P nodes join and leave the network frequently. It is worth exploring whether the use of Vivaldi synthetic coordinates can help mitigate the topological mismatch of the Gnutella P2P network discussed in Subsection 2.2.2. If Gnutella servents had coordinates that could reliably predict the round trip time between nodes, they could chose to form neighbour relationships with nodes that are near to them. In Subsection 4.1, the syntactic modifications to the Gnutella messages are outlined. In Subsection 4.2, the behavioural modifications to the nodes using the Gnutella protocol are presented. Finally, in Subsection 4.3, the costs and risks associated with these proposed modifications are discussed.

4.1 Syntactic Modifications

The proposed modification to Gnutella messages is that they include Vivaldi coordinates. As discussed in Subsection 3.3.4, 2D Euclidean coordinates augmented with height vectors appear to provide sufficiently accurate results, therefore these three coordinate components may be inserted in Gnutella binary and text messages. It is important that they be in the text messages because these messages are the vehicle for connection requests, based upon which Gnutella nodes form neighbour relationships. This information must also be included in binary messages so that the communicating nodes' coordinates will converge as messages circulate in the network.

In addition to communicating nodes' coordinates, the Vivaldi algorithm requires the estimated coordinates error and a means to estimate the RTT (such as a "send timestamp"). This timestamp represents the time a message is sent and may employ a common time base used by all Gnutella nodes to synchronize when they join the network. Assuming that the latency is symmetrical, an estimate of the RTT is twice the difference between the timestamp and the time a message is received. Therefore, the proposed modifications include nodes' estimated coordinate error and send timestamp in all Gnutella messages.

It is necessary to strike a balance between adequate precision for the coordinates, error, and timestamp, and the extra bandwidth required to transmit this information. For the purpose of evaluating the proposed protocol modifications, the implementation uses double precision floating point numbers for all quantities, whilst recognizing that the introduction of forty additional octets (each of the coordinate components, the error and the timestamp account for 8 octets)of overhead to all binary message headers might unreasonably increase the load on the network. It might be feasible to use only 32-bit, single precision floating point representations of the coordinates, error, and timestamp, given the millisecond scale of RTT values. The 32-bit representation is not investigated in this thesis. Instead, the thesis focuses on examining the viability of the synthetic coordinate approach itself. The modified structure of the Gnutella binary message header is shown in Table 4.1.

This information could be encoded into text (connection sequence) messages by adding an additional field named *Vivaldi-Data*. Because the data is ASCII text [29], each character added to the message requires 8 bits. It would therefore be economical to use a hexadecimal representation for the coordinates, error, and timestamp, and concatenate them into a single string to be included after the Vivaldi-Data header. This is shown in Fig. 4.1 for a *connect* message.

The ten-hexadecimal-digit sequence *aaaaaaaaaa* represents the forty octets of information required to convey the Vivaldi 5-tuple of $(X, Y, height, error, timestamp)$.

Table 4.1: Modified header structure of Gnutella binary messages.

Octets	Description
0-15	Message globally unique identifier.
16	Payload type
17	TTL (Time to live)
18	Hops
19-22	Payload length
23-30	Vivaldi X coordinate
31-38	Vivaldi Y coordinate
39-46	Vivaldi *height* coordinate
47-54	Vivaldi coordinate error
55-62	Vivaldi send timestamp

```
GNUTELLA CONNECT/0.6
User-Agent: BearShare/1.0
Pong-Caching: 0.1
Vivaldi-Data: aaaaaaaaaa
```

Figure 4.1: Gnutella connect message bearing Vivaldi data.

Gnutella servents ignore headers they do not support in the text messages exchanged during the connection phase [29]. Thus, the use of the additional header *Vivaldi-Data* does not pose a backwards compatibility problem. The proposed binary messages, however, will be incompatible with servents that do not implement the modifications. For this reason, both servents involved in a connection must agree to use the Vivaldi enhancements during the connection phase, when they exchange capability headers. If either servent does not support Vivaldi, then they cannot communicate using the modified protocol.

4.2 Behavioural Modifications

To make use of the proposed Vivaldi enhancements to the Gnutella protocol, servents must behave differently. This subsection describes the proposed behavioural modifications.

Firstly, The proposed modifications only operate in ultrapeer-to-ultrapeer communication: leaves and legacy Gnutella 0.4 servents do not employ the enhancements.

In the two-tiered Gnutella hierarchy, peers were divided into ultrapeers and leaves to conserve the networking and processing resources of leaves [29]. Hence, the proposed modifications are not applied to leaves. We therefore avoid adding extra bytes to the messages sent to leaves and eliminate the burden of processing each message bearing Vivaldi coordinates. New protocol modifications do not apply to legacy peers because they employ an old version of the protocol.

4.2.1 Initialization

The coordinates of a node joining the network are initialized to the origin $(0, 0, 0)$ and its local error is set to 5×10^6 (an arbitrary large value). These initial values are used every time a node enters the network, regardless of whether it had previously been a participant. This is necessary because node coordinates will *drift* significantly over time due to the highly dynamic nature of the P2P overlay topology. Coordinates held previously by the node are of little use.

4.2.2 Coordinate Updates

When a node receives any Gnutella message bearing Vivaldi data, it uses the information to update its coordinates and local error according to the algorithm described in Chapter 3. It uses twice the difference between the current time and the send timestamp of the received message as an estimate of the RTT to the sending node. Vivaldi information is "piggybacked" on Gnutella control traffic, and, hence, nodes' coordinates are continuously updated through the normal exchange of messages in the network.

4.2.3 Message Forwarding

When a Gnutella servent forwards a message to one of its neighbours either through flooding (*ping* or *query*) or backrouting (*pong* or *query hit*), it includes its coordinates, timestamp, and error estimate in the message. The updated values included in the received messages replace the previous values recorded at a node.. Nodes must only consider the Vivaldi information of their immediate neighbours when updating their coordinates because the RTT estimate is only meaningful for nodes that are directly connected in the overlay topology.

4.2.4 Optimal Neighbour Selection

The topological mismatch between the overlay and physical topology is addressed by servents electing neighbors that are physically close. Thus, nodes need to judiciously select how to respond to *connect* messages they receive. The number of connections (a finite maximum) a Gnutella servent can accept is decided by the client implementation. It is a small number: of the order of tens rather than hundreds of connections [9]. If this maximum has not been reached, Vivaldi-modified Gnutella clients will accept any connection request, as in the case of standard clients. The receiving client stores an estimate of the distance to the node on the other side of the new connection. The estimate is based on the Euclidean distance to that node's coordinates. Every time a new message with new coordinates is received from that node, the distance estimate is purged. If a node reaches its maximum allowed number of connections and a new connection request is received, it will consider dropping an existing connection to accommodate the new request. It first estimates the distance to the requesting node based on its coordinates. It then searches its own connection records for the node estimated to be the farthest. If this node is farther than the requesting node, the existing connection is terminated and the new connection is accepted. Otherwise, the new connection request is rejected. Thus, the Gnutella nodes use the Vivaldi information to form neighbour relationships with nodes that are physically close. The pseudocode for the neighbour selection algorithm (NSA) is shown in Fig. 4.2.

Peers initiate connection requests because they have not reached their maximum number of connections. They should, therefore, attempt to connect to any peer they ar eaware of; no special logic is required in this situation. When a peer receives a positive response to a connection request it initiated, it records a distance estimate to the responding node. The new connection will be eligible for discard if the servent reaches the maximum number of allowed connections.

In summary, when a node has room for more connections, it initiates and accepts connections exactly as a normal Gnutella client would. When the maximum number of connections has been reached, it only accepts connections to peers that are closer

```
// Accept or reject a new connection
// request based on the estimated
// RTT to the requesting node
AcceptOrRejectConnectionRequest(RTT, NodeId) {
  if (neighborList.roomForMore()) {
    // We still have connection slots
    // available.
    acceptNewConnection(NodeId)
    return
  }

  currentWorstDistance   = 0
  foundNeighborToReplace = false
  neighborToReplace = null

  // Find the farthest neighbour. Must be farther than the RTT for
  // the requesting node.
  foreach neighbor in neighborList {
    if ((neighbor.Distance < RTT) and (neighbor.Distance > currentWorstDistance)) {

       currentWorstDistance = neighbor.Distance
       neighborToReplace = neighbor
       foundNeighborToReplace = true
    }
  }

  if (foundNeighborToReplace) {

    disconnect(neighborToReplace)
    neighborList.erase(neighborToReplace)
    acceptNewConnection(NodeId)
    neighborList.add(NodeId, RTT)
  }
  else
    rejectNewConnection(NodeId)
}
```

Figure 4.2: Neighbour selection algorithm.

than the peers to which it is connected.

4.3 Costs and Risks of the Modifications

There are costs and risks associated with the proposed modifications. The tradeoffs involved with the neighbour selection algorithm and its implementation in Gnutella are discussed in this subsection.

4.3.1 Costs of the Modifications

As with most protocol enhancements, the costs are mainly associated with increased message size and additional processing. By adding by adding an additional forty octets to each Gnutella binary message, the header size would be almost trippled. Even if only 32-bit values were used for the coordinates, error, and timestamp, the algorithm could still be adding an additional twenty octets, or almost doubling the header size. To keep this seemingly enormous cost in perspective, it is important to recall that Gnutella control traffic volumes are orders of magnitude less than the associated file transfer activity, which this proposal does not modify. Hence, even though the amount of control traffic would be increased, the overall effect would be slight: traffic volumes would not be significantly increased and the *bit cost* of the proposed modifications is not a major concern. The processing costs at the nodes are far more important. Each node must update its coordinates with every message received. Assuming a control traffic rate of 6 kb/s [9], and knowing that the vast majority of messages (91%) [9] are queries with a minimum size of 25 bytes, a lower bound for the number of coordinate updates per second can be estimated as:

$$\begin{aligned} U \geq \tfrac{6{,}000 \text{ bits/sec}}{25 \text{ bytes} \times 8 \text{ bits/byte}} &= 30 \text{ messages/s} \\ &= 30 \text{ updates/s.} \end{aligned} \tag{4.1}$$

While this is a fair number of operations, it is not unreasonable for today's fast processors.

4.3.2 Risks of the Modifications

In addition to the costs, the proposed modifications do carry certain risks.

One of the most important risks is that the enhancements leave the network vulnerable to malicious nodes. A node that misreported its coordinates, especially with a very low reported error, could mislead other nodes and reduce the accuracy of their coordinates. Furthremore, a node could send connection requests with false coordinates engineered to be close to the target node's coordinates, and thus cause it to discard other legitimate connections. For this proposal to be safely deployed, it would have to be augmented with a trust management algorithm. Such security concerns are beyond the scope of this particular research project.

Another potential risk is that the dropping of existing connections (*connection churn*) and the resulting network instability may actually degrade the user experience and make it harder to locate content quickly. While the network is undoubtedly more unstable as a result of connections being dropped and new ones being established according to the neighbour selection algorithm, instability is not inherently detrimental to the location of content and is therefore not directly measured.

One final risk is that by causing nodes to preferentially associate with nodes that are physically close, the algorithm may be fragmenting the Gnutella network and limiting nodes' search horizon. If distant nodes are always rejected in favour of closer ones, disjoint networks may form (one on each continent, for example). If some nodes have not filled all their available connection slots, they would still accept connections from distant servents. Thus, inter-continental connections would still be possible, although less frequent. The presence of disjoint networks is not directly tested, but the simulations discussed in subsequent chapters do measure the time required to locate content, which is the ultimate indication of the network's success and the user's QoE.

Chapter 5

System Architecture

In this chapter, the newly developed *Gnutaldi* (Gnutella + Vivaldi) simulation framework is presented. This simulator is based on the GnutellaSim work presented in Subsection 2.4 as well as the Vivaldi coordinate system described in Chapter 3. In Subsection 5.1, the purpose of the simulator is discussed. In Subsection 5.2, the simulator's software architecture is presented. In Subsection 5.2.5, the platform upon which the simulations were run is described.

5.1 Gnutaldi Objectives

The purpose of the Gnutaldi simulator is to evaluate the performance of the proposed modifications to the Gnutella protocol.

It is neither feasible nor desirable to implement modified Gnutella clients in a deployed network on any meaningful scale. Hence, we rely on network simulations. In order to observe the effects of the proposed neighbor selection algorithm on the performance of the Gnutella network, we developed a new network simulator: Gnutaldi (Gnutella + Vivaldi). Gnutaldi evolved from the GnutellaSim simulator [11] based on *ns-2* [14]. The choice of the *ns-2* simulator was motivated by a desire to capture packet-level details. It has the disadvantage of a fairly unscalable platform. As discussed in Subsection 2.3.4, *ns-2*-based simulations do not scale to the tens of thousands of nodes required to model a realistic Gnutella network. Nevertheless, even simulating small networks with tens of nodes provided a useful test of the proposed modifications.

The GnutellaSim simulator [11] provides a starting point for evaluating the performance of the existing Gnutella network. With some modifications, it was possible to customize it to collect statistics on query propagation times and the time required for nodes to receive query hits, thus establishing a baseline for comparison with the modified version. After implementing the modifications to Gnutella in the simulator, it was possible to obtain results for comparison with the unmodified protocol, which was a major objective of the project.

A second objective of the Gnutaldi simulator is to provide a means to observe the effect of parameter modifications within the enhanced Gnutella protocol or the network environment.

In summary, the Gnutaldi simulator is intended to model the modified Gnutella protocol and allow network observations.

5.2 Gnutaldi Architecture

The software architecture of the Gnutaldi simulator is discussed in this subsection. As stated, Gnutaldi is based on GnutellaSim [11], which, in turn, is based on *ns-2* [14]. The relationship between the three simulators is illustrated in Fig. 5.1.

5.2.1 Protocol Layer Operation

The modules discussed in this subsection are largely unchanged from the original GnutellaSim implementation [11], but are included for completeness. These classes deal with forwarding control messages according to protocol specifications, but do not contain any logic related to the generation of these messages or to user behaviour. This is in the domain of the application layer modules.

PeerAgent

This class is the base for all protocol layer modules. The class hierarchy is shown in Fig. 5.2. It defines virtual methods for the basic operations of a peer in any P2P network, such as callbacks for packet reception and connection establishment. It provides no functionality for these callbacks; this is left to subclasses.

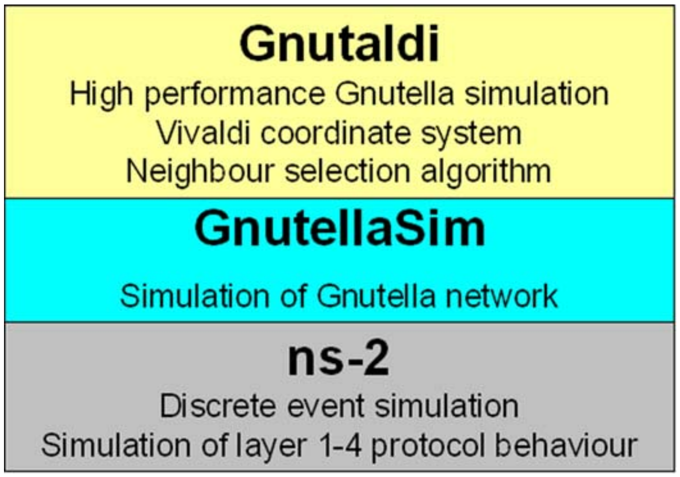

Gnutaldi
High performance Gnutella simulation
Vivaldi coordinate system
Neighbour selection algorithm
GnutellaSim
Simulation of Gnutella network
ns-2
Discrete event simulation
Simulation of layer 1-4 protocol behaviour

Figure 5.1: Relationship between *ns-2*, GnutellaSim and Gnutaldi. *ns-2* is the fundamental building block for the other two simulators. GnutellaSim is a set of classes that extends *ns-2* to include simulation of the Gnutella network. Gnutaldi is an extension of GnutellaSim which adds higher performance and implements the Vivaldi coordinate system and the proposed neighbour selection algorithm.

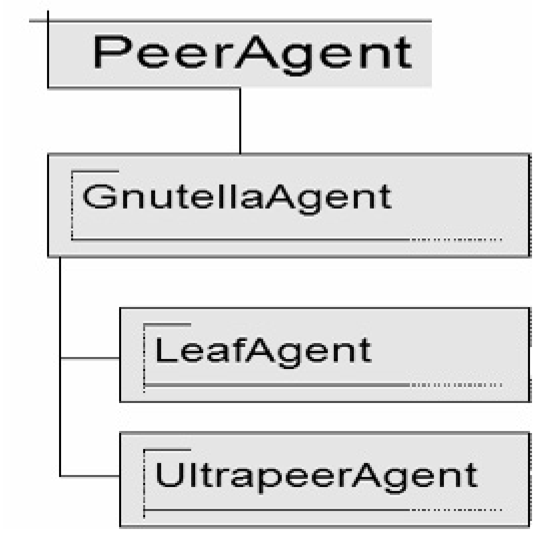


Figure 5.2: Class hierarchy for Gnutaldi protocol layer modules.

GnutellaAgent

This class is derived from *PeerAgent*, and encapsulates the protocol operation of a Gnutella. It contains virtual methods for the sending and receiving of all Gnutella messages, such as *ping*, *pong*, *push*, *query*, and *query hit*. The *GnutellaAgent* class provides default implementations for these methods: they behave as for a legacy Gnutella v. 0.4 peer. The *GnutellaAgent* handles the forwarding of Gnutella messages in a way transparent to upper layers, and only notifies the application when a message of interest is received. The application, in turn, communicates with the *GnutellaAgent* by instructing it to connect to particular peers, or to send queries and query responses. The *GnutellaAgent* participates in statistics collection and interacts with the *GnutStats* class.

LeafAgent

This is a specialization of the *GnutellaAgent* class adapted for Gnutella 0.6 leaf nodes. If differs principally in that it will not consider forwarding received messages: leaves only terminate messages in Gnutella. It will also only accept connection requests from ultrapeers, as mandated by the protocol.

UltraAgent

This is another specialization of *GnutellaAgent*, adapted to encapsulate the behaviour of Gnutella 0.6 ultrapeer nodes. Ultrapeers do not forward ping messages to their shielded leaf nodes. Furthermore, they only forward queries to leaves with a certain probability. This is done because GnutellaSim does not yet implement the query routing protocol (QRP).

5.2.2 Application Layer Operation

The modules described in this section encapsulate the functionality of Gnutella clients. They are responsible for the initiation of Gnutella connections and searches. They also model user behaviour.

