Simulating Peer-to-Peer Networks

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

By

Nyik San Ting

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in this thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or in part should be addressed to:

Head of the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan, Canada

S7N 5A9

Abstract

Peer-to-Peer (P2P) systems are emerging as a new form of distributed computing with a strong emphasis on self-organization, decentralization, and autonomy of the participating nodes. The characteristics of self-organization, autonomy, and decentralization allow for highly adaptive, robust, and scalable networks, making P2P an increasingly interesting way to design distributed systems.

Since the deployment of P2P systems involves significant resources, e.g., hundreds of hosts and users, it is often not possible to run realistic tests prior to the rollout of the system. Consequently, simulation is the only realistic approach for testing or predicting the behavior of large P2P networks. However, the majority of the existing simulators tend to provide limited flexibility in simulating the details of the users, application, protocol, and physical network.

In this research, the impact of user behavior, protocol, and physical network characteristic on the overall P2P system are being observed. The aim is to investigate the importance of simulating P2P systems in such detail.

Acknowledgement

I would like to express my deep gratitude to my supervisor Dr. Ralph Deters for his constant guidance, patience and support. His wide knowledge in distributed computing has enlightened and motivated me in the P2P research. He has guided me and kept me on track.

I would like to say thanks to the students and faculties in the Computer Science Department for providing supportive and resourceful environment.

Last but not least, I greatly thank my friends and family for the companionship in the journey of completing my thesis. Thanks for their nudge when I was behind and their encouragement when I felt like this journey is not going to end.

Table of Contents

List of Figures

List of Tables

List of Acronyms

3LS	3 Layer Simulator
CHK	Content-Hash Key
DNS	Domain Name System
GDL	Graph Description Language
GUI	Graphical User Interface
GUID	Globally Unique IDentifier
HTTP	HyperText Transfer Protocol
I/O	Input/Output
ID	Identifier
IP	Internet Protocol
IPC	Inter Process Communication
PID	Peer Identifier
RMI	Remote Method Invocation
SHA	Secure Hash Algorithm
SMPD	Single Process Multiple Data
SSK	Signed-Subspace Keys
TCP/IP	Transmission Control Protocol/Internet Protocol
TTL	Time-To-Live
QH	Query_Hit

Chapter 1

Introduction

In the middle of the 90's, the computational resources of ordinary desktop computers began to exceed the needs of their users, creating an ever-growing pool of unused computational resources. Intel [16] estimates that in the average organization the combined computational resources of the desktop machines are 2.5 times larger than those of their centralized servers and high-end computer nodes.

SETI@Home [45] demonstrated that by using simple P2P techniques it is possible to harvest the scattered resources of thousands of desktop computers in an efficient way enabling the analysis of a large amount of data at virtually no cost. The success of the SETI@Home project, which forms one of the most powerful computing networks today, has resulted in an increasing interest in the study and deployment of P2P networks.

The field of P2P networks is still in its infancy with new applications and protocols emerging on a nearly daily basis. Testing a system's performance prior to its deployment is fairly common in the development of software applications. There are two main possible experimentation streams: experimentation with the actual system and experimentation with a model of the system.

P2P networks tend to be large, heterogeneous systems with complex interactions between the physical machines, underlying networks, applications, and users. Hence, testing a "running" P2P network or protocol in a realistic environment is often not feasible. Due to the difficulties involved in evaluating the protocols prior to their large-scale deployment, they are often short-lived and disappear as fast as they emerge, normally due to poor performance.

According to Manjoo [25], the Gnutella [9] network encountered massive performance problems similar to a denial-of-service attack in September 2001. Consequent investigations traced the cause to the use of a new Gnutella client called "Xolox" [49] that unintentionally swamped the network with useless messages. Xolox was configured to have a re-querying algorithm, which frequently checks the network for desired files, to enable faster downloads for the user.

Xolox demonstrates the crux of the problem in developing P2P applications – the inability to test the algorithms in realistic settings due to the absence of an appropriate simulator. Currently, a new generation of protocols and systems, which emphasize self-organization as a result of "learning" from past experiences, is emerging. These protocols are tightly coupled to the network topologies, applications, and the users' behavior. To experiment with such networks, a P2P simulator must be capable of representing the tight dependencies between the users, applications, protocols, and physical networks. Researchers who want to simulate a P2P system tend to avoid the development of complex simulators and focus on some selected areas (such as caching schemes). While some start an implementation from scratch, an increasing number of researchers build their simulators on top of existing agent platforms like JADE [6] to speed up the development.

The general problem of having only special-purpose simulators is that the results obtained with one simulator are difficult to validate and often impossible to achieve with another simulator due to the many hard-coded assumptions in every simulator. A faithful P2P network simulation is complex. It involves multiple layers such as the physical network, application/protocol, and user modeling. It also involves large numbers of nodes on the different layers.

This thesis intends to answer the following questions:

- Is it really necessary to simulate the P2P system in such detail?
- Is there any impact on the overall P2P system performance as a result of minor changes in the physical network, application/protocol, and user modeling?

Hence in this research, a simulator that supports manipulation of the characteristics for each layer is developed to enable the evaluation of the overall P2P system performance. In addition, a series of P2P simulations is run to observe the impact on the overall P2P system performance.

The thesis is organized as follows. Chapter two gives an overview of existing protocols. Chapter three introduces the challenges of and approaches to simulating P2P protocols and provides an overview of existing P2P simulators. Chapter four describes the design of a novel multi-layered P2P network simulator. Chapter five details the experiments with analysis results and concludes the thesis work with future plans.

Chapter 2

Overview of P2P Protocols

Over time, a variety of P2P protocols and systems have emerged. In this section an overview of the most influential protocols is given. Since resource allocation is a central element in every P2P protocol, P2P protocols can be grouped [28] into one of the following three basic categories: centralized directory model, document routing model, and flooded request model.

2.1 Centralized directory model

This model is based on the availability of a server (used as a centralized directory service) that manages the information of all the participating peers in the P2P network. In order for a peer to join and participate in the network, it must directly connect to the central server. While the peers are in full control of their locally owned resources, they need the server to advertise their resources or locate the resources of other peers.

This centralized approach solution provides good resource allocation, performance and network manageability. However, using a centralized component introduces a single point of failure.

2.1.1 Napster

Napster [31] is a centralized P2P file-sharing network that was forced to shut down as a result of copyright infringements. In Napster, a cluster of central servers maintained the information on the content offered by the peers in the network. All the peers in the

Napster network had to connect to the server, which also handled their resource request queries. Upon receiving the results of the match from the central server, the peer establishes a connection to the resource-providing peer and starts to consume the resource, i.e., downloads the file directly.

2.1.2 SETI@Home

SETI@Home is designed to aid the process of searching for Extraterrestrial Intelligence by harvesting the unused processing power of idle desktops via the Internet. A centralized server stores and packs the signal data of radio telescopes into small chunks that can be processed by an average desktop machine. The role of the desktops is to perform a series of signal processing operations to detect patterns or abnormalities.

Users who are willing to participate have to download and install the signal processing packages provided by SETI@Home. Users download an OS specific SETI@home screen-saver that is used as a means of detecting idle-cycles or software designed to run as a background process. When the SETI@Home software recognizes that the host has idle resources (based on the settings of the user), it contacts the central server of SETI@Home to download a chunk of data (200 KB). Upon the successful download, the peer starts a series of signal processing activities, sends the results (80 KB) back to the server, requests new data and restarts the compute cycle.

Despite the impressive performance of the SETI@Home network, it is important to point out that it can only handle single process multiple data (SPMD) computing problems due to the inability of SETI@Home peers to communicate and therefore coordinate their data processing.

2.2 Document routing model

The document routing model is the most recent model for resource allocation in P2P networks. In this model, the system has no single point of control. It assigns a PID (Peer Identifier) to each peer and a key to each resource. The files (resources) are routed to other peers using algorithms that decide the location of the resource in the network according to the key of the resource and the PID of the nodes in the network. Consequently, the request for the resource can be routed to the destination peer without replication and broadcast. This model obtains significantly better performance at the price of reduced autonomy of the peers in regards to how resources and data about resources are handled.

2.2.1 FreeNet

FreeNet [14] is a P2P file-sharing system that provides features such as information anonymity, high security, and encryption. Each FreeNet peer resides on a node that contributes storage spaces to the FreeNet network. A unique key of the file is generated using SHA-1 [32] (Secure Hash Algorithm). There are several kinds of hash keys used within FreeNet; the two most important ones are content-hash keys (CHK) and signed-subspace keys (SSK) [8]. The content of the file to be stored is hashed to generate the CHK, ensuring the uniqueness of the keys as it is considered "nearly impossible" [8] for two different files to be hashed to the same key due to the large space a key can be picked from. Each SSK is associated with a pair consisting of a public and private key. This provides a secure system because anyone who has the public key can read the file but only those who have the private key can perform write operations on it.

Upon joining the network, a new node first generates a public-private key pair for itself. The peer advertises its presence by connecting to a remote peer that is already in the network and sending an announcement message that contains the public key, the physical

address, and the TTL (Time-To-Live) of the message. Once the peer receives the announcement message, it randomly chooses a connection to which it forwards the announcement message. The announced message is propagated throughout the network until the maximum TTL is reached. Then, the peers that know of this new peer assign a unique random PID (also called GUID or Globally Unique IDentifier) to the new node and update their routing tables.

A FreeNet peer stores knowledge about the PIDs of the nearby peers and the keys of the files contained. As each file is assigned a unique key, upon insertion of the file into the network, the file is rerouted to a neighboring peer with a PID closest to the key of the file and it is replicated for storage before it is rerouted again. The process of rerouting and replication is repeated until a user-defined number of copies have been stored in the network. Similarly, users can retrieve the file using the key of the file. When the file is being rerouted for retrieval, it is replicated and stored in the nodes along the path. Hence a frequently retrieved file tends to have a larger number of replicated files in the network and the search can be done faster, whereas a less frequently retrieved file requires a longer time to be retrieved and might even be replaced by the more frequently retrieved files due to the limited storage space.

2.2.2 Pastry

Pastry [40] supports a variety of P2P applications, such as file-sharing and group communication. Similar to other document routing models, each node in the Pastry network has a PID (or node ID) and each resource in the network has a unique key. A resource is stored on a user-predefined number of nodes with PIDs that are the closest to the 128 most significant bits of the key. Each Pastry node keeps a routing table, a neighborhood set, and a leaf set. The routing table contains $\lceil log_{2^b} N \rceil$ rows with $2^b - 1$ entries, where N is the number if nodes in the network and b is a configuration parameter

with typical value of 4. Each entry in row *i* contains the IP address of the other node that matches the node's own IP address in the first *i* positions.

The leaf set is the set of the closest PIDs, such that half of the PIDs are larger than the node's PID and the other half of the PIDs are smaller. The neighborhood set contains the PIDs and IP addresses of nodes that are closest to the local node. A Pastry node first forwards the message – which contains the key of the targeted object – to those nodes with PIDs at least one digit longer than the matching number of prefixes between the key and the current node's PID that are closest to the key. If this attempt fails, the message is routed to the node with a PID that shares the same number of matching prefixes with the current PID and is numerically closest to the key as the current node's ID. Using the information stored and the two steps described above, a Pastry node can route a message with the minimum distance/hops needed. To ensure the information of the tables is up-to-date, after network-changing events, such as the node arrival and departure, the routing tables are exchanged among the affected nodes.

2.3 Flooded request model

The flooded request model is a "pure" P2P model since it does not require any centralized infrastructure. Due to the absence of any predefined structures, resource requests of a peer are handled by use of message flooding.

2.3.1 Gnutella

Gnutella [9] is a good example for the flooded request protocols. Gnutella has been designed for resource searching in distributed systems with decentralized control. In this thesis, the two main Gnutella protocols (0.4 and 0.6) are being studied. Due to its

widespread use in the research community, it was chosen as the protocol to be used in the simulator for this research and a more detailed description is given below.

2.3.1.1 Gnutella v0.4

A Gnutella v0.4 peer (X) connects to the network by establishing a TCP/IP connection to a peer (Y) that is currently on the network. The IP addresses of the peers on the network can be obtained from the predefined repositories. After connecting, X sends a request string **"GNUTELLA CONNECT/0.4\n\n"** to establish a link to peer Y. Peer Y can accept the connection, by sending **"GNUTELLA OK\n\n"** string, or refuse the connection, by sending a different response. Once the peer X is connected to the network, it can communicate with other peers by sending and receiving Gnutella messages (also called descriptors).

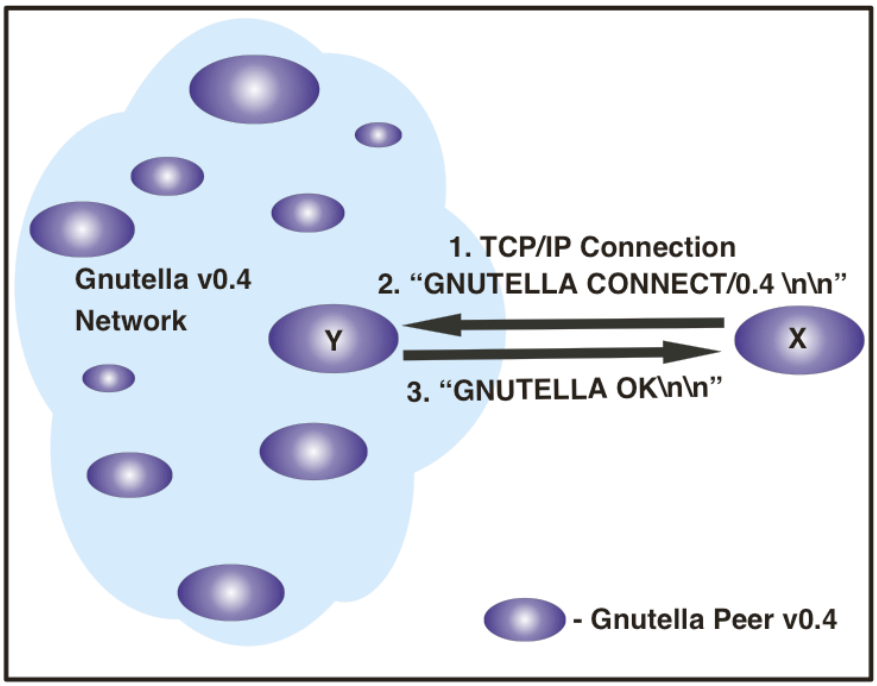

Figure 2.1. Peer X Initializes Connection to Gnutella Network.

There are five kinds of common messages in Gnutella v0.4: ***Ping***, ***Pong***, ***Query***, ***Query_Hit***, and ***Push***. The Push message is used to allow file-based data contribution of

a peer behind a firewall. There are 60% of LimeWire [24] (a P2P file sharing application using Gnutella Protocol) users behind firewalls today; however, the contribution of the Push messages to the overall Gnutella traffic is not determined. An analysis [3] over a period of 35 hours in 2001 showed that the Push messages contribute about 6% of the traffic on the Gnutella network, while another quantitative analysis [50] of the Gnutella network in 2002 shows that there are only 3000 Push messages in the sample of 56 millions messages. It also shows that among 56 million messages, about 63% of the messages are Ping/Pong, 47% of the messages are Query/Query_Hit, and 0% (about 3000 messages) are Push messages. Hence, for simplification, Push messages are being ignored in this thesis.

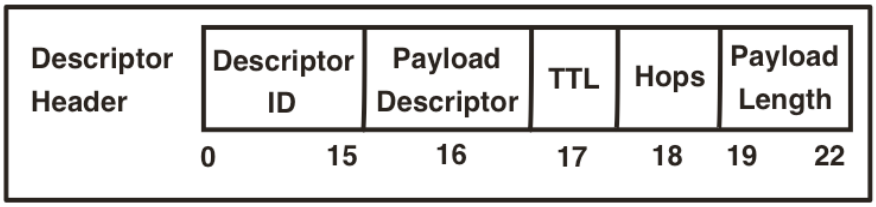

Figure 2.2. General Structure of a Message Header.

Each of the messages has a message/descriptor header (see Figure 2.2). Every Gnutella message has 22 bytes in the header: Descriptor ID (bytes 0 to 15), Payload Descriptor (byte 16), TTL (byte 17), Hops (byte 18), and Payload_Length (bytes 19 to 22). The Descriptor ID is a unique identifier of the Gnutella message in the network and is typically created by using a random generator. The TTL is the number of times the message will be forwarded by Gnutella peers before being dropped from the network. Hops represent the number of times the message has been forwarded by peers. Each time a message is forwarded, the message's TTL is decremented by one and the Hops are increased by one.

The Payload Descriptor contains the code for the type of the message: 0x00 for a Ping message, 0x01 for a Pong message, 0x40 for a Push message, 0x80 for a Query message, and 0x81 for a Query_Hit (QH) message. The Payload_Length contains the length of the

remainder message (payload) following the header. Hence, a message's total length is 22 bytes + the Payload_Length; the next message is located exactly a Payload_Length after the current header.

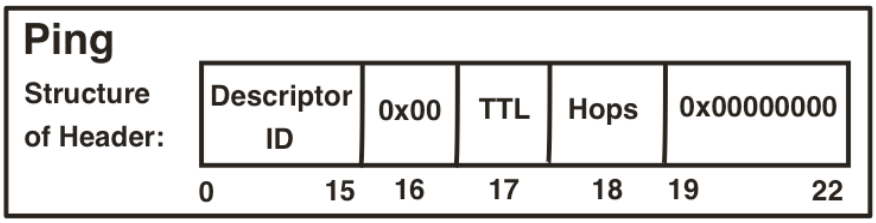

Figure 2.3. Structure of a Ping Message.

A Ping message is used to probe the other peers in the network. It is used to obtain information on the active peers on the network. A Ping message has no payload, hence the Payload_Length is zero (see Figure 2.3).

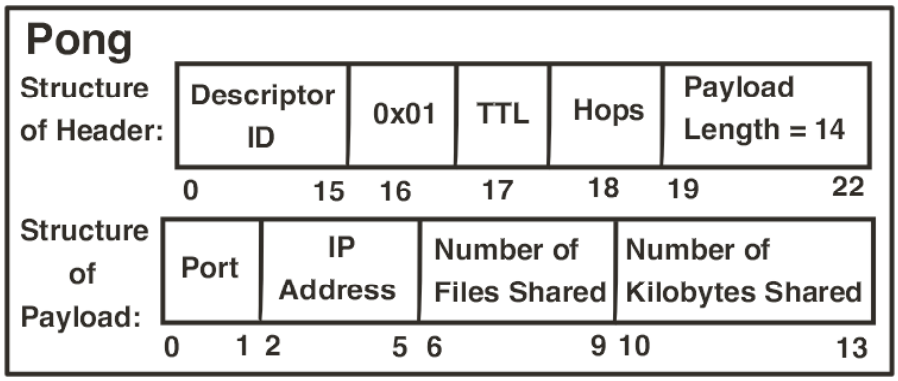

Figure 2.4. Structure of a Pong Message.

A Pong message is used as a reply to a Ping message. It includes the IP address and the port number of the responding peer, together with the number of files and number of kilobytes shared on the peer (see Figure 2.4).

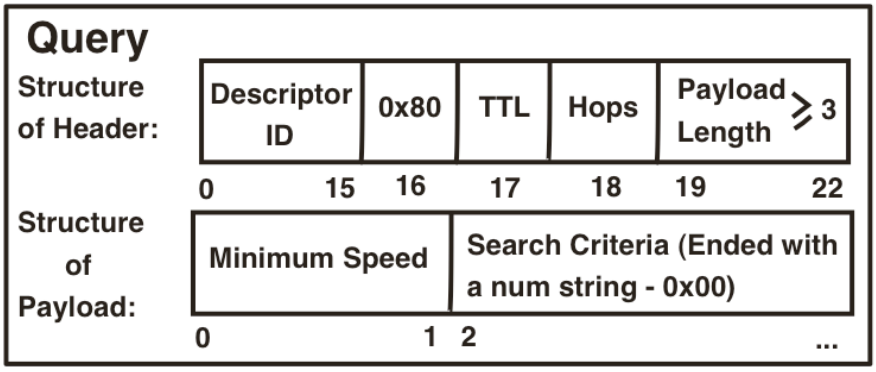

Figure 2.5. Structure of a Query Message.

A Query message is used to search a file in the distributed network. It contains a list of search criteria and a minimum speed in KB/second (see Figure 2.5). A peer only responds to a Query message if it has files that match the search criteria and a minimum communication speed as stated.

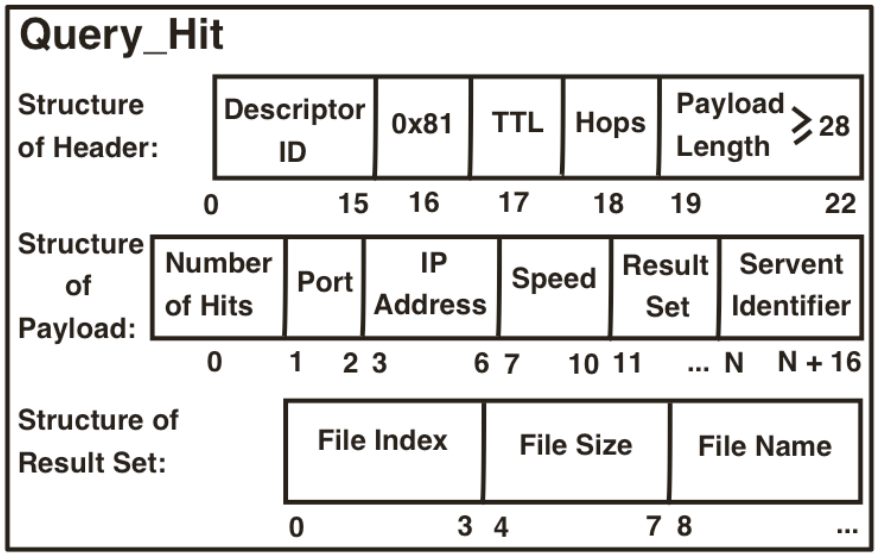

Figure 2.6. Structure of a Query_Hit Message.

A Query_Hit (QH) message is sent as a response to a Query message. It contains the number of hits, port number, the IP address, and the minimum speed of the responding peer (see Figure 2.6). The result set contains a list of file indexes with corresponding file sizes and file names. The Servent Identifier in the QH message is a 16-byte string

uniquely identifying the responding servent, and is used as an important element in the Push message.

A peer can discover other peers and their features (e.g., IP address and port number, number of resources, etc.) by sending a Ping message. Upon receiving a Ping message, a peer will replicate the Ping message and forward the Ping messages to all the directly connected peers, except to the peer who sent the Ping message. At the same time, the peer will send a Pong message, which has the same Descriptor ID as the corresponding Ping message, in response to the peer who sent the ping message. A Pong message has a payload that contains the number of files shared, the number of kilobytes shared, IP address, and port number of the responding peer.

While Ping and Pong messages are used for discovering other peers in the network, Query and QH messages are used for locating resources. To locate a resource, a peer sends a Query message that has a payload containing the required minimum network speed of a potential resource providing peer and a search string for describing the requested resource. Similar to the Ping messages, a Query message received by a peer is broadcasted to all the directly connected peers (except the sender of the request). In addition, Query messages are also subject to Hops increment and TTL decrement as a means of limiting the range of the Query message. Each message will be replicated and propagated throughout the networks until it either comes to a peer that matches the search criteria or exceeds its TTL value. Each peer that is capable of offering the requested resource responds with a QH message. A QH message contains the information about the resource-providing peer.

With the use of TTL, a resource contained in the network is not searchable if the minimum distance (Hops) between the peers is greater than the lifetime of the request. However, a resource located within the lifetime of the request might not be reachable as the result of the “short-circuiting” effect due to the latency of the network [21]. Short-

circuiting can occur as a result of the race condition in the network. If the message with the same ID travels faster on a longer path than those using a shorter path, nodes that have been visited by the faster messages might refuse to forward the slower messages effectively cutting off potential search paths (see Figure 2.7).

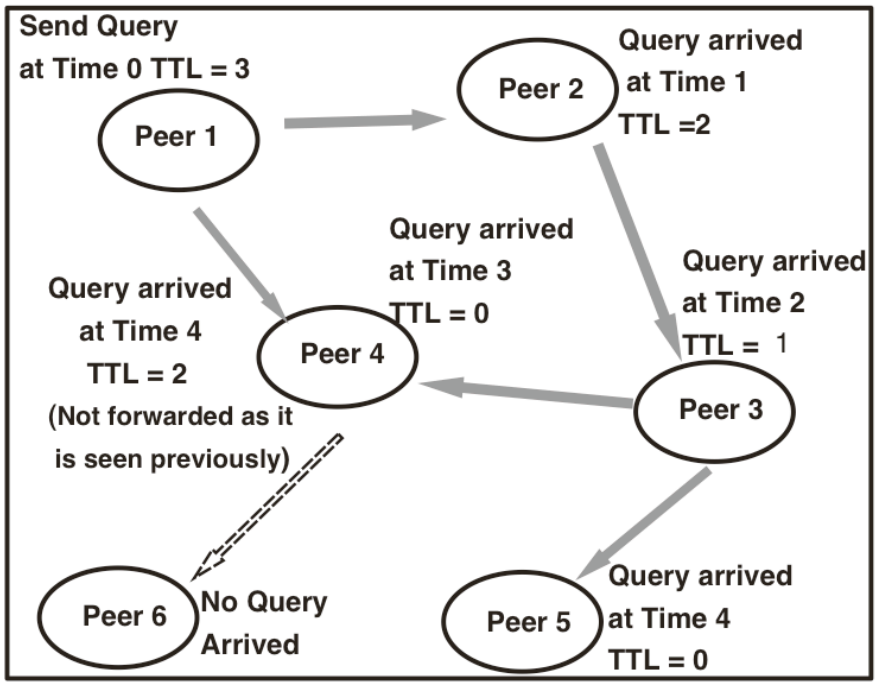

Figure 2.7. The Flat Gnutella 0.4 Network

Pong messages and QH messages are routed along the same path back to the peer that launched the original Ping and Query messages. This is made possible by keeping a record of a predefined number of messages' Descriptor IDs and their corresponding Payload Descriptors. When a peer received a response message (Pong or QH), it checks if it generated the original message (Ping or Query). If the peer is not the destined receiver, it checks if it has seen such a Ping or Query that has the descriptor ID. If no such Ping or Query message is passed through the peer, the message is dropped; otherwise the peer sends the response message to the connection that sent the corresponding Ping or Query message. Once a peer receives the QH message, it accesses the resource using the http protocol.

2.3.1.2 Gnutella v0.6

Similar to Gnutella v0.4, a Gnutella v0.6 [23] peer connects to the network by first establishing a link to another already connected peer through TCP/IP. A peer wishing to join the network sends the request to connect string "**GNUTELLA CONNECT/0.6<cr><lf>**" and optional data to describe itself thus making a connection more likely.

```
GNUTELLA CONNECT/0.6<cr><lf>
User-Agemt:BearShare<cr><lf>
X-Ultrapeer:True<cr><lf>
Listen-IP:12.134.4.23:6349<cr><lf>
<cr><lf>
```

Figure 2.8. Connection Request in 0.6.

The sending of additional data allows for the introduction of UltraPeers [46] (super nodes) in the peer network. The concept of UltraPeers is not part of the v0.6 specification; it is an optional implementation that helps reduce the network bandwidth consumption resulting from the flood of Ping and Query messages. However, since the concept of UltraPeers is a way to reduce the network load, almost all the Gnutella v0.6 peers are implemented to support the UltraPeer functionality. A peer can announce its planned departure to its connected peers by sending a Bye message thus minimizing the problems of dead links.