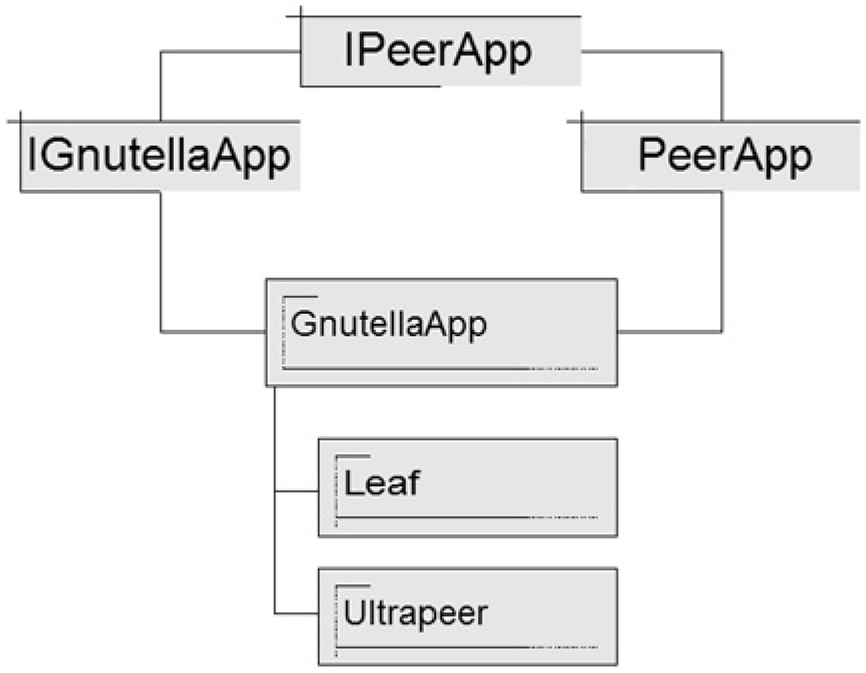


Figure 5.3: Class hierarchy for Gnutaldi application layer modules.

IPeerApp

This interface class declares pure virtual methods which could be implemented by any application layer entity in a P2P network. These methods are: join, leave, search, share, setState, bootstrap (search for nodes to connect to), maintenance (update connections), and connect. In this work, it was added to the original GnutellaSim project because it helps make the class interface clearer and allows for the eventual manipulation of any P2P applications implementing the interface through pointers or references to the base class. It is the base for all other classes in this subsection. The class hierarchy is shown in Fig. 5.3.

PeerApp

This class, derived from *IPeerApp* provides empty implementations for the methods declared in the interface. It is meant to be a null implementation: a peer that does nothing. It stores settings for the application such as the peer's address, the number of files it is sharing, and its connection speed.

IGnutellaApp

This interface class is derived from *IPeerApp* and defines additional pure virtual methods to be implemented by Gnutella application classes, which are divided into two categories: instructions to the protocol layer and callbacks invoked from the protocol layer to notify the application of events. Examples of instructions to the protocol layer include disconnecting from a particular node, sending a ping message, and replying to a query with a query hit. Callbacks invoked by the protocol layer objects include notifications of bootstrap results, indications that query has been received, and confirmation that a connection request has been accepted by another node.

GnutellaApp

This is the concrete implementation of *IGnutellaApp*, and also derives from *PeerApp*. This class encapsulates the behaviour of a Gnutella v. 0.4 servent. In addition to performing all the operation mandated by its parent interface, it stores connection information for each of its neighbours and interacts with the *VivaldiManager* in order to maintain its coordinates and make connection decisions based on nodes' coordinates. The flowchart illustrating this decision process is shown in Fig. 5.4. With each Gnutella message arrival, *GnutellaApp* updates its estimate of the distance to the originating node and its local coordinates and error, as shown in Fig. 5.5. *GnutellaApp* is also heavily involved in gathering statistics, which it maintains via the *GnutStats* manager.

Leaf

The *Leaf* class is derived from the *GnutellaApp* base and specializes it to encapsulate the behaviour of a Gnutella v. 0.6 leaf servent. Leaves will only attempt to connect to ultrapeers and will reject any other connection requests. This is the only difference, from an application layer perspective, between this class and *GnutellaApp*.

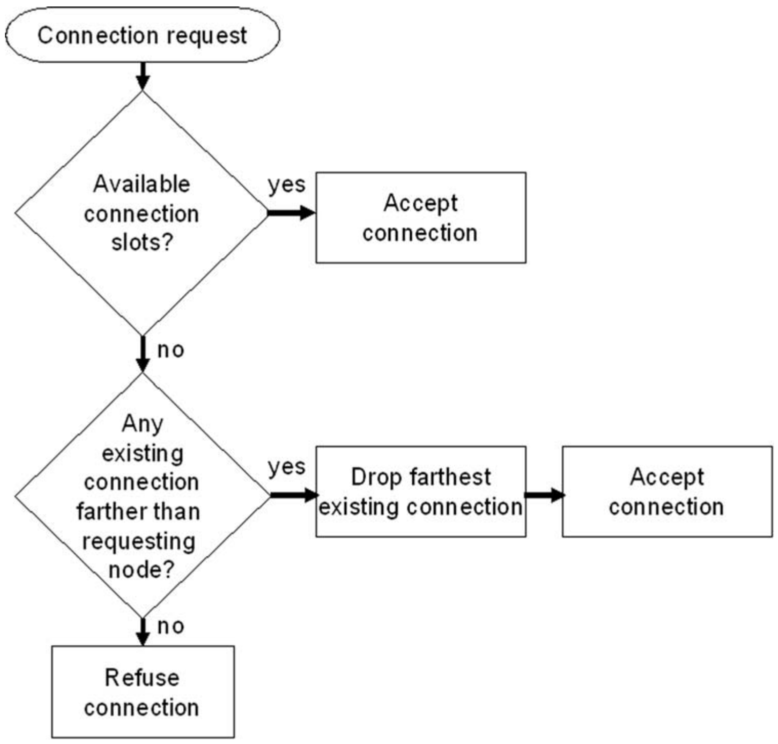

Connection request
Available connection slots?
yes
Accept connection
no
Any existing connection farther than requesting node?
yes
Drop farthest existing connection
Accept connection
no
Refuse connection

Figure 5.4: Connection selection process implemented within *GnutellaApp* as part of the neighbour selection algorithm.

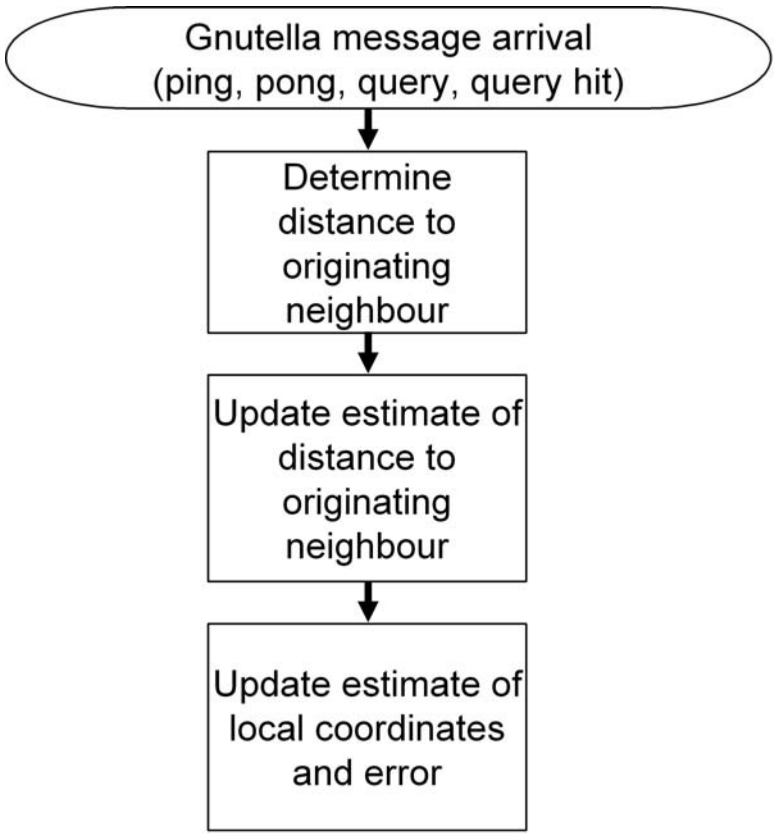

Gnutella message arrival
(ping, pong, query, query hit)
Determine distance to originating neighbour
Update estimate of distance to originating neighbour
Update estimate of local coordinates and error

Figure 5.5: With every received Gnutella message, *GnutellaApp* updates its estimate of the distance to the originating node. It also updates the local coordinates and error estimate.

Ultrapeer

This class is another subclass of *GnutellaApp*, specialized to represent a Gnutella v. 0.6 ultrapeer servent. Ultrapeers have different maximum connection limits for legacy (v. 0.4) peers, ultrapeers, and leaves. Thus, this class contains logic for responding differently to each type of connection request. This is the main difference from the base class.

SmpBootServer

This class models a simple bootstrap server, which provide new nodes in the network with the addresses of a few existing servents. Nodes simply call this class directly instead of sending messages through the network to it. Since bootstrapping dynamics are not of interest in this research project, this is not a concern. The *SmpBoot-Server* implements methods to store peer addresses in its database and to respond to bootstrap requests from new nodes.

ActivityController

The original implementation of the *ActivityController* contained logic to distribute queries according to the UMASS model [45]. Since the comparison of the enhanced Gnutella protocol and the unmodified version required precise control on the timing of queries, this functionality was disabled. The ActivityController still retains code to probabilistically determine whether peers should be online or offline. This behaviour is governed by the parameters of the UMASS model.

5.2.3 Messaging

The messaging classes encapsulate the Gnutella protocol messages exchanged by peers. The message transmission and parsing algorithms were entirely rewritten for this work and consequently, all these classes are new in the Gnutaldi simulator. The reason for redesigning this portion of the simulator is twofold. Firstly, the initial implementation was inefficient. It copied and simulated the transmission of a great deal of superfluous Gnutella fields that were never examined by any nodes in GnutellaSim. While this was quite rigorous, the larger data structures and extensive use of C++ memory copying functions caused the simulation to scale poorly. With the new design, only the fields that are actually used in the simulation are implemented. Secondly, the initial implementation was fairly complex and unmaintainable. The implementation was rationalized by introducing a class hierarchy. Also, the maintainability of the code was improved by handling all the message parsing in a single class instead of throughout the code, as was the case prior to the modifications.

IGnutMsg

This interface declares the methods which all message classes must implement. The message class hierarchy is shown in Fig. 5.6. Declaring the virtual methods as high up the class hierarchy as possible allows the use of generic code in the parser class and elsewhere, which does not need to know what type of message it is dealing with. These pure virtual methods include primitives for writing a message to an*ns-2* packet structure, getting the size of a message in bytes, and retrieving the Vivaldi tuple contained in the message.

GnutConnectMsgBC

This abstract base class is derived from *IGnutMsg*. It is a base for all Gnutella connection messages. *GnutConnectMsgBC* stores the Vivaldi data (coordinates, error, and time sent) for a message. It also contains logic for writing this information to raw*ns-2* packets.

GnutBootcacheUpdateMsg, GnutBootstrapMsg, and GnutBootstrapResMsg

These three classes, derived from *GnutConnectMsgBC*, encapsulate the interaction between nodes and the *SmpBootstrapServer*. *GnutBootcacheUpdateMsg* is used by nodes to inform the server of their presence in the network, so that the server may give their address to new nodes seeking peers with which to connect. *GnutBootstrapMsg* is “sent” (as discussed earlier, the message is not actually sent to the simplified server) to the bootstrap server to request the addresses of peers with available connections. The response, *GnutBootstrapResMsg*, contains a list of such peers.

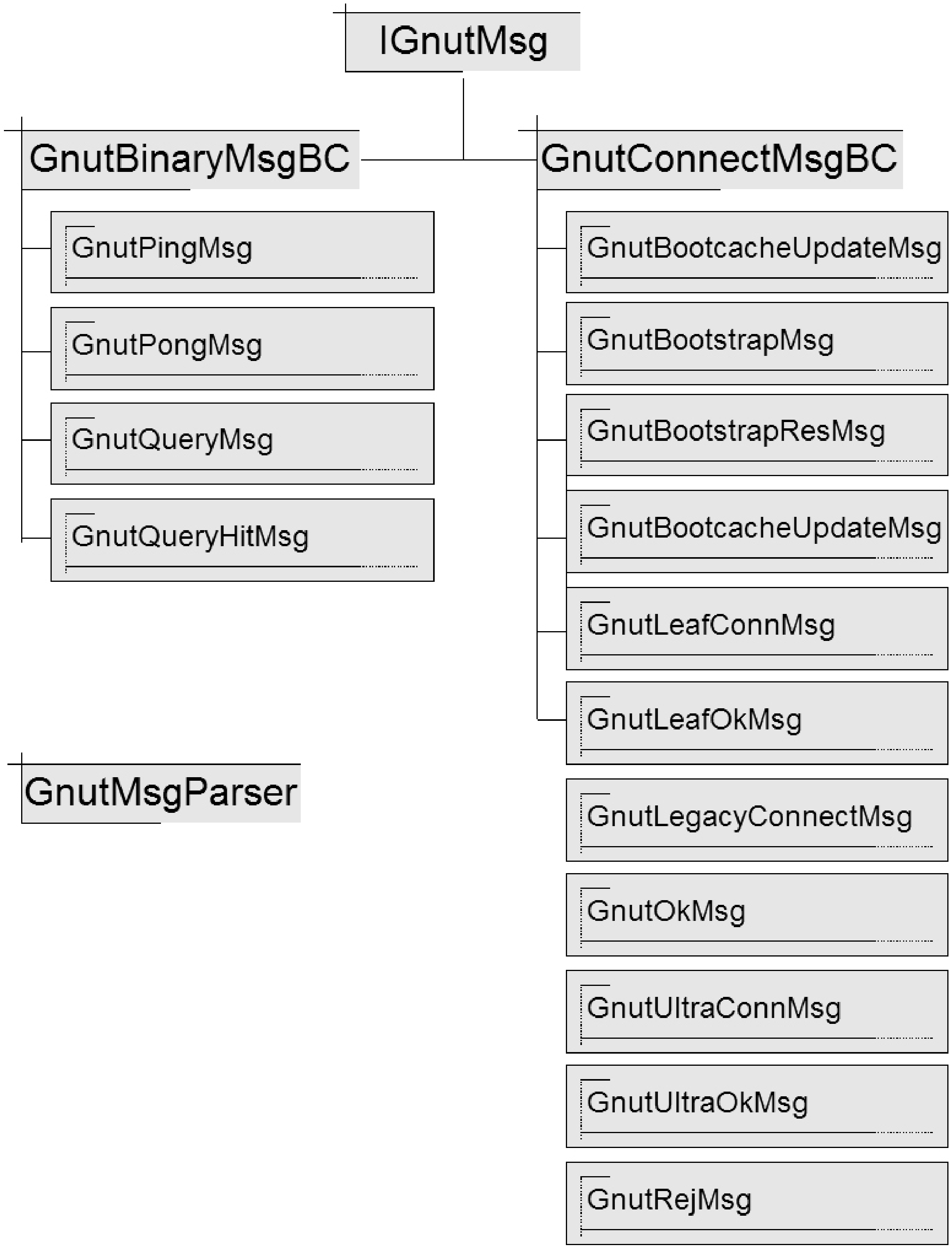


Figure 5.6: Class hierarchy for Gnutaldi message modules.

GnutLeafConnMsg and GnutLeafOkMsg

These two classes, again, derived from *GnutConnectMsgBC*, represent the messages sent by leaf nodes. The *GnutLeafConnMsg* is a request to connect to an ultrapeer. The *GnutLeafOkMsg* is a positive response to an ultrapeer's request for a connection.

GnutLegacyConnectMsg and GnutOkMsg

These message classes, subclasses of *GnutConnectMsgBC*, encapsulate the connection messages sent by a legacy Gnutella 0.4 peer. GnutLegacyConnectMsg is a message requesting a connection with another legacy peer or an ultrapeer. GnutOkMsg signals that a v 0.4 peer is willing to accpet a connection request sent by a legacy peer or an ultrapeer.

GnutUltraConnMsg and GnutUltraOkMsg

These subclasses of *GnutConnectMsgBC* are analogous to GnutLeafConnMsg and GnutLeafOkMsg, but for ultrapeers: they encapsulate requesting and accepting a Gnutella connection.

GnutRejMsg

This subclass of *GnutConnectMsgBC* encapsulates a connection rejection message sent by any Gnutella servent.

GnutBinaryMsgBC

This abstract base class is derived from *IGnutMsg*. It is a base for all Gnutella binary messages. *GnutBinaryMsgBC* stores the Vivaldi data (coordinates, error, and time sent) for a message. In addition to the Vivaldi data, it stores the Gnutella binary message header introduced in Table 2.1. This class contains the logic for writing the Vivaldi and header information to raw*ns-2* packets.

GnutPingMsg

This subclass of *GnutBinaryMsgBC* represents a ping message sent by a Gnutella servent to discover potential neighbours.

GnutPongMsg

This subclass of *GnutBinaryMsgBC* encapsulates a pong, the response to a ping message. It stores the address of the initiating node and implements the logic to write this information to raw*ns-2* packets.

GnutQueryMsg

This class, which is derived from *GnutBinaryMsgBC*, represents a query message used by Gnutella servents to search for content in the network. Normally, it should contain information about the search criteria, but in the simulation, all content is assumed to be identical (i.e., the same file). Thus, it caries no additional data associated with Gnutella. For the simulation, however, these objects are instrumented with the time the query was sent. This is used for gathering statistics about how fast queries sent at a particular time are traversing the network.

GnutQueryHitMsg

This subclass of *GnutBinaryMsgBC* represents a Gnutella query hit. This type of message, sent in response to a query, indicates that a node has the requested content. Normally, a query hit would contain a list of files matching the search criteria, but since all content in the simulation is identical, this is not required. Furthermore, the Gnutaldi simulator do not model the actual download of content, so this information would be superfluous. These query hit message objects are instrumented with the time the query that triggered them was sent. This is done so that the last node receive the query hit — the one that initiated the query — will be able to record statistics about how long it took for the query to be satisfied.

GnutMsgParser

This singleton class is responsible for parsing all the Gnutella messages (subclasses of *IGnutMsg*) throughout the Gnutaldi application. When passed raw data, it segments it into the message into type, header, payload, Vivaldi coordinates, error, and send timestamp. It relies on the services of the message classes to reconstruct IGnutMsg-derived objects that are readily usable by the application code. The message parser

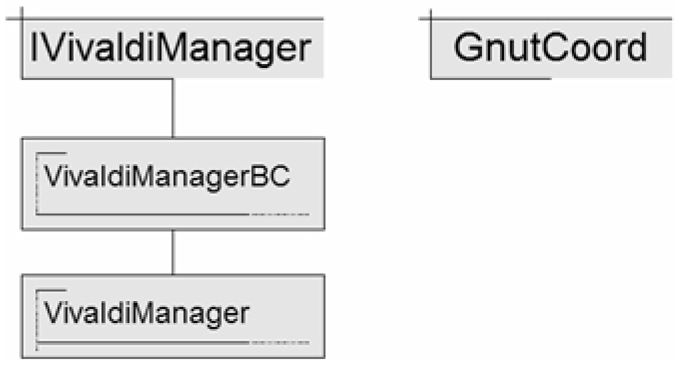


Figure 5.7: Class hierarchy for Gnutaldi's Vivaldi-related modules.

acts as a factory for *ns-2 PacketData* objects, which are the raw packets. It also manages the allocation of message GUIDs.

5.2.4 Vivaldi-Related Classes

This subsection describes the modules related to implementing the Vivaldi algorithm.

IVivaldiManager

This interface is the base class which defines the behaviour of an entity that manages Vivaldi coordinates for a node. It defines only pure virtual methods to update coordinates, reset them to the initial value, retrieve the local node error, and estimate the RTT given a pair of coordinates. The class hierarchy for the Vivaldi classes is shown in Fig. 5.7

VivaldiManagerBC

This class provides a null implementation of the *IVivaldiManager*: it implements all the methods, but they do nothing. This is used when running the normal Gnutella network simulations.

VivaldiManager

This subclass of *IVivaldiManager* fully implements the functionality to maintain coordinates according to the Vivaldi algorithm. An instance of this class is stored in each *GnutellaApp* instance in order to calculate that node's coordinates.

5.2.5 Gnutaldi Platform

The Gnutaldi simulations for this research were executed on a dual-Xeon processor Red Hat Linux workstation. The machine was equipped with 2 gigabytes of RAM. At the time of the simulations, this was considered a relatively high-end platform. It is important to note that *ns-2* is a single-threaded process, so the performance of the simulations was not directly improved by virtue of having two processors. Nevertheless, the fact that processes other than *ns-2* could be served by the additional CPU improved performance somewhat. CPU utilization was at 100% for the duration of the simulations. The scale of the simulations was bounded by the available memory on the system. When too many nodes were included, the system failed to complete the simulation. After all available RAM had been filled, the system entered a *thrashing* state where data was constantly being swapped between RAM and the hard disk cache and progress of the simulation was halted. Because memory requirements increased exponentially with the number of peers sending queries (due to the flooding nature of query propagation), the limit on the number of peers was quickly reached. With 92 nodes, the simulation took several hours. With more than 150 nodes, the simulations never completed due to lack of memory.

Chapter 6

Network Topology Generation

In order to evaluate the performance of the Gnutella network through simulation, it was necessary to generate a synthetic network topology. There are a variety of tools available to accomplish this task, many of which are able to export their information to native*ns-2* scripts. There are also many mathematical models employed by the tools to produce realistic topologies.

A very simple model is the Erdös-Rényi model, which generates a graph $G = (V, E)$ where each of the possible $\|V\| (\|V\| - 1)$ edges has a probability p of appearing in the graph [53]. This has the property of producing a graph that is not necessarily connected. Furthermore, the average degree of the vertices is $(\|V\| - 1) p$.

A more recent effort at generating random topologies is the random graph model introduced by Waxman [54]. With this model, nodes are randomly placed in a plane [55] and connected with the following probability [54]:

$$P(x, y) = \alpha e^{-\frac{d(x,y)}{\beta L}}, \tag{6.1}$$

where $d(x, y)$ is the distance between vertices x and y, L is the distance between the two farthest nodes in the graph, and α and β are model parameters. Increasing α yields more edges, whilst increasing β increases the ratio of long edges to short ones [53].

Since the Internet includes a notion of hierarchy, a refinement to the "flat" models presented thus far was for them to incorporate that characteristic. The *transit-stub* [56] model uses this approach by recognizing that routing domains in the Internet are either transit or stub domains. Transit domains are typically operated by large service providers such as AT&T and Worldcom, and are used to connect the stub domains operated by their customers and to forward traffic between them. Stub domains terminate traffic but do not forward it. More precisely, a domain is a stub if and only if the path connecting two nodes goes through that domain only if either of the nodes is contained in the domain [56]. The following algorithm is used to build transit-stub topologies [56]:

1. Construct a connected random graph using any suitable method. The vertices in this graph represent transit domains.

2. Replace each node in the graph with another connected random graph. This graph represents the backbone of the transit domain.

3. For each node in the transit domains, generate a certain number of random connected graphs. These are the stub domains connected to the node in the transit domain.

4. Add a number of edges between nodes in transit domains and stub domains, and between nodes in different stub domains. The hierarchical topology is now complete.

Because there is strong evidence to suggest that the Internet topology exhibits power-law characteristics [31], Barabási and Albert proposed the model known as Barabási-Albert [32]. This scale-free (power-law) model is based on the premises of incremental growth and preferential attachment. Incremental growth refers to the fact the network did not come online all at once: nodes were added progressively, over time. This is modelled by starting with a small number, m_0, vertices and, at each step, adding a vertex and connecting it to the existing vertices with m edges. Preferential attachment implies that *popular* nodes (i.e., ones with many incident links) get more popular as the network grows. More formally, the probability Π of a new vertex i connecting to vertex j, which is already in the network, is given by [32]:

$$\Pi\left(i,j\right) = \frac{d_j}{\sum_{k \epsilon V} d_k}, \tag{6.2}$$

where d_j is the degree of node j and the overall equation is the ratio of j's links to the sum of all the vertex degrees. The recommended m-value to most closely approximate the router-level topology of the Internet is 2 [55]. This topology represents the core of the network. The routers are connected by links with a latency uniformly distributed between 0 and 4 milliseconds. The technologies deployed in the core of the Internet at this time are connected with high speed links, with a bandwidth of 10 Gb/s being quite common. To be conservative, 100 Mb/s links were employed, although beyond a small threshold, the actual capacity of the links is irrelevant: no TCP window closures nor traffic throttling would be observed even if a slow link speed such as 10 Mb/s were used. Link latency variation, however, is of significance. The intent was to create paths of different lengths (in terms of time) through the network so that the Vivaldi algorithm would have the opportunity to select the shortest one.

Using this core topology as a starting point, the sample network was completed by attaching a random number of nodes (either 1 or 2) to each leaf (nodes with the smallest degree) in the core topology. A similar approach was proposed by the authors of GnutellaSim in their usage notes [44] and the hierarchical transit-stub model [56]. These nodes, newly attached to the leaves, contain the Gnutella servents. They are connected by slower links, distributed according to observed peer bandwidths [43]. (Note that the bandwidth of the nodes is of little importance here). The latency for the peer access links was distributed uniformly between 2 and 6 milliseconds. The resulting network contained 42 Gnutella servents and a total of 92 nodes. Because of the scalability limitations of the *ns-2* simulator, it was not possible to simulate networks with a large number of nodes. Even with a small number of nodes, it was still possible to examine the behaviour of the proposed algorithm. We generated 10 different networks and used these as the basis for the simulations described in Chapter 7.

Chapter 7

Evaluation of the Modified Gnutella Protocol

In this section, simulations using the 92-node network described in Chapter 6 are presented. In Subsection 7.1, a range of tuning parameters for the Vivaldi algorithm is explored. In Subsection 7.2, the observed operation of the neighbour selection algorithm (NSA) is discussed. In Subsection 7.3, the convergence properties of the Vivaldi coordinates in the simulation are reported. Finally, in Subsection 7.4, the performance evaluation results are shown and the extent to which the neighbour selection algorithm improves performance in the Gnutella network is discussed.

7.1 Tuning Parameter Selection

Although the Vivaldi algorithm tuning parameter c_c is recommended to be 0.25 [10], no value is specified for the tuning parameter c_e, which is used to balance the contribution of new samples to the weighted moving average of the local error (3.8). Higher c_e values cause recent samples to be weighted more heavily. In order to determine the c_e value to use in subsequent simulations, the performance of different c_e values was explored for one of the 92-node networks described in Chapter 6. The median relative RTT prediction error, which is an indicator of coordinate accuracy, is shown for values of c_e ranging from 0.01 to 0.75 in Fig. 7.1 – 7.5.

As shown in Fig. 7.1, with $c_e = 0.01$, approximately 1000 seconds are required for the relative error to drop below 10%. The higher c_e values converge much more

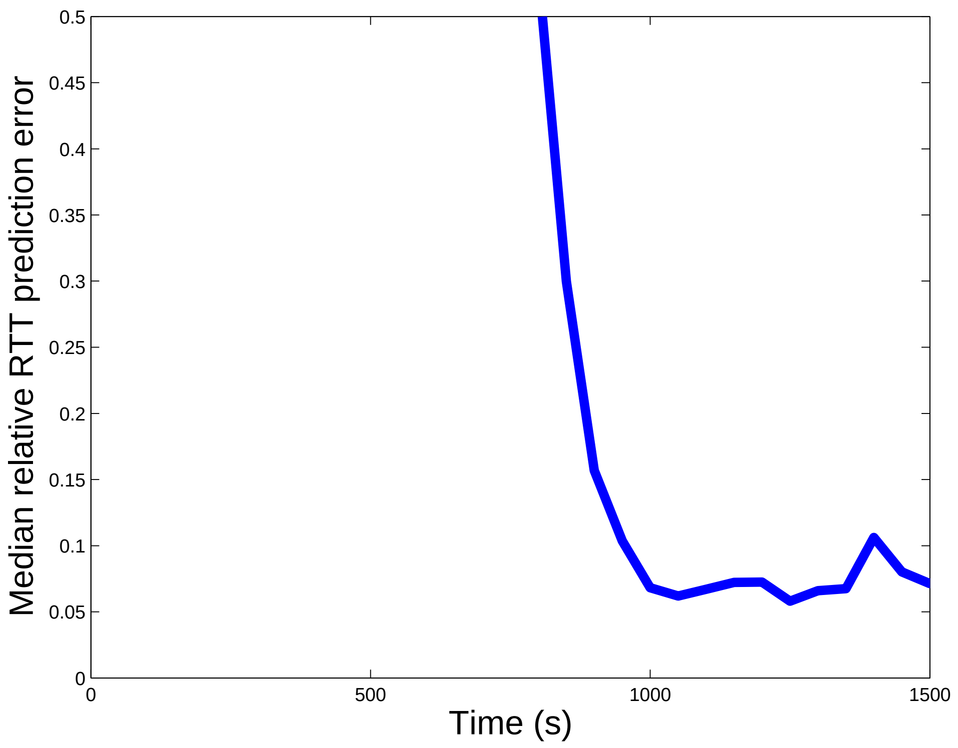


Figure 7.1: Median relative RTT prediction error as a function of time for $c_e = 0.01$.

rapidly: they drop below 10% within 200 – 400 seconds. The steady-state errors for the c_e values shown in Fig. 7.2 – 7.5 are quite similar, but $c_e = 0.10$ (Fig. 7.2) causes less oscillation of the error. It is expected that lower c_e values would lead to less error dramatic variations since the impact of each new sample is smaller (3.8).

The lower the coordinate error, the more likely the coordinates are to accurately predict RTT between nodes and lead to optimal neighbour selection behaviour. Rapid convergence and little oscillation is also desired. For these reasons, with the networks used in this simulation, the most suitable c_e value is 0.10. This value was used in all simulations discussed in this chapter. Networks with distinct properties, in particular a different number of nodes, may require different values for the c_e tuning parameter.

7.2 Neighbour Selection Behaviour

In this subsection, simulation results are examined in order to observe the behaviour of the neighbour selection algorithm.

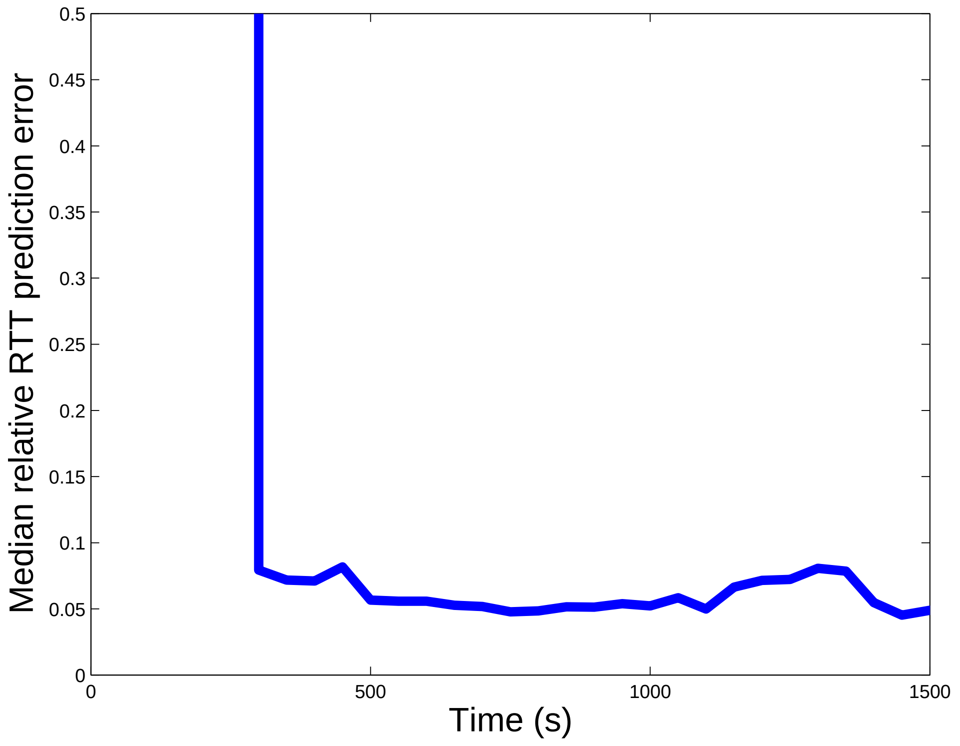


Figure 7.2: Median relative RTT prediction error as a function of time for $c_e = 0.10$.

Table 7.1: Simulation Parameters for a Stable 42-Peer Network

Nodes	92
Gnutella servents	42
Maximum number of connections per servent	8
Minimum node start time	0 s
Maximum node start time	50 s
Probability of going offline after a successful query	0%
Number of nodes with the desired content	12
Proportion of nodes sending queries	25%
Query interval	100 s
Simulation time	1,500 s

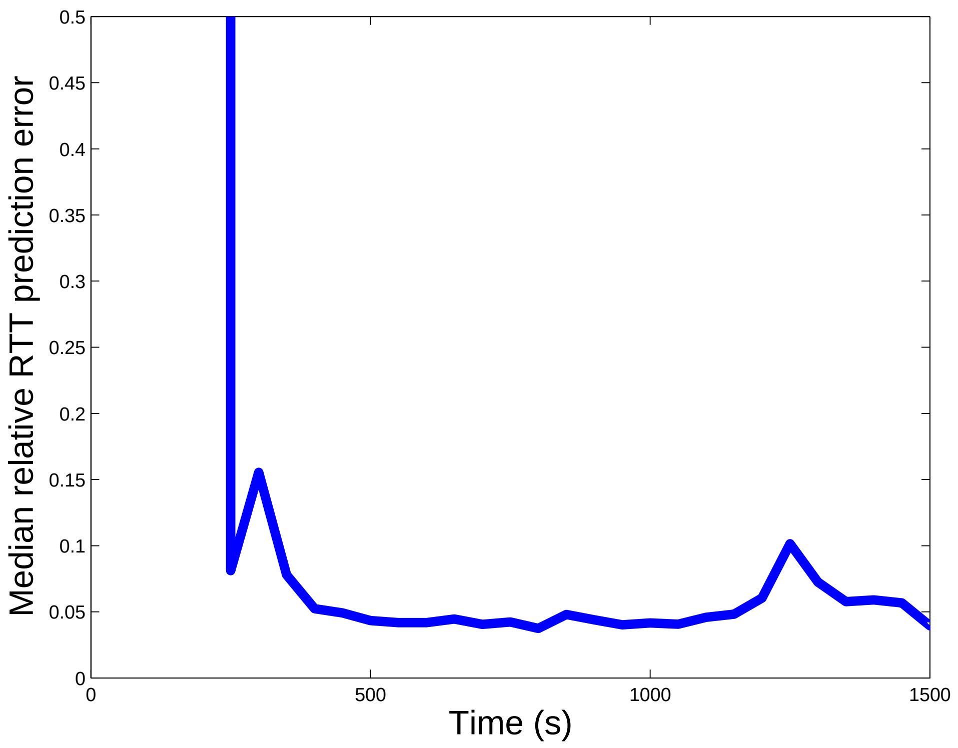


Figure 7.3: Median relative RTT prediction error as a function of time for $c_e = 0.25$.

The first simulation consisted of a 42-servent Gnutella network where nodes join the network and never disconnect. This is an unrealistically stable environment, but it is ideal for observing the raw behaviour of the neighbour selection algorithm. The parameters describing this network, which was generated using the method outlined in Chapter 6, are summarized in Table 7.1. Each dot in Fig. 7.6 marks the time where a Gnutella servent disconnected from one of its existing connections in order to select a peer deemed to be closer, in accordance with the neighbour selection algorithm. Although 10 different physical networks were actually used in the simulation iterations, only the results for a single network are presented in this subsection because many interesting characteristics of the neighbour selection behaviour are masked when the aggregate results are considered. There are no connection drop events early in the simulation: the first occurrence is at instant 72.034588 s. This initial period of calm is expected, because the nodes are still joining the network and have not exhausted all of their available connection. Hence, there is no need for them to drop existing connection in favour of closer nodes. Once the network contains more nodes and their

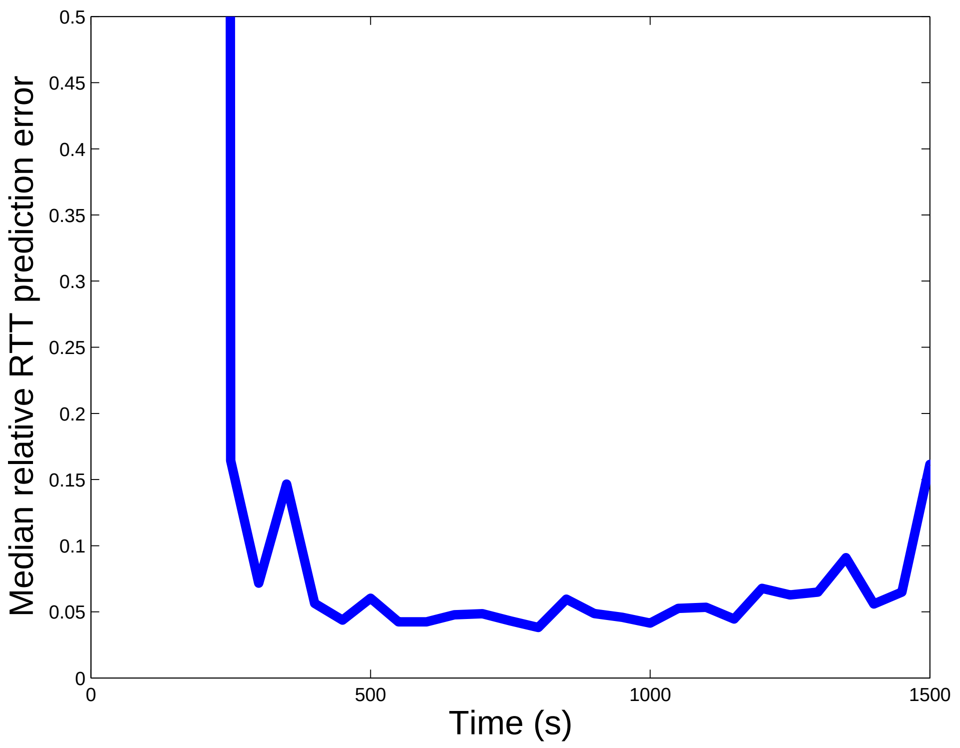


Figure 7.4: Median relative RTT prediction error as a function of time for $c_e = 0.50$.