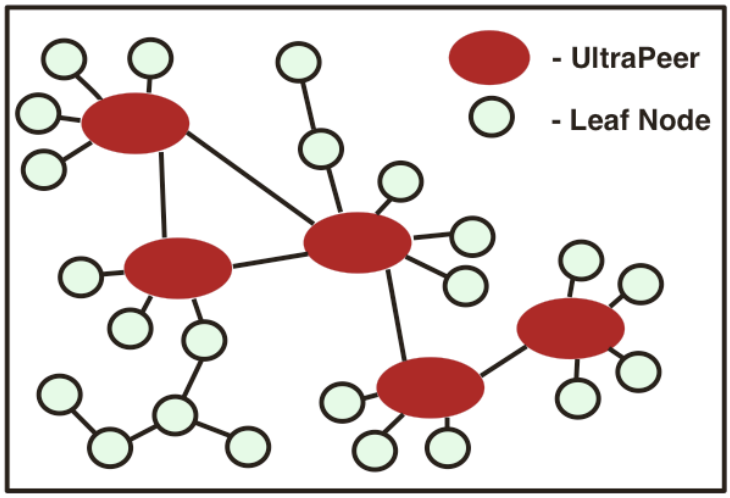

Figure 2.9. Hierarchies in Gnutella v0.6

In a Gnutella v0.6 network that uses UltraPeer functionality, there are two kinds of peers: UltraPeers and leaf nodes. Leaf nodes are the nodes that are not powerful enough to be an UlraPeer or fail to provide the UltraPeer functionality. Leaf nodes only maintain connections to UltraPeers and/or to another leaf node that wants to joint the network but can't find a suitable UltraPeer. An UltraPeer is a Gnutella peer that maintains many connections to other UltraPeers and a large set of leaf nodes. The UltraPeer maintains a record of the leaf nodes' resources and acts as a shield for its leaf nodes to the flood of Ping and Query messages.

2.3.2 NeuroGrid

NeuroGrid [18] was initially designed as an alternative routing model for Gnutella. It is designed to "provide a method of communication that will piggy-back on top of http"[19]. In NeuroGrid, the resources in the P2P network are assumed to be associated with a set of keywords. When searching for a resource, the requesting peer needs to know the keywords that are associated with the resource. It focuses on minimizing the sending of messages by increasing the query processing cost for peers. Hence, unlike Gnutella, the searches are not being forwarded to all the connections automatically; NeuroGrid expects the peer to decide how to route the request. A NeuroGrid node uses a model for

neighboring peers and their contents. Using a decentralized routing model allows a more efficient routing at the expense of added processing overhead.

2.4 Summary

The centralized directory model is straightforward but lacks the robustness of the other models since it introduces a single point of failure as it relies on the reliability and availability of the central server. The flooded request model and document routing model depend largely on the efficiency of the protocols and algorithms used. The success of the document routing model relies on the protocol of routing and replicating the resources. Since the information like routing tables are being exchanged among the nodes that are affected during node arrival/departure, the efficiency of the protocol relies on peers' frequency of entering/leaving the network and network stability.

Table 2.1. Summary of P2P Protocols.

Protocols	Characteristics
Centralized Directory Model (e.g. Napster, SETI@Home)	• Centralized, hence introduces single point of failure • Availability of the server • Peer's computing power/speed and user absence for computation (SETI@Home)
Document Routing Model (e.g. FreeNet, Pastry)	• Depends on the efficiency of the algorithm that reroutes and reallocates the resources. • User activity and network stability is important because network changing events will change the routing tables and PID info of peers
Flooded Request Model (e.g. Gnutella, NeuroGrid)	• Large number of messages floating in the network • Efficiency of the protocol/algorithm • Short circuiting effect shows the importance of network structure/latency

P2P Users' behavior, such as switching on/off the application, querying, and choosing the connection to forward the message in the case of NeuroGrid, can greatly impact the P2P network topology. Consequently, it affects the efficiency (such as the success rate of query) and scalability of the system.

While the user is a factor in P2P system efficiency, the underlying physical network of the P2P system can also be a factor. This can be seen clearly from SETI@home system, because its success relies on the peers' computing power and speed. Through the short-circuiting phenomena observed in Gnutella protocol, network delays also appear to affect the system efficiency.

As summarized in Table 2.1, the **physical network characteristics, protocol, user presence, and activity** appear to have an influence on the efficiency of the P2P system. A P2P network can span across the world and involve large number of computers with different hardware/network composition. Hence, simulation is sought to provide a testbed for P2P system experimentation and various P2P simulators are studied in Chapter three.

Chapter 3

P2P Network Simulators

As concluded in Chapter two, areas such as efficiency of the protocol/algorithm, user behavior, network, and hardware specifications appear to be important parts for the specific P2P system. Because it is not feasible to involve a large number of peers hosted on a wide range of heterogeneous hardware, P2P network simulators are needed to study the behavior of complex P2P systems. This section presents the few P2P simulators that have been implemented to aid the research on P2P-applications and P2P-protocol development. Most of the P2P simulators listed below are implemented in Java [44].

3.1 NeuroGrid Simulator

The NeuroGrid simulator 0.1.0 [33] is a Java-based P2P simulator that has been extended to support the simulation of FreeNet, Gnutella, and NeuroGrid protocols. NeuroGrid is a single-threaded discrete event simulator. It uses properties files that enable the user to modify the parameters for a simulation run. The user can specify the type of protocol, the number of searches to simulate and the type of preferred user interface (e.g., applet-based GUI). The applet displays the search messages being sent to the searching nodes at each step (see Figure 3.1).

The statistics (e.g., the number of messages parsed and the state of the simulation) can be saved into files for later analysis. The NeuroGrid Simulator 0.1.0 assumes that the distances between the nodes are constant – messages with the same TTL are sent through the network in parallel. After a search message is sent out to other predefined nodes, the nodes that received the messages take turns in forwarding the messages, due to the single-threaded design of the simulator. Until a search event has terminated, no other new

search event will be active (sequential execution of search events). NeuroGrid is still very much a work in progress and efforts are being made to improve the level of detail in the network models [20]. NeuroGrid enables the user to specify the number of nodes to simulate (this is also the number of nodes to add to the current simulation after a number of searches is done), the initial number of connections for each node, the number of searches to be generated, and the initial network topology (ring or random networks).

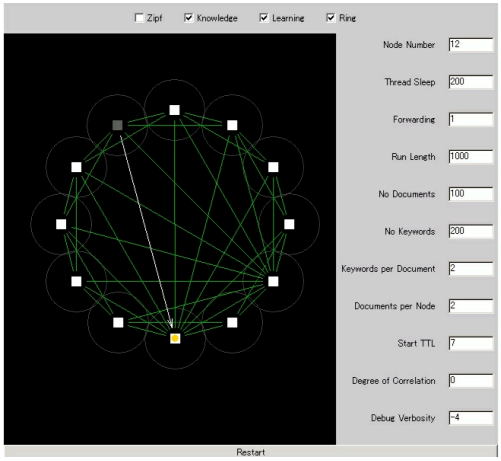

Figure 3.1. NeuroGrid Simulator

Since it is designed to simulate the searching algorithms of three different protocols, it is necessary to let the user specify the number of keywords used for the simulation, the number of the documents used for the simulation, the number of keywords per document, and the number of documents stored on each node (document and keyword assignments are all randomized). The latest release of NeuroGrid Simulator version 0.2.1 (June 24th, 2003) included power law network topology. By extending the classes provided, the simulator can simulate the user-defined application on the three protocols.

3.2 FreePastry

FreePastry [15] is an ongoing open-source implementation of the Pastry protocol in Java. The latest version of FreePastry was released on Jan 31th, 2006. FreePastry includes the

implementation of the PAST [39] archival storage system based on Pastry and an implementation of the Scribe [7] group communication infrastructure. It is a Pastry peer application, but it can also emulate a Pastry network. It provides three choices of transport protocols for the user application: direct, RMI [43], and Wire. With the direct transport protocol, FreePastry emulates a network with a user-defined number of Pastry nodes in a single Java Virtual Machine without modeling the physical network. In this situation, the main thread sets the network up and initiates the search events; because it is single threaded, the searches are done in sequence as in NeuroGrid Simulator. RMI and Wire refer to the distribution and use of RMI or Socket for IPC (Inter Process Communication).

The settings of the simulator parameters, such as the number of nodes to simulate and the number of events to generate, is done by providing the values in the command line upon starting the local simulators. The results are displayed on the command prompt screen as the messages are being processed. Since the Pastry routing uses a proximity metric, it is necessary to represent proximity in the simulation. Euclidian, random, and sphere network topologies are currently available. In the Euclidean network topology, the nodes are randomly placed in an Euclidean plane and the proximity is based on the Euclidean distance in the plane. In the Sphere network topology, the nodes are randomly placed on a sphere, and the proximity is based on the Euclidean distance on the sphere. However, the network delay for the message passing is not simulated, because the simulator is not designed to simulate time. Subsequent future releases on FreePastry are planned to support strong security that allow deployment in the insecure Internet.

3.3 Query-Cycle Simulator

As mentioned in the name, the Query-Cycle simulator [42] is a discrete event simulator. It advances the simulation process in query cycles. During each cycle, a peer can either be in a state of actively issuing a query, inactive, or down and not responding to any

queries passing by. Upon receiving the responses to the query issued at the beginning of the query cycle, the peer selects a download source for file downloading. The query cycle ends when all the peers that have issued queries are done with satisfactory file downloading. Then, another query cycle begins.

This simulator focuses on file-sharing of Gnutella-like networks. The content distribution is based on the category of the files and their popularity. The parameters are divided into Content, Peer-Behavior, and Network parameters. Content parameters are responsible for content distribution. Peer-Behavior parameters are responsible for the characteristics of the peer, such as uptime, query type and its frequency, and download selections. The Network parameters are responsible for the network topology and available bandwidth.

3.4 PeerSim

Bison [30] stands for Biology-Inspired techniques for Self-Organization in dynamic Networks. It was a three year project conducted at the University of Bologna from January 2003 to April 2006. Its aim was to study complex adaptive systems in a dynamic network environment. Prior to the Bison project, the Anthill project [4] was developed as a testbed for studying and experimenting with complex adaptive systems. The Anthill project has now been suspended with the development of Bison. As the need for a simulated environment to enable experimentation with both ad-hoc and overlay networks increases, PeerSim [29] [37] has been partly developed as part of the Bison project for large-scale overlay network simulation. It was developed in Java and designed to provide flexible configuration with many pluggable and extendable components. It provides the cycle-based simulation which proceeds through time steps/cycles. During each cycle, each peer has a chance for execution; however, the execution of searches are done sequentially. The simulations are executed with simplified assumptions (such as ignoring the communication transport details in the network) to improve scalability. PeerSim 0.3,

which was released on 06 Jan 2005, includes the first prototype of an event discrete simulator, which allows the simulation of latency of transport communication.

The PeerSim simulation engine is responsible for instantiation of the network at simulation startup. The network in PeerSim composed of nodes, where each node contains a list of connections (i.e., a list of neighbor nodes). Initializers are run to initialize the state of the system, for example, to initialize network connections to construct overlay network topology and to initialize values for the PeerSim Protocols, which instantiate the nodes with the appropriate protocols. During the simulation, the PeerSim Dynamics are run to modify the state of the system, such as by removing nodes from the network. The PeerSim observer objects are run periodically with the predefined interval to analyze and collect statistics on the state of the network.

3.5 GnutellaSim

GnutellaSim is a scalable packet-level simulator that enables the evaluation of the Gnutella system with a detailed network model. It is based on the framework for the Packet-level Peer-to-Peer Simulation [17], consisting of three layers: socket adaptation layer, P2P agent layer, and P2P application layer. The bottom layer can be adapted with a packet level network simulator, such as ns-2 [34]. The P2P agent layer contains the protocols that are being simulated and studied, such as Gnutella protocol or FreeNet protocol. The upper layer is the peer application layer that defines the behavior of the peer with a separate entity, namely an activity controller, which generates user actions based on behavioral model.

Without the use of an external network simulator such as ns-2 or pdns [38], the packet-level peer-to-peer simulator no longer involves any simulation on the packet-level, and

bandwidth is not simulated. A packet-level simulation is actually a discrete event simulation with each packet treated as an event.

3.6 Narses Simulator

Narses [10] is a network simulator implanted in Java for large distributed applications. It overcomes the scalability and speed of ns-2 due to the detail in packet-level simulation and introduces a flow-based network simulation. It groups packets into flows and reduces the total number of events in the simulation. The bandwidth on a node is shared equally between the ongoing flows. Through experiments, it is shown that the Narses simulator is 45 times faster than ns-2 and consumes only 28% of the memory used by ns, while the accuracy of the simulation is within 8% on the average.

3.7 SimP2 Simulator

SimP2 [22] is designed to support and analyze the development of ad-hoc peer-to-peer resource sharing networks. It focused on Gnutella-like networks. It uses an analytical model to create a set of network instances and to simulate file searches. The underlying P2P network is modeled on a non-uniform graph. It provides control over nodal degree distribution. The simulation does not have the mapping of P2P networks onto physical network.

3.8 GnucNS Network Simulator

GnucNS Network Simulator [11] was developed by the Gnucleus (an open source Gnutella client) community. It is built for studying the performance of Gnutella networks in different configurations, such as “the proper supernode upgrade/downgrade

algorithms, the right number of connections and TTL for a target network size, and the effect of different settings on the amount of bandwidth needed or wasted." It is a time-stepped simulation with each time unit as one second. It is implemented in C# [27] with an accurate representation of the connected nodes at each second.

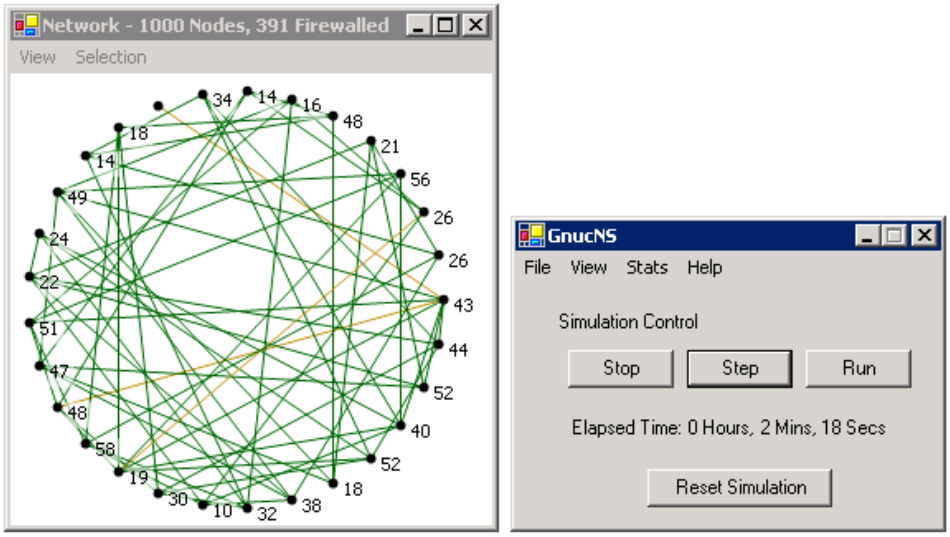

Figure 3.2. The Network View (Left) and Simulation Controller (Right).

3.9 P-sim

P-sim [26] is a simulator for P2P networks on top of representative Internet topologies. Upon initialization of peer information, P-sim needs to construct an underlying physical network at the router-level using an Internet Topology. It uses the Transit-Stub graph model (a representation of the hierarchical router-level topology of the Internet) of a GT-ITM topology generator. It attaches the end-hosts to the leaf routers and computes a routing table of both distance and next hop information between every pair of end-hosts. It allows simulation of a dynamic peer environment where a peer can join and leave the networks at various time. In P-sim, there are two implementations of search protocols (scoped flooding and random walk) for use in simulated P2P networks. In scoped flooding, the query is sent to all the peer's neighbors, while in random walk protocol, the query is only forwarded to a randomly chosen neighbor. The following metrics are used

for evaluation of various aspects of a P2P network in P-sim: File Success Rate, Nearest Search Result, Connectivity, Stress, and Relative Stretch Penalty.

3.10 P2PSim

P2PSim [36] is a multi-threaded discrete event simulator. It is part of the IRIS (Infrastructure for Resilient Internet Systems) project. It is still under development and is available in its pre alpha version. It "maximizes concurrency for performance, minimizes the need for synchronization, and avoids deadlocks" [35]. It is currently supporting Chord, Accordion, Koorde, Kelips, Tapestry, and Kademlia. The P2PSim source code is actually part of Chord. Hence, in running P2Psim, Chord's source code is needed.

3.11 Summary

In Chapter two, a few P2P protocols were reviewed and it became clear that user and physical network characteristics should be taken into consideration when a P2P system is being analyzed. However, lots of simulators have ignores the possible interaction between the user behavior and physical network characteristic. Most of the current P2P simulators do not support the customization of the initial network state (connections between the simulated computers and the network delay) and are limited in the level of detail and the scalability of the supported models. Furthermore, the simulators are mostly focused on the caching algorithms and ignore the fact that other activities can also impact the efficiency of the system.

FreePastry executes search events only in a serial fashion and do not support the modeling of network latency and heterogeneous hardware. Due to the absence of a GUI in most of the simulation, such as FreePastry, the modification of parameters is cumbersome. Some of the settings are made through the command line and some have to

be encoded in the program. Some of the simulators, such as the NeuroGrid simulator and GnucNS provide very good network visualization using an applet; however, it does not currently simulate the user events, network latency, and the processor delay of the nodes.

Table 2.2. Summary of existing P2P simulators

Simulator	*Summary*
NeuroGrid	• Simulate limited constant network delay • Enable different network topology for the simulation • Extendable protocols • Enable content distribution settings • Sequential execution of simulation
FreePastry	• Provide proximity metrics • Does not model the physical network (no network delay/latency simulation) • Sequential execution of simulation
Query-Cycle	• Enable network topology settings • Simulate available bandwidth • Simulate Peer up-time/down-time • Discrete event; cycle based (non realistic) • Content distribution
PeerSim	• Simulate latency (in event-driven simulation) • Cycle-based (sequential execution) • Event-Driven (serial execution that supports concurrency and latency) • Provide extendable protocols and network
GnutellaSim	• Extendable to use ns-2 or pdns; if not extending it's not Packet-level simulation and will not simulate bandwidth. • Discrete event simulator (where each packet is treated as an event)
Narses	• Flow-based network simulation. It groups packets into flows (bigger chunk)
SimP2	• No mapping of P2P network onto the physical network • Analytic model
GnucNS	• Enable simulation of different protocol algorithms • Bandwidth simulation
P-Sim	• Support dynamic network • Event-driven simulation
P2PSim	• Multi-threaded discrete event simulator • Simulate different protocol • Enable topology simulation

Adapting the simulator to new or modified protocols is a question of great practical importance. The NeuroGrid Simulator did not simulate the network overlay, and hence, it is hard to extend the simulation to handle new protocols that need the network proximity information, e.g., the Pastry protocol. Though FreePastry is focused on Pastry protocol, the simulator is more decoupled and some of the code can be reused and extended to implement a simpler protocol, such as Gnutella. However, the serialized event handling and the lack of simulation of time make it hard to extend FreePastry to simulate the network delay and processor delay. Furthermore, since there is no P2P simulator that simulates the hardware of the computer, compute-sensitive protocols, such as the one used in SETI@Home, cannot be simulated by extending from the existing simulators.

Chapter 4

Research Goal

Chapter two identified that protocol/algorithm, user behavior, network, and hardware specifications are key elements in a P2P system. To simulate a realistic P2P network, the simulation of numerous and diverse hardware (PCs and network), resources, protocols, and user behavior are needed. However, it is shown in Chapter three that very few P2P simulators are designed with all these features in mind.

Is the lack of details in the simulator acceptable or is that a limitation that needs to be addressed? We must ask how important is it to simulate a P2P network in such detail? **This thesis research aims to investigate the importance of simulating a P2P system in detail and to discover whether minor changes in user behavior, protocol and physical network characteristics have an impact on the overall P2P network performance.**

In order to be able to conduct a series of experiments to answer this question, a fine-grained simulator is needed. This simulator needs to provide flexibility in protocol/algorithm, user presence/behavior, network, and hardware specifications. Due to the absence of an appropriate P2P network simulator that enables a simulation with this level of detail, an infrastructure for a complex, large-scale P2P network simulation is introduced in this thesis.

4.1 3 Layer Simulator (3LS)

To implement a simulation, general-purpose languages (such as FORTRAN, Pascal, C, and Java) or simulation languages (such as GPSS, SIMAN, or SLAM II) can be used.

While simulation languages provide most of the features needed in programming, a general-purpose language was selected to provide greater programming flexibility. Since Java is the preferred language of many P2P programmers it seems reasonable to use it as the host-language.

The simulator design is intended to provide easy manipulation of peer characteristics in detail without the user having previous knowledge of simulation implementation. The usability of the simulation is provided through the use of XML for data manipulations on the simulation. 3LS provides auto-saving of the states of the simulation into XML files. Hence, the result of the simulation can be reproduced with added monitoring/statistic analysis if necessary. And as the same XML file can be used as the simulation setup file, the file can easily be reconfigured for further simulation experiments. A detailed and complex simulation is both time- and resource-consuming. However, in order to observe the emergent behaviors of a P2P system, it is very important to have a large-scale simulation. Therefore, 3LS is designed with the possibility to adapt multi-threaded and multi-processes execution to accommodate a large number of peers.

4.1.1 Architecture

This study has identified three layers (network, protocol, and user model) as the main components of a P2P simulator. The Network layer should contain a desired **number of PCs** with the desired hardware specifications (**processor delay**) and physical network links (**network delay/latency and network bandwidth**) of the system. Note that the physical network is different from the **overlay network** of a peer system. Physical network refers to the hardware connectivity of the computer nodes. The overlay network refers to the connectivity of the application peers. Even though the computer nodes may physically be connected and located close to each other, the peers on the connected computer nodes may not know of each other and hence may not be connected to each other. The overlay network connection information is stored on the Application/Protocol

layer. This layer also represents peer node behavior with specific content distribution and tasks scheduling. Each peer is a peer application with various peer **behaviors** using a certain **protocol**. The User Model layer focuses on the **resource distribution** and simulates the different user behaviors with a certain **frequency** of specific **resource requests**.

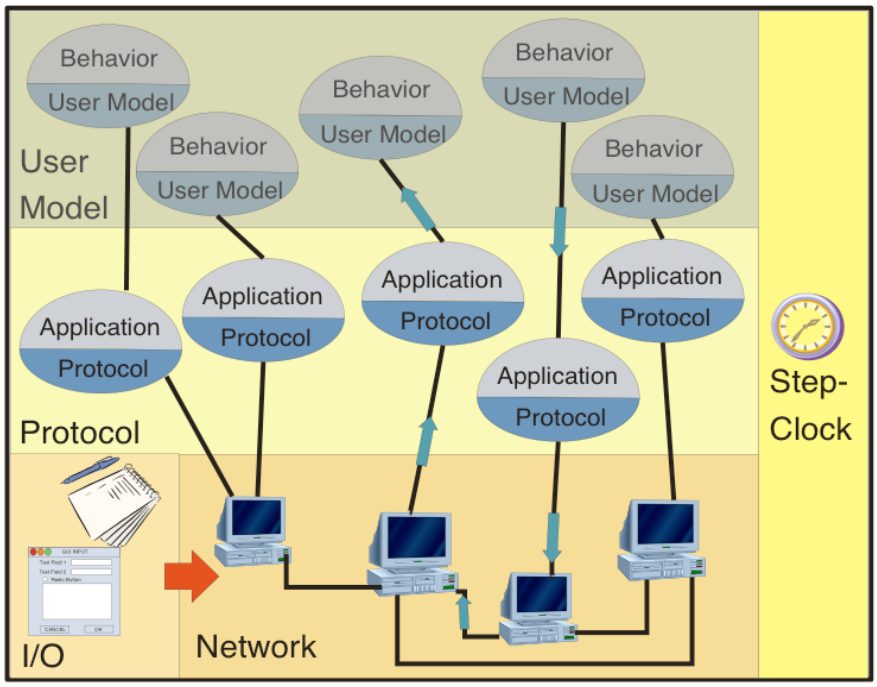

Figure 4.1. Architecture of the P2P Simulator

Following the desired criteria to simulate protocol/application, user presence/behavior, network, and hardware specifications as identified in Chapter two, 3LS [35] has the architecture shown in Figure 4.1. By separating the network, protocol/application, and users from each other, the simulation of various network topologies, for different protocols, applications, and user models becomes possible. A global clock (called step-clock in the thesis) is used as the counter for the simulation time.

There are two assumptions made to simplify the implementation of simulation:

- There is no time zone difference in the different locations of the network.

- The physical machine will not change the IP address in the system; however, the application peer may hop from one host to the other.

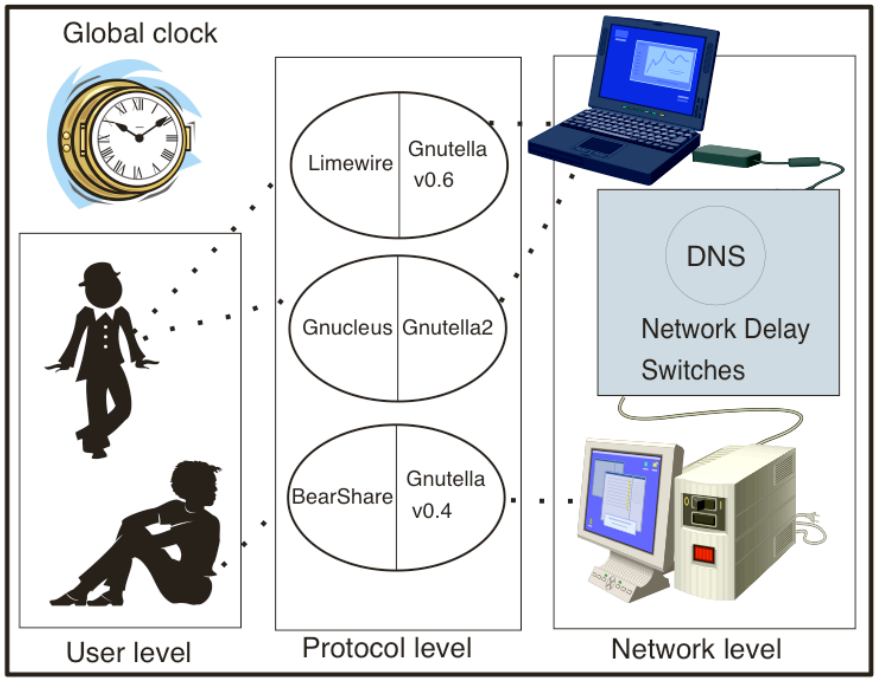

Figure 4.2. A Possible Simulation Example on 3LS.

Communication in 3LS can only happen between the directly connected layers. The Protocol layer, which is responsible for simulating the protocol of a desired application, acts as the interface between the User Model layer and the Network layer. The following sections describe the three layers of 3LS in detail. Section 4.1.1.1 describes the User Model layer, Section 4.1.1.2 describes the Protocol layer, and Section 4.1.1.3 describes the Network detail. Information that contains the initial settings of the simulation is fed into the Network layer using input files in XML format. All the events (such as issueing a Query message) are inputs from the user upon the initiation of the simulation. During the simulation, there are 3 types of output files: the current simulation state file in XML format (which can be used as an input file to repeat the simulation), the current simulation state file for visualization (refer to Section 4.1.3), and the simulation event files, which record the events that occurred in the simulation with time stamps in XML format. The simulation event files can then be parsed and analyzed.

4.1.1.1 User Model Layer

This layer models the user behaviors that tend to affect the overall network. The location of the resources (i.e., the file) shared by the user, for example, can greatly affect the overall file distribution of the system and ultimately the accessibility of the file. The uptime/downtime of a peer can be modeled in this layer because it affects the connection and availability of the file in the system; however, in this research, the peers do not leave the network after successfully joining the network.

Another user behavior in this layer that has great impact on the system is the frequency of the file requests. Note that the event of generating a single file request can cause multiple Query messages to be fired, replicated, and propagated in the network. In 3LS, a list of tasks is fed to the peer. The task could be turning the simulated peer on or off, or firing a file request for a specified resource. Each task is associated with a timestamp that states when the task should be executed.

4.1.1.2 Protocol Layer

Gnutella is an open file sharing network with a publicly specified protocol. Most of the Gnutella peers in the network are open source Gnutella clients like LimeWire, Gtk-Gnutella [13], and Gnucleus [12]. However, a few Gnutella clients are closed source, such as Xolox and BearShare [5]. Though each Gnutella client application has followed the basic public protocol specifications, additional features provided by each application make a difference in the peer behavior. Even the different settings of the same application with the same feature may make a difference in the emergent Gnutella peer network. Hence, the protocol layer enables the simulator to model the applications with certain settings using the protocols.