connections are filled, however, a considerable number of drop events are observed, as shown by the densely packed dots between 100 and 500 seconds. This is the expected pattern, where the network is reorganizing itself in order to more closely match the underlying physical topology. Between instant 530 and 675, no connection drop events are observed. This is followed by a period of moderate connection drop activity ending at instant 899.921195 s. An inspection of the connection dynamics of the network explains this somewhat surprising result. When the statistics were gathered at instant 550 s, all 42 servents in the network had filled their 8 available connections except for a single node. This node was attempting to connect to any other node in the network, but could not find one that was close enough to be willing to drop one of its existing connections until instant 675.551901. At that time, one of the other nodes deemed that the orphan node was closer than at least one of its neighbours, dropped its connection to the most distant of its peers, and accepted the connection request from the orphan node. As a result, the "formerly orphan" node had 7 more connection slots to fill and the newly dropped neighbour had 1 vacant slot. As these

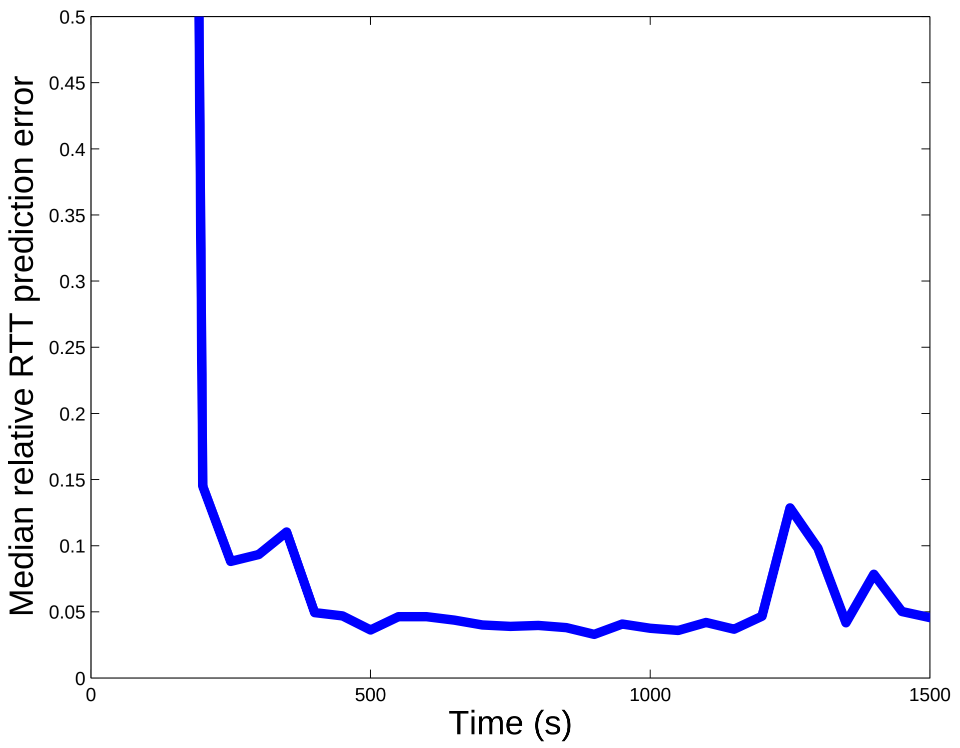


Figure 7.5: Median relative RTT prediction error as a function of time for $c_e = 0.75$.

nodes searched for neighbours, they found peers that were close enough to drop an existing connection, which in turn caused more nodes to search for connections. This *domino effect* caused the period of moderate connection drop activity observed up until instant 899.921195. During this period, a total of 18 connection drop events were observed. The activity was less intense than earlier in the simulation because the network had already reorganized into a form where the neighbour selection was quite optimized: nodes in search of peers willing to drop their connections succeeded less frequently. There were also fewer empty connection slots to fill. Ultimately, from instant 900 onwards, the network remained in a stable state where all nodes except one filled their 8 connection slots and a single node had 6 connections. This convergence to a stable overlay topology is an expected result for a scenario where the peers do not leave the network. After the initial turbulence in the network topology, peers eventually fill their connection slots with neighbours that are closer than any peers still seeking connections. Hence, no peers are willing to drop connections to accept connection requests from peers with available connection slots: the topology

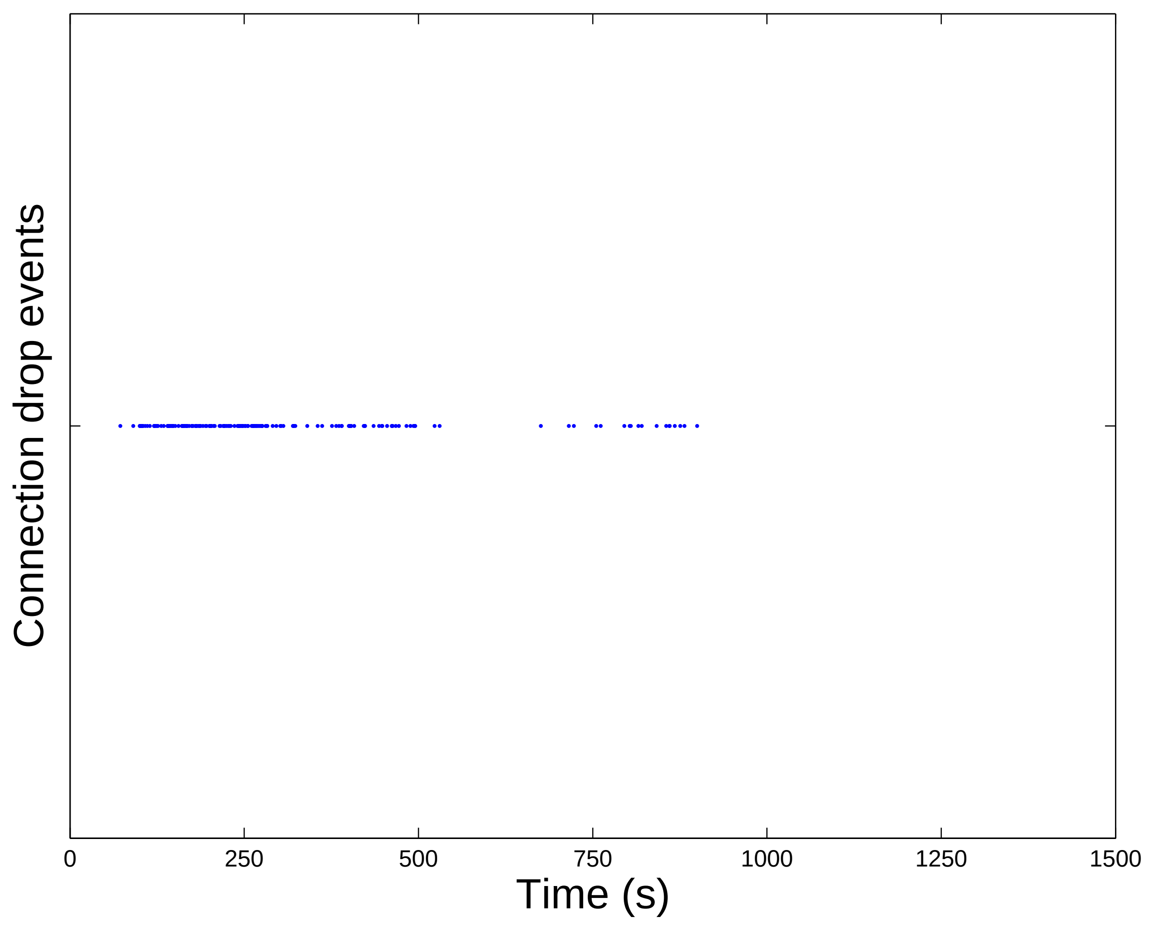


Figure 7.6: Connection drop events for a stable network of 42 Gnutella servents.

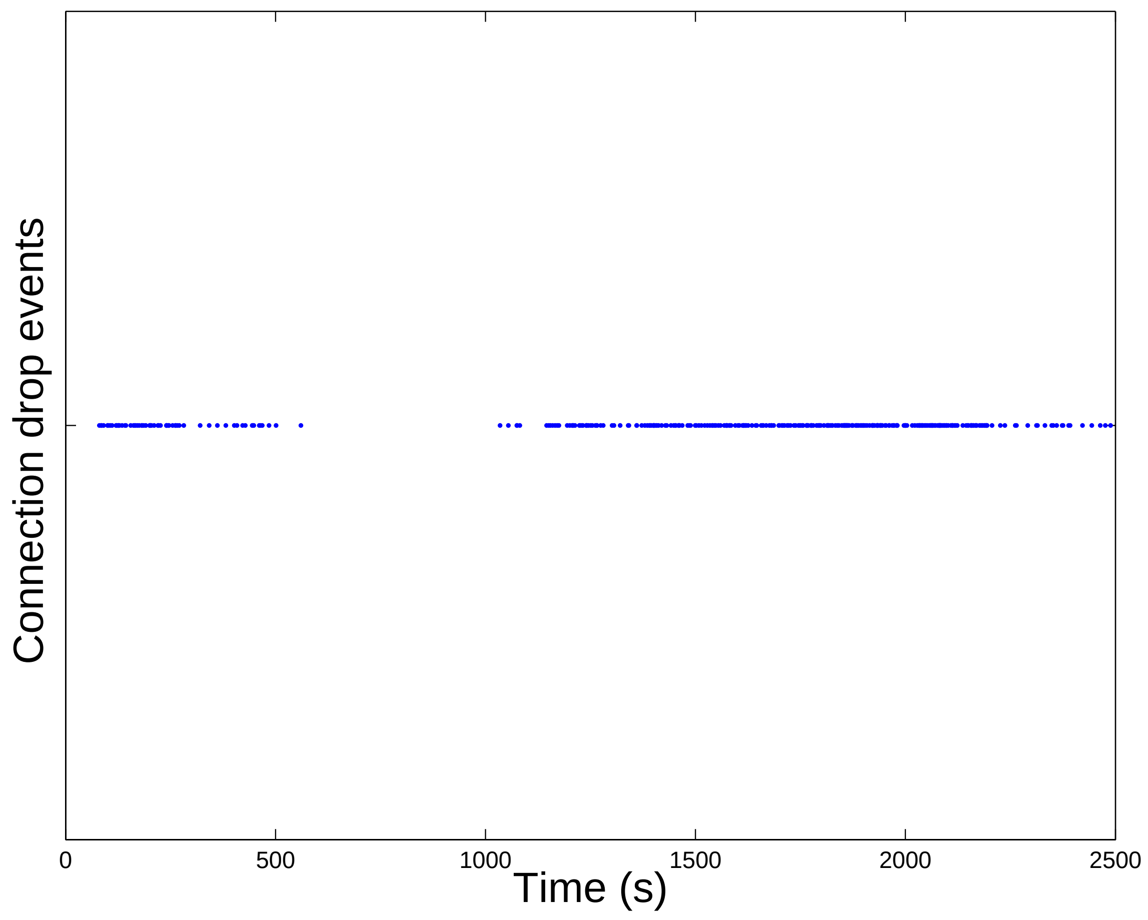


Figure 7.7: Connection drop events for a dynamic network of 42 Gnutella servents.

is frozen. It is not guaranteed that this is *the* optimal state, but it is certainly somewhat optimized, since neighbour relationships are guided by topological information. The extent to which this improves the performance of the network is evaluated in Subsection 7.4.

The simulation parameters for the second scenario are shown in Table 7.2. In this case, the 42-servent topology discussed in Chapter 6 was used as the basis for a more realistic dynamic Gnutella network. Since large P2P networks tend not to spring up all at once [10], a period of stability of 1,000 seconds was imposed at the beginning of the simulation, where nodes will never disconnect after a successful query (UMASS model [45]). Half of the 42 servents are started at the beginning of this period. This is intended to model a *core* topology of servents that have been in the Gnutella network long enough to have somewhat stable Vivaldi coordinates. After

Table 7.2: Simulation Parameters for a Dynamic 42-Peer Network

Nodes	92
Gnutella servents	42
Maximum number of connections per servent	8
Minimum node start time (core overlay)	0 s
Maximum node start time (core overlay)	50 s
Earliest allowed node disconnection time	1,000 s
Minimum new node start time	1,000 s
Maximum new node start time	1,999 s
Probability of going offline after a successful query	10%
Number of nodes with the desired content	12
Proportion of nodes sending queries	25%
Query interval	100 s
Simulation time	2,500 s

1,000 seconds, nodes will disconnect after a successful query (i.e., the receipt of a query hit) with a probability of 10%. This 10% value is meant to be representative of realistic Gnutella node behaviour [11]. Once a node goes offline, it does not rejoin the network. Queries are sent at 100-second intervals throughout the simulation. Between 1,000 and 2,000 seconds of the simulation, at uniformly distributed random time intervals, the remaining 21 servents were introduced into the network. This period is meant to model the normal conditions in a Gnutella network, with nodes joining and leaving. The connection drop events for this scenario are shown in Fig. 7.7. From the beginning of the simulation up until instant 560.539315, there is fairly intense connection drop activity, as the core network of 21 servents reorganizes itself according to the neighbour selection algorithm. This is similar to the previous scenario, except that the network converges to a stable state faster because of the smaller number of nodes. The next connection drop event is at instant 1,034.465494 s, which is after the introduction of dynamic network behaviour. As more and more nodes are introduced into the network between instant 1,000 and 2,000 seconds, and as nodes leave due to successful queries during the same period, connection drop activity (as shown by the dots in Fig. 7.7) increases and continues throughout the period. This continuous reorganization of the network is the expected result of the neighbour selection algorithm in a dynamic, realistic P2P environment. After instant 2,000, no new nodes are introduced into the network, and connection drop activity decreases, as expected. Nodes are still leaving the network after successful queries, some level of drop activity is indeed expected.

In summary, with a stable network, the neighbour selection algorithm causes the overlay to converge to a stable form. With a dynamic Gnutella network, connection drop activity continues throughout the simulation, as the network reorganizes itself to optimize the integration of the new nodes and replace the departing nodes with neighbours that are close.

7.3 Convergence of Vivaldi Coordinates

In this subsection, the convergence properties of the Vivaldi coordinate system implemented in the Gnutella protocol are examined.

The experiments conducted in the seminal Vivaldi publication [10] show that with as few as 8 neighbours, most nodes exhibited a relative RTT estimate error of less than 20%. This result was obtained using a set of 1,740 DNS servers, which are considered to be stable nodes. In order to compare the proposed implementation to this result, the Gnutaldi tool was used to simulate the 92-node networks discussed in Chapter 6, with the simulation parameters shown in Table 7.1.

The median relative RTT prediction error as a function of time for a stable network where nodes do not disconnect from the P2P overlay once they have joined is shown in Fig. 7.8. The neighbour selection algorithm described in Subsection 4.2 is operating on all the Gnutella servents. The Vivaldi coordinates converge with a median relative error of approximately 5% in the steady-state. It is appropriate to consider the median error rather than the arithmetic mean (for example) because the error for new nodes is initialized to an extremely large value. This outlier value would disproportionately affect the evaluation if the mean were used. Even though the neighbour selection algorithm introduces additional instability in the network, as connections to distant nodes are replaced with connections to closer peers, it can be observed from Fig. 7.8 that the Vivaldi coordinates converge adequately. Note that this is not a realistic situation: P2P networks are quite dynamic in nature. It is, nevertheless, a solid basis for comparing the implementation of Vivaldi in Gnutella with the Vivaldi experiments conducted on the stable DNS servers [10]. The fact that the coordinates converge with a smaller error than in the original experiments may be attributed to the greater

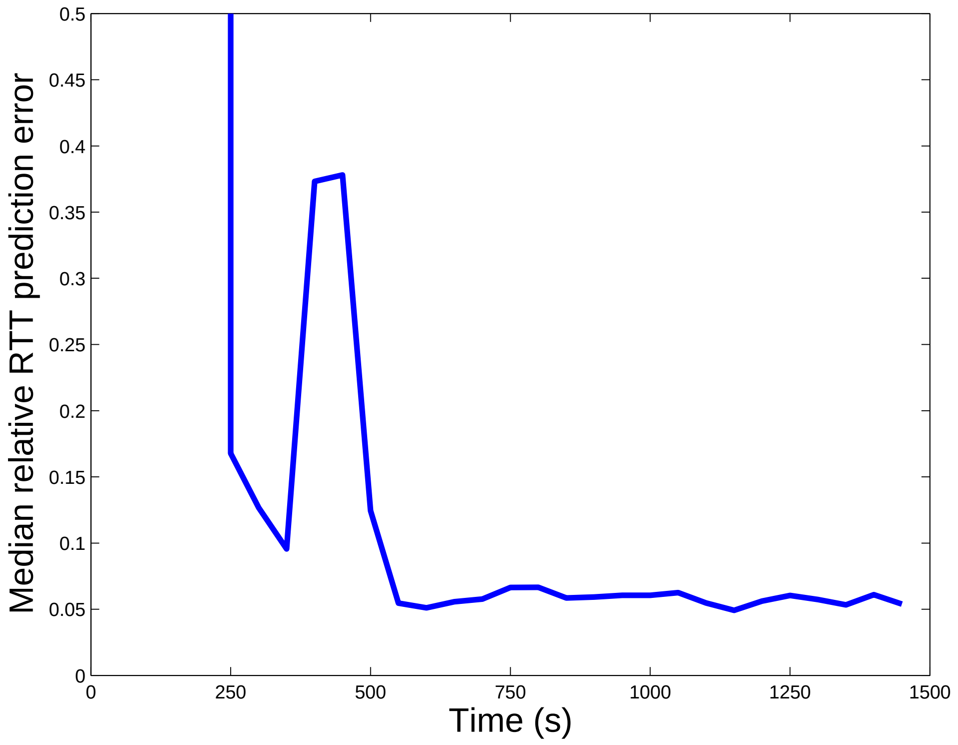
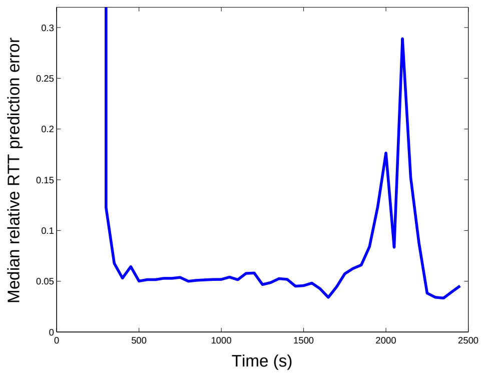


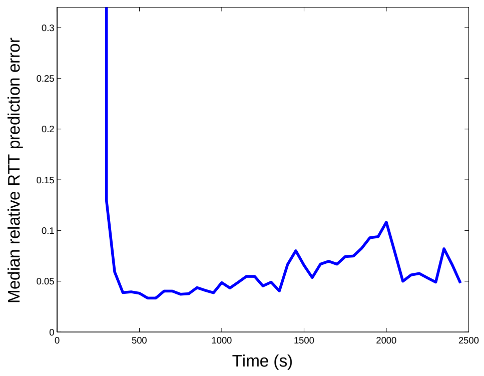
Figure 7.8: Median relative RTT prediction error as a function of time for a 92-node network with 42 stable Gnutella servents running the neighbour selection algorithm. Each node has up to 8 neighbours.

connectivity of the simulated overlay network: the 8 connections allowed for each peer to reach a larger proportion of the network than the equivalent 8 connections would in the DNS experiment with a network containing forty times as many nodes. Spikes such as the one observed near 400 seconds are not unexpected as the coordinates converge. When a node connects to a new node, it obtains information from different regions of the network, and must adjust its coordinates. During this adjustment period, its median RTT prediction error is high. furthermore, as it adjusts its coordinates, the nodes to which it is connected will also have to adjust their coordinates, as the Vivaldi spring system attempts to return to a state of minimal potential energy. The *domino* effect of these coordinate adjustments percolates through the network and the median RTT prediction error may rise quickly, only to fall as the coordinates converge based on the new information from their neighbours.