This layer serves as an interface layer in between the Network Layer and the User Model Layer. Events scheduled in the Protocol layer are bound to the application's tasks such as issuing Pings or replying to received Pings and Query messages with Pongs and QHs correspondingly. It also provides a schedule of repeating tasks like a thread in an application.

Each application node on a single computer node is differentiated using the port number associated with the host computer node, which has a certain IP address. The combination of IP address and port number is a unique key that can identify the application node. Hence, the application node keeps track of a table with the unique key of the directly connected peers. By exchanging and injecting the two peers' unique keys into each other's connection tables, it can be said that there is a connection established between the peers. With this, the overlay network topology of the Gnutella network can be determined.

4.1.1.3 Network Layer

The Network layer is responsible for modeling physical network aspects deemed relevant for the simulation, e.g., varying network load due to increased P2P communication. Due to assumption 1 in Section 4.1.1, there is no local clock on each PC. They all share the same simulation time through the use of global clock.

In this layer, there is an IP address registry that functions as a DNS server. The user can specify a number of subsets to divide the networks into groups. There is a two-dimensional matrix for keeping track of the physical network delay between the subgroup, and within each subgroup, there is one individual two-dimensional matrix for keeping track of the physical network delay between the simulated hosts (computer node).

The Network layer simulates a user-defined number of computer nodes. Each node in the Network layer represents a computer with a user-defined hardware specification to simulate the processor delay. Each node is a potential host for the application peers. Nodes keep track of the application nodes in the Protocol layer with the port number (see Figure 4.7). Nodes also keep a list of the open connections from the local application nodes to the remote application nodes on the other host. Interactions between the Network layer and the Protocol layer are made through referencing the application node in the Protocol layer obtained from its hash table.

Each node with the user-specified characteristics also simulates available bandwidth on the host. In the simulation, for simplification, only the uplink bandwidth is being simulated. The bandwidth of a host is shared between the communication objects. Hence, connection bandwidth = (total available bandwidth on the host) / (number of communication objects that are being uploaded from the host). Each of the message objects is pre-assigned with an estimated unit of bandwidth consumption size.

4.1.2 Message delivery process

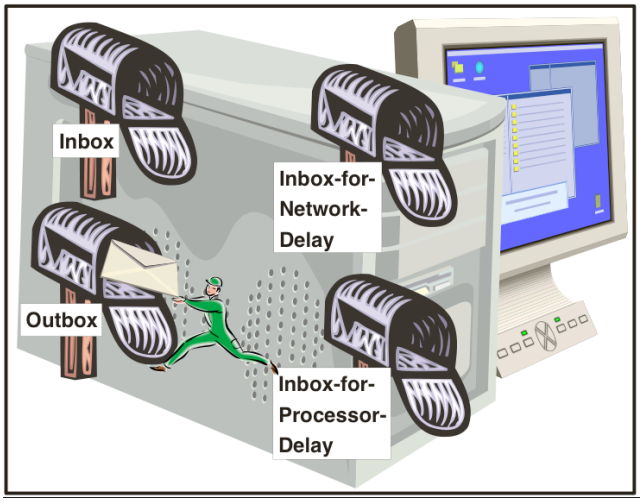

Figure 4.3. Peer Sending Message Through Outbox.

Each node contains four queues for storing the message object (see Figure 4.3):

- Outbox,
- Inbox-For-Network-Delay,
- Inbox-For-Processor-Delay, and
- Inbox.

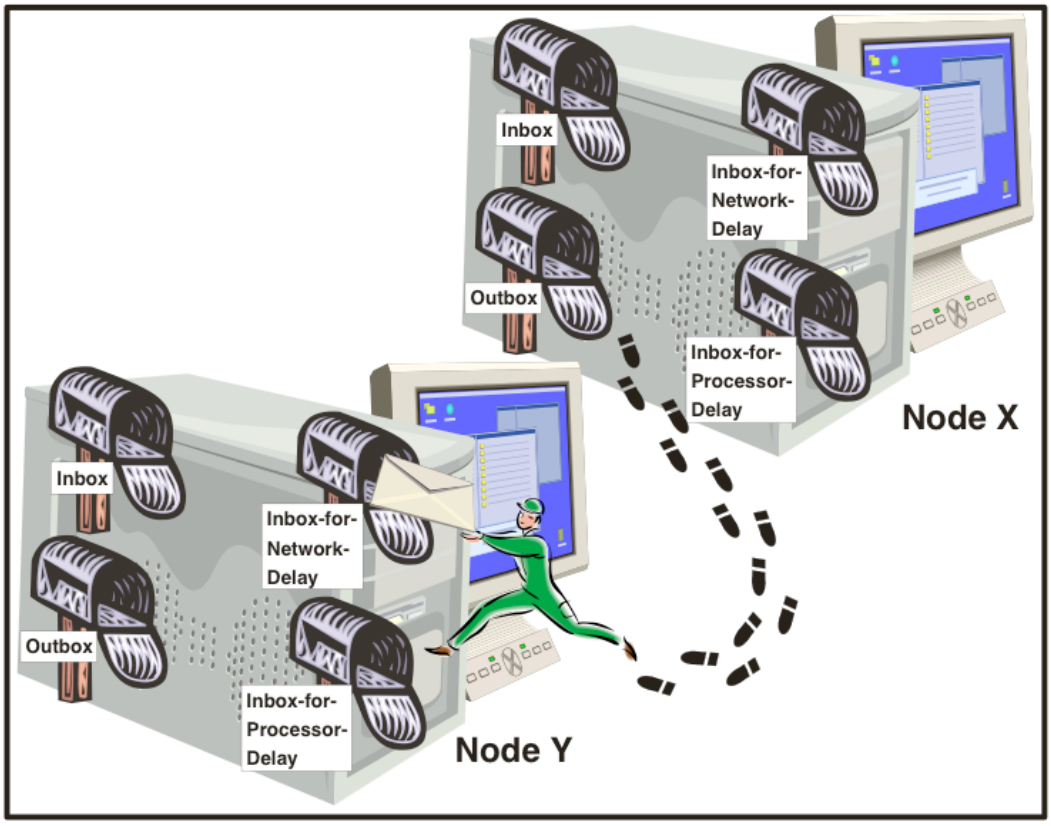

Figure 4.4. Node X Sending Message to Node Y.

When the application node sends a message object from computer node X to another application node at computer node Y, the following steps will be executed:

i. The message object (with time-stamp) that is stored in the Outbox of the computer node X is obtained by the worker. The available uplink bandwidth of the host is shared between the outgoing message objects. **A** message object may be delivered to the Inbox-For-Network-Delay at the destination node Y (see Figure 4.4) if all the units of the message object have been transferred to the destination node or have stayed in the Outbox for more unit transfer in the next time-step.

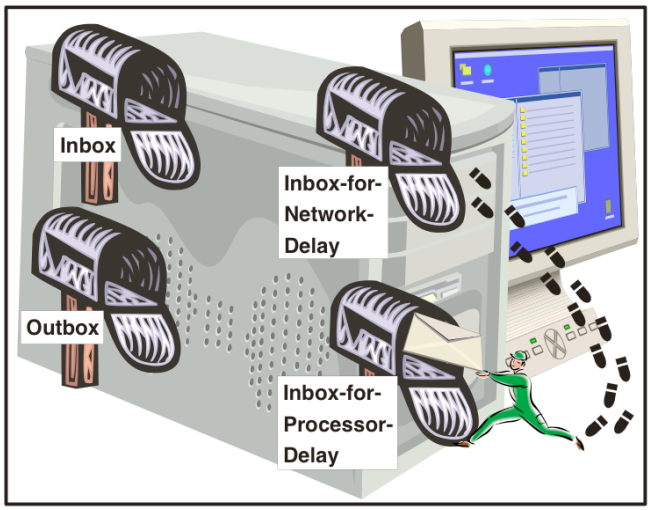

Figure 4.5. Node Received Message from the Network.

ii. When the delivery process is done, the step-clock is incremented and the message objects in the Inbox-For-Network-Delay are considered for the network delay simulation. If the network delay (the total value of the network delay in 2-D distance matrix and the congestion network delay) has been reached, the message is stored in the Inbox-For-Processor-Delay, with another time-stamp, or else it remains in the Inbox-For-Network-Delay (see Figure 4.5).

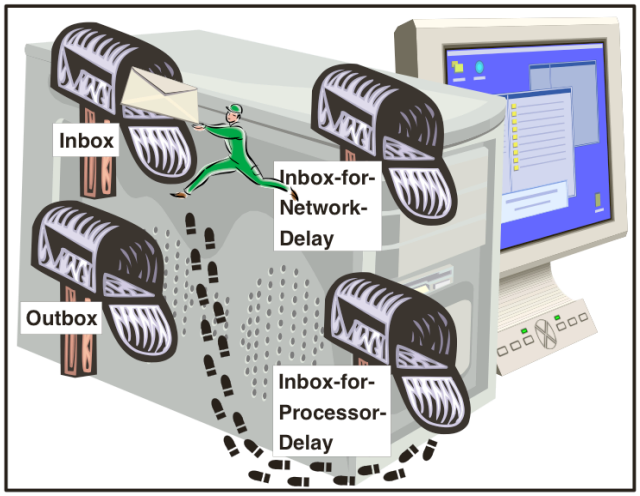

Figure 4.6. Node Received and Processed the Message.

iii. When the step-clock is incremented, the main thread of the simulator will look at the processor delay of the computer node Y and check whether the message object in the Inbox-For-Network-Delay should be moved into the Inbox for the application node to process (see Figure 4.6). If the processor delay has not been reached, the message object remains in the Inbox-For-Processor-Delay.

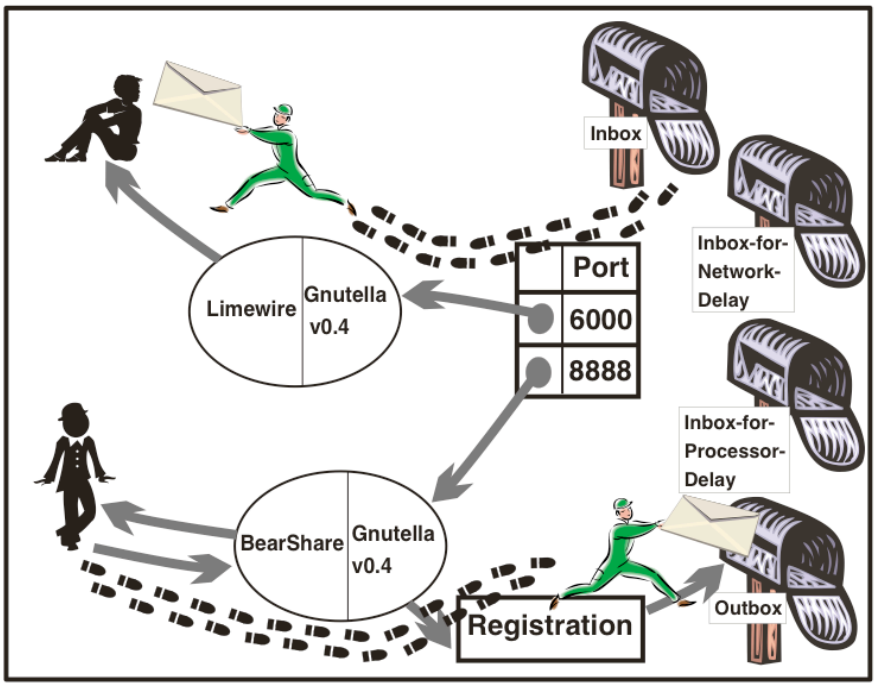

Figure 4.7. Peers Sending and Receiving Messages.

iv. When the step-clock is incremented again, the destination port number (encoded in the message object in the Inbox) will be obtained and the reference to the application node will be obtained to send the message object to the application node for processing (see Figure 4.7). After the application node processes the received message object, it will check the table that contains the scheduled task for execution. In this step, the peer can create and send a message object. Any created message object is sent through a Registration object that has the reference to the Outbox. The message object is stored in the Outbox and the iteration is repeated from (i) when the step-clock is incremented again.

The simulator follows a 2-step mechanism: each unit of user-time takes two step-times in the simulation (i.e., the step-clock is incremented twice). Hence, in the first step-time, iteration step ii and iv are executed. In the second step-time, iteration step i and iii are executed. With this design, the network delay and processor delay can be simulated without changing the order of the message arrival at each box. And most importantly, this enables several tasks (or events) to be carried out at any time. As the simulation runs, the events are saved into a file in XML format.

4.1.3 Visualization

Visualization of the network is done with the aid of AiSee [2]. There are a variety of graph visualization tools (such as Otter, GraphViz, and H3Viewer); however, AiSee has been chosen in this development for its wide-availability for multiple operating systems, easy installation, fast rendering, provision of various implemented functions, and finally, its free non-commercial usage. When a snapshot of the network is to be visualized, a file containing the information from the graph is written using the Graph Description Language (GDL). This file is generated with a specified interval during the simulation run 3LS. AiSee reads the GDL file and calculates the layout for displaying the graph.

Chapter 5

Experiments

Since 3LS has an internal step clock, each time-step of the time can be described as equivalent to simulating a user defined unit of time in the real word system. In the experiments, 1 second of the real world system is simulated by 10 time-steps in 3LS.

5.1 Terminology

5.1.1 User Model Layer

"Free-riders" or "Free-loaders" are peers who issue a lot of queries for resources, but have very little or no contribution to the resource pool of the P2P system. A study on the Gnutella system showed that there nearly 70% of the Gnutella users are free-riding [1]. On the other hand, a study [41] shows that about 75% of the Gnutella peers share 100 files or less (as many as 25% of the peers do not share any files), and only 7% of the peers share more than 1000 files.

Hence, in this thesis, there are three types of peers: contributing, low resource, and leech peers. To simplify the simulation setup, the experiments in this thesis use random file distribution. A **contributing peer** is a peer that has a high volume of files/resources to share. It randomly chooses 80% of the files from the central file pool and locates the files locally for sharing. A **low resource peer** is a peer that has low volume of files/resources (10% of the files) to share in the system, whereas a **leech peer** is a peer that does not share any files. Among the 100 peers being simulated in this thesis, 10 of the peers are contributing peers simulating the peers that share more than 1000 files in the system, 25 peers are leech peers simulating the 25% of the peers that share no files, while the rest of

the peers are low resource peers, simulating the peers that include less than 100 files from the system. To preserve the file distribution, only the initial files being distributed onto the peers are shared. Hence a file that is being downloaded during the experiment is not shared by the peer. There are 100 unique files in the experiments for this thesis.

The query frequency of a peer depends on the user's usage. An **active peer** is a peer that frequently initiates a file request for resources, a **moderate peer** is a peer that initiates file requests at greater intervals, whereas an **inactive peer** is a peer that does not issue any file requests at all. In the experiments, an active peer initiates a total of 20 file requests with an interval of 60 seconds. A moderate peer initiates a total of 6 file requests at an interval of 200 seconds. The first request is initiated at time 9.1 seconds. The file being queried by a peer is chosen randomly. It is a file that exists in the system but is not located on the peer who is initiating the file request.

5.1.2 Protocol Layer

In the Protocol Layer, there are many settings that can be studied in the experiments. The more common factors used in the studies are the TTL of a Query message and the settings on the caching of QH (size and expiry time). Most of the experiments conducted in the P2P world ignore the Ping/Pong scheme. Hence, in this thesis, the Ping/Pong scheme has been chosen to be a factor for experimentation in the protocol layer. Comtella [47] was developed at the University of Saskatchewan as a P2P file sharing system that enables Computer Science students to share papers. Due to the author's access to the Comtella system, it was chosen as the protocol to be simulated in this thesis.

A **flooding peer** is a peer that frequently sends pings to the network. A **non-flooding peer** has a lower pinging frequency. Because the implementation of the peer used in the experiments closely follows the implementation of Comtella file-sharing system, the setting of the pinging mechanism in the experiment closely corresponds to the real setting

used in the Comtella systems for the non-flooding peer. The non-flooding peer checks to maintain the desired number of connections every 10 seconds and sends ping messages to all the connected peers every 11 seconds. For a flooding peer, the peer maintains the connections every 5 seconds and sends ping messages every 5.5 seconds. The first ping message for each peer is initiated at timestamp 0.1 second and the connection maintenance thread starts at times stamp 0.2 seconds.

5.1.3 Network Layer

On the network layer, the hardware specification of a PC that is hosting the peer can greatly affect the speed of the peer processing the messages. A **fast computer** simulates a powerful computer (such as a Pentium 4 Processor at 3.2GHz with 4G of RAM), which speeds up the message processing of the peer. A **slow computer** has poor hardware (such as a 386 Processor with 128M of RAM), which has a higher processor delay. Due to the pace at which technology is evolving, it is not easy to specify how short a delay should be for a fast computer, or how long a delay should be for a slow computer. In the experiments, a fast computer has a processor delay of 0.1 seconds, and a slow computer has a processor delay of 0.5 seconds.

Network latency and bandwidth are the two factors that affect the speed and capacity of message delivery from node to node. Latency is the amount of time a packet takes to arrive at one node from another. To simplify the simulation, network latency is not included as a variable in the experiment. Therefore, network latency is set as a constant with a value of 0.2 seconds. However, a network link that has low bandwidth limits the number of packets to be sent from one peer to another at a time. Hence, if the bandwidth is low and there are a lot of messages being sent through the network, the bandwidth will contribute delays to the propagation of the messages.

Bandwidth between two nodes depends on the type of the connection being used to link the nodes. If one of the connection links between the two nodes has low bandwidth, that link will be the bottleneck. The bandwidth of a network link is shared by all the nodes that are using the network link. The data rate for the bandwidth is usually measured in bytes per seconds. The Internet service providers usually have different data rate for upstream (sending of a packet from the node) and downstream (receiving a packet from the node) connection. The upstream data rate usually is lower than the downstream data rate. The data rate of a network connection varies for different connection types, such as a dial-up modem (57.6 kbps); ISDN Digital Subscriber Line (128 kbps); Asymmetric Digital Subscriber Line or ADSL (1.544 to 6.1 Mbps for downstream, and 16 to 640 kbps for upstream); Rate-Adaptive Digital Subscriber Line, which is similar to ADSL (640 kbps to 2.2 Mbps for downstream, and 272 kbps to 1.088 Mbps for upstream); or Very High Digital Subscriber Line (12.9 to 52.8 Mbps for downstream, and 1.5 to 2.3 Mbps for upstream) [48].

To speed up the 3LS processing time, only upstream bandwidth is being implemented for simulation in this thesis. Each simulated packet in the thesis is set to simulate approximately 15 bytes. A **high bandwidth** node has a higher bandwidth limit and allows more packets per message to be sent at a time. In this thesis, a high bandwidth node allows 14000 simulated packets sent out from the node per 0.1 second in the simulation (approximately 2.1MBps). A **low bandwidth** node has limited upstream bandwidth, with 384 simulated packets sent out from the node per 0.1 second (57.6 kbps).

As discussed in Section 2.3.2.1, there are four basic types of Gnutella messages (Ping, Pong, Query, and QH) being simulated. The sizes of Query (>24 bytes, depending on the size of the search criteria) and QH (>46 bytes, depending on the number of result sets) messages are bigger than Ping (22 bytes) and Pong (35 bytes) messages. Each Ping and Pong message is estimated as 2 simulated packets. The size of the search criteria for the simulated Query messages is set as a constant, and each Query message is estimated as 4 simulated packets. Because the files on the peer are unique without duplication on the

peer and the file distribution is unchanged throughout the simulation, if a peer is generating a QH to a Query, the result set will always be one. Hence, the size of each QH message is being estimated as 6 simulated packets.

Apart from the basic Gnutella v0.4 messages, there are a few messages that are introduced in order to enable the simulation of connection establishment in the system. ***Message Connect*** is used to simulate the initiation of requesting a connection from a peer. When a peer receives a Connect message, it will either accept the connection by sending an ***OK message*** or reject the connection by sending a ***Bye message***. A Bye message is also sent when a peer is disconnecting from a connection, just like the function of a Bye message in Gnutella v0.6 as discussed in Section 2.3.1.2.

For downloading, a ***Request message*** is used as a request for download and a ***Response message*** is sent as a response to the peer saying the download is done. The Comtella client, which is being simulated in this thesis, is used for sharing web addresses. The downloading process of this Comtella client is not really a process of a file transmission; it is a notification message from the recipient of the web address, which is returned in the QH message. This thesis focuses more on the peer message propagation in the network rather than on the other messages. Hence, all the messages for simulating download and connection establishment are considered to be the size of 1 simulated packet in this simulation.

5.2 Experiment Setups

In this research, a series of simulations with different settings on the three layers is executed to investigate the importance of simulating details on the layers. Schlosser [42] has identified that there is no realistic measurement of the query rates of peers in a P2P network. In order to compare and identify the effect (if any) of the query rate on the

overall P2P system performance, a different proportion of peers with the query rate defined in Section 5.1.1 is used. For the User Model Layer, there are three types of worlds with different User Setups (Distributed, Active, and Moderate), as specified in Table 5.1.

Table 5.1. The distributions of peers in different User Setups.

Resource Sharing	*Querying Frequency*	*Distributed World*	*Active World*	*Moderate World*
Contributing	***Active***	3	10	0
	Moderate	5	0	10
	Inactive	2	0	0
Low Resource	***Active***	20	65	0
	Moderate	32	0	65
	Inactive	13	0	0
Leech	***Active***	8	25	0
	Moderate	12	0	25
	Inactive	5	0	0

In the ***Distributed World***, for each category (contributing, low resource, and leech peers), 30% of the peers are active peers, 50% are moderate peers, and 20% of the peers are inactive peers. In the ***Active World***, all the peers are active, and in the ***Moderate World***, all the peers are moderately firing queries. Since the files are distributed randomly, the simulations are repeated three times. The total number of files being shared in the system, the file distribution, and the overlay network topology stay the same in the three simulation runs. The only difference is the pattern and distribution of the file requests being initiated.

For the Protocol layer and Network layer, there are eight setups as illustrated in table 5.2. For each scenario, the settings are applied to all the peers in the simulation, except for **the server** (with high bandwidth settings in all scenarios), which serves as the single entry point for Comtella peers to join the network. The server is a special peer that is first connected by all the Comtella peers that want to join the network. It sends Pings to the connected peers every 5.0 seconds, then puts the peers into a waiting list to be

disconnected. The server stores the Pong messages received in a cache. When a connected Comtella peer sends a Ping message to the server, the server will not forward the Ping message; instead, it will respond with 5 cached Pong messages so the Comtella peers may learn about each other's presence. Hence, the server does not stay in the Comtella network. In this server, there are two disconnecting schemes for the server during each Ping cycle:

- A **good server** disconnects all peers in the waiting list.
- A **faulty server** disconnects 50% of the peers in the waiting list.

Theoretically, the higher frequency of the pinging/connection maintenance should result in better system performance because the connections are established faster and the peers can have better access to the files in the systems, provided that bandwidth is not the bottleneck. Higher bandwidth enables higher message throughputs and faster Query message parsing, hence, the system performance is increased. With faster processor speed, the peers handle the messages faster, resulting in better system performance.

Table 5.2. The settings of the peers for Protocol and Network Layer.

	Ping/Connection Maintenance	*Bandwidth*	*Processor Speed*
Scenario 1	Non Flooding	High	Fast
Scenario 2	Non Flooding	High	Slow
Scenario 3	Non Flooding	Low	Fast
Scenario 4	Non Flooding	Low	Slow
Scenario 5	Flooding	High	Fast
Scenario 6	Flooding	High	Slow
Scenario 7	Flooding	Low	Fast
Scenario 8	Flooding	Low	Slow

Another aspect of the simulation that will be examined is the initial overlay network topology of the system. In the simulations, the system is dynamic; although no peers leave the network, the peers will try to maintain the pre-defined number of connections in the protocol. Hence, as time goes by, there are more connections established in the

network. There are two types of initial network topologies being observed; no initial network (initially no peer is connected) and Small World topology (each peer has 4 initial connected neighboring peers, where 5 is the maximum number of peers that a peer can have at any time). In this thesis, all the peers are started at the same time, and since they have the same protocol setting, the maintenance and pinging of the peers are all executed at the same time. In the real world, this is not the case, because not all the peers are initialized to join the network at the same time.

5.3 Analysis Metrics

In this research, five metrics are used to analyze the system performance. A file request is considered successful if at least one QH message is returned to the peer that initiated the file request. The ***Query Success Rate***, or ***Success Rate***, is the percentage of the successful queries over the total number of file requests being initiated. The higher the *Success Rate*, the better the system performance.

However, *Success Rate* is not the only metric that expresses the success of the resource search protocol. ***Query Coverage*** determines the success of search protocols in finding all the files in the network that match the search of a Query. If there are X copies of a file in the network, when a peer initiates a Query message for this file, it may get Y number of QH messages in return (where $X \geq Y$). Hence, the *Query Coverage* of a Query is said to be Y/X*100%. The *Query Coverage* of all the Queries are added together and divided by the total number of successful Queries to obtain the ***Average Query Coverage per Successful Query***, which is referred to as ***File Coverage*** in the rest of the thesis. The higher the *File Coverage*, the better the performance of the search protocol.

QH Time indicates the time elapsed between the initiation of the Query (file request) and the reception of a corresponding QH. For each file request, the query hit time of each

successful query is added up, and the ***Average QH Time of File Request*** is calculated for the file request. Then the *Average QH Time of File Request* for all successful file requests are being added up, and the ***Average QH Time per Successful*** **Query,** which is called the ***Average QH Time*** in the rest of the thesis**,** is obtained for the analysis of the simulation. The lower the value for the *Average QH Time*, the better the performance of the system is.

QH Count indicates the number of QH being received by the peer that initiated the corresponding query. The ***Average QH Count*** is equal to the sum of all *QH Count* of each query divided by the total number of Successful Queries. The higher the *Average QH Count*, the better the performance of the system.

QH Hop indicates the distance (how many hops away) between the querying peer and the peer that has the resources. For each Query, the ***Average QH Hop of File Request*** is calculated. Then the *Average QH Hop of File Request* for all successful queries are added up to obtain the ***Average QH Hop***. In this thesis, the greater the distance, or the higher the *Average QH Hop*, the better the system performance is because the query is propagated further.

5.4 Results

Using the experimental setups as described in Section 5.2, a number of simulations were carried out and grouped to produce a series of graphs to be analyzed in this chapter. The total number of simulations needed to produce the graphs (refers to Appendix A and B) for this chapter was calculated as shown below:

Total Number of Simulations

= (simulations to produce a set of graphs to be analyzed) * (# of Initial Overlay Network Topologies) * (# of Server Implementations)

= [(# of User Setup Worlds) * (# of Random Generated File Distributions) * (# of Combination Settings on Protocol and Network Levels)] * (# of Initial Overlay Network Topologies) * (# of Server Implementations)

= [(3 * 3 * 8)] * 2 * 2 = 288

5.4.1 System Performance over time

The simulation results of the 5 metrics described above are calculated with an interval of 100 seconds (1000 time-steps). The graphs of the results for each scenario over the simulated time are drawn for each user setup of each initial network topology. The complete collection of graphs drawn for analysis is available in Appendix A and Appendix B. Appendix A contains all the graphs for system performance over time with a faulty server while Appendix B contains all graphs for system performance over time with a good server. With these graphs, the system performance for each setting over time can be observed in detail. Table 5.3 shows a breakdown of the scheduled query initiation for the different worlds.

Table 5.3. Breakdowns of query initiation and timestamps for different worlds.

Time Interval	*Timestamp*	*Distributed World*	*Active World*	*Moderate World*
0.0s – 99.9s	***9.1s***	80 peers	100 peers	100 peers
	69.1s	31 peers	100 peers	0 peer
100.0s – 199.9s	***129.1s***	31 peers	100 peers	0 peer
	189.1s	31 peers	100 peers	0 peer
200.0s – 299.9s	***209.1s***	49 peers	0 peer	100 peers
	249.1s	31 peers	100 peers	0 peer
300.0s – 399.9s	***309.1s***	31 peers	100 peers	0 peer
	369.1s	31 peers	100 peers	0 peer
400.0s – 499.9s	***409.1s***	49 peers	0 peer	100 peers
	429.1s	31 peers	100 peers	0 peer
	489.1s	31 peers	100 peers	0 peer
500.0s – 599.9s	***549.1s***	31 peers	100 peers	0 peer
600.0s – 699.9s	***609.1s***	80 peers	100 peers	100 peers
	669.1s	31 peers	100 peers	0 peer
700.0s – 799.9s	***729.1s***	31 peers	100 peers	0 peer
	789.1s	31 peers	100 peers	0 peer
800.0s – 899.9s	***809.1s***	49 peers	0 peer	100 peers
	849.1s	31 peers	100 peers	0 peer
900.0s – 999.9s	***909.1s***	31 peers	100 peers	0 peer
	969.1s	31 peers	100 peers	0 peer
1000.0s – 1099.9s	***1009.1s***	49 peers	0 peer	100 peers
	1029.1s	31 peers	100 peers	0 peer
	1089.1s	31 peers	100 peers	0 peer
1100.0s – 1199.9s	***1149.1s***	31 peers	0 peer	0 peer

5.4.1.1 Active World

With a faulty server, the *Success Rate* for Small World topology is close to 100% for all scenarios as shown in Figure A.1. Figure A.3 showed fluctuation of the *Average QH Hop* over time with very slight overall improvement. The *File Coverage* (Figure A.2) and *Average QH Count* (Figure A.4) show great improvement over time, especially on the

flooding network. Figure A.5 showed no change in *Average QH Time* over time; however, it also showed the *Average QH Time* of all slow processor networks is more than double the value in the fast processor networks. The figures showed that processor speed is the bottleneck for this simulation of system performance.

With a good server, however, the graph of the *Success Rate* (Figure B.1) of Small World topology shows a very similar result with almost 100% *Success Rate* over time. Though the *File Coverage* (Figure B.2) and *Average QH Count* (Figure B.4) of both faulty and good server networks increase over time, the reading for a good server has lower range (e.g., 39-48% for *File Coverage*) than the network with a faulty server (e.g., 37-55% for *File Coverage*), showing that the faulty server have better system performance. The *Average QH Hop* (Figure B.3) and the *Average QH Time* (Figure B.5) of the network for both servers is similar.

With a faulty server and no initial network topology, *File Coverage* (Figure A.7) and *Average QH Count* (Figure A.9) improve greatly over time, which can be seen most vividly for flooding peers with slow processors, followed by flooding peers with fast processors, non flooding peers with slow processors, and finally, non flooding peers with fast processors. Because there is no initial network in the system, the network connections develop and evolve quickly. A flooding peer pings more frequently, and therefore, the network evolves faster and the system performance improves faster. With slow processors, the total time for a ping to be processed by other peers (especially the peers that are further away because the delays add up when the Ping message is being forwarded by other peers) is longer and therefore later in the simulation, the network is more developed and File Coverage is better.

The *Average QH Hop* (Figure A.8) and *Average QH Time* (Figure A.10) of a network with a faulty server and no initial network topology increase over time, especially for the first 300 seconds. The slow processor network at least doubled the *Average QH Time*

compared to the fast processor network. The graph shows that as time goes by and more connections have developed, the network is more congested with messages and hence the *Average QH Time* increases, especially with flooding peer networks.

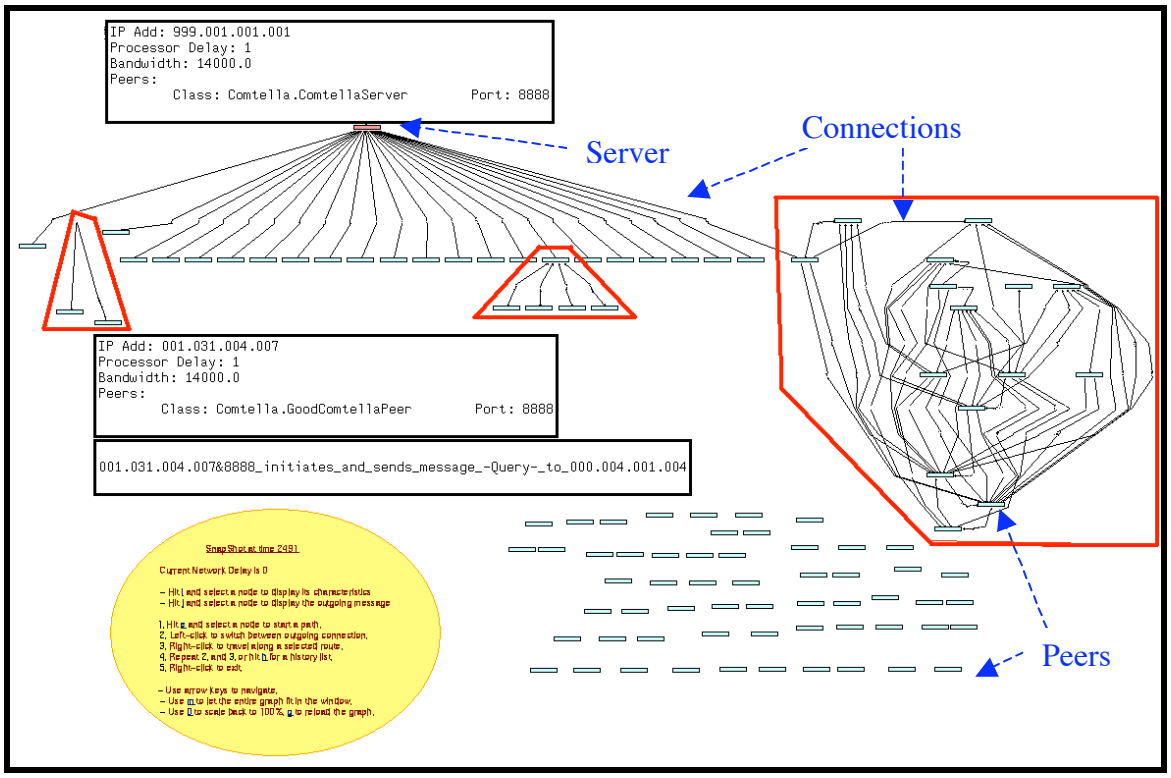

Figure 5.1. Snapshot of Active World at 249.1s without initial network topology (with faulty server and Scenario 1).

There is an interesting drop in system performance in the *Success Rate* (Figure A.6) for non-flooding peers. The *Average QH Hop* for those Queries from non-flooding peers are smaller than those from flooding peers, showing that the network is not well expanded/developed in the non-flooding network. In Table 5.3, the queries for time range 200.0 to 299.9 seconds are generated at time 249.1 seconds. Figure 5.1 shows that there are lots of disconnected peers (peers without any connection links or peers with only a connection to the server) and three disconnected Comtella networks for the *Active World* at time 249.1 seconds. In order for a Query to be issued from a peer, the peer must have at least one connection to another peer (not the server). In this particular simulation run, only 13 Queries have corresponding QH out of the 22 Queries issued. The two of the

disconnected Comtella networks are very small (2 peers and 5 peers relatively) and hence the 7 Queries issued by these peers have a very small chance of QH success rate. Figure 5.2 and Figure 5.3 show that the disconnected network doesn't exist at time 189.1 (the previous Query issue time before 249.1 seconds) and 309.1 seconds (the next Query issue time after 249.1 seconds). This explained the dramatic drops in performance.

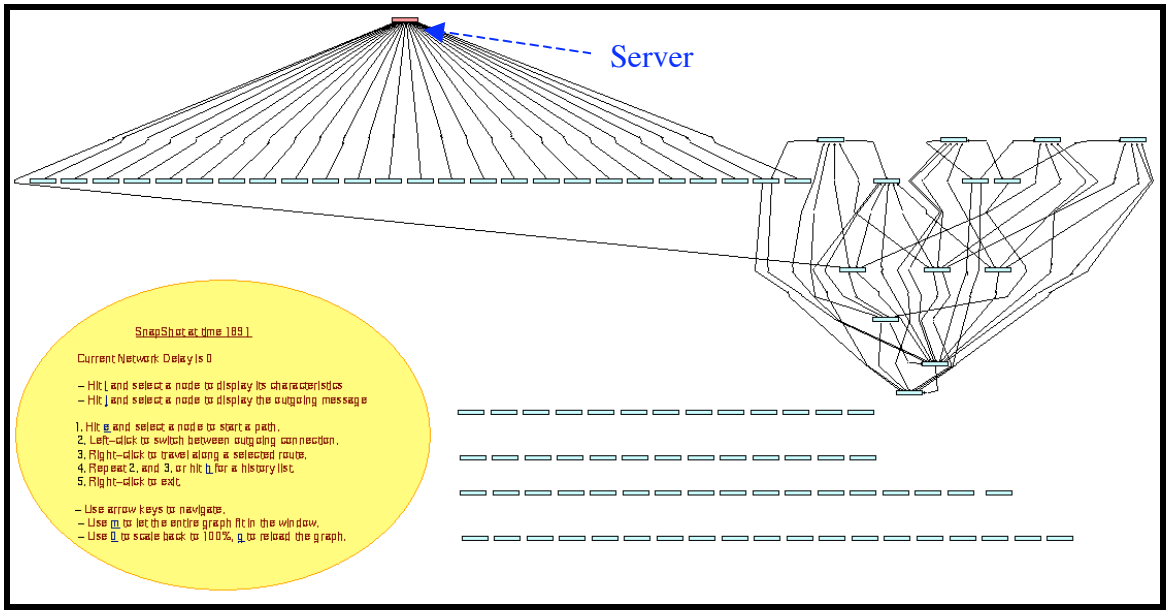

Figure 5.2. Snapshot of Active World at time 189.1 seconds without initial network topology (with faulty server and Scenario 1).

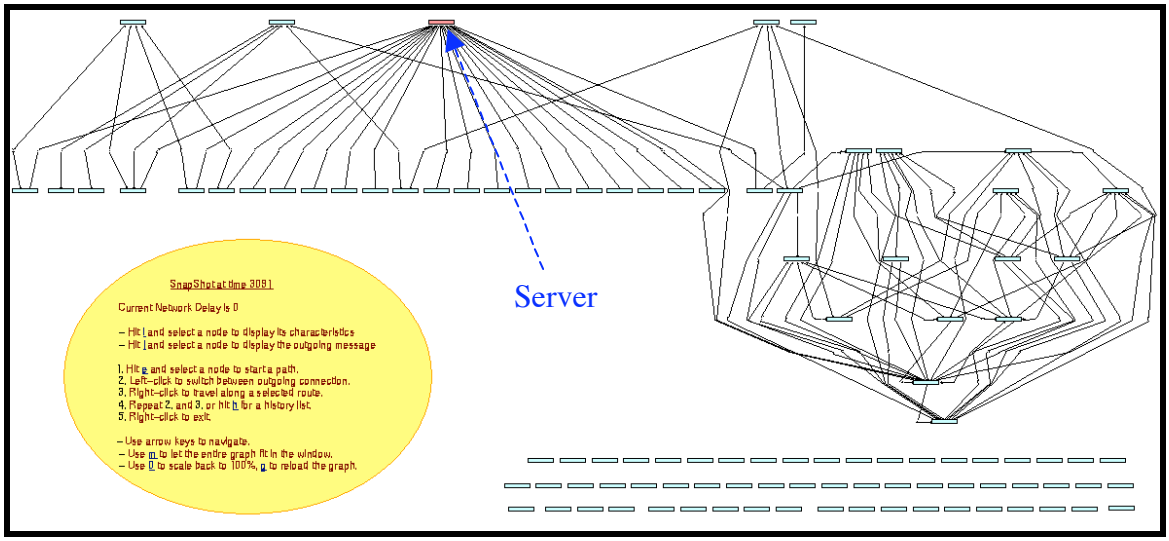

Figure 5.3. Snapshot of Active World at time 309.1 seconds without initial network topology (with faulty server and Scenario 1).

With a good server, though the *Success Rate* (Figure B.6) of the network without initial overlay topology has a lower reading for the first 100 seconds of the simulation, the dramatic drops in performance that was observed in the network with a faulty server does not happen with a good server. The graphs (Figure B.6-10) show that the system performance of the network with a good server is lower than the network with a faulty server. The network is less developed during the given simulation period with a good server because the peers are disconnected too quickly; they are disconnected from the server before the peers can send pings to the server for connections.

5.4.1.2 Moderate World

With a faulty server, the graph of the *Success Rate* for the *Moderate World* with Small World topology (Figure A.11) shows all settings have 100% *Success Rate* at all times. The *File Coverage* (Figure A.12) and *Average QH Count* (Figure A.14) for the *Moderate World* with Small World topology show improvement over time, which can most vividly be seen for flooding peers with slow processors, followed by flooding peers with fast processors, non flooding peers with slow processors, and finally, non flooding peers with fast processors (just like the *Active World* without initial network topology). The *Average QH Hop* (Figure A.13) increases over time for all settings, while the *Average QH Time* (Figure A.15) is unchanging with slow processor settings taking more than double the time of fast processor settings. Just as in the *Active World*, the graph of the Small World overlay network topology with a good server (Figure B.11-15) shows slightly lower performance than the network with a faulty server. However, the graph of the *Success Rate* (Figure B.11) for the *Moderate World* with Small World topology shows no decrement in performance and has 100% *Success Rate* at all time for all settings.

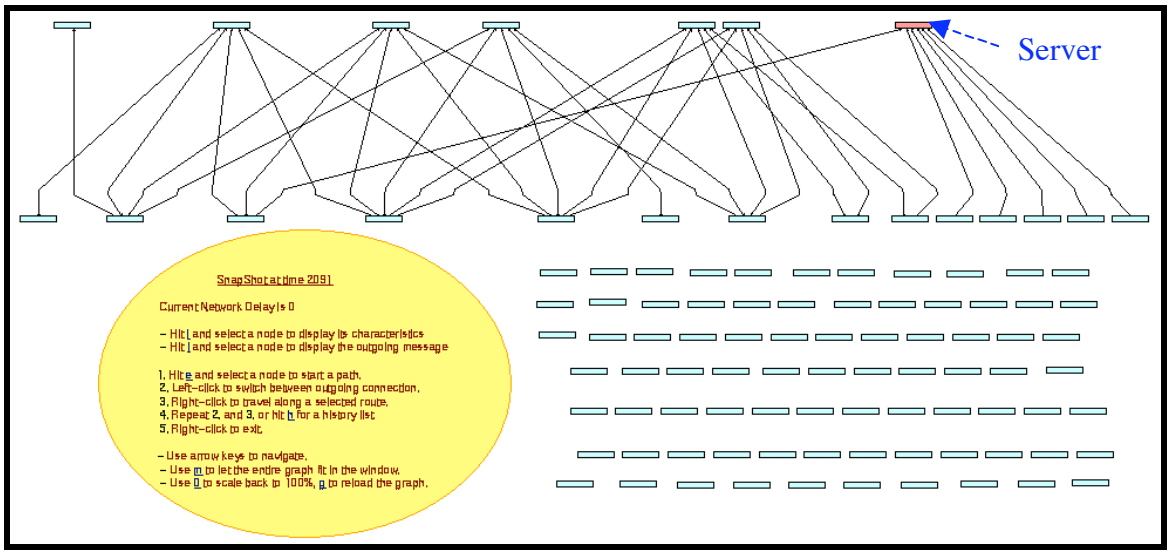

Figure 5.4. Snapshot of Moderate World at time 209.1 seconds without initial network topology (with faulty server and Scenario 1).

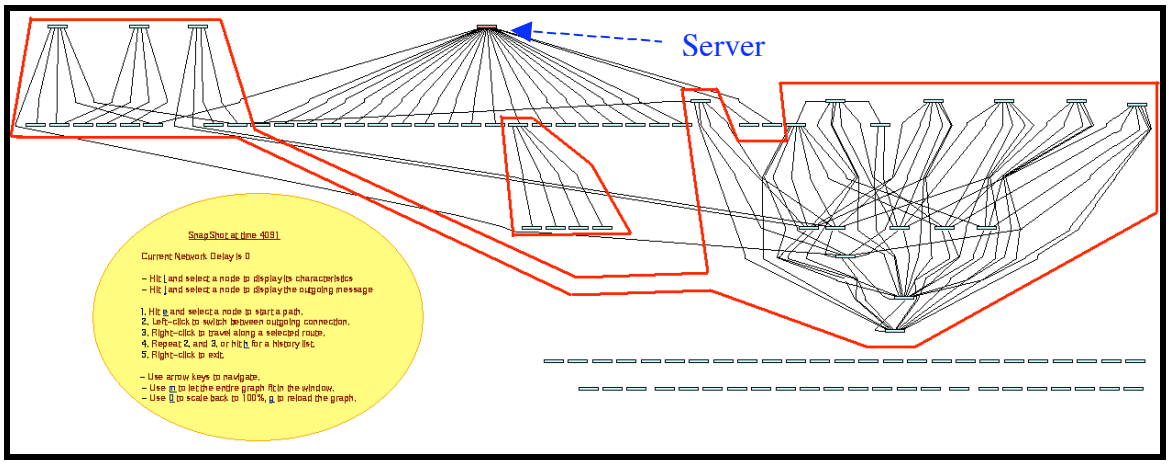

Figure 5.5. Snapshot of Moderate World at time 409.1 seconds without initial network topology (with faulty server and Scenario 1).

The *File Coverage* (Figure A.17), *Average QH Hop* (Figure A.18), *Average QH Count* (Figure A.19), and *Average QH Time* (Figure A. 20) of the network with a faulty server and no initial network topology increase over time. Once again, the *Average QH Time* for the slow processor network uses twice the amount of time compared to the fast processor network. Without an initial network topology, the *Success Rate* (Figure A.16) fluctuates over time and the result is inconclusive. For the non flooding network, there is a slight drop in the *Success Rate* at time interval 400 to 499.9 seconds (unlike the *Active World* where the drop is at 200 to 299.9 seconds). For the *Moderate World*, the Queries are

issued at time step 209.1 seconds, where the overlay network is a single connected network (Figure 5.4); while for interval 400 to 499.9 seconds, the Queries are issued at time step 409.1 seconds in two disconnected networks (Figure 5.5).

For the network with a good server and no initial overlay network topology, the graphs (Figure B.16-20) show lower performance than for the network with a faulty server, as was observed in the *Active World*. The drop in performance at time interval 400-499.9 seconds with a faulty server in a non-flooding network does not occur in the network with a good server. However, the performance drops for the network setting of flooding peers with fast processors are observed at time interval 400-499.9 seconds with a good server, which is not observed in the network with a faulty server.

5.4.1.3 Distributed World

The *Success Rate* (Figure A.21) for the *Distributed World* with Small World topology stays close to 100% throughout the simulation period with a faulty server, while the graph of the *Average QH Hop* (Figure A.23) shows fluctuations around 3 hops for all settings. Just like in the *Active* and *Moderate Worlds*, the *Average QH Time* (Figure A.25) for the slow processor network uses twice the amount of time compared to the fast processor network. The *File Coverage* and (Figure A.22) *Average QH Count* (Figure A.24) increase over time. Once again, the network with a good server (Figure B.21-25) shows similar performance as the network with a faulty server with a lower range of readings, while the *Success Rate* still stays close to 100% for both settings.

Without initial network topology and with a faulty server, the *Success Rate* (Figure A.26) shows a mixture of results in the *Active* and *Moderate Worlds*, where the non-flooding network has a slight drop at interval 200-299.9 seconds and 400-499.9 seconds when the network is disconnected. As time goes by, the network grows, resulting in an improvement in the *File Coverage* (Figure A.27) and *Average QH Count* (Figures A.29).

The *Average QH Hop* (Figure A.28) and *Average QH Time* (Figure A.30) of the network without initial overlay network topology increase over time, especially for the first 300 seconds. For the network with a good server and no initial overlay network topology, the graphs (Figure B.26-30) show lower performance than for the network with a faulty server.

5.4.2 Overall System performance

To observe the impact of the user, protocol, and network factors on the performance of the simulated system, the results of the 5 metrics are calculated over the whole simulation period for each initial network topology. To easily compare the performance of the simulation runs, the simulation result of scenario 8 (flooding, low bandwidth, slow processors) with *Moderate World* setting is chosen as the control set. The following performance value is obtained for each set of simulations:

Performance value of set X compared to control set (in %)

= (value of set X – value of control set) * 100 / value of control set,

where X = 1, 2, 3, 4, 5, 6, 7, 8 for all User Setup Worlds

A positive performance value (e.g., 30) indicates a better system performance, with an improvement of 30% compared to scenario 8, and a negative performance value (e.g., -30) indicates the system performance decreases 30% compared to scenario 8.

As illustrated in Table 5.2, there are 8 scenarios that can be compared with each other. Table 5.4 shows the 4 possible comparisons derived from the scenarios in Table 5.2. To observe the effect of the protocol layer with the frequency of pinging/connection maintenance, the performance value of *Scenario 1* is compared to *Scenario 5*, then

Scenario 2 vs. *Scenario 6*, *Scenario 3* vs. *Scenario 7*, and *Scenario 4* vs. *Scenario 8*. To observe the impact of bandwidth, the performance values are compared in the order of *Scenario 1* vs. *Scenario 3*, *Scenario 2* vs. *Scenario 4*, *Scenario 5* vs. *Scenario 7*, *Scenario 6* vs. *Scenario 8*. To observe the impact of processor speeds on the overall performance, the performance values are compared in the order of *Scenario 1* vs. *Scenario 2*, *Scenario 3* vs. *Scenario 4*, *Scenario 5* vs. *Scenario 6*, *Scenario 7* vs. *Scenario 8.* And to observe the impact of User Setup Worlds, comparison is done between the corresponding scenarios of the different User Setup Worlds (e.g., *Scenario2* of *Distributed World* vs. *Scenario2* of *Active World*).

Table 5.4. Comparisons of Scenarios for different factors.

Ping/Connection Maintenance	***Bandwidth***	***Processor Speed***
Scenario 1 vs. Scenario 5 (Non Flooding vs. Flooding), High Bandwidth, Fast Processor	Scenario 1 vs. Scenario 3 Non Flooding, (High vs. Low Bandwidth), Fast Processor	Scenario 1 vs. Scenario 2 Non Flooding, High Bandwidth, (Fast vs. Slow Processor)
Scenario 2 vs. Scenario 6 (Non Flooding vs. Flooding), High Bandwidth, Slow Processor	Scenario 2 vs. Scenario 4 Non Flooding, (High vs. Low Bandwidth), Slow Processor	Scenario 3 vs. Scenario 4 Non Flooding, Low Bandwidth, (Fast vs Slow Processor)
Scenario 3 vs. Scenario 7 (Non Flooding vs. Flooding), Low Bandwidth, Fast Processor	Scenario 5 vs. Scenario 7 Flooding, (High vs. Low Bandwidth), Fast Processor	Scenario 5 vs. Scenario 6 Flooding, High Bandwidth, (Fast vs. Slow Processor)
Scenario 4 vs. Scenario 8 (Non Flooding vs. Flooding), Low Bandwidth, Slow Processor	Scenario 2 vs. Scenario 4 Flooding, (High vs. Low Bandwidth), Slow Processor	Scenario 7 vs. Scenario 8 Flooding, Low Bandwidth, (Fast vs. Slow Processor)

However, as this thesis is not trying to answer the question of "Which experiment settings has the best performance?", the comparison is done by looking at the absolute value of the difference between the performance values of the two scenarios. All the comparisons are done within the same initial network topology using the same server behavior. The minimum and maximum of the four comparisons are obtained for each scene/factor (User Setup, Protocol, and Bandwidth) with each analysis metric. Tables for the comparison results are drawn below along with the analysis.

Table 5.5 shows that for a network with Small World topology as initial topology and a faulty server, protocol appears to have the most impact on the system performance in *Average QH Count*, *Average QH Hop*, and *File Coverage*. Processor speed has the most impact on *Average QH Time*. For the *Success Rate*, User Setups appear to have the more impact on the system, thought the value is small (less than 1%). While Table 5.6 shows that with a proper server, the Protocol is not the factor that affects the server performance most. User Setups appear to have a bigger impact on network performance in the *Success Rate* and *Average QH Hop*. Processor delay is the factor that affects the system performance most in the *Average QH Count*, *Average QH Time*, and *File Coverage*.

Table 5.5. Difference of performance value for Small World topology with a faulty server.

Scene		Success Rate	Average QH Count	Average QH Hop	Average QH Time	File Coverage
User Setups	Min	0.02706054	0.584190939	0.002287872	0.00228485	0.064716204
	Max	**0.1**	3.324170119	0.51447384	0.570682271	3.025422947
Protocol	Min	0	3.472729386	0.117463933	0.117308772	3.47321285
	Max	0.03646973	**7.867748442**	**1.031941133**	0.511598553	**7.934227756**
Bandwidth	Min	0	0	0	0.000671465	0
	Max	3.335E-05	0.261422704	0.083422769	0.162117492	0.251514819
Processor	Min	0	0.986927539	0.305099102	56.5402984	1.048010998
	Max	0.03646973	2.773220056	0.583312514	**57.13252836**	2.933848171

Table 5.6. Difference of performance value for Small World topology with a good server.

Scene		Success Rate	Average QH Count	Average QH Hop	Average QH Time	File Coverage
User Setups	Min	0.022922785	0.031924669	0.008339061	0.007777575	0.272777858
	Max	**0.07293946**	2.427323059	**0.695290154**	0.777362443	2.480757919
Protocol	Min	0	0.714763663	0.06289211	0.029276323	0.702041194
	Max	1.6675E-05	1.853277818	0.162023149	0.161883748	1.82325083
Bandwidth	Min	0	0	0	0.002929249	6.76126E-14
	Max	1.6675E-05	0.468647483	0.039786748	0.125635151	0.468566107
Processor	Min	0	1.854473741	0.181828102	56.48771751	1.818360944
	Max	1.6675E-05	**3.064954541**	0.51479155	**57.06532432**	**3.006222549**

Table 5.7 shows that without an initial network topology, User Setup has the most impact on *Average QH Hop*. Protocol has the most impact on *Average QH Count* and *File Coverage*. Processor delay has the most impact on *Success Rate* and *Average Query Hit Time*, while Table 5.8 shows that the protocol has the most impact on the performance for

Average QH Count, *Average QH Hop*, and *File Coverage*, and the processor delay has the most impact on *Success Rate* and *Average QH Time*.

Table 5.7. Difference of performance value without initial network topology with a faulty server.

Scene		*Success Rate*	*Average QH Count*	*Average QH Hop*	*Average QH Time*	*File Coverage*
User Setups	***Min***	*0.064729505*	*0.087932685*	*0.367007816*	*0.157141025*	*0.147250225*
	Max	*1.610515733*	*3.441647511*	***4.552116638***	*2.294501286*	*4.55895856*
Protocol	***Min***	*0.20748746*	*7.937878098*	*0.735440508*	*0.314892136*	*7.135899838*
	Max	*2.315881658*	***21.44467346***	*3.728748929*	*3.554561144*	***20.89343078***
Bandwidth	***Min***	*0*	*0*	*0*	*0*	*0*
	Max	*1.16999903*	*3.623413484*	*0.790403153*	*0.291512345*	*3.672356496*
Processor	***Min***	*0.453377007*	*4.644430204*	*0.309278174*	*55.56613337*	*5.029881982*
	Max	***2.912465948***	*15.90381287*	*3.811241835*	***58.50532421***	*16.21373496*

Table 5.8. Difference of performance value without initial network topology with a good server.

Scene		*Success Rate*	*Average QH Count*	*Average QH Hop*	*Average QH Time*	*File Coverage*
User Setups	***Min***	*0.052943139*	*0.765571793*	*0.249561823*	*0.106955067*	*0.10493707*
	Max	*6.161723452*	*8.135341704*	*10.34715703*	*10.34715703*	*8.261267106*
Protocol	***Min***	*0.085027166*	*7.646463536*	*1.770663815*	*0.758855921*	*7.449052444*
	Max	*5.719511189*	***22.64893344***	***10.57015149***	*10.57015149*	***22.56348961***
Bandwidth	***Min***	*0*	*0*	*0*	*0*	*0*
	Max	*5.88932116*	*8.215966569*	*6.538512256*	*6.538512256*	*8.423304031*
Processor	***Min***	*0.131863643*	*0.922559397*	*2.017512436*	*48.464813*	*1.068119338*
	Max	***6.291594633***	*10.27233131*	*5.6464472*	***59.56276309***	*10.61412606*

5.5 Conclusion

From the analysis done in Section 5.4.1, it can be seen that as time goes by, the network evolves and becomes more well-connected, hence the system performance increases. It can be seen that with different disconnection schemes on the server, the performance of the system is different, especially for networks without initial overlay network topology, which rely on the server to form the connections for the network. At any time, if the network is disconnected, the system performance (i.e., *Success Rate*) drops. Consequently, in a dynamic network, the overall system performance depends on the state of the network (which is different at different points in time) from which the

Queries are being issued. This can be concluded by looking at the results of the three different worlds.