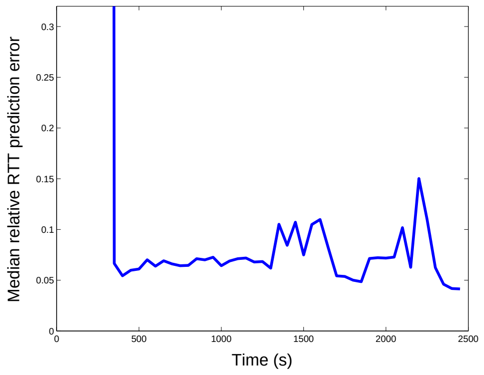
The convergence of the Vivaldi coordinates for the same network, but with peers joining and leaving the network, according to the parameters in Table 7.2 is shown in Fig. 7.9. Each subfigure shows an iteration of the same simulation, differing only by the random seed used to generate the underlying physical network. The random seed also affects which nodes have the desired content, which nodes will send queries, as well as the times nodes join and leave the network. The subfigures show different median relative RTT prediction error values because the networks evolve differently over time as a result of random events. Overall, however, the error tends to vary between 5% and 10%, never exceeding 31% (observed at instant 2,050 s, in Fig. 7.9(c)) in any of the simulations. This is of the same magnitude as the error in the original Vivaldi experiments [10]. The fact that no significant spike or general increase in median relative RTT prediction error is observed immediately after time 1,000 s, when new nodes are introduced into the stable network, indicates that the new nodes' coordinates are converging very quickly. The convergence delay of several hundred seconds observed at the beginning of the simulation is for an entirely new network coming into existence all at once. This is not a realistic situation and is only depicted to show that the coordinates do indeed converge, even in this extreme case. Evidently the coordinates in the simulated dynamic network converge quickly to within a moderate error when a core network of nodes with reliable coordinates exists. This would be the case in the real Gnutella network, where new nodes would encounter nodes that had been in the network for some time and had acquired reliable coordinates: the



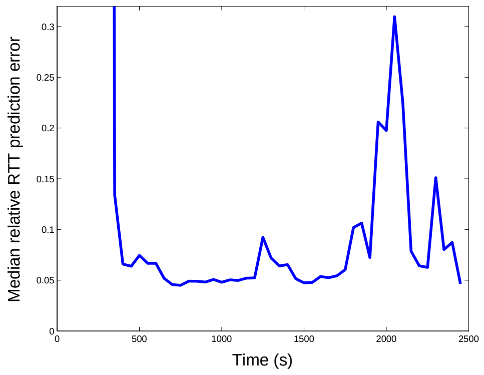
(a) Network 1.



(b) Network 2.



(c) Network 3.



(d) Network 4.

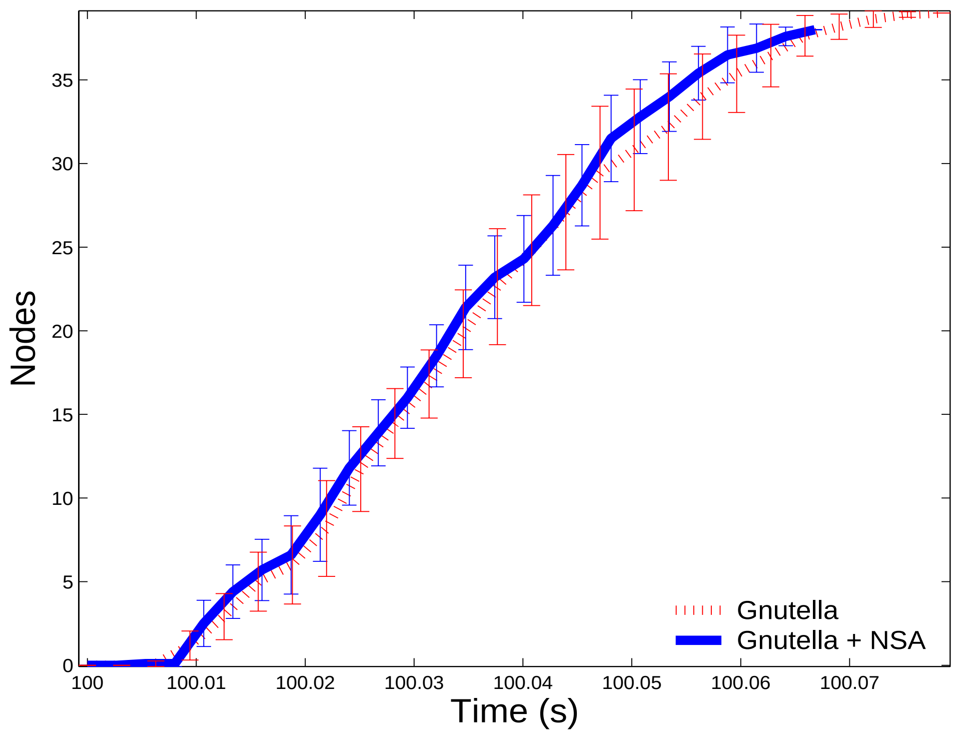
Figure 7.9: Median relative RTT prediction error as a function of time for a 92-node network with 42 Gnutella servents running the neighbour selection algorithm. Each node has up to 8 neighbours. The peers have a 10% chance of leaving the network after a successful query. Each subfigure represents a different iteration of the simulation, operating on a different physical network.

network does not come into existence all at once.

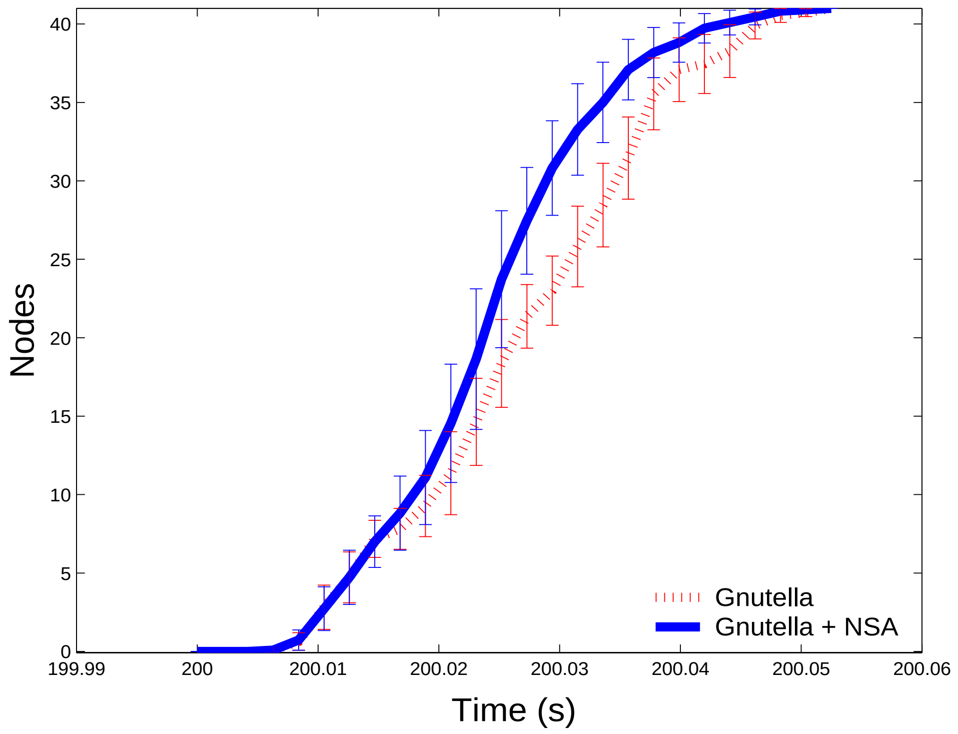
7.4 Performance Evaluation

In this subsection, the actual performance improvements the neighbour selection algorithm brings to the simulated Gnutella network are evaluated. Two metrics are of interest: the average number of distinct nodes reached by queries, as a function of time, and the average number of query hits received by the node that originated the query, as a function of time. These quantities ultimately give an indication of the users' QoE when using the Gnutella network. The more nodes are reached faster, the better. The more query hits received faster, the better. In each of the simulations, one quarter of the Gnutella servents were randomly selected to generate queries at regular 100-second intervals. To ensure a valid comparison between the unmodified Gnutella simulation and the simulation using the neighbor selection algorithm, in both experiments the identical set of nodes was selected to send queries. We observed the average for both QoE metrics for every 100-second flight of queries. All simulations were repeated with 10 different BRITE-generated networks. The results reported represent the mean simulation results within a 95% confidence interval.

The average number of nodes reached by queries sent *early* in the simulation described in Table 7.1 (i.e., a stable 42-servent network) is shown in Fig. 7.10. Early, in this connection, refers to times before the Vivaldi coordinates have converged with an error less than 100%. Since the network is still forming, some nodes have not yet had a chance to find neighbours. In both plots, the performance of the neighbour selection algorithm (labled *Vivaldi* in the legend) closely tracks that of the unmodified Gnutella protocol. Not significant performance improvements were expected early in the simulation because the coordinates provided by the Vivaldi system are not reliable, with a median error in excess of 100%. Hence, the decisions made by the neighbour selection algorithm are ill informed and, as expected, do not cause queries sent early in the simulation to reach more servents faster. This is also reflected in Fig. 7.11, which shows the average number of query hits received for queries sent early in the simulation, as a function of time: query hits are not received significantly faster as a result of employing the neighbour selection algorithm. In summary, no significant performance improvement were observed nor expected in the simulated

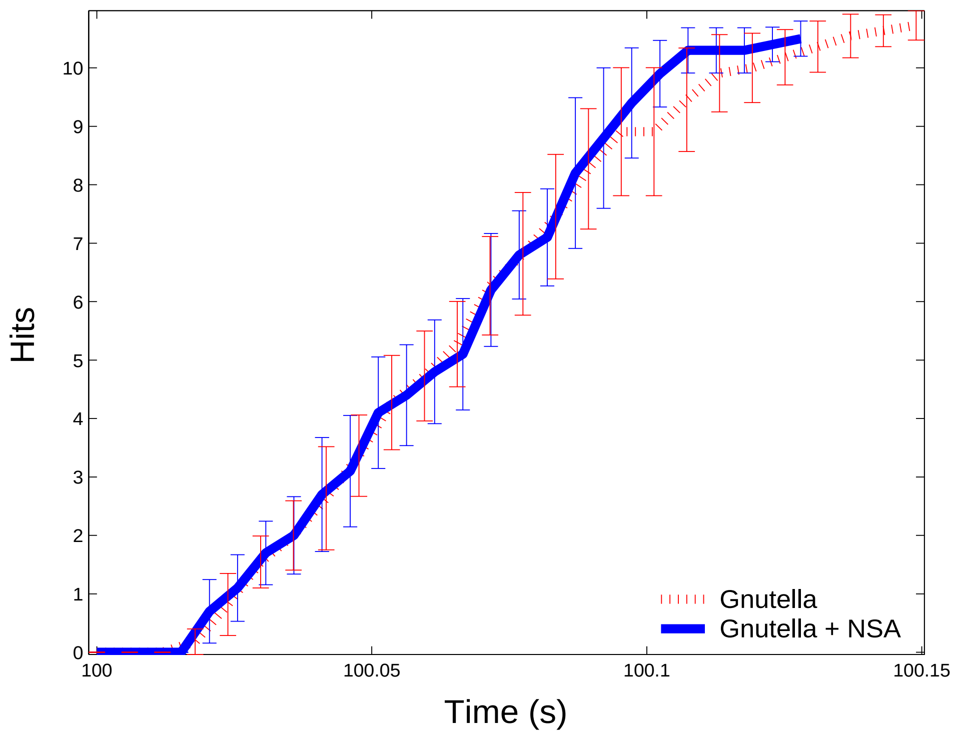


(a) Queries sent at 100 s. Median RTT prediction error = 2,147,740.66

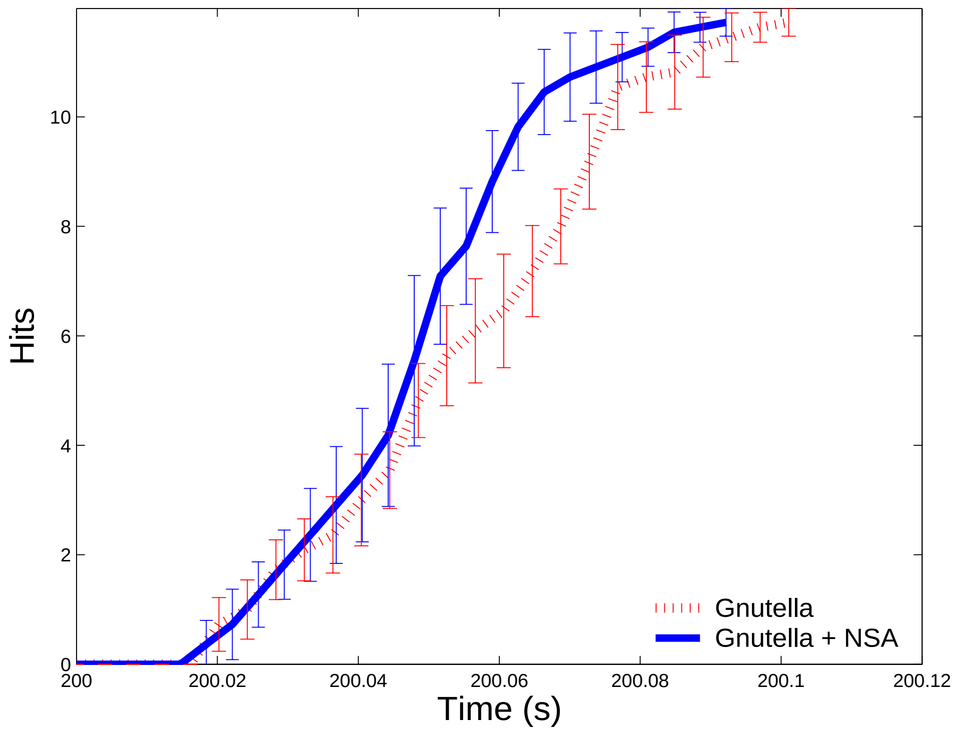


(b) Queries sent at 200 s. Median RTT prediction error = 122.63

Figure 7.10: Average number of nodes reached by queries sent *early* in the simulation with 42 stable Gnutella servents.



(a) Queries sent at 100 s. Median RTT prediction error = 2,147,740.66



(b) Queries sent at 200 s. Median RTT prediction error = 122.63

Figure 7.11: Average number of query hits received for queries sent *early* in the simulation with 42 stable Gnutella servents.

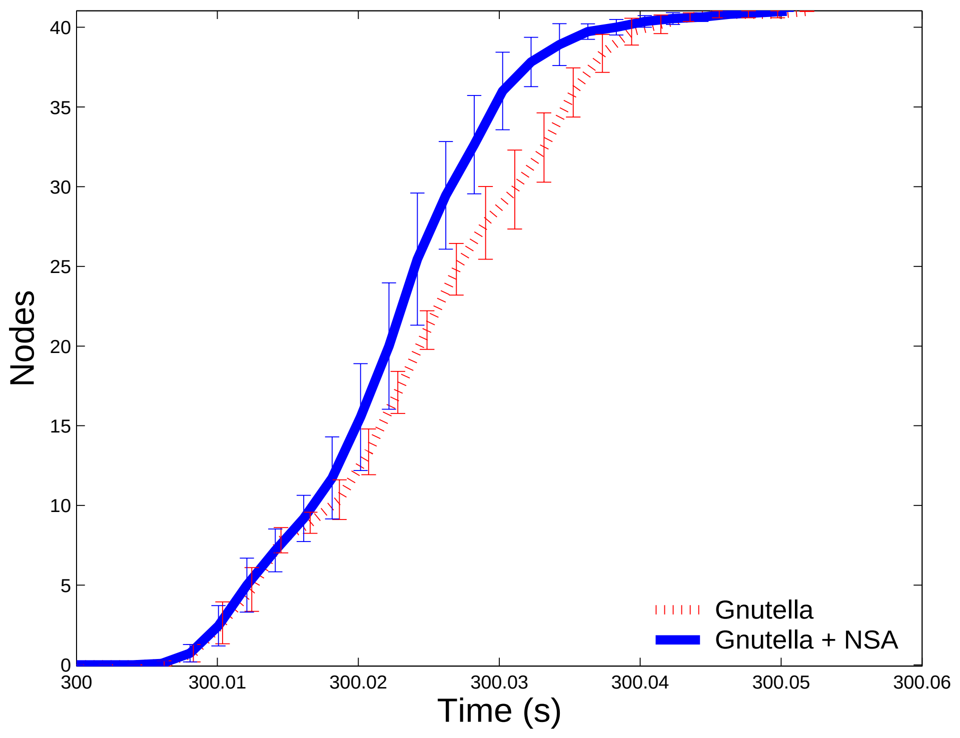
Gnutella network whilst the Vivaldi coordinates were still converging.

As the simulation progresses in time, the Vivaldi coordinates converge to values that predict the RTT between nodes with a lower median error (see Fig. 7.8). At instants 300 and 400, the median RTT prediction error is below 100%, but above 10%. Fig. 7.12 and Fig. 7.13 compare the performance of the unmodified Gnutella protocol to the modified version for this moderate level of coordinate convergence. The modified protocol which includes the neighbour selection algorithm performs better than the unmodified Gnutella protocol. As an example, in Fig. 7.13, it can be observed that it takes about 20 ms longer to receive 8 query hits with the unmodified protocol than with the neighbour selection algorithm.