Section 5.4.2 shows that though the simulated system of Small World initial overlay network topology (Table 5.5) begins with nearly saturated connections (4 out of 5 connecting peers for each peer), the protocol layer (i.e., the frequency in the pinging/connection maintenance scheme) still has the most impact (< 8%) on the system with a faulty server for most analysis metrics (except for *Average QH Time,* which is most affected by the Processor with a 57% performance change). However, with a good server, the system performance of the simulated system with *Small Word* initial overlay network topology (Table 5.6) is most affected by User Setups (< 3%) and Processor delay (57%).

Section 5.4.2 also shows that without initial network topology (Table 5.8), the system performance of the network with a good server is affected mostly by the Protocol (< 22.6%) and Processor (6-59.6%), depending on the analysis metrics that are being used. However, the system performance of the network with a faulty server and no initial network topology (Table 5.7) shows that the User Setup (< 5%), Protocol (<21%), and Processor speed (3-58.5%) are all important factors because it is inconclusive which factor has the most impact on the system since the result depends on the analysis metrics being chosen.

As the hardware quickly evolves and becomes more affordable, it will be interesting to see what may happen to the P2P network with powerful machines with fast processors or adequate bandwidth. Hence a series of experiments were repeated with the settings as described in Section 5.1, except the fast computer has 0.0 seconds delay and the slow computer has 0.1 seconds delay. Table 5.9 shows that with the improved processors and a faulty server, the system performance of the network without initial network topology is most affected by Protocol (<22.8%, rather than <8% with the original settings), except

the *Average QH Time* is most affected by the Processor (in the range of 19.9-31.6%, rather than the range of 55.6-58.5% with the original settings).

Table 5.9. Difference of performance value without initial network topology with a faulty server and improved processors.

Scene		*Success Rate*	*Average QH Count*	*Average QH Hop*	*Average QH Time*	*File Coverage*
User Setups	***Min***	*0.112062489*	*0.008048645*	*0.080484*	*0.29717166*	*0.008976405*
	Max	*5.450864976*	*5.720563727*	*7.943057219*	*11.01857991*	*7.15031167*
Protocol	***Min***	*0.036139759*	*14.20054534*	*3.054040024*	*2.224031669*	*13.11723987*
	Max	***5.877325451***	***22.82672336***	***9.960937068***	*14.48636821*	***22.76145894***
Bandwidth	***Min***	*0*	*0*	*0*	*0*	*0*
	Max	*5.044814169*	*4.758489501*	*5.031732137*	*9.425870838*	*4.806637122*
Processor	***Min***	*0.067987674*	*0.127997387*	*1.225797392*	*19.88809854*	*0.17203965*
	Max	*5.446411824*	*4.630492114*	*3.321965912*	***31.61658726***	*4.634597472*

Table 5.10. Difference of performance value without initial network topology with a good server and improved processors.

Scene		*Success Rate*	*Average QH Count*	*Average QH Hop*	*Average QH Time*	*File Coverage*
User Setups	***Min***	*0.097396315*	*0.773914627*	*0.258280106*	*0.258280106*	*0.281952249*
	Max	*2.776078802*	***14.66773776***	***9.550557423***	***9.550557423***	***13.10209525***
Protocol	***Min***	*0.179589401*	*0.177713779*	*0.006122538*	*0.006122538*	*0.120312109*
	Max	***4.530021274***	*12.72574019*	*5.149383827*	*5.149383827*	*13.05987155*
Bandwidth	***Min***	*0*	*0*	*0*	*0*	*1.7164E-13*
	Max	*3.014978428*	*8.951272044*	*5.123619477*	*5.123619477*	*9.056087187*
Processor	***Min***	*0*	*0*	*0*	*0*	*0*
	Max	*2.679980825*	*8.951272044*	*5.123619477*	*5.123619477*	*9.056087187*

However, with improved processors and a good server (Table 5.10), the system performance of the network without initial network topology is most affected by User Setups (<14.7) including *Average QH Time*, except the *Success Rate* that is most influenced by the Protocol (4.5%), whereas with the original settings, the Protocol is the most influential factor (<22.6%) followed by the Processor (6.3%-59.6% depending on the metrics chosen).

Table 5.11 shows that with improved processors and a faulty server, the system performance of the network with Small World network topology is most affected by Protocol (<7.8%, rather than <7.9% with the original setting), except for Success Rate

(<0.07%, rather than >0.03% with the original setting), which is most affected by the User Setup. The Processor delay is not a factor after the improvement. With improved processors and a good server (Table 5.12), the system performance of the network without initial network topology is most affected by the User Setup (<2.4%) for all metrics used.

Table 5.11. Difference of performance value for Small World network topology with a faulty server and improved processors.

Scene		*Success Rate*	*Average QH Count*	*Average QH Hop*	*Average QH Time*	*File Coverage*
User Setups	***Min***	*0.030196937*	*0.804669692*	*0.151700237*	*0.151178713*	*0.384127387*
	Max	***0.07293946***	*3.243291904*	*0.512387821*	*0.833510678*	*2.962284529*
Protocol	***Min***	*0*	*6.380455907*	*0.975737794*	*0.972383349*	*6.43387063*
	Max	*0.03646973*	***7.742350735***	***1.021195476***	***1.189899455***	***7.805647449***
Bandwidth	***Min***	*0*	*0*	*0*	*0.009543741*	*0*
	Max	*1.6675E-05*	*0.037657815*	*0.010297299*	*0.372843807*	*0.029196805*
Processor	***Min***	*0*	*0*	*0*	*0*	*0*
	Max	*1.6675E-05*	*0.037657815*	*0.010297299*	*0.010261898*	*0.029196805*

Table 5.12. Difference of performance value for Small World network topology with a good server and improved processors.

Scene		*Success Rate*	*Average QH Count*	*Average QH Hop*	*Average QH Time*	*File Coverage*
User Setups	***Min***	*0.022922785*	*0.027850615*	*0.00310551*	*0.011925976*	*0.465147421*
	Max	***0.07293946***	***2.310249428***	***0.72931595***	***0.859710949***	***2.410962041***
Protocol	***Min***	*0*	*0.674965276*	*0.03956022*	*0.048525898*	*0.654251152*
	Max	*1.6675E-05*	*1.331288795*	*0.149785407*	*0.149474781*	*1.312757503*
Bandwidth	***Min***	*0*	*0*	*0*	*0.007816547*	*0*
	Max	*1.6675E-05*	*0.065929908*	*0.043674869*	*0.207381068*	*0.079725404*
Processor	***Min***	*0*	*0*	*0*	*0*	*0*
	Max	*1.6675E-05*	*0.065929908*	*0.043674869*	*0.043584295*	*0.079725404*

In all results, it is shown that the bandwidth being simulated in this thesis is high and sufficient for the system. Therefore it does not have much effect on the system. Another series of simulations with a faulty server have been run to observe the system performance with low bandwidth, where the high bandwidth peers simulate 4.5 kbps and the low bandwidth peers simulate 1.5 kbps. Table 5.13 and Table 5.14 show that with limited bandwidth, the User Setup (<141%), Bandwidth (<46.1%), and Processor (52.2%) have the most impact on the system (both with and without network topology),

depending on the analysis metrics chosen. The Protocol is deemed less important than the high bandwidth simulations in the system.

Table 5.13. Difference of performance value for Small World topology with a faulty server and limited bandwidth peers.

Scene		Success Rate	Average QH Count	Average QH Hop	Average QH Time	*File* Coverage
User Setups	**Min**	0.020947857	0.686441568	0.125911252	0.084631776	1.132227824
	Max	7.71468218	**140.9954963**	34.40853541	15.44754133	**141.2624946**
Protocol	**Min**	0	0.789174598	0.157615839	0.030890932	0.716658277
	Max	2.804878049	23.67478246	11.23380535	7.947071061	23.33478568
Bandwidth	**Min**	0	10.7051035	1.578910994	0.10239736	10.44529991
	Max	**9.695121951**	129.0469238	**46.07759692**	33.05606879	128.2524341
Processor	**Min**	0	4.15328851	0.220044546	9.68244612	4.384780534
	Max	6.585365854	108.3039626	29.52653327	**52.15142808**	108.3882675

Table 5.14. Difference of performance value without initial network topology with a faulty server and limited bandwidth peers.

Scene		Success Rate	Average QH Count	Average QH Hop	Average QH Time	*File* Coverage
User Setups	**Min**	0.302897154	1.071394754	0.622258353	0.187024274	0.932590618
	Max	9.85575296	**90.20243469**	28.7052244	21.46586662	**87.04586637**
Protocol	**Min**	0.396981585	0.452839922	0.11104594	0.018095391	0.927459992
	Max	8.791304254	45.0193356	7.615225992	13.11230266	43.85957537
Bandwidth	**Min**	0.010909286	14.37991597	1.252402469	1.596818661	14.5204918
	Max	**12.56009087**	71.57586919	**39.58600523**	39.62359859	70.643596
Processor	**Min**	0.080535961	0.128896316	1.902348576	9.557601267	0.298920546
	Max	7.949398866	59.73962582	21.45138028	**46.73178765**	58.70010105

This thesis concludes that user behavior and physical network characteristics depend on each other and impact the system performance, especially in a dynamic network or one with limited hardware. Lots of simulators can initiate the experiments with certain overlay network topology; however, it is shown in this thesis that the results can be quite different with different network topology. Though the result shows that the performance difference may not be too significant for some settings, especially with good hardware specifications, because real world P2P networks are dynamic, it is important to simulate

these constraints in the experiment to fully express the real world scenario. It is shown that in order to analyze a P2P system and capture its emergent behavior, it is necessary to take into account the **physical network characteristics, protocol, user presence, and user activities**.

5.6 Future Work

This thesis has shown that protocol/application, user presence/behavior, network topology, network, and hardware specifications are interconnected with each other. With different configurations on each layer, the overall P2P system performance differs. It is found that the current implementation of the simulator with the current settings (where all peers fire the message at the same time) can support up to 1250 peers with faulty server. Currently, the simulator is single threaded on a single processor. The nature of the simulator is designed to support multi-threaded simulation. Future work focuses on a distributed simulation scheme that will enable the simulation of a larger number of peers on multiple processors/computers.

It will also be interesting to be able to reproduce the result of an improved protocol (say introducing UltraPeers in the Gnutella network as illustrated in Section 2.3.1.2). Then repeat the simulation with a different configuration on each layer to investigate if the improvement on the system performance persists or not.

References

[1] Adar, E., Huberman B. A. "Free riding on Gnutella," First Monday, vol. 5, no. 10, page 8 October 2000. Available at http://www.firstmonday.org/issues/issue5_10/adar/ – Accessed on August 20, 2006.

[2] Aisee. The Aisee homepage. Available at http://www.aisee.com/ – Accessed on August 20, 2006.

[3] Anderson, K. "Analysis of the Traffic on the Gnutella Network" November 2001. Available at http://www.cs.ucsd.edu/classes/wi01/cse222/projects/reports/p2p-2.pdf – Accessed on August 20, 2006.

[4] Babaoglu, O. , Meling, H. and Montresor, A. "Anthill: a Framework for the Development of Agent-Based Peer-to-Peer Systems". In proceedings of 22nd International Conference on Distributed Computing Systems ICDCS, pages 15-22, Vienna, Austria, July 2002.

[5] BearShare. The BearShare homepage. Available at http://www.bearshare.com/ – Accessed on August 20, 2006.

[6] Bellifemine, F., Poggi, A., Rimassa, G. JADE–A FIPA-compliant agent framework. In Proceedings of PAAM'99, pages 97-108, London, UK, April 1999.

[7] Castro, M., Druschel, P. Kermarrec A-M., Rowstron A. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC), vol.20, no. 8, pages 100-110, October 2002.

[8] Clarke, I., Miller, S. G., Hong, T. W., Sandberg, O., Wiley, B. Protecting Free Expression Online with Freenet. IEEE Internet Computing, vol. 6, no. 1, pages 40-49, January-February 2002.

[9] Clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.2). Available at http://rfc-gnutella.sourceforge.net/Development/GnutellaProtocol0_4-rev1_2.pdf – Accessed on August 20, 2006.

[10] Giuli, T. J., Baker, M. Narses: A Scalable Flow-Based Network Simulator. Techinical Report, Standford University, 17th May 2006. Available at http://wotan.liu.edu/docis/dbl/xxcspf/0211024.html – Accessed on August 20, 2006.

[11] GnucNS. The GnucNS homepage. Available at http://www.gnucleus.com/GnucNS/ – Accessed on August 20, 2006.

[12] Gnucleus. The Gnucleus homepage. Available at http://www.gnucleus.com/ – Accessed on August 20, 2006.

[13] Gtk-Gnutella. The Gtk-Gnutella homepage. Available at http://gtk-gnutella.sourceforge.net/ – Accessed on August 20, 2006.

[14] FreeNet. The Free Network Project homepage. Available at http://freenetproject.org/ – Accessed on August 20, 2006.

[15] FreePastry. The FreePastry homepage. Available at http://freepastry.org/FreePastry/ – Accessed on August 20, 2006.

[16] Fried, I. Intel chip will be bigger, more expensive to make. Cnet News.com, June 2002. Available on http://news.com.com/2100-1001-241376.html?tag=mainstry – Accessed on August 20, 2006.

[17] He, Q., Ammar, M., Riley, G., Raj, H., Fujimoto, R. Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems. Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS), pages 71-78, October 2003.

[18] Joseph, S.R.H. Adaptive Routing in Distributed Decentralized Systems: NeuroGrid, Gnutella, and Freenet. Proceedings of workshop on Infrastructure for Agents, MAS, and Scalable MAS, at Autonomous Agents, Montreal, Canada, 2001.

[19] Joseph, S.R.H. NeuroGrid Protocol Version 0.1. Available at http://www.neurogrid.net/ng-protocol.html – Accessed on Auguest 20, 2006.

[20] Joseph, S.R.H. NeuroGrid Simulation Mailing Archive, January 2003. Available at http://sourceforge.net/mailarchive/forum.php?thread_id=1593250&forum_id=8271 – Accessed on August 20, 2006.

[21] Jovanovic, M. A. Modeling Large-scale Peer-to-Peer Networks and a Case Study of Gnutella. M. Sc. Thesis, University of Cincinnati, 2001.

[22] Kant, K., Iyer, R. Modeling and Simulation of Adhoc/P2P Resource Sharing Networks. In Proceedings of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Septempber 2003.

[23] Klingberg, T., Manfredi, R. Gnutella version 0.6. Available at http://rfc-gnutella.sourceforge.net/developer/testing/index.html – Accessed on August 20, 2006.

[24] LimeWire. The LimeWire homepage.

Available at http://www.limewire.com/english/content/home.shtml - Accessed on August 20, 2006.

[25] Manjoo, F. Gnutella Bandwidth Bandit.

Available at http://www.salon.com/tech/feature/2002/08/08/gnutella_developers/ – Accessed on August 20, 2006.

[26] Merugu, S., Srinivasan, S., Zegura, E. p-sim: A Simulator for Peer-to-Peer Networks. Proceedings of 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS), October 12-15, 2003, Orlando Florida.

[27] Microsoft Corporation. The Microsoft Visual C# homepage.

Available at http://msdn.microsoft.com/vcsharp - Accessed on August 20, 2006.

[28] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne J., Richard, B., Rollins, S., Xu, Z. Peer-to-Peer Computing. Technical Report HP, HPL-2002-57R1, July 3rd, 2003. Available at http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf – Accessed on August 20, 2006.

[29] Montresor, A., Caro, G. D., Ducatelle, F. Jesi, G. P. BISON project deliverable number D12-D13, February 25, 2005, University of Bologna. Available at http://www.cs.unibo.it/bison/deliverables/D12-13.pdf – Accessed on August 20, 2006.

[30] Montresor, A., Caro, G. D., Heegaard, P. E. BISON project deliverable number D11, January 29, 2004, University of Bologna. Available at http://www.cs.unibo.it/bison/deliverables/D11.pdf – Accessed on August 20, 2006.

[31] Napster. The Napster homepage. Available at http://www.napster.com – Accessed on August 20, 2006.

[32] National Institute of Standards and Technology. Secure Hash Standard. Federal Information Processing Standards Publication 180-1, April 17, 1995. Available at http://www.itl.nist.gov/fipspubs/fip180-1.htm – Accessed on August 20, 2006.

[33] NeuroGrid. The NeuroGrid homepage. Available at http://www.neurogrid.net/ – Accessed on August 20, 2006.

[34] Network Simulator (ns-2). The ns-2 homepage. Available at http://www.isi.edu/nsnam/ns – Accessed on August 20, 2006.

[35] Ting, N., Deters, R. 3LS-A P2P Network Simulator. Poster in The 3rd IEEE International Conference on Peer-to-Peer Computing, Linkoping, Sweden, 1-3 September, 2003.

[36] P2PSim. The P2PSim homepage. Available at http://www.pdos.lcs.mit.edu/p2psim/ – Accessed on August 20, 2006.

[37] PeerSim. The PeerSim homepage. Available at http://peersim.sourceforge.net – Accessed on August 20, 2006.

[38] Riley, G., Fujimoto, R., Ammar, M. A Generic Framework for Parallelization of Network Simulations. In Proceeding of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 128-135, October 1999.

[39] Rowstron, A., Druschel, P. PAST: A large-scale, persistent peer-to-peer storage utility. In Proceeding HotOS VIII, pages 75-80, Schoss Elmau, Germany, May 2001.

[40] Rowstron, A., Druschel, P. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems IFIP/ACM international Conference on Distributed Systems Platforms (Middleware), pages 329-350, Heidelberg, Germany, November 2001.

[41] Saroiu, S., Gummadi, P. K., Gribble, S. D. "A Measurement Study of Peer-to-Peer File Sharing Systems" in Proceedings of Multimedis Computing and Networking (MMCN), pages 156-170, San Jose, CA, USA, January 2002.

[42] Schlosser, M. T., Condie, T. E., Kamvar, S. D. Simulating a P2P File-Sharing Network. 1st Workshop on Semantics in Peer-to-Peer and Grid Computing, 12th International World Wide Web Conference, December 2002.

[43] Sun Microsystems. The Java Remote Method Invocation homepage. Available at http://java.sun.com/products/jdk/rmi – Accessed on August 20, 2006.

[44] Sun Microsystem. The Java Technology homepage. Available at http://java.sun.com – Accessed on August 20, 2006.

[45] SETI@Home. The SETI@Home homepage.

Available at http://setiathome.ssl.berkeley.edu/ – Accessed on August 20, 2006.

[46] Single, A., Rohrs, C. Ultrapeers: Another Step Towards Gnutella Scalability. Available at http://www.limewire.com/developer/Ultrapeers.html – Accessed on August 20, 2006.

[47] Vassileva, J. Motivating participation in Peer to Peer Communities. Proceeding of Workshop on Emergent Societies in the Agent World, ESAW'02, pages 141 – 151, Madrid, 16-17 September, 2002.

[48] Whatis.com. Fast Guide To DSL.

Available on http://whatis.techtarget.com/definition/0,289893,sid9_gci213915,00.html – Accessed on August 20, 2006.

[49] Xolox. The Xolox homepage. Available at http://www.xolox.nl – Accessed on August 20, 2006.

[50] Zeinalipour-Yazti, D., Folia, T. "A Quantitative Analysis of the Gnutella Network Traffic" Available on http://www.cs.ucr.edu/~csyiazti/courses/cs204/project/gnuDC.pdf – Accessed on August 20, 2006.

Appendix A. Simulations with A Faulty Server

A.1 Active World

Small World as initial network topology

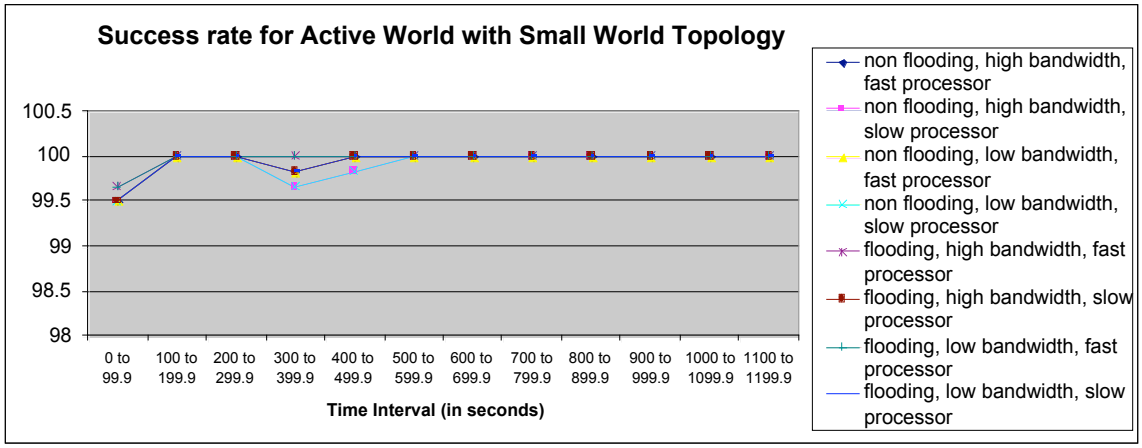

Figure A.1. Success Rate over time for Active World with Small World Topology.

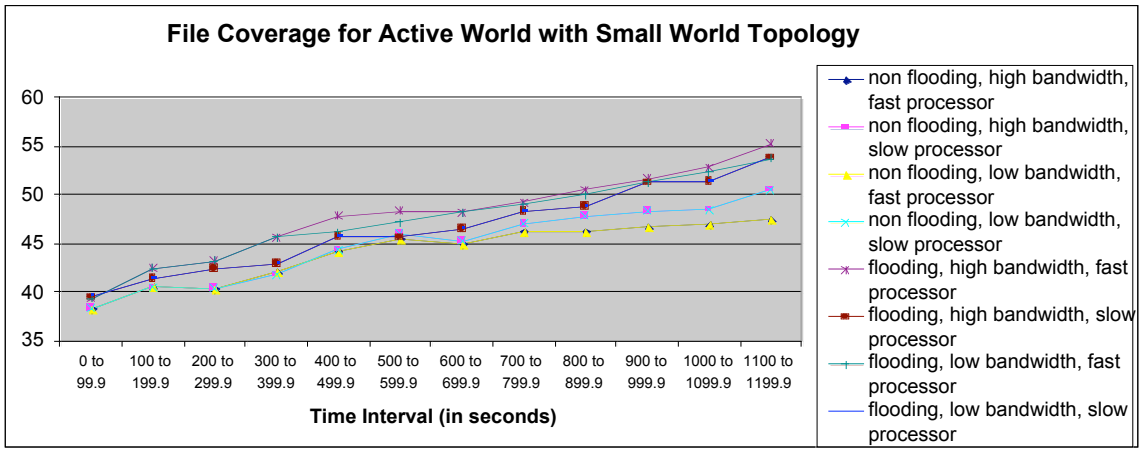

Figure A.2. File Coverage for Active World with Small World Topology.

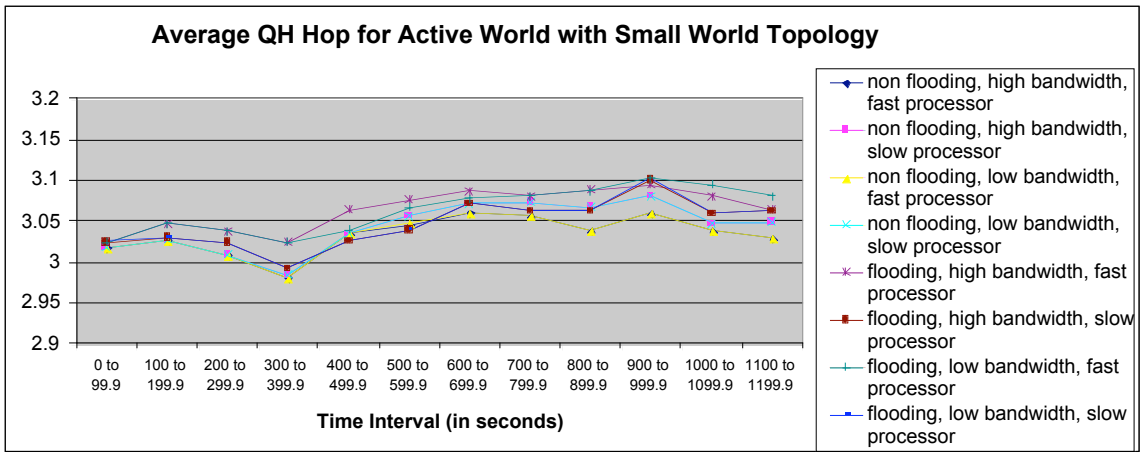

Figure A.3. Average QH Hop for Active World with Small World Topology.

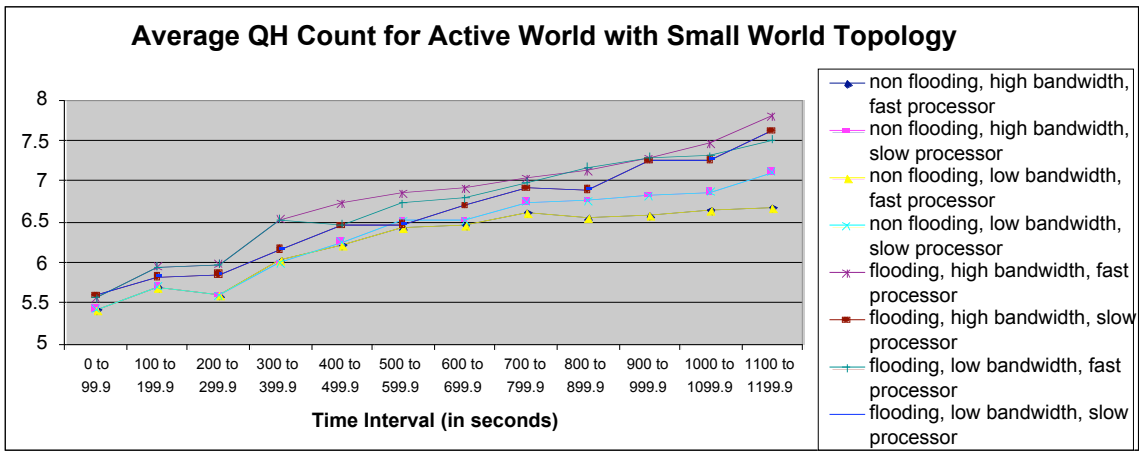

Figure A.4. Average QH Count for Active World with Small World Topology.

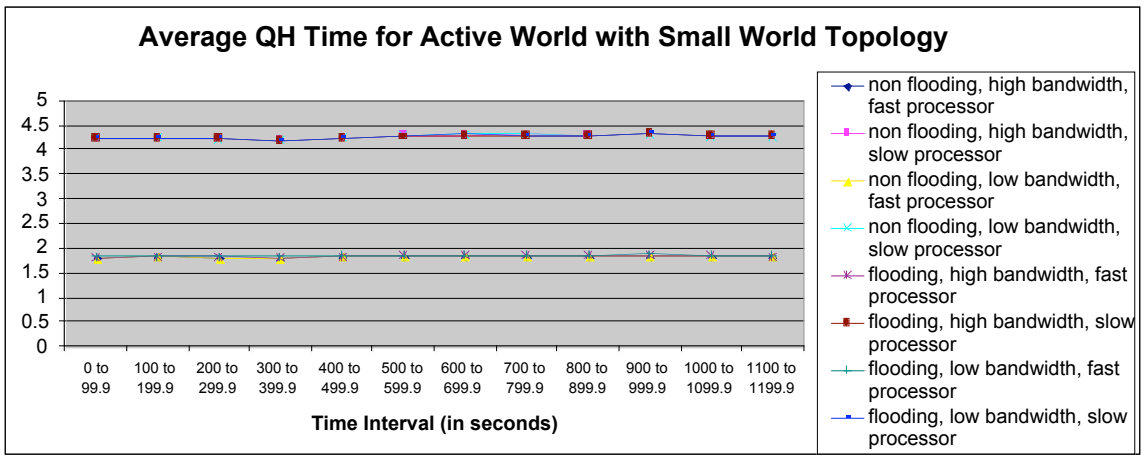

Figure A.5. Average QH Time for Active World with Small World Topology.

Without initial network topology

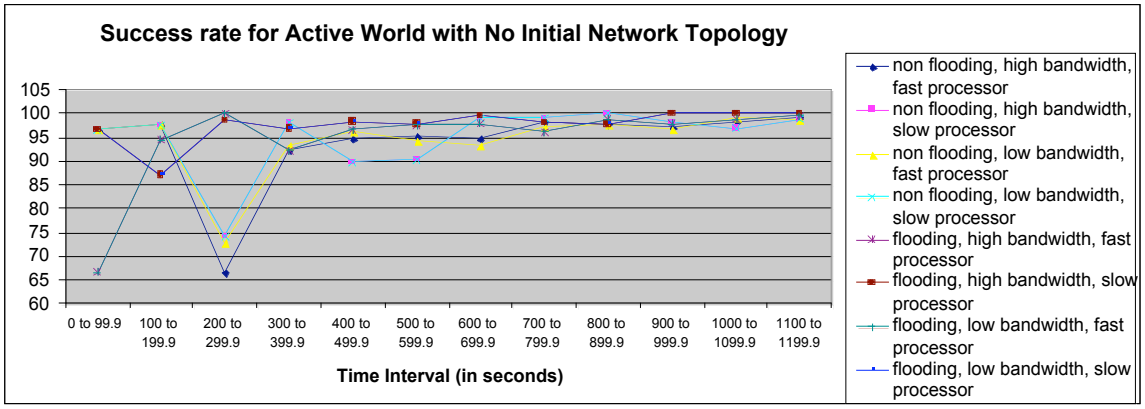

Figure A.6. Success Rate for Active World without Initial Network Topology.

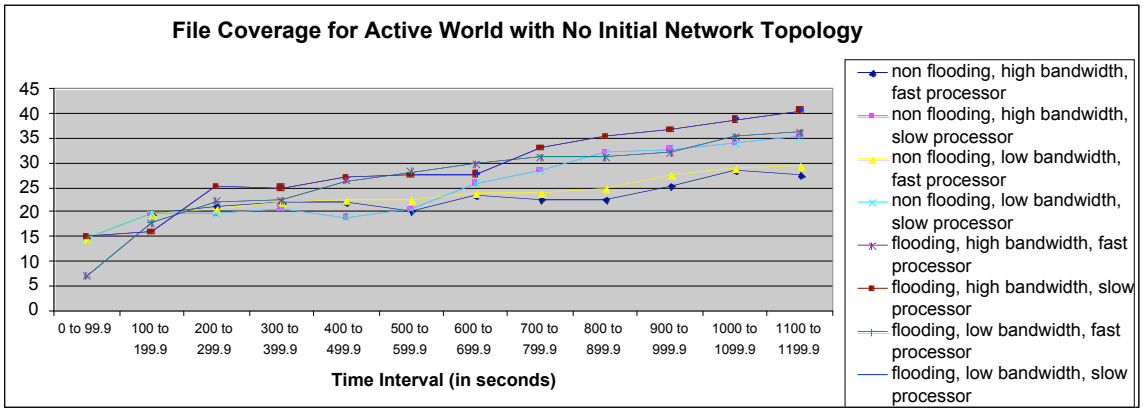

Figure A.7. File Coverage for Active World without Initial Network Topology.

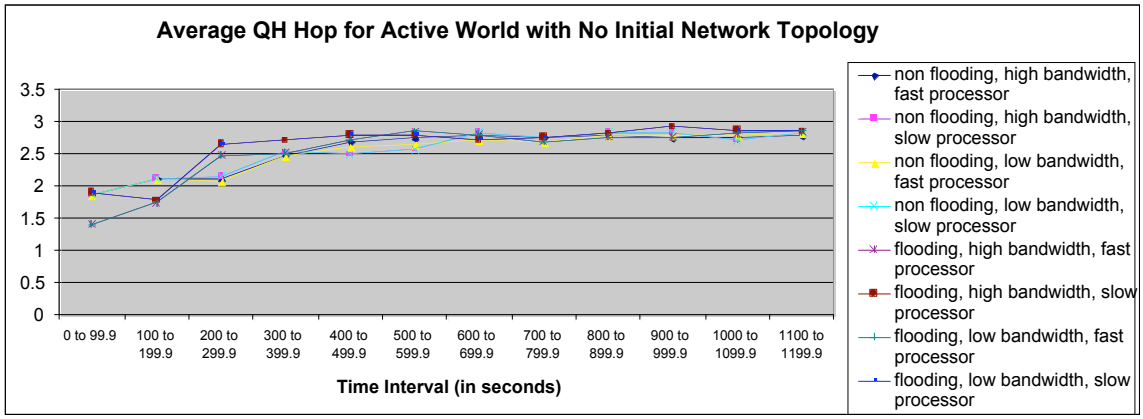

Figure A.8. Average QH Hop for Active World without Initial Network Topology.

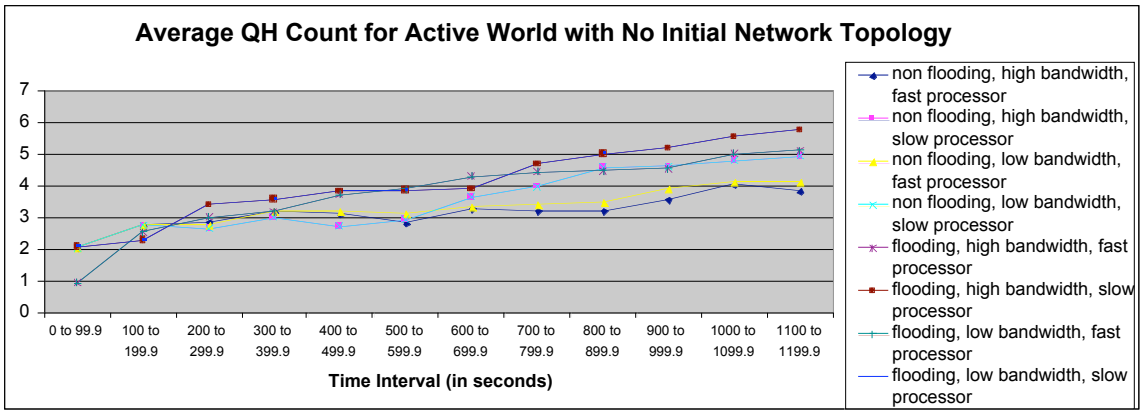

Figure A.9. Average QH Count for Active World without Initial Network Topology.

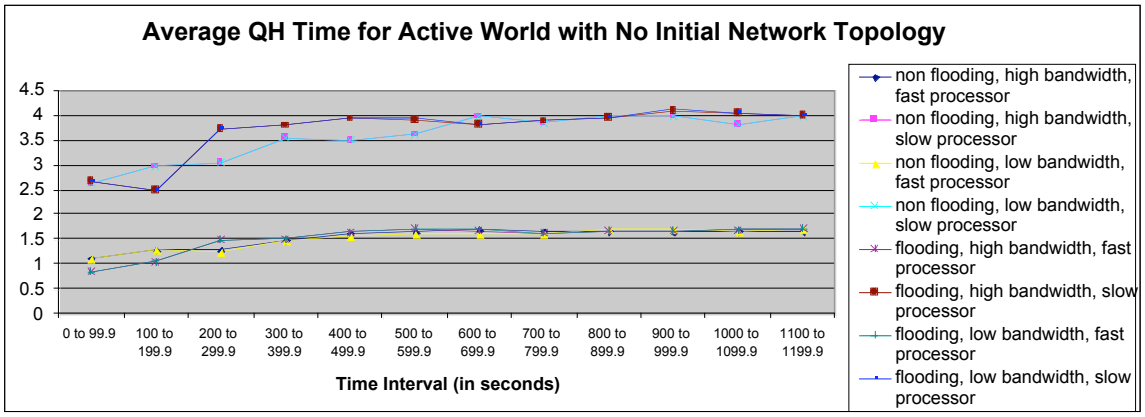

Figure A.10. Average QH Time for Active World without Initial Network Topology.

A.2 Moderate World

Small World as initial network topology

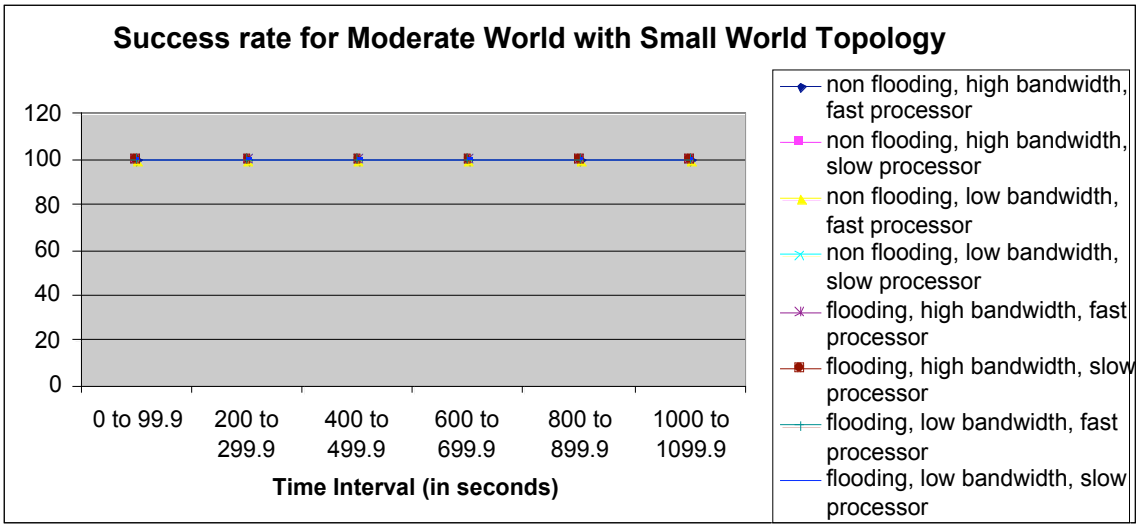

Figure A.11. Success Rate for Moderate World with Small World Topology.

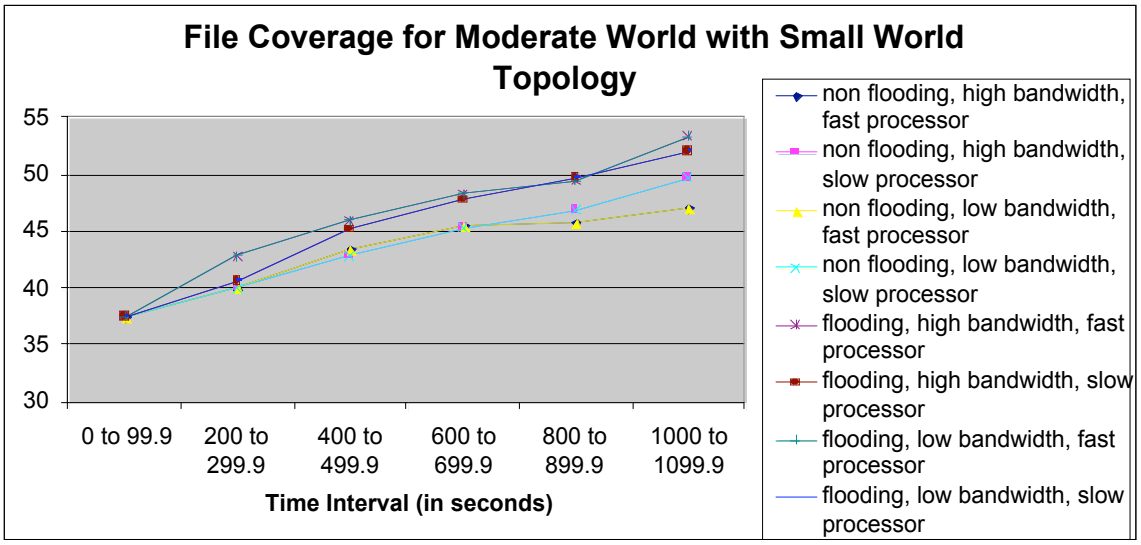

Figure A.12. File Coverage for Moderate World with Small World Topology.

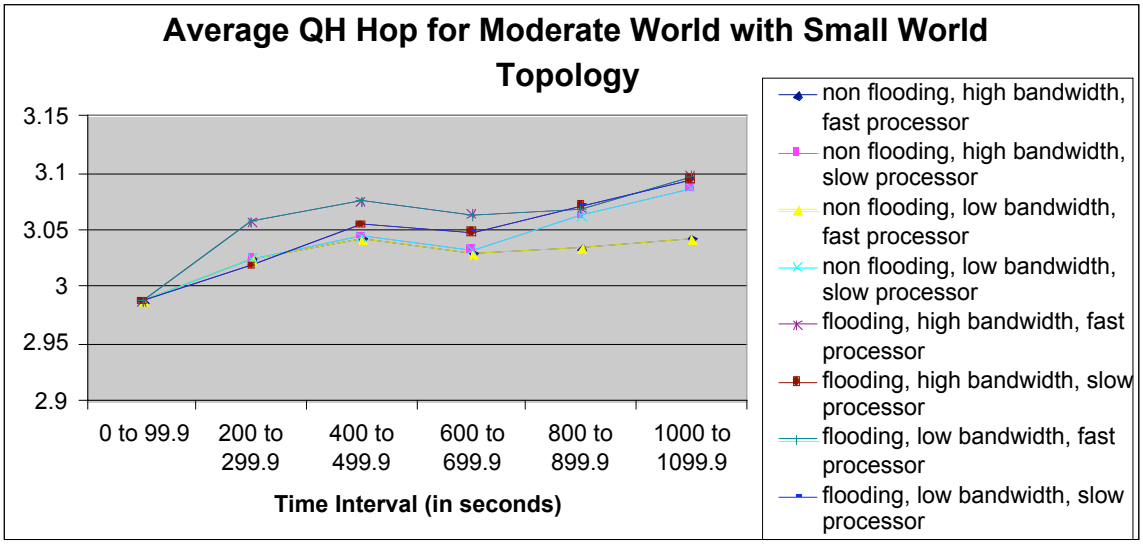

Figure A.13. Average QH Hop for Moderate World with Small World Topology.

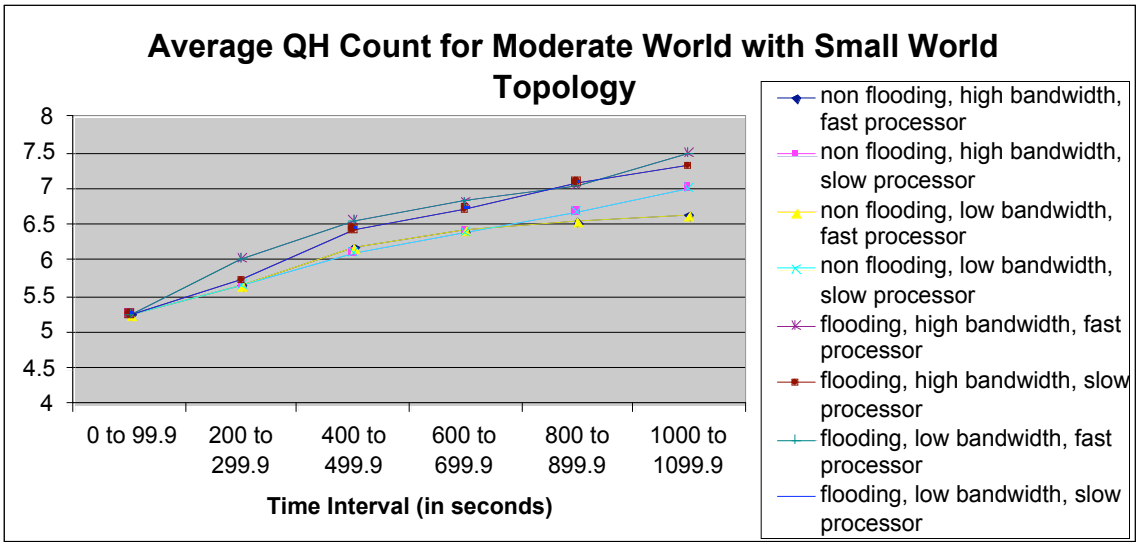

Figure A.14. Average QH Count for Moderate World with Small World Topology.

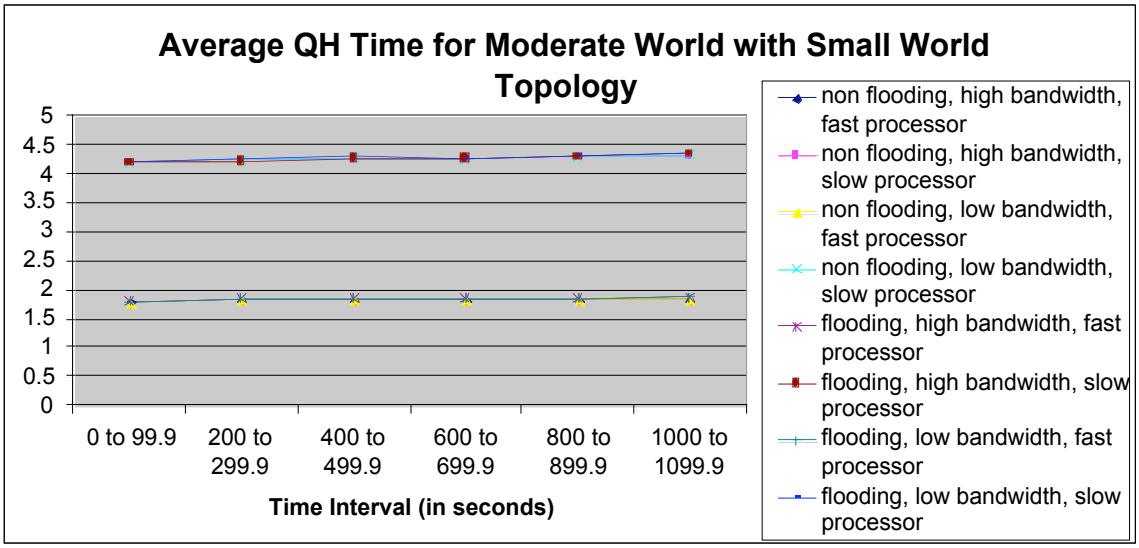

Figure A.15. Average QH Time for Moderate World with Small World Topology.

Without initial network topology

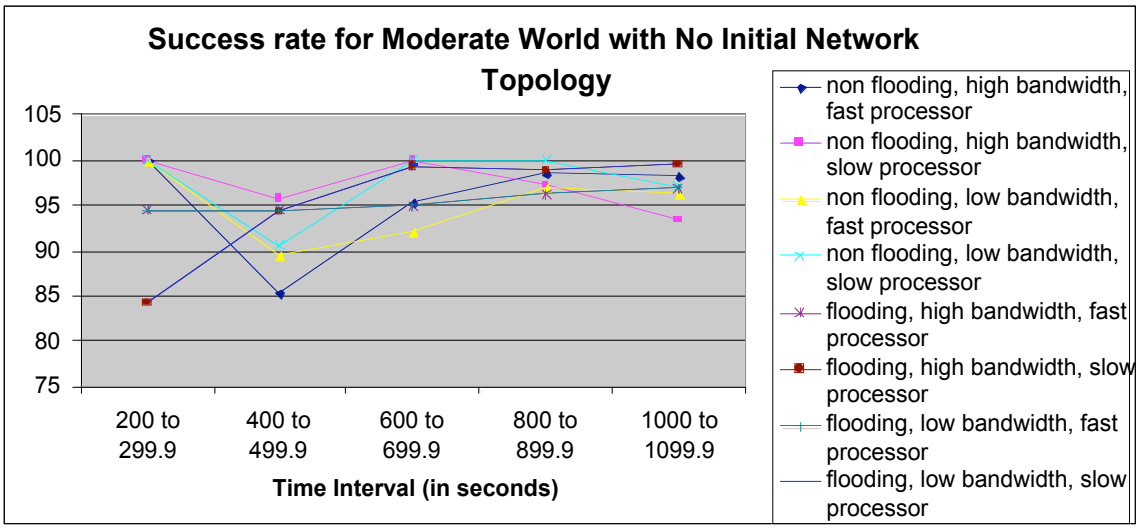

Figure A.16. Success Rate for Moderate World without Initial Network Topology.

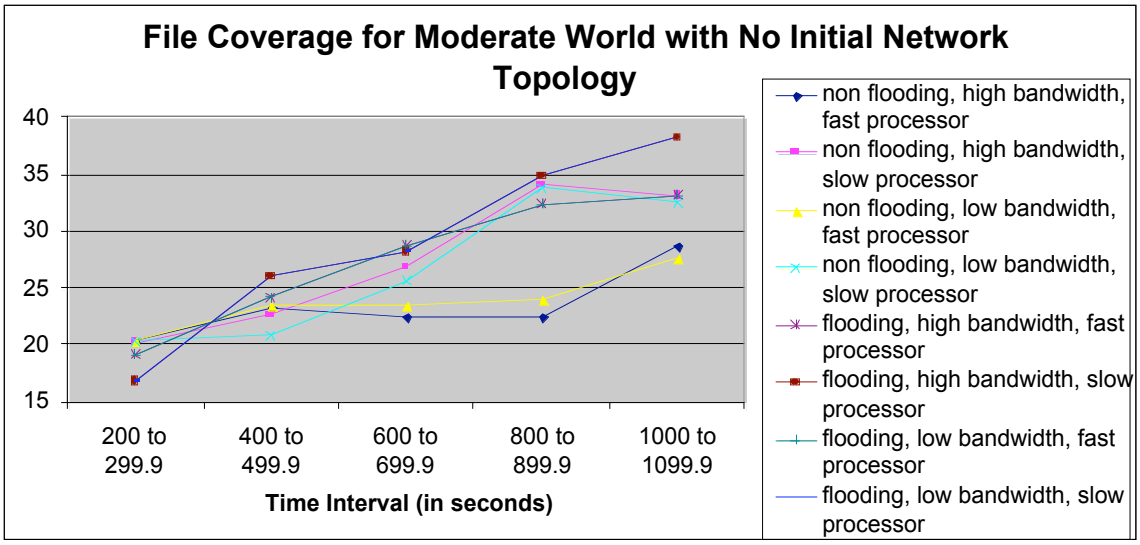

Figure A.17. File Coverage for Moderate World without Initial Network Topology..

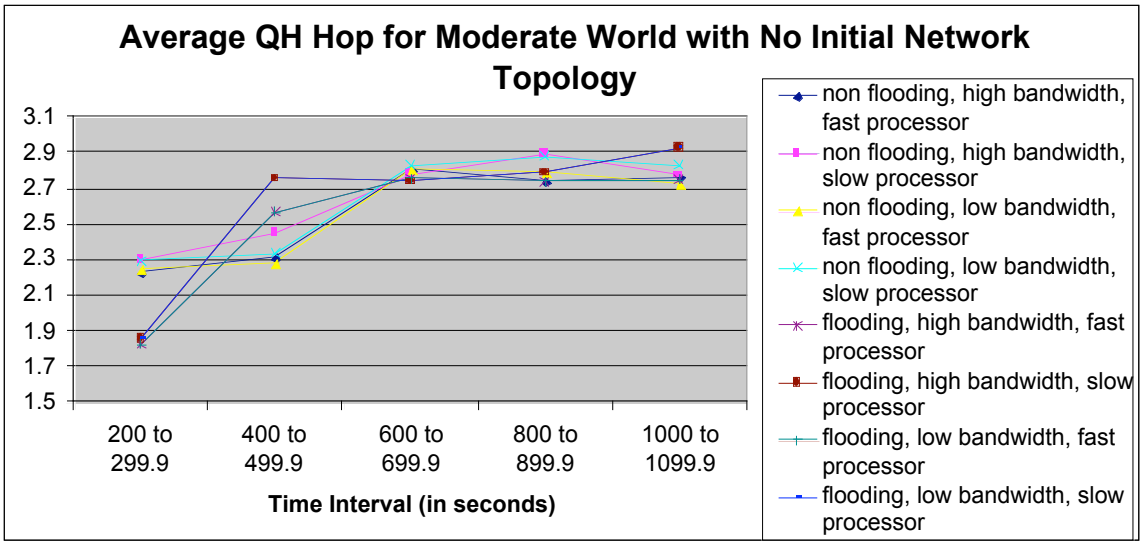

Figure A.18. Average QH Hop for Moderate World without Initial Network Topology..

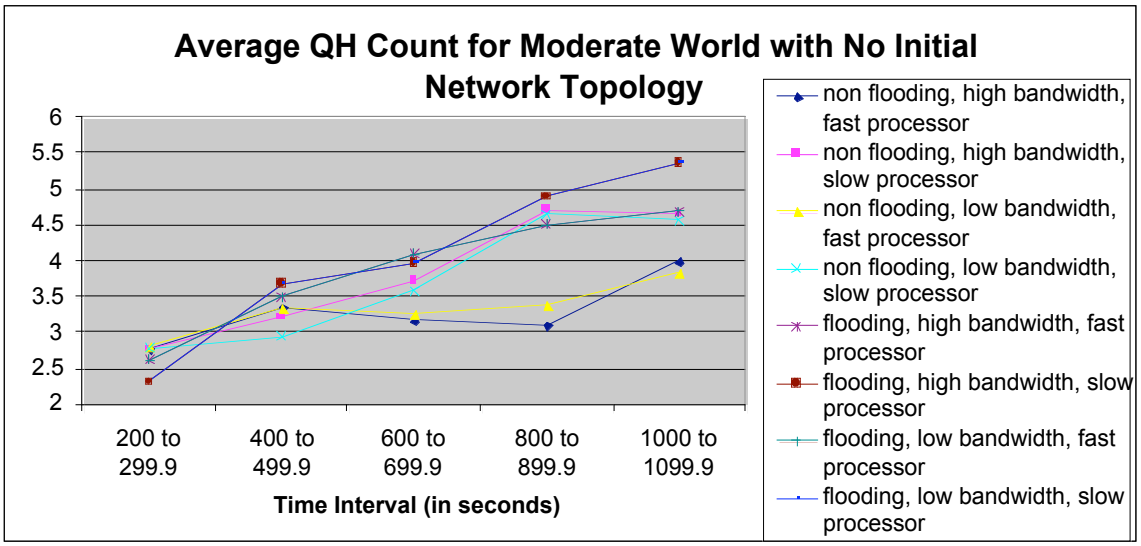

Figure A.19. Average QH Count for Moderate World without Initial Network Topology.

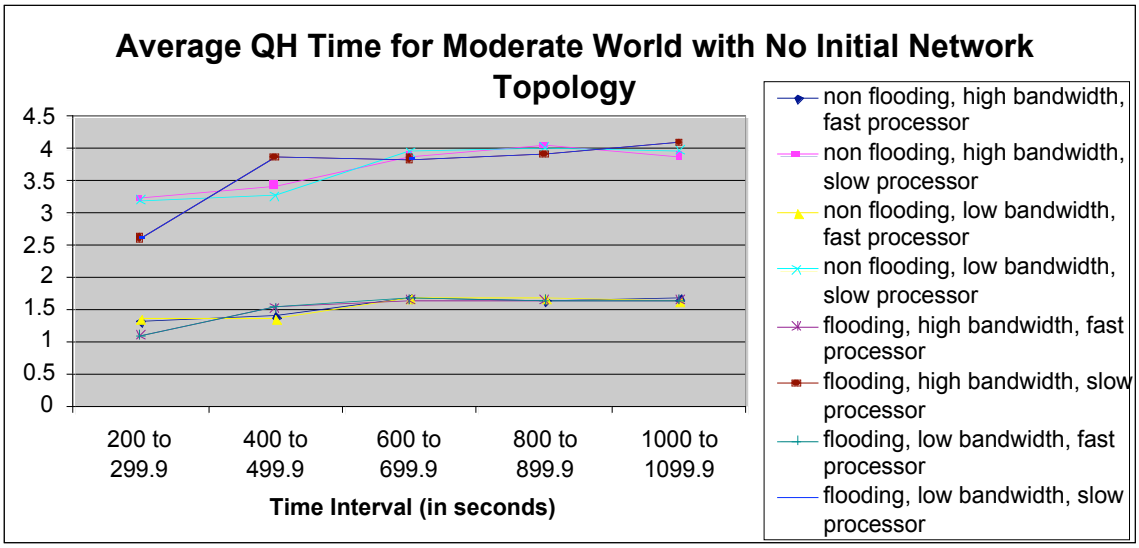

Figure A.20. Average QH Time for Moderate World without Initial Network Topology.