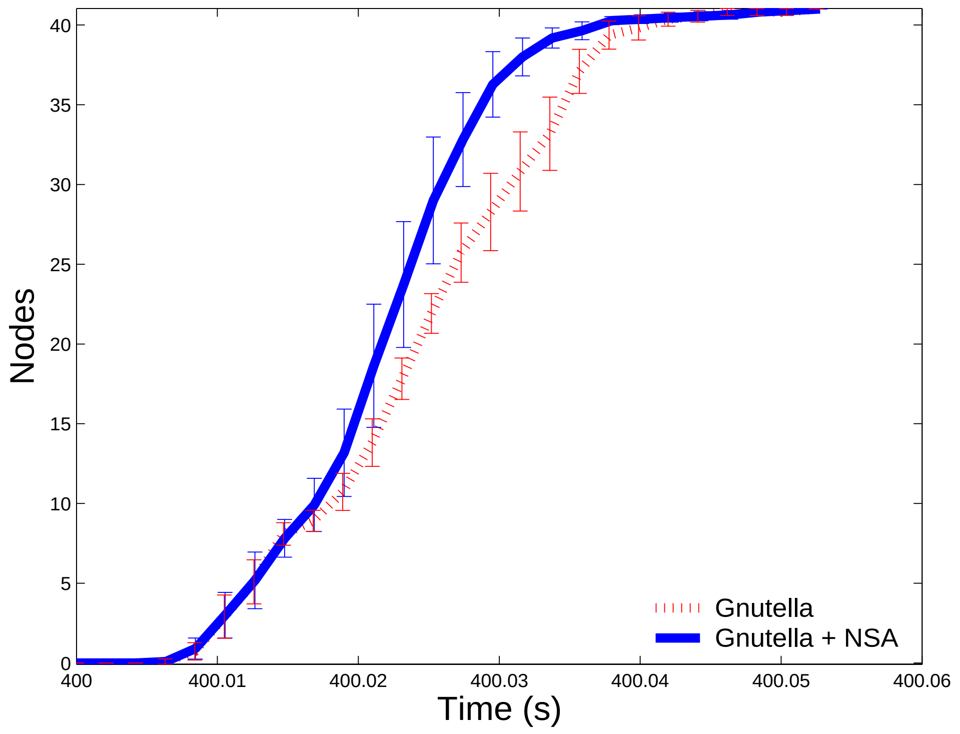
Fig. 7.14 and Fig. 7.15 show the number of nodes reached by queries and the number of query hits received at a time when the network has converged to a higher degree of stability. The instants 700 and 900 which are shown in the plots are in a period when connection drop activity, shown in Fig. 7.6, is relatively low. The neighbour selection algorithm again outperforms the unmodified Gnutella protocol. The performance improvement is of the same order as that observed for instants 300 and 400, when the network was less stable and the Vivaldi coordinates were less reliable. This observation leads to the conclusion that even with a high error of 10% or more, the neighbour selection algorithm can influence the overlay topology enough to enhance performance.

When the network has completely stabilized and no connection drop events are observed, the neighbour selection again outperforms the unmodified Gnutella protocol. Fig. 7.16 and Fig. 7.17 show the average number of nodes reached by queries and the average number of query hits received for queries sent at instant 1,200 s, well after all connection drop activity has ceased. Again, the performance improvements over the unmodified protocol are similar to those observed when the network was still adjusting as a result of the neighbour selection algorithm.

While the above results show that the neighbour selection algorithm performs well in a network where nodes join and never leave, this is known to be an unrealistic situation. It is nevertheless an interesting best case against which to compare. The following results apply for a dynamic, more realistic 42-servent, 92-node network where Gnutella servents' behaviour is governed by the parameters in Table 7.2. These values are intended to approximate realistic peer behaviour [11], to the extent that it

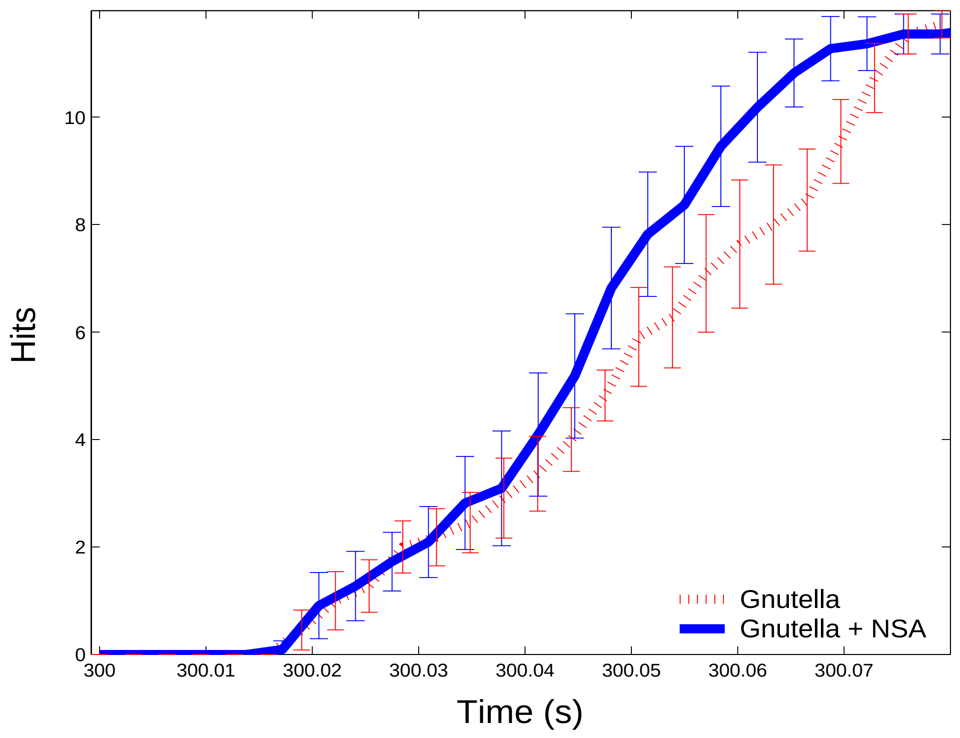


(a) Queries sent at 300 s. Median RTT prediction error = 0.1267

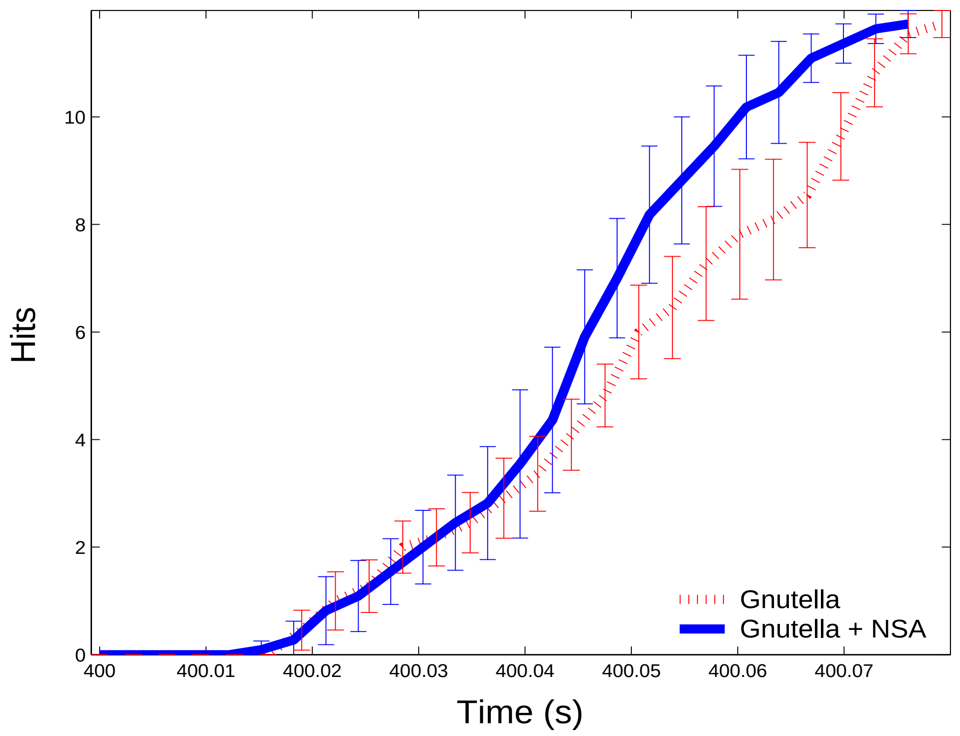


(b) Queries sent at 400 s. Median RTT prediction error = 0.3732

Figure 7.12: Average number of nodes reached by queries sent with 42 stable Gnutella servents when the median RTT prediction error is above 10%.

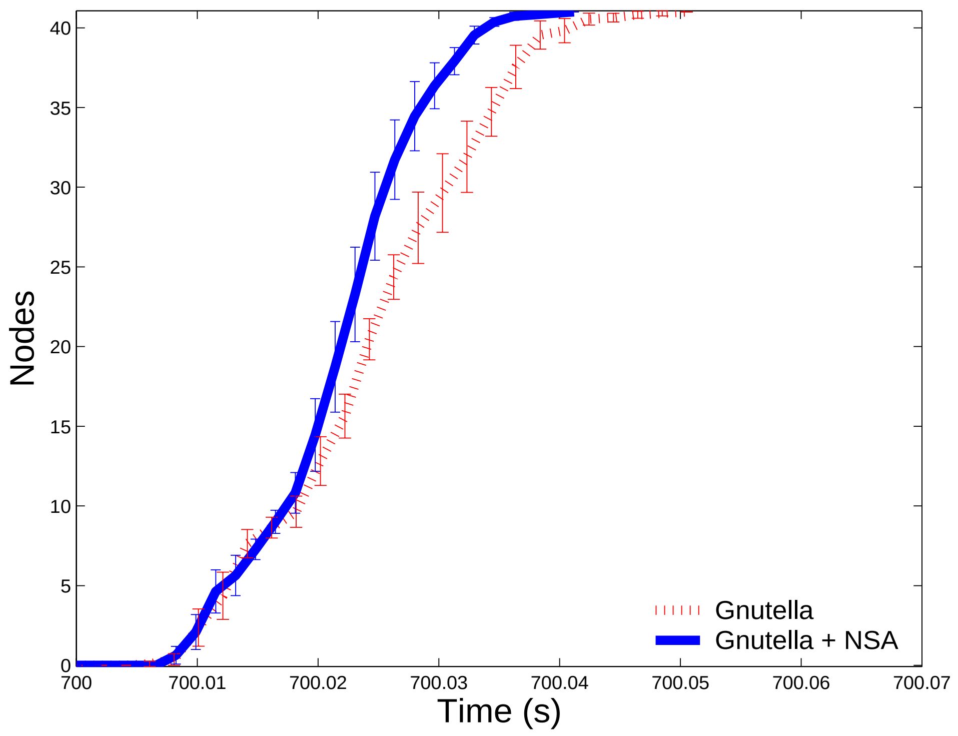


(a) Queries sent at 300 s. Median RTT prediction error = 0.1267

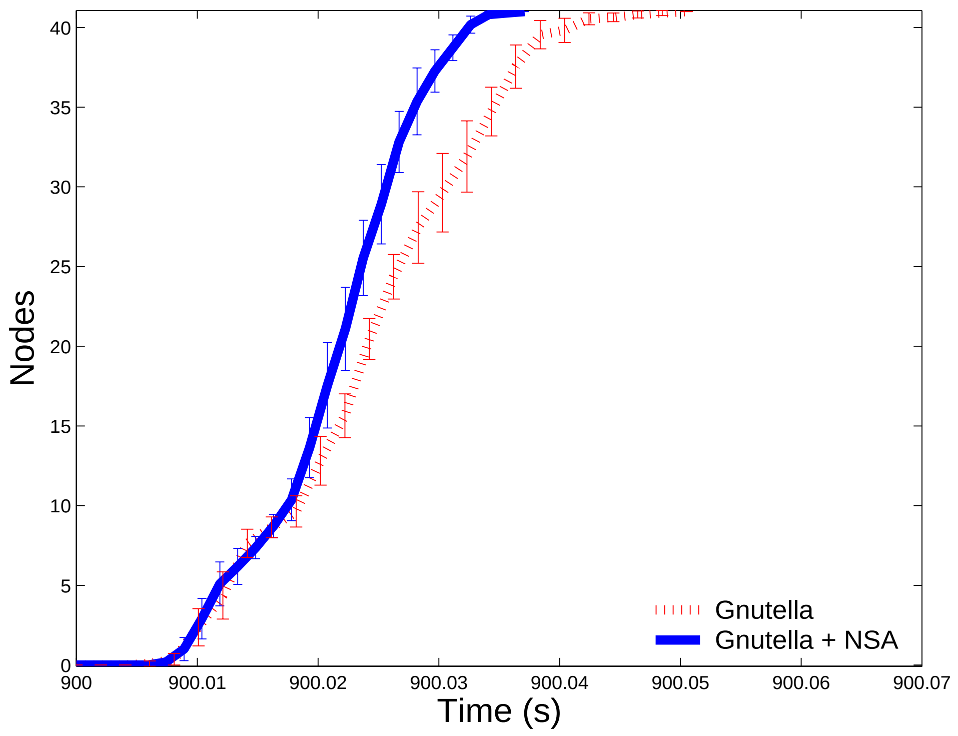


(b) Queries sent at 400 s. Median RTT prediction error = 0.3732

Figure 7.13: Average number of query hits received for queries sent with 42 stable Gnutella servents when the median RTT prediction error is above 10%.

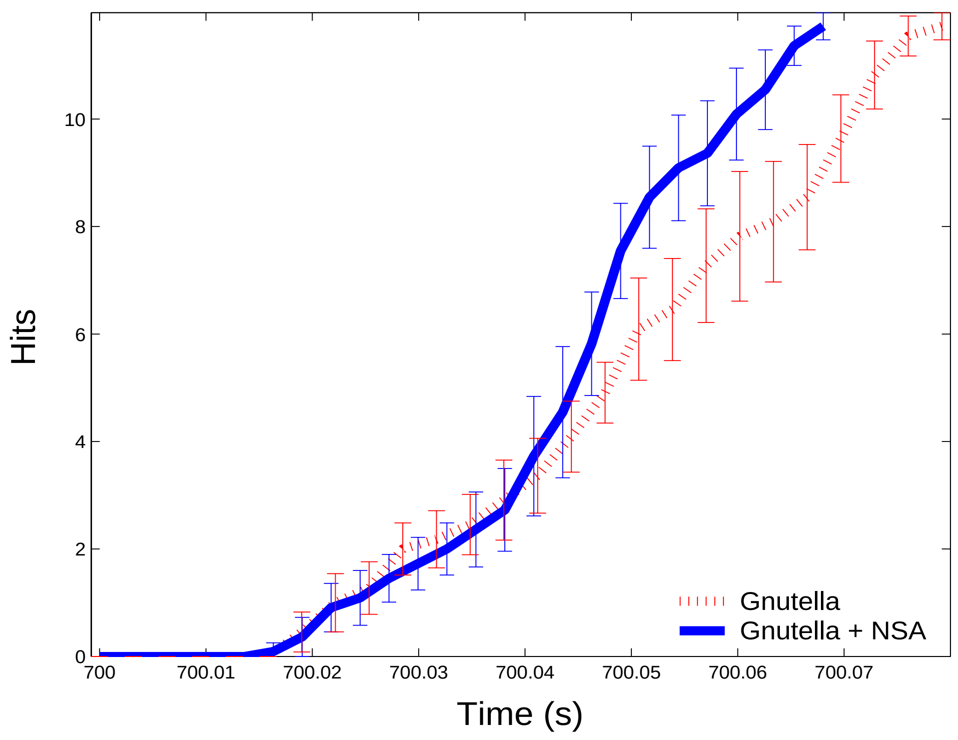


(a) Queries sent at 700 s. Median RTT prediction error = 0.057736

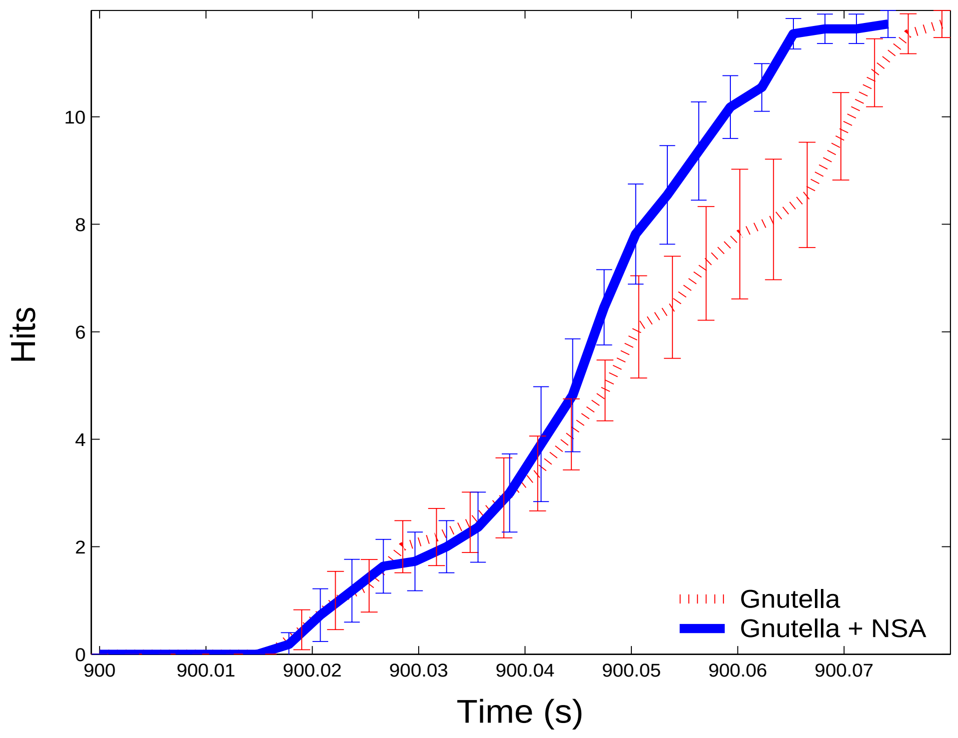


(b) Queries sent at 900 s. Median RTT prediction error = 0.059233

Figure 7.14: Average number of nodes reached by queries sent with 42 stable Gnutella servents when the network has reached a higher degree of stability.



(a) Queries sent at 700 s. Median RTT prediction error = 0.057736



(b) Queries sent at 900 s. Median RTT prediction error = 0.059233

Figure 7.15: Average number of query hits received for queries sent with 42 stable Gnutella servents when when the network has reached a higher degree of stability.

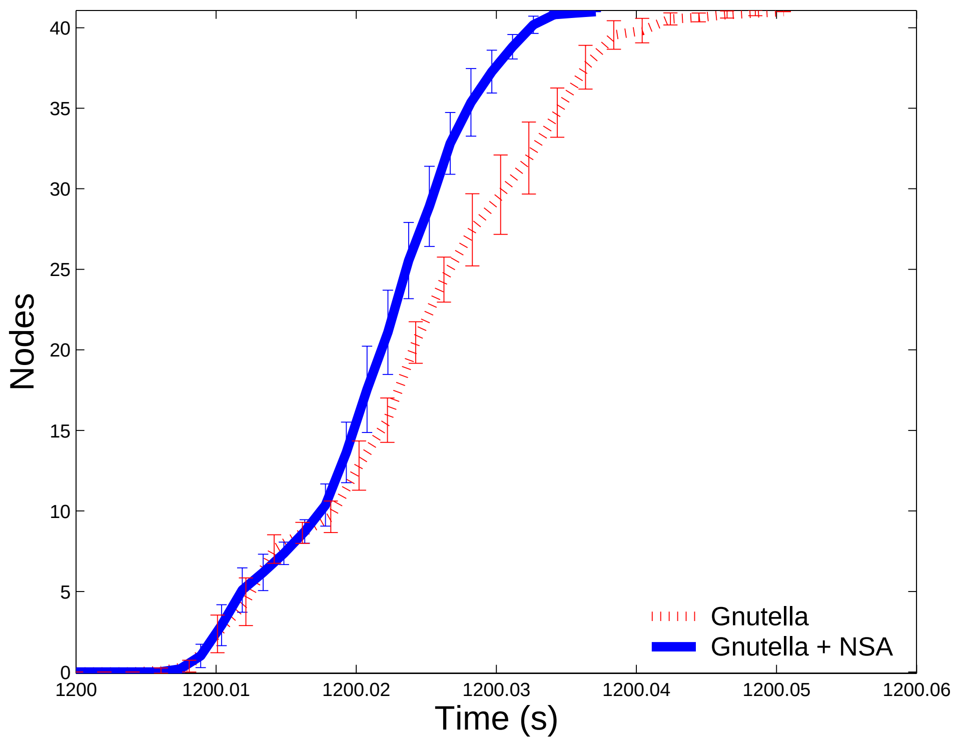


Figure 7.16: Average number of nodes reached by queries sent with 42 stable Gnutella servents when the network has completely stabilized.