A.3 Distributed World

Small World as initial network topology

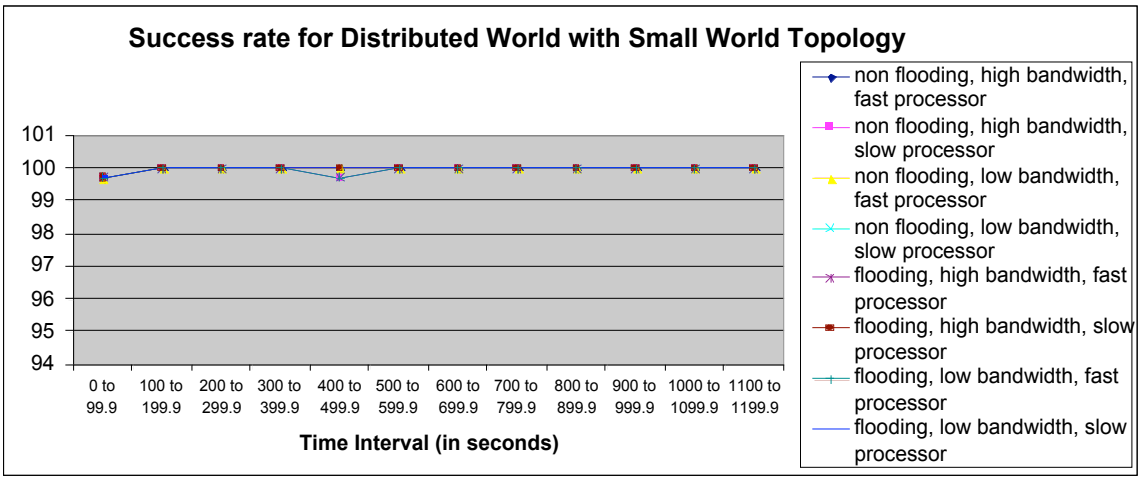

Figure A.21. Success Rate for Distributed World with Small World Topology.

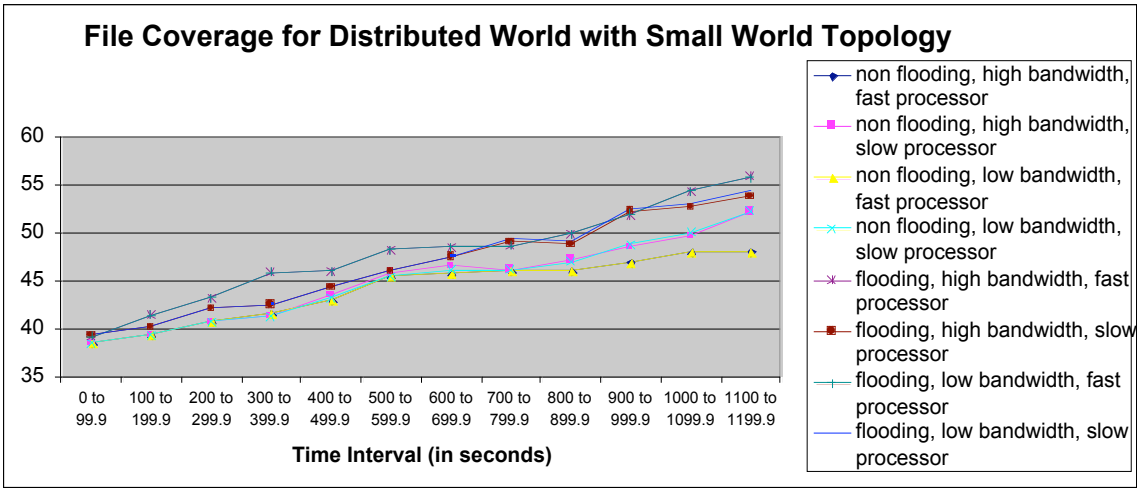

Figure A.22. File Coverage for Distributed World with Small World Topology.

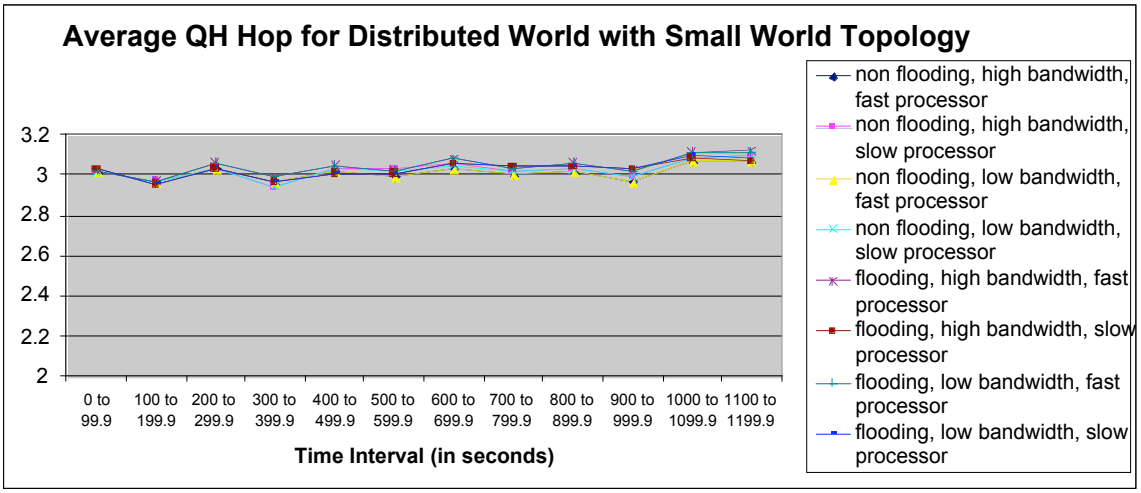

Figure A.23. Average QH Hop for Distributed World with Small World Topology.

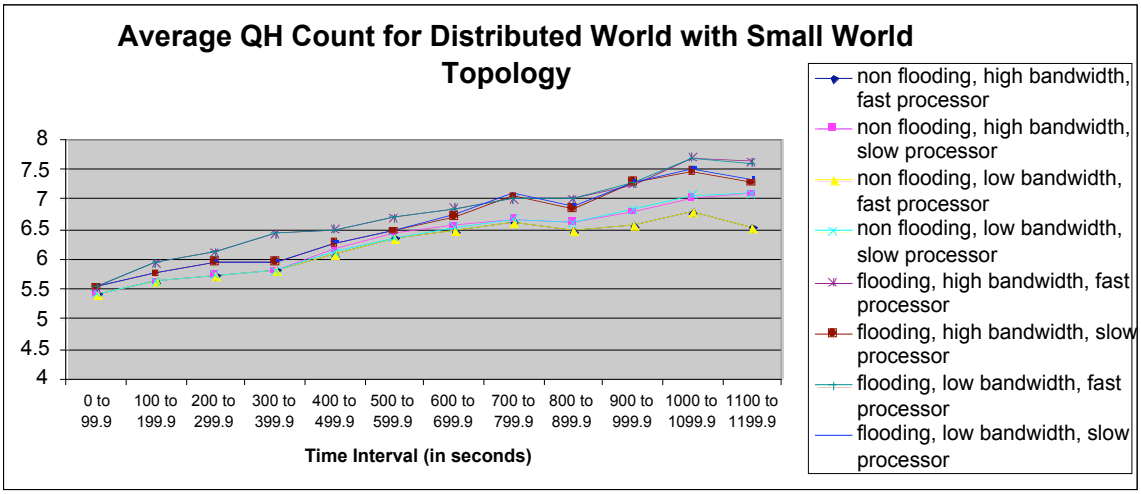

Figure A.24. Average QH Count for Distributed World with Small World Topology.

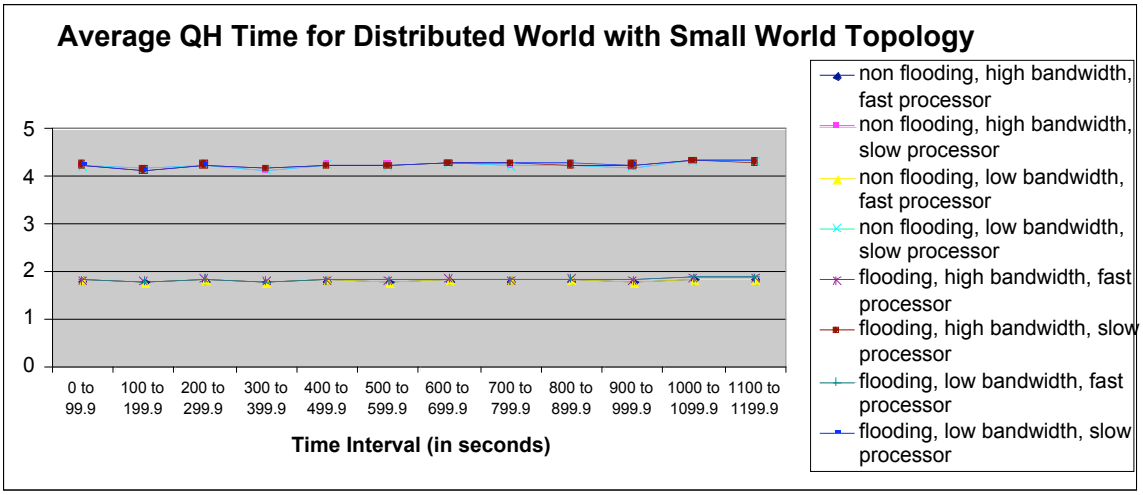

Figure A.25. Average QH Time for Distributed World with Small World Topology.

Without initial network topology

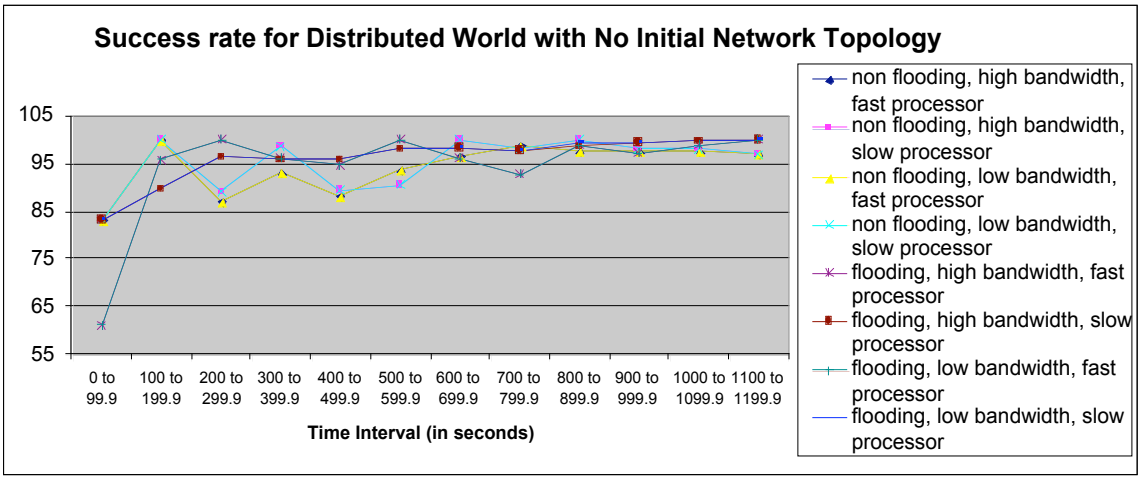

Figure A.26. Success Rate for Distributed World without Initial Network Topology.

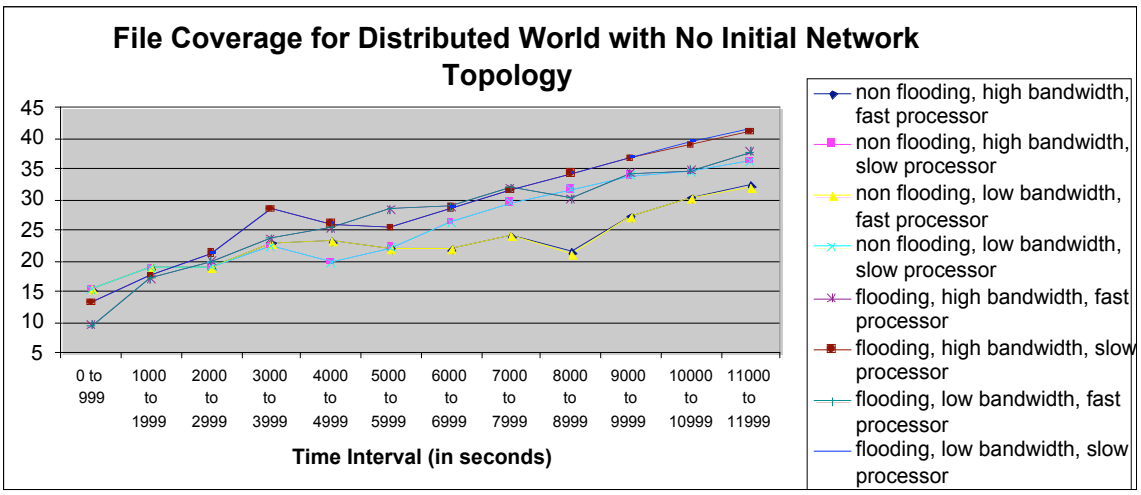

Figure A.27. File Coverage for Distributed World without Initial Network Topology.

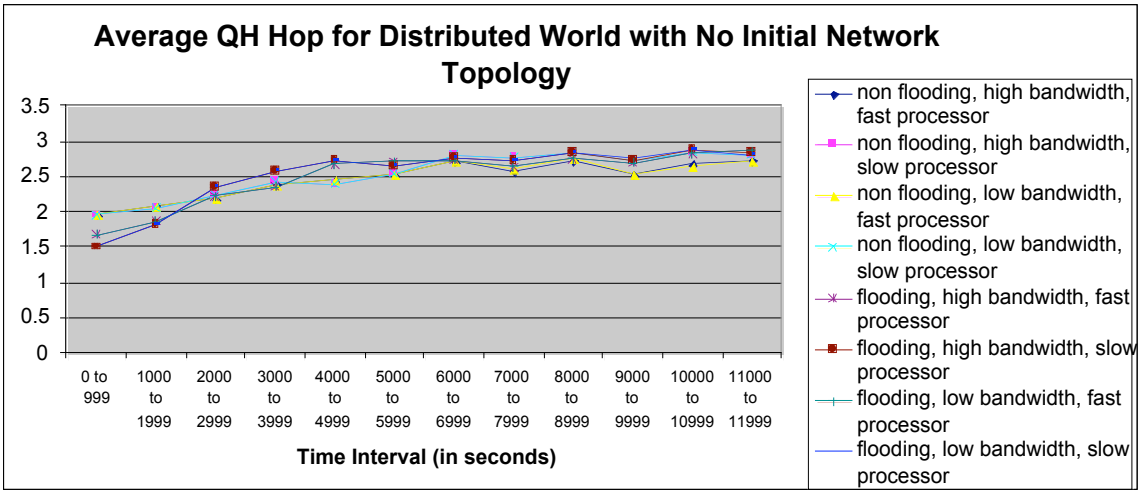

Figure A.28. Average QH Hop for Distributed World without Initial Network Topology.

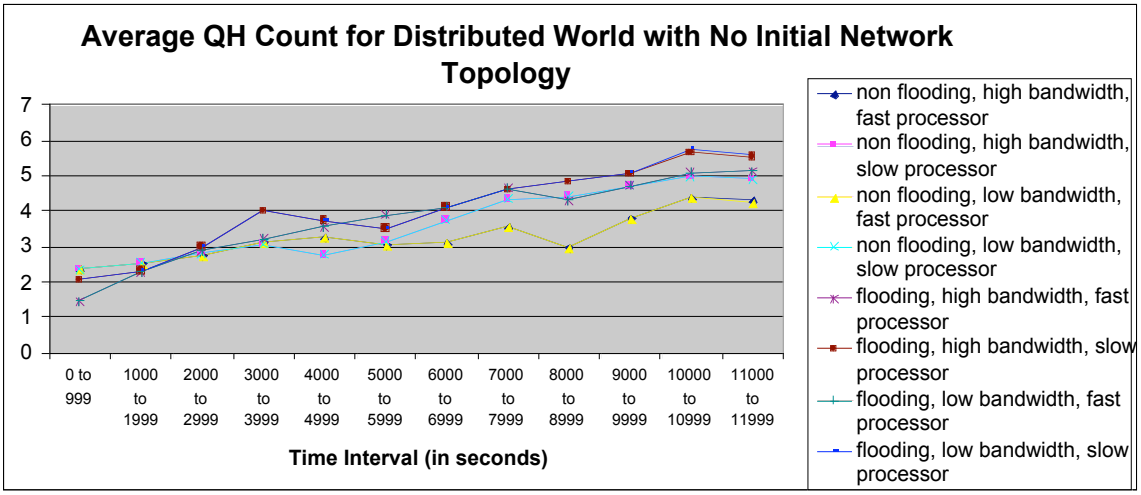

Figure A.29. Average QH Count for Distributed World without Initial Network Topology.

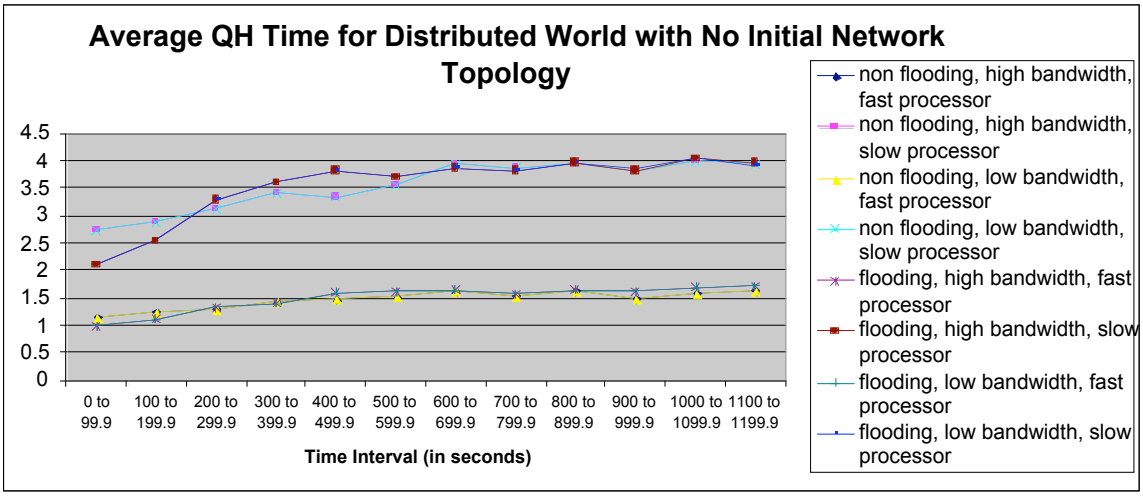

Figure A.30. Average QH Time for Distributed World without Initial Network Topology.

Appendix B. Simulations with A Good Server

B.1 Active World

Small World as initial network topology

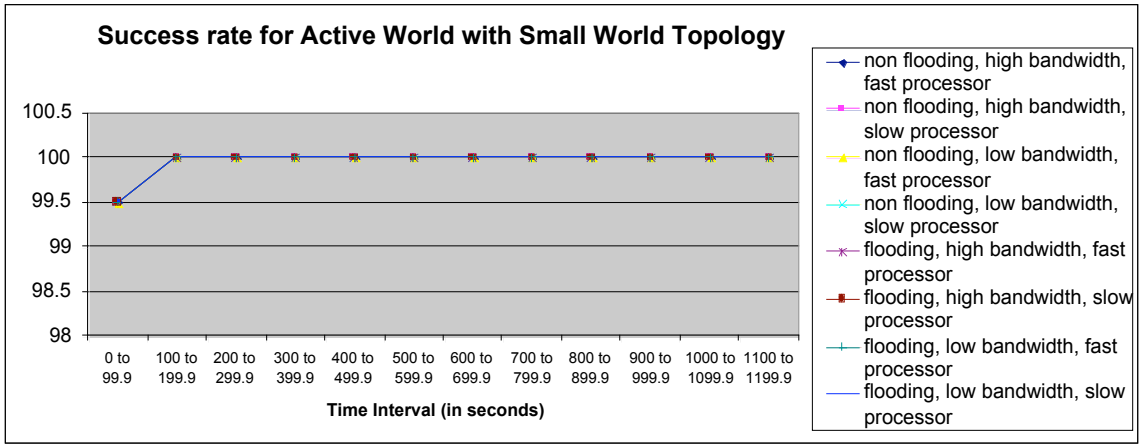

Figure B1. Success Rate for Active World with Small World Topology.

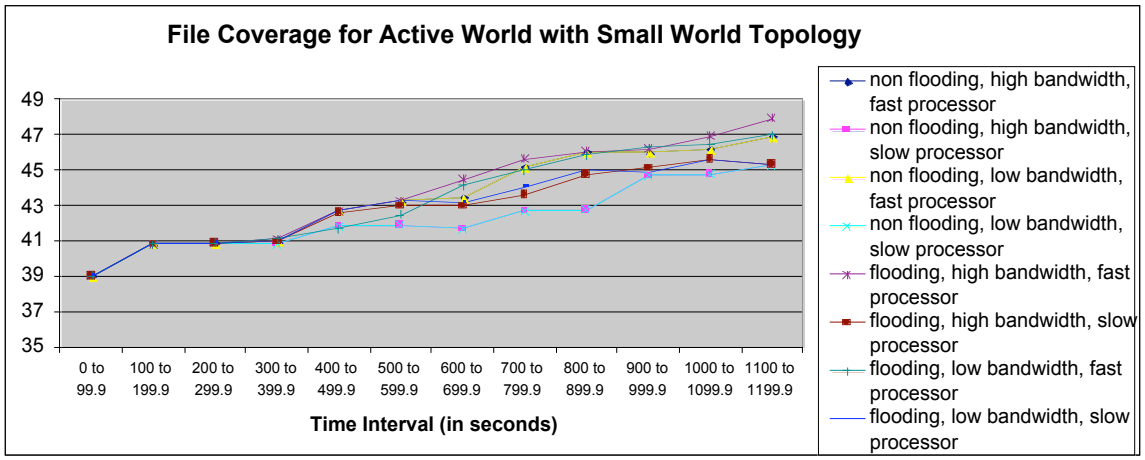

Figure B.2. File Coverage for Active World with Small World Topology.

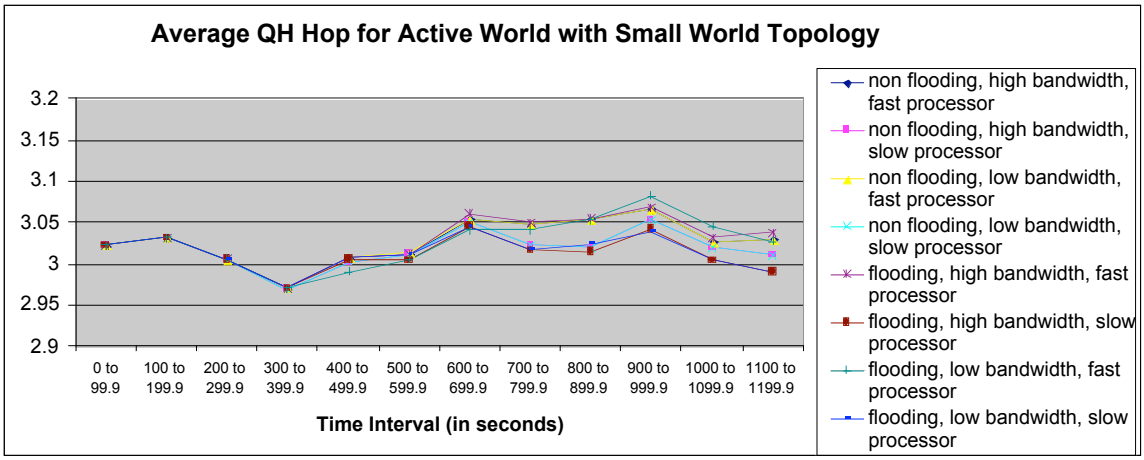

Figure B.3. Average QH Hop for Active World with Small World Topology.

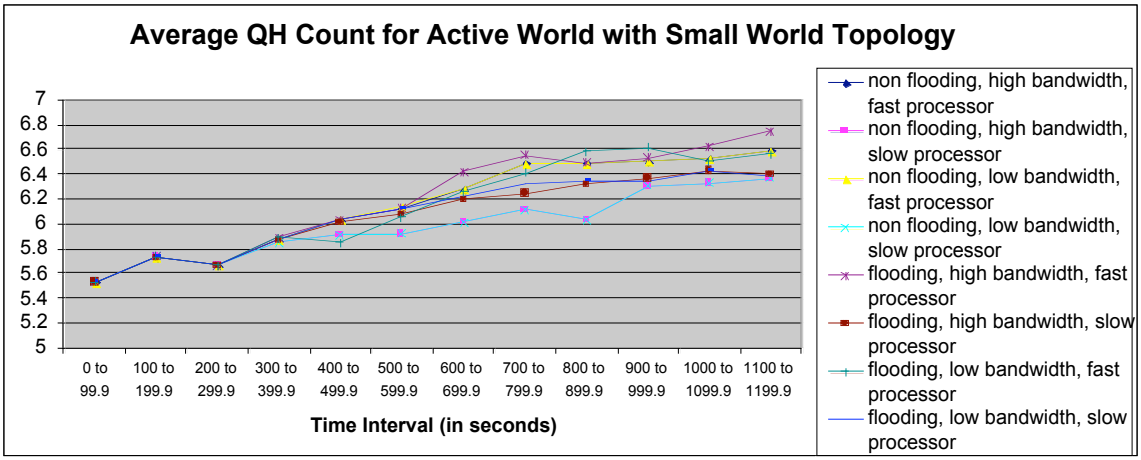

Figure B.4. Average QH Count for Active World with Small World Topology.

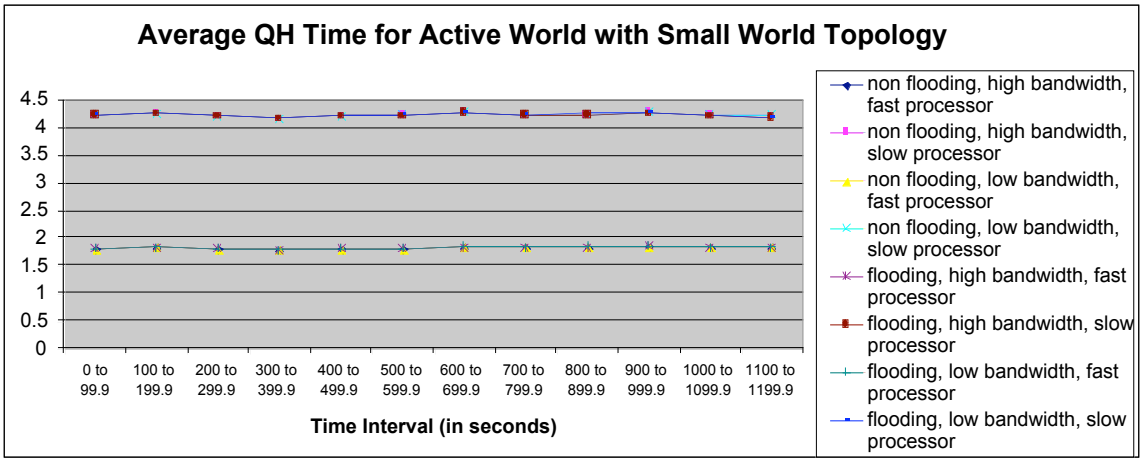

Figure B.5. Average QH Time for Active World with Small World Topology.