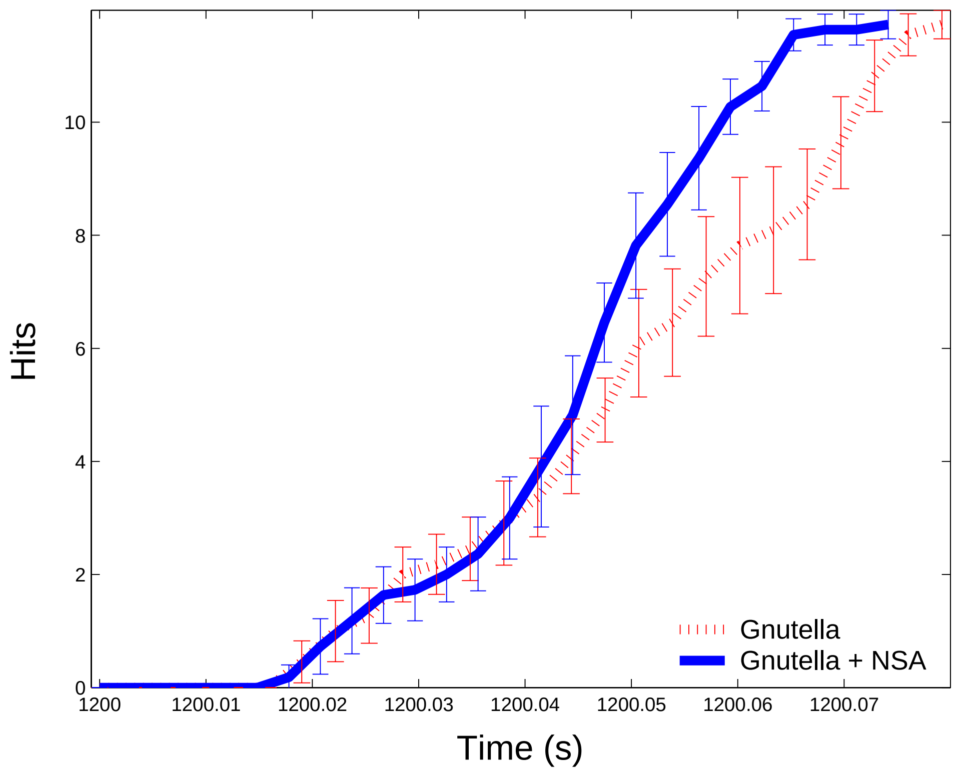


Figure 7.17: Average number of query hits received for queries sent with 42 stable Gnutella servents when the network has completely stabilized.

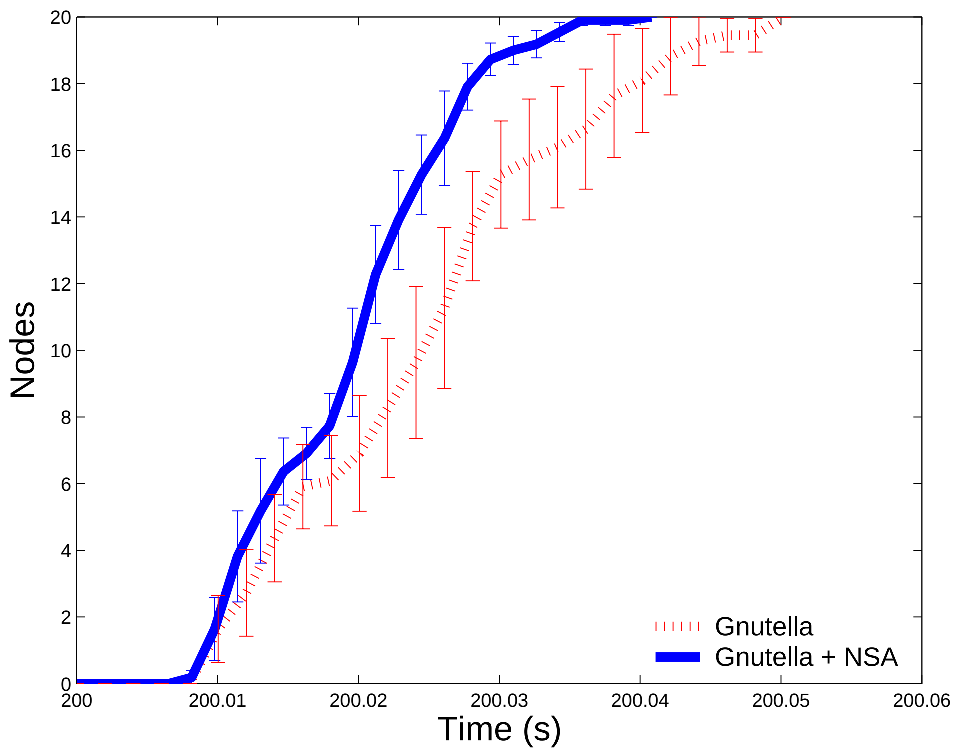


Figure 7.18: Average number of nodes reached for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 100%. At this stage in the simulation, only 21 servents are online. The median relative RTT prediction error at instant 200, when these queries originated, is 497,760.252921.

is possible to capture it in the small simulated network.

Fig. 7.18 and Fig. 7.19 compare the performance of the neighbour selection algorithm and the unmodified Gnutella protocol for queries sent at instant 200 s. They respectively plot the average number of unique nodes reached by queries and the average number of query hits received as a function of time. At instant 200 s, all nodes in the network are still very new, and their coordinates have not yet converged to values that reliably predict the RTT between servents. The median relative RTT prediction error at this time is 497,760.252921. Hence, no significant performance improvement were expected due to the neighbour selection algorithm, because any decisions made by peers based on their coordinates would be ill informed. However, as seen in both figures, the neighbour selection algorithm does indeed outperform the unmodified Gnutella protocol. Results for two additional simulation runs, with random seeds of 12 and 17, are shown and confirm the result. Closer scrutiny of the connection

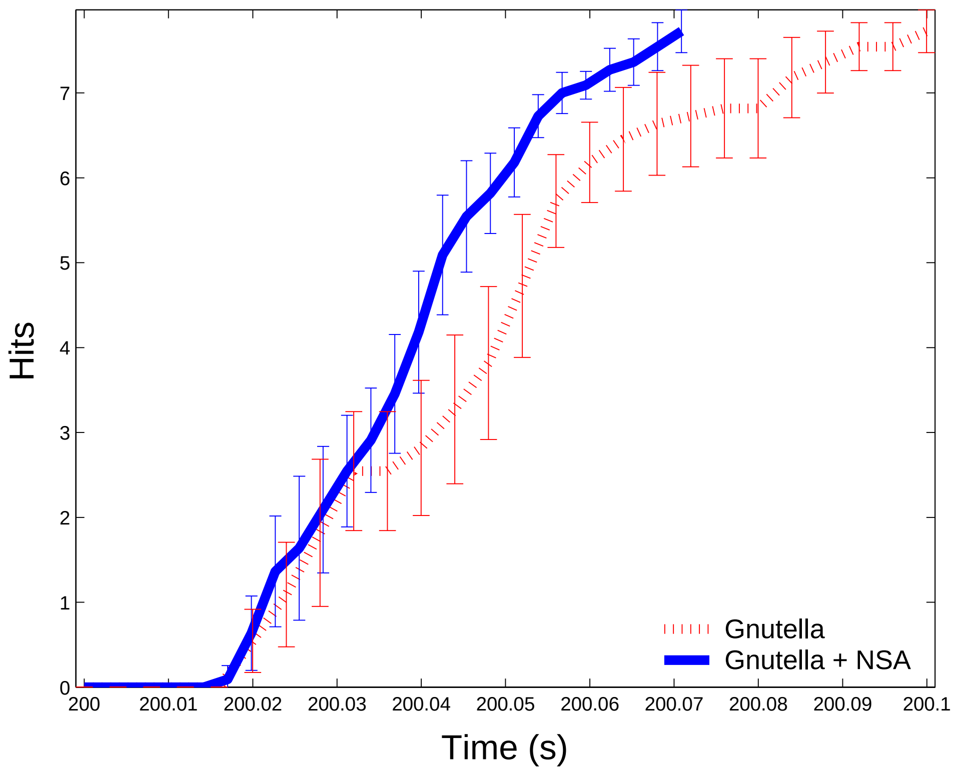


Figure 7.19: Average number of query hits received for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 100%. At this stage in the simulation, only 21 servents are online. The median relative RTT prediction error at instant 200 s, when these queries originated, is 497,760.252921.

Table 7.3: Connectivity at 200 Seconds

Random Seed	Average Connections per Servent		Standard Deviation	
	Gnutella	Gnutella + NSA	Gnutella	Gnutella + NSA
7	6.5714	6.8571	2.2265	1.8784
12	7.0476	7.5238	1.3593	0.8136
17	6.7619	7.3333	1.8949	0.7303

dynamics in the simulations reveals that the neighbour selection algorithm has the effect increasing connectivity in the network. Table 7.3 shows the average number of connections per servent at instant 200 s in the three simulation runs. It also captures the standard deviation on the number of connections. The results show that the neighbour selection algorithm (Vivaldi column in the table) increases connectivity in the network. This is not surprising, since the algorithmic modification to Gnutella give new nodes a greater chance of finding a neighbour to connect to, since nodes with no available connections will consider the possibility of dropping existing connections to accept a request from a closer node. Although this connection success for the new node leads to a concomitant connection drop for another node, it may increase overall connectivity since the dropped node, by virtue of having been in the network longer, may have learned of many other nodes to which it may easily connect. This would not have been the case for the new node. Moreover, it can be observed from Table 7.3 that the standard deviation of the number of connections is much smaller with the modified Gnutella protocol. This is again because new nodes may "steal" connections from existing ones with many neighbours, thus leading to a tighter distribution of the number of connections. Both the greater connectivity and the fact that there are fewer outlier nodes – in particular, fewer bottleneck nodes with a very low number of connections – lead to the improved performance observed at 200 seconds, even though the median relative RTT prediction error is very high.

Fig. 7.20 and Fig. 7.21 compare the performance of the unmodified Gnutella protocol and the neighbour selection algorithm according to the chosen two metrics: average number of nodes reached and average number of query hits received as a function of time. In both cases the neighbour selection algorithm outperforms the unmodified Gnutella protocol: queries traverse the network faster and query hits are received faster. The neighbour selection algorithm is yielding benefits event though the median relative RTT prediction error is above 12%. These results are similar to those reported for the stable 42-node network described in Table 7.1, which is as expected, since before 1,000 seconds, nodes are forbidden to leave the network because this period is used to build a stable Gnutella *core*.

The times which most closely resemble real-world network conditions are those between 1,000 and 2,000 seconds, in the dynamic network. In this period, nodes are

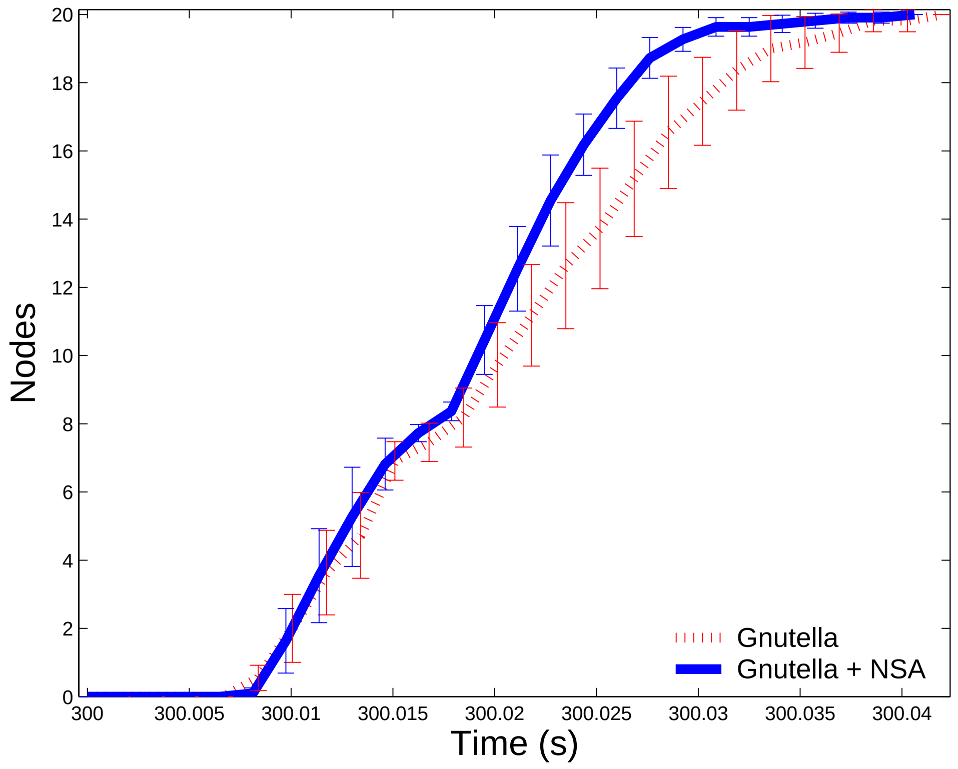


Figure 7.20: Average number of nodes reached for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 10%. At this stage in the simulation, only 21 servents are online. The median relative RTT prediction error at instant 300, when these queries originated, is 0.122698.

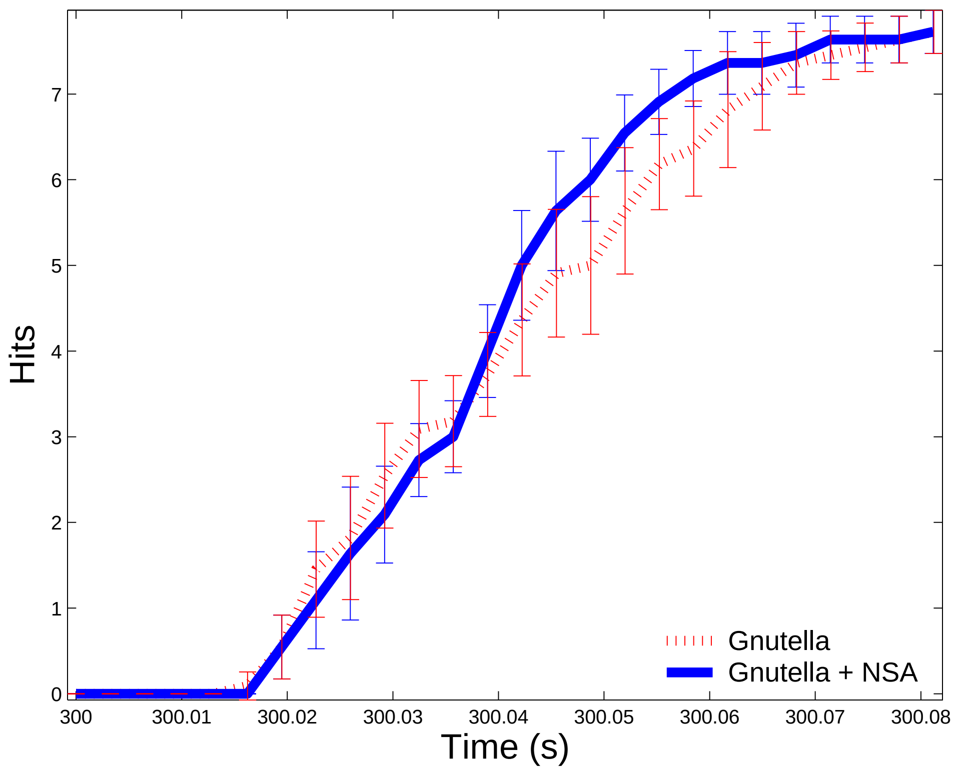


Figure 7.21: Average number of query hits received for queries sent with 42 dynamic Gnutella servents when the median relative RTT prediction error is above 10%. At this stage in the simulation, only 21 servents are online. The median relative RTT prediction error at instant 300 s, when these queries originated, is 0.122698.

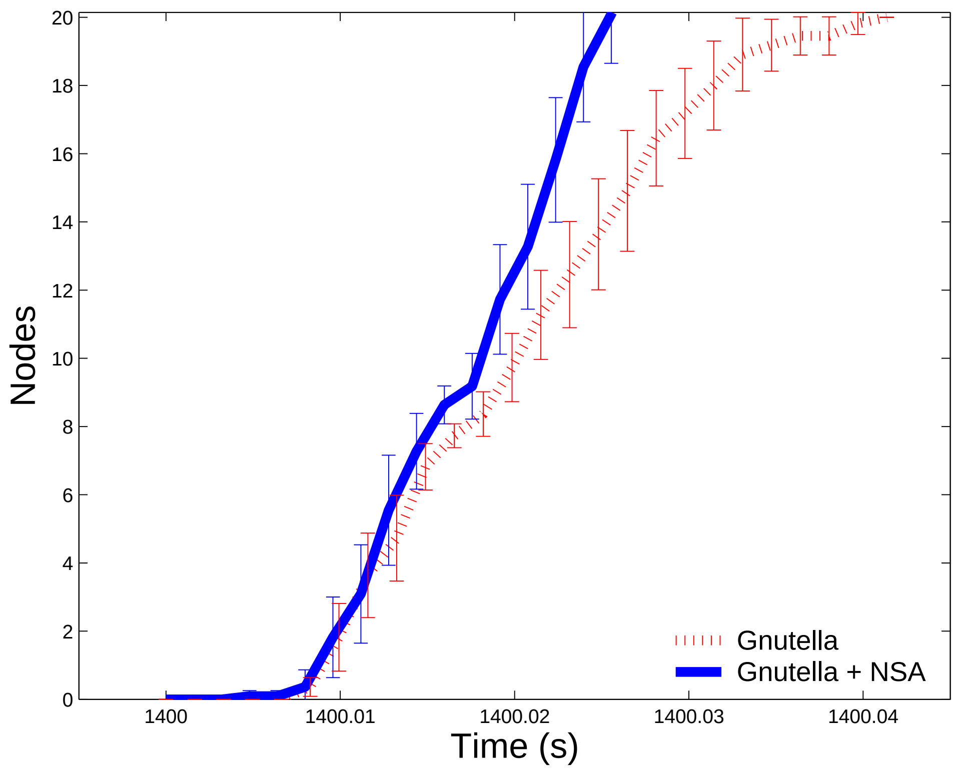


Figure 7.22: Average number of nodes reached for queries sent with 42 dynamic Gnutella servents at instant 1,400 s. At this time, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.051884.

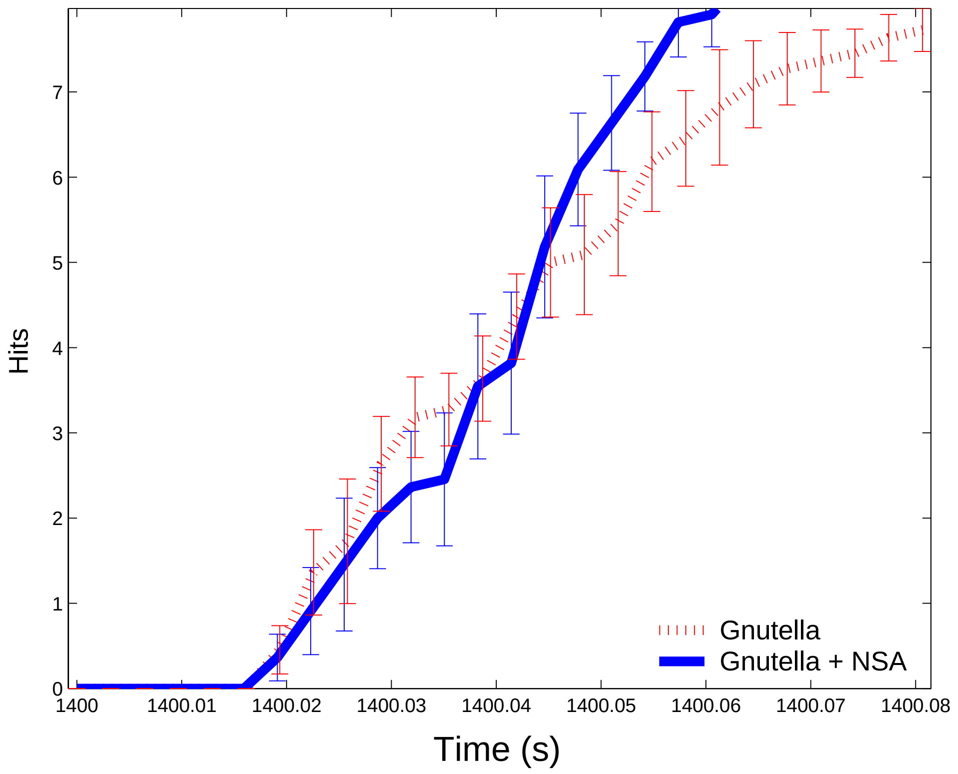


Figure 7.23: Average number of query hits received for queries sent with 42 dynamic Gnutella servents at instant 1,400 s. At this time, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.051884.

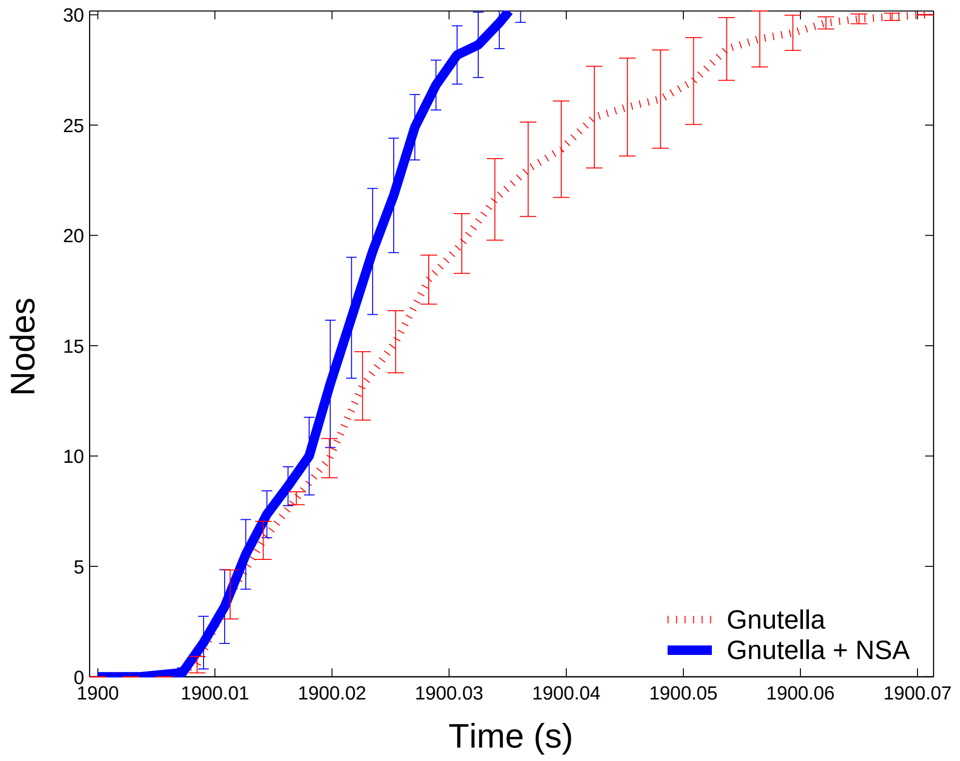


Figure 7.24: Average number of nodes reached for queries sent with 42 dynamic Gnutella servents at instant 1,900 s. At this time, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.084133.

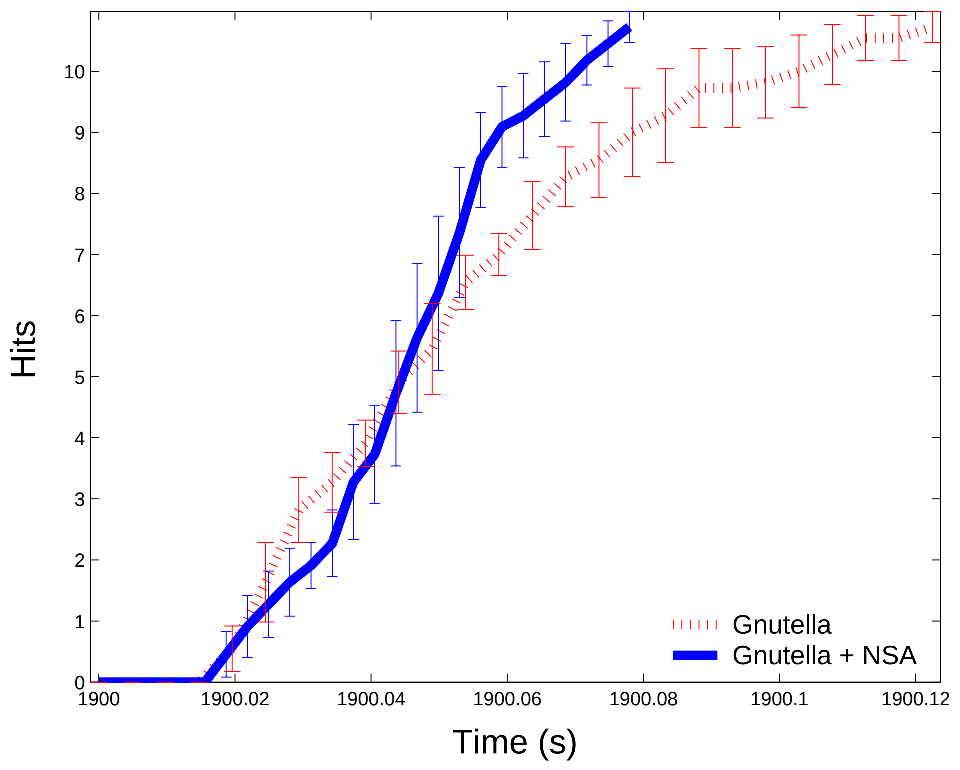


Figure 7.25: Average number of query hits received for queries sent with 42 dynamic Gnutella servents at instant 1,900 s. At this time, the network most closely resembles real-world network conditions. The median relative RTT prediction error is 0.084133.

joining the network at random intervals, and leaving according to the UMASS behavioural model [45]. Fig. 7.22, 7.23, 7.24, and 7.25 show the results for instants 1,400 and 1,900. Again, it can be seen that the neighbour selection algorithm outperforms the normal Gnutella protocol. The results in Fig. 7.23 are slightly equivocal, since the first query hits are received faster without the modifications. Ultimately, however, the rest of the query hits are received faster with the neighbour selection algorithm.

Using the neighbour selection algorithm in the simulations, the performance improvements are observed to be of the order of tens of milliseconds. The originating nodes at instant 1,900 s receive the last query hit on average about 10 milliseconds earlier with the neighbour selection algorithm. It would appear that this is a very small difference, one which would make no difference to the users' QoE. Is it really worth the extra overhead involved in using the neighbour selection algorithm to achieve this meagre improvement? In order to answer, it is important to consider the numbers in context. The simulations involve a network with only 92 nodes. The actual Internet has millions of nodes, and Gnutella has of the order of 1 million participating hosts. End-to-end latencies in the Internet can be much greater than those simulated: they can reach hundreds of milliseconds or more, especially across AS boundaries and intercontinentally. The potential for performance improvement based on locality is much greater than in the simulation. The fact that a consistent performance improvement was observed at all with only 42 servents proves the concept that performance gains can be achieved by constructing an overlay that takes the underlying physical network into account. It is difficult to quantify what scale the improvements would reach in a real-world deployment, but since they are quite measurable with a 92-node network, it is reasonable to expect that they would be significant enough to positively impact users' QoE in a realistically sized deployment. Also, although it does not directly affect users' QoE, an overlay that maps well to the underlying physical network and avoids many of the inefficiencies discussed in Subsection 2.2.2 would reduce the P2P traffic on service providers' networks, leading to reduced congestion, and ultimately to better performance for all network applications. While the transfer of content that users can initiate after a successful query was not modelled, it would be significantly faster for users to download content from a node that is closer to them: the use of Vivaldi coordinates provides the means to determine this proximity without explicit measurements. Also, due to the higher traffic volumes involved in these types of transfers, as compared to Gnutella control traffic, downloading from a peer that is close in the physical topology would reduce network congestion by utilizing the physical network more efficiently.

In summary, the results show that by using the proposed neighbour selection algorithm to form the Gnutella overlay based on physical topology information achieves a better QoE for users by reaching more nodes faster with queries and receiving query hits more quickly.

Chapter 8

Conclusions

In this thesis, a neighbour selection algorithm for the Gnutella P2P network was proposed with the intention of improving the performance of queries and query hits, ultimately providing a better QoE to users. The algorithm made use of the Vivaldi coordinate system in order to assign synthetic coordinates to each node participating in the P2P overlay. These coordinates are then used to make informed decisions about which neighbours a peer in the network should choose: nodes will preferentially select neighbours that are close to them in the physical topology, according to the Vivaldi coordinates.

In order to characterize the performance of the proposed algorithm, a new network simulator named Gnutaldi was developed, based on the existing GnutellaSim and*ns-2* simulators. In conjunction with topologies generated with the BRITE tool, Gnutaldi was used to simulate a small Gnutella network. It was found that the Vivaldi coordinates converged with a low error. It was also observed that in a network using the neighbour selection algorithm, queries reached nodes faster and query hits were returned to the originator more quickly. This result shows that the proposed neighbour selection algorithm improves users QoE.

In order to extend this research, it is appropriate to consider opportunities for further investigation. One obvious area is larger-scale simulations. While the scalability of the *ns-2* simulator on the available hardware resources was a limitation, it would be worthwhile to pursue other means of simulation in order to validate the algorithm with a larger network. Also, it would be desirable to simulate the content download that can occur after a successful query, to determine the performance improvement the neighbour selection algorithm provides in this area. It would be interesting to simulate coordinate systems other than Vivaldi in the Gnutella network and compare their performance. The performance of the various coordinate systems and the neighbour selection algorithm could also be compared to other Gnutella optimization techniques that do not act on the overlay topology. Finally, it would be useful to deploy the modified protocol in a controlled network environment such as PlanetLabs [57] in order to observe its performance.

References

[1] (2005) Napster - all the music you want. any way you want it. [Online]. Available: http://www.napster.com/

[2] G. Bennett. (2003) Controlling P2P traffic. [Online]. Available: http://www.lightreading.com/document.asp?doc_id=44435&site=lightreading

[3] V. Pande, "Folding@home: advances in biophysics and biomedicine from world-wide grid computing," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, CO, Apr. 2005, pp. 101–107.

[4] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "SETI@home-massively distributed computing for SETI," *Computing in Science & Engineering*, vol. 3, no. 1, Jan.

[5] (2005) Msn messenger version 7.5. [Online]. Available: http://messenger.msn.com/Xp/Default.aspx

[6] (2005) Icq.com - community, people search and messaging service! [Online]. Available: http://www.icq.com/

[7] A. El Saddik and A. Dufour, "Peer-to-peer suitability for collaborative multiplayer games," in *Proc. Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications*, Delft, Netherlands, Oct. 2003, pp. 101–107.

[8] ——, "Peer-to-peer communication through the design and implementation of xiangqi," in *Proc. International Conference on Parallel and Distributed Computing*, Klagenfurt, Austria, Aug. 2003, pp. 1309–1313.

[9] M. Ripeanu and I. Foster, "Mapping the Gnutella network," in *Proc. 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002, pp. 85–93.

[10] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. SIGCOMM'04*, Portland, OR, Aug. 2004, pp. 15–26.

[11] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto, "Mapping peer behavior to packet-level details: a framework for packet-level simulation of peer-to-peer systems," in *Proc. 11th IEEE/ACM MASCOTS*, Orlando, FL, Oct. 2003, pp. 71–78.

[12] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: an approach to universal topology generation," in *Proc. MASCOTS 2001*, Cincinnati, OH, Aug. 2001.

[13] A. Dufour and L. Trajković, "Using synthetic coordinates to improve the performance of the gnutella network," in *Proc. The Third International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks*, Waterloo, Canada, Aug. 2006, to be published.

[14] (2005) The network simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/

[15] (2005) P-Cube - global leader in service control & bandwidth management for service providers. [Online]. Available: http://www.p-cube.com/indexold.shtml

[16] (2005) Caspian networks. [Online]. Available: http://www.caspian.com/home.asp

[17] (2005) A brief history of Napster. [Online]. Available: http://iml.jou.ufl.edu/projects/Spring01/Burkhalter/Napster%20history.html

[18] (2003) The annotated Gnutella protocol specification v0.4. [Online]. Available: http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[19] (2003) Limewire: The official site for the fastest file sharing program on the planet. [Online]. Available: http://www.limewire.com/english/content/home.shtml

[20] (2005) Bearshare - the world's best gnutella client. [Online]. Available: http://www.bearshare.com/

[21] B. Cohen. (2004) The official BitTorrent home page. [Online]. Available: http://bitconjurer.org/BitTorrent/

[22] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[23] (2005) The Freenet project - index - beginner. [Online]. Available: http://freenet.sourceforge.net/

[24] A. Barabási, *Linked: the New Science of Networks.* Cambridge, MA: Perseus, 2002.

[25] A. Rowstron and P. Druschel, "Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001, pp. 329–350.

[26] A. Rowstron, A. Kermarrec, P. Druschel, and M. Castro, "Scribe: the design of a large-scale event notification infrastructure," in *Proc. NGC2001*, London, UK, Nov. 2001.

[27] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in a cooperative environment," in *Proc. SOSP'03*, Lake Bolton, NY, Oct. 2003.

[28] L. Gong, "JXTA: a network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, May/June 2001.

[29] T. Klingberg and R. Manfredi. (2002) Gnutella protocol development. [Online]. Available: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[30] V. Pareto, *Cours d'Economie Politique.* Geneva, Switzerland: Droz, 1964.

[31] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power laws and the as-level internet topology," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 514–524, Aug. 2003.

[32] A. Barabasi, R. Albert, and H. Jeon, "Scale-free characteristics of random networks: the topology of the world wide web," *Physica A*, vol. 281, no. 1-4, pp. 69–77, June 2000.

[33] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," *Networks: the international journal of computer and telecommunications networking*, vol. 33, no. 1-6, 2000.

[34] S. Joseph. (2004) P2P simulation and reality. [Online]. Available: http://apan.net/meetings/honolulu2004/materials/p2pgrid/SamJoseph-P2P-SimulationAndReality.ppt#1

[35] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, Aug. 2001.

[36] (2005) Making networks and applications perform. [Online]. Available: http://www.opnet.com/

[37] (2005) Scalable simulation framework. [Online]. Available: http://www.ssfnet.org/

[38] S. Merugu, S. Srinivasan, and E. Zegura, "P-sim: a simulator for peer-to-peer networks," in *Proc. 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, Orlando, FL, Oct. 2003, pp. 213–218.

[39] N. Ting and R. Deters, "3ls - a peer-to-peer network simulator," in *Proc. Third International Conference on Peer-to-Peer Computing*, Linkoping, Sweeden, Sept. 2003, pp. 212–213.

[40] K. Kant and R. Iyer, "Modeling and simulation of ad-hoc/P2P file-sharing networks," in *Proc. Performance TOOLS 2003*, Urbana, IL, Sept. 2003.

[41] J. Banks, J. Carson II, B. Nelson, and D. Nicol, *Discrete-Event System Simulation.* Upper Saddle River, NJ: Prentice Hall, 2001.

[42] G. Riley, M. Ammar, R. Fujimoto, A. Park, K. Perumalla, and D. Xu, "A federated approach to distributed network simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 2, pp. 116–148, Apr. 2004.

[43] S. Saroiu, P. Krishna Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. Multimedia Computing and Networking*, San Jose, CA, Jan. 2002.

[44] Q. He. (2005) Packet-level peer-to-peer (gnutella) simulation. [Online]. Available: http://www.cc.gatech.edu/computing/compass/gnutella/usage.html

[45] Z. Ge, D. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling peer-peer file sharing systems," in *Proc. IEEE INFOCOM'03*, San Francisco, CA, Apr. 2003, pp. 2188–2198.

[46] M. Penrose, "On k-connectivity for a geometric random graph," *Wiley Random Structures and Algorithms*, vol. 15, no. 2, pp. 145–164, Sept. 1999.

[47] T. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM'02*, New York, NY, June 2002, pp. 170–179.

[48] ——, "A network positioning system for the internet," in *Proc. of the 1st USENIX Symposium on Networked Systems Design and Implementation*, San Francisco, CA, Mar. 2004, pp. 141–154.

[49] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for scalable distributed location," in *Proc. 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, Feb. 2003, pp. 278–291.

[50] M. Costa, M. Castro, A. Rowstron, and P. Key, "Pic: Practical internet coordinates for distance estimation," in *Proc. 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, Mar. 2004, pp. 178–187.

[51] H. Benson, *University Physics.* Hoboken, NJ: John Wiley & Sons, 1995.

[52] K. Gummadi, S. Saroiu, and S. King, "Estimating latency between arbitrary internet end hosts," in *Proc. SIGCOMM'02*, Pittsburgh, PA, Nov. 2002, pp. 5–18.

[53] A. Haeberlen. (2004) Topology measurement and modelling. [Online]. Available: http://www.cs.rice.edu/ eugeneng/teaching/f04/comp529/lectures/lecture14.ppt

[54] B. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[55] D. Magoni and J. Pansiot, "Evaluation of internet topology generators by power law and distance indicators," in *Proc. 10th IEEE International Conference on Networks*, Singapore, Aug. 2002, pp. 401–406.

[56] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM'02*, San Francisco, CA, Mar. 1996, pp. 594–602.

[57] (2005) PlanetLab: home. [Online]. Available: http://www.planet-lab.org/