Without initial network topology

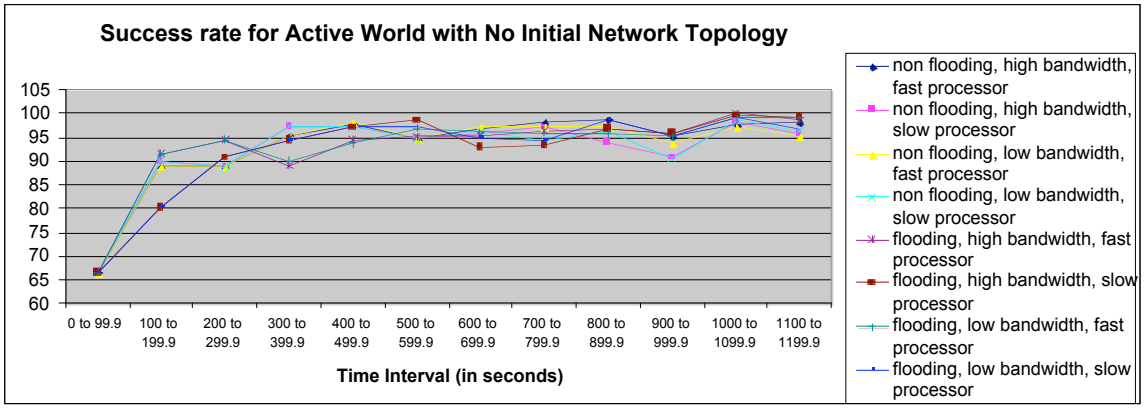

Figure B.6. Success Rate for Active World without Initial Network Topology.

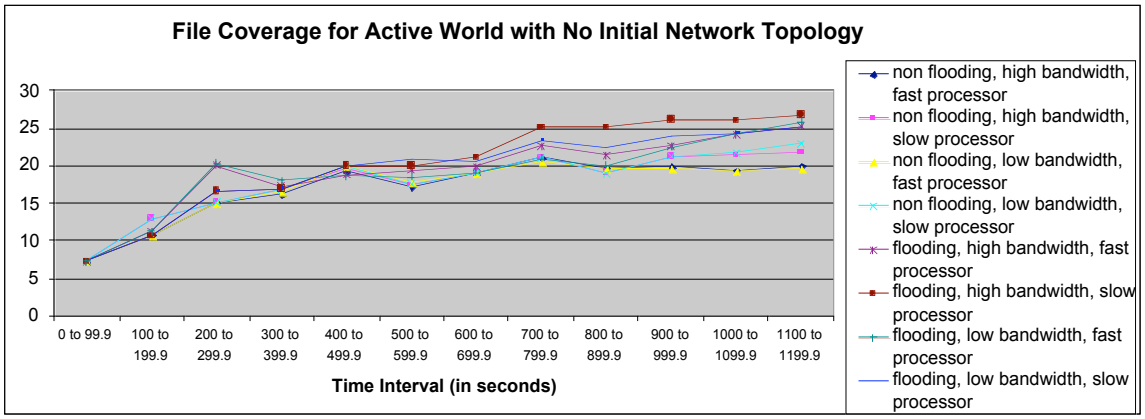

Figure B.7. File Coverage for Active World without Initial Network Topology.

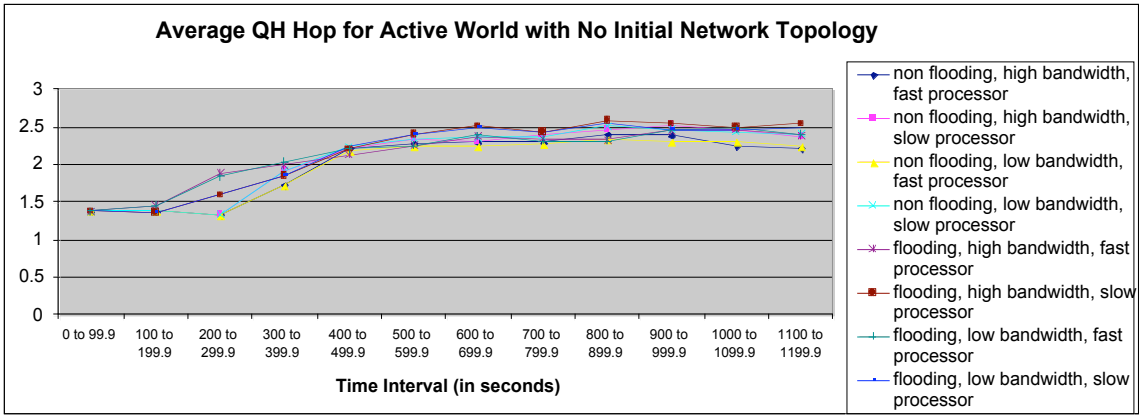

Figure B.8. Average QH Hop for Active World without Initial Network Topology.

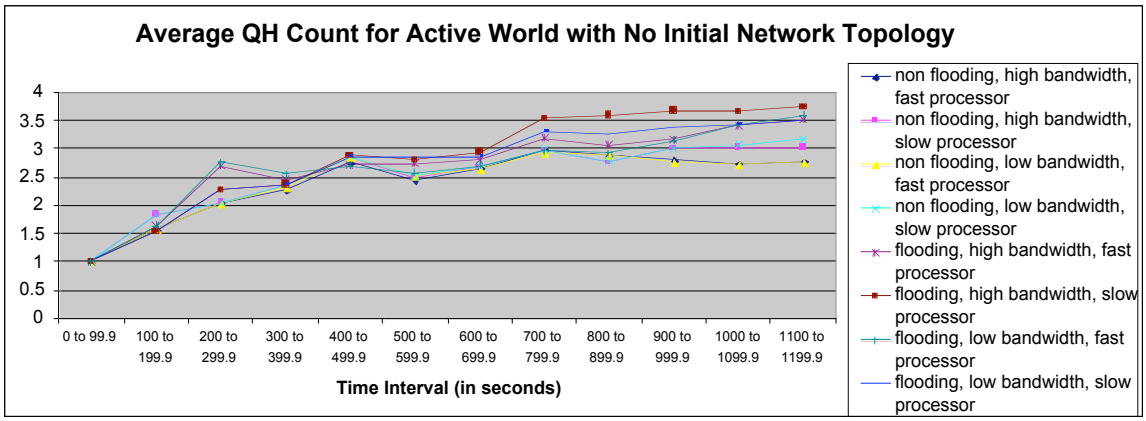

Figure B.9. Average QH Count for Active World without Initial Network Topology.

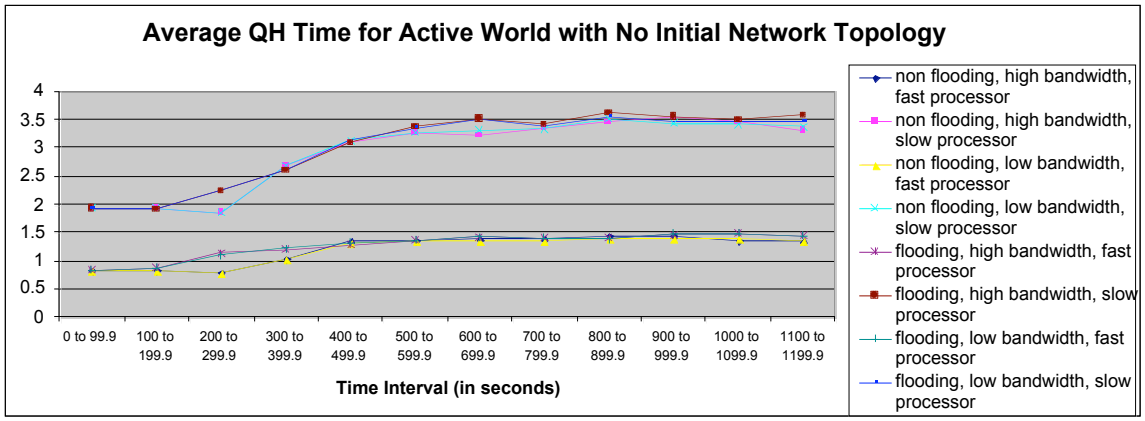

Figure B.10. Average QH Time for Active World without Initial Network Topology.

B.2 Moderate World

Small World as initial network topology

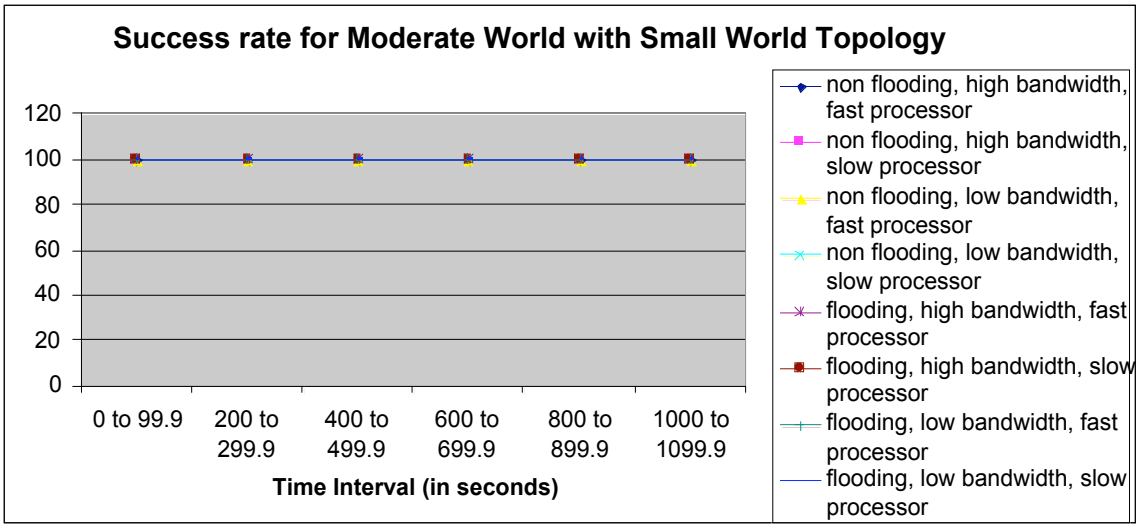

Figure B.11. Success Rate for Moderate World with Small World Topology.

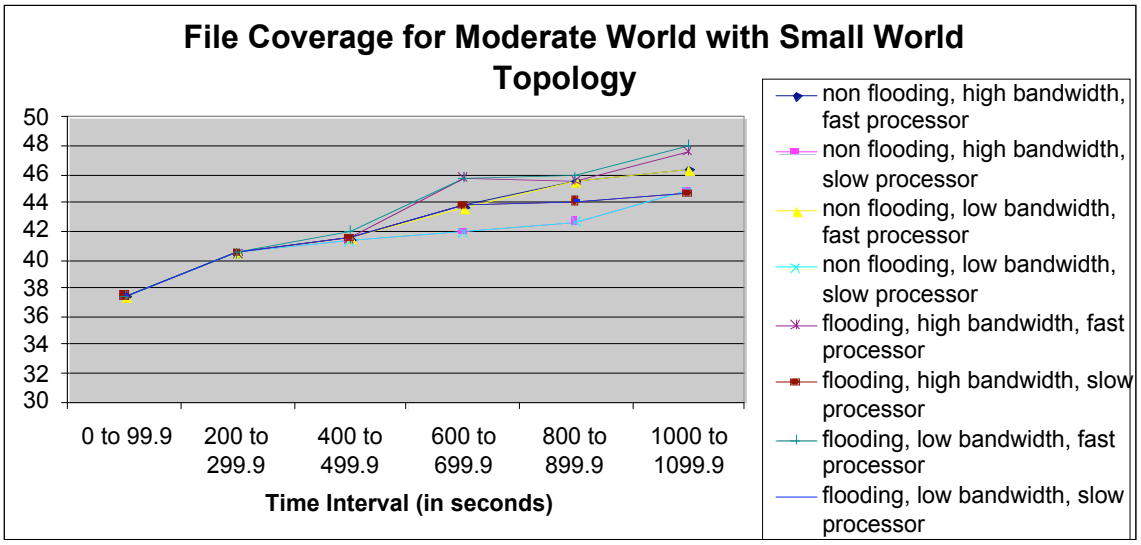

Figure B.12. File Coverage for Moderate World with Small World Topology.

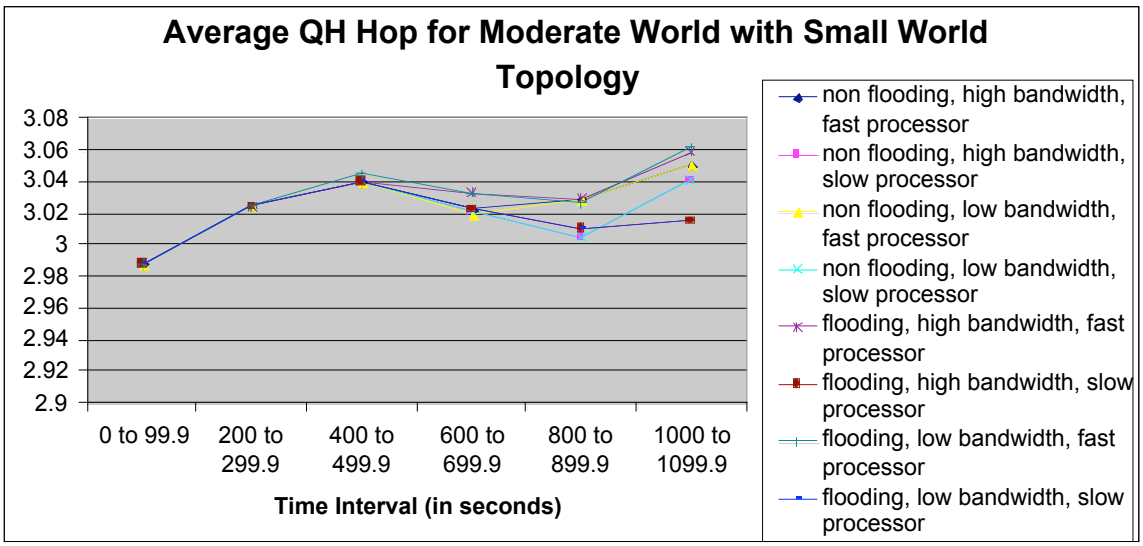

Figure B.13. Average QH Hop for Moderate World with Small World Topology.

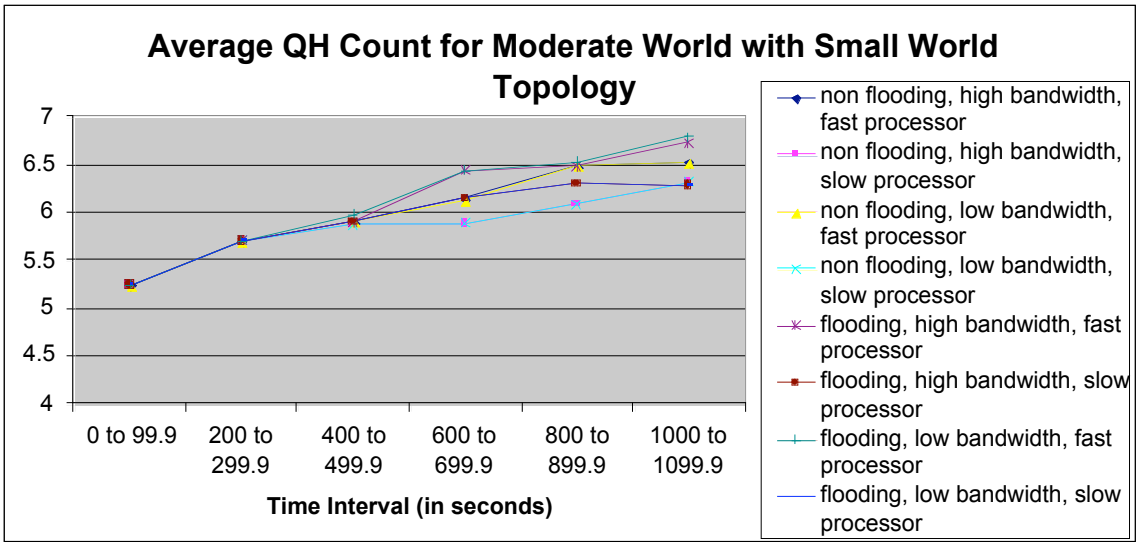

Figure B.14. Average QH Count for Moderate World with Small World Topology.

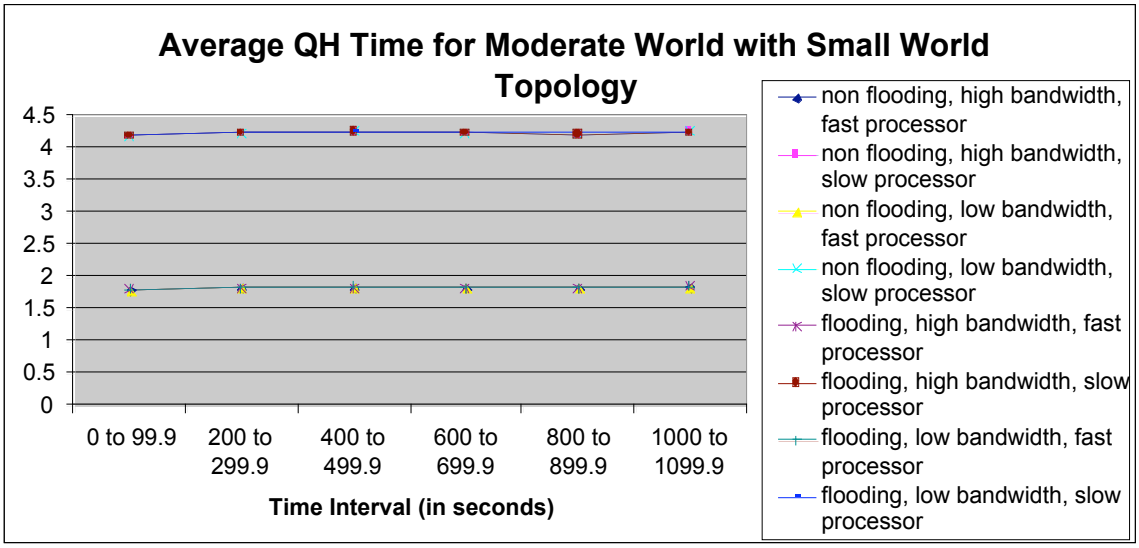

Figure B.15. Average QH Time for Moderate World with Small World Topology.

Without initial network topology

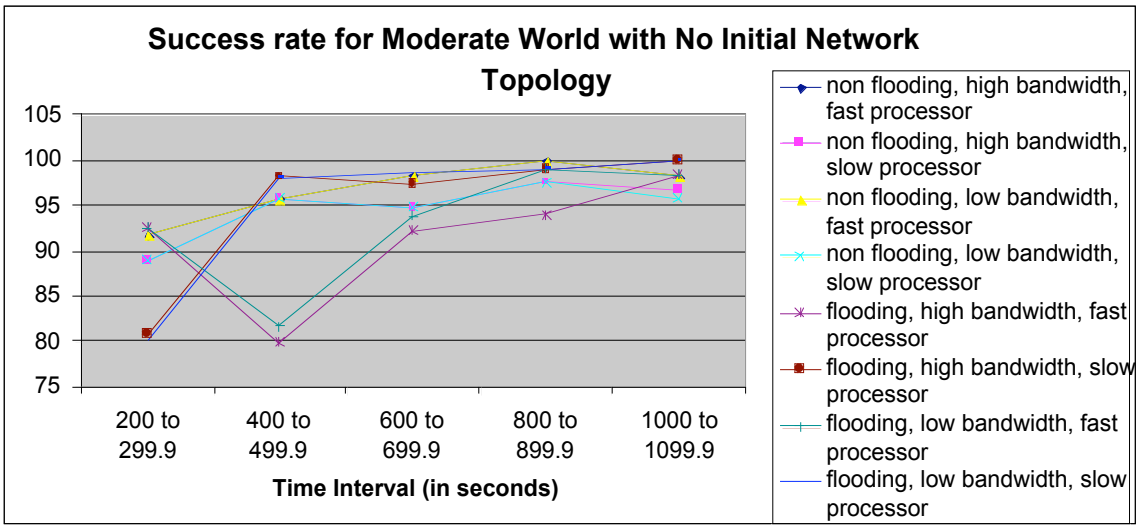

Figure B.16. Success Rate for Moderate World without Initial Network Topology.

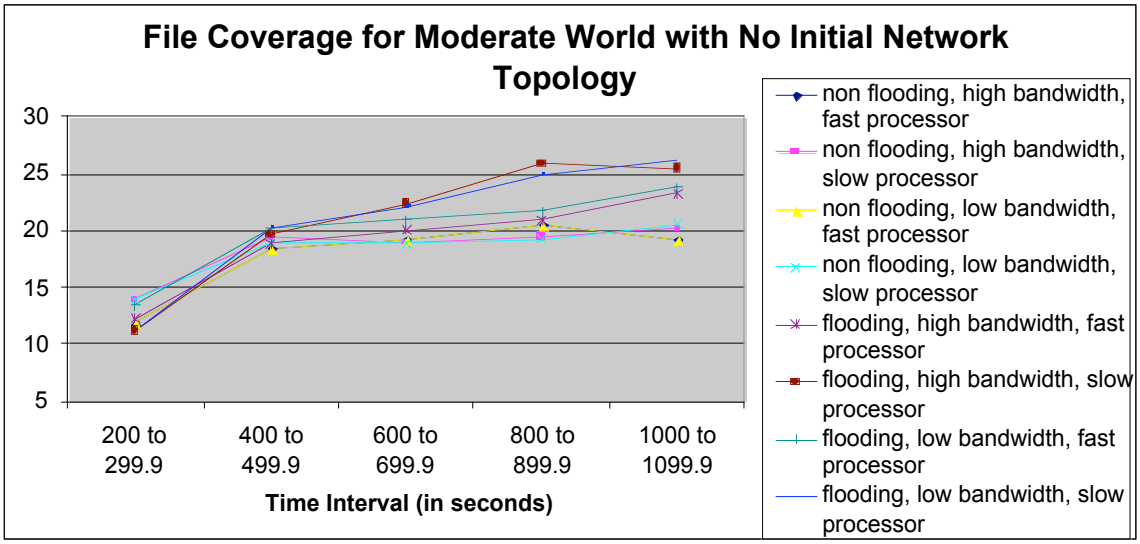

Figure B.17. File Coverage for Moderate World without Initial Network Topology..

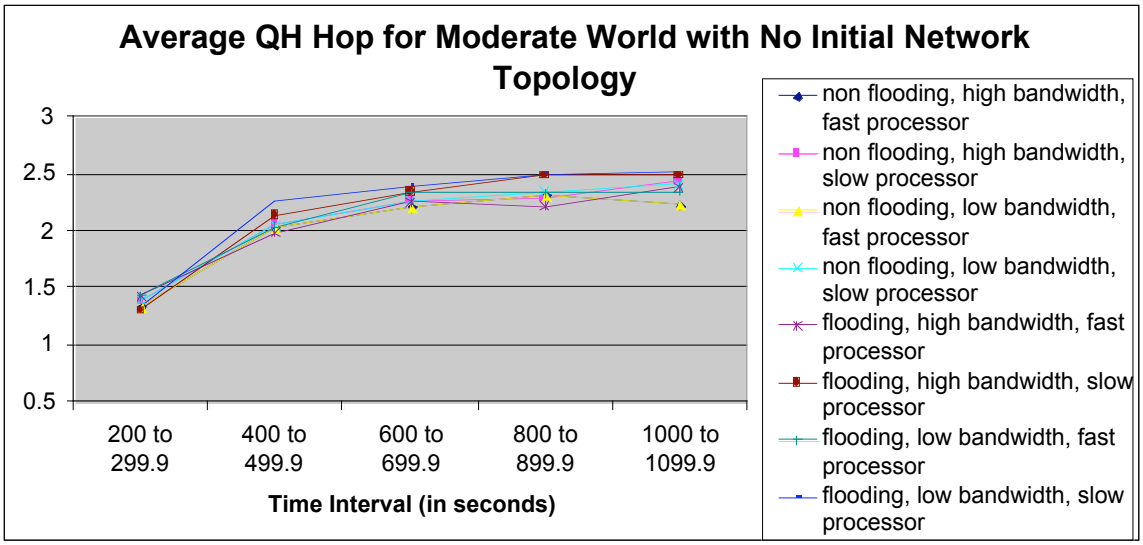

Figure B.18. Average QH Hop for Moderate World without Initial Network Topology..

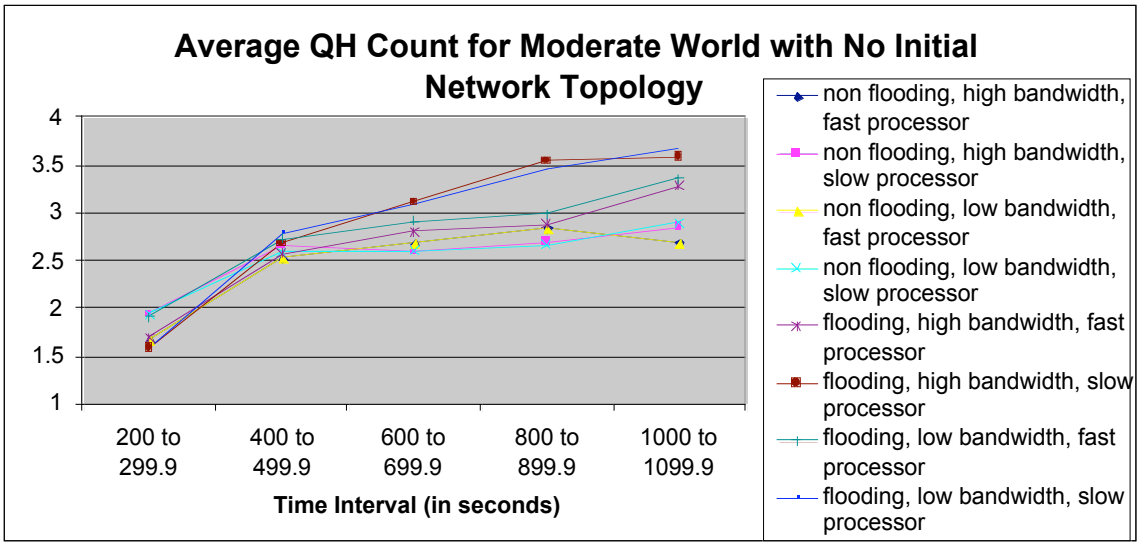

Figure B.19. Average QH Count for Moderate World without Initial Network Topology.

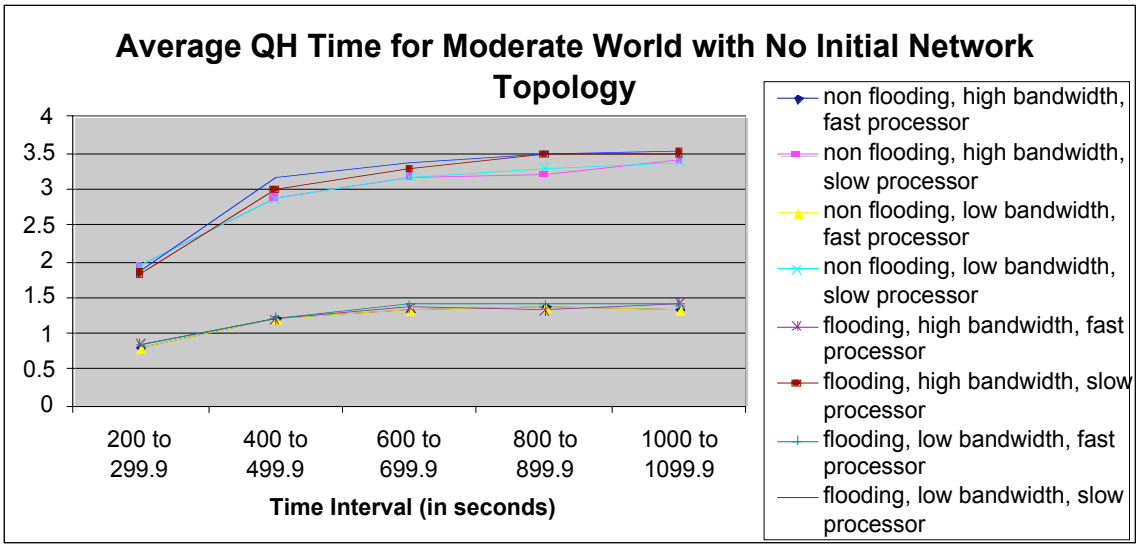

Figure B.20. Average QH Time for Moderate World without Initial Network Topology.

B.3 Distributed World

Small World as initial network topology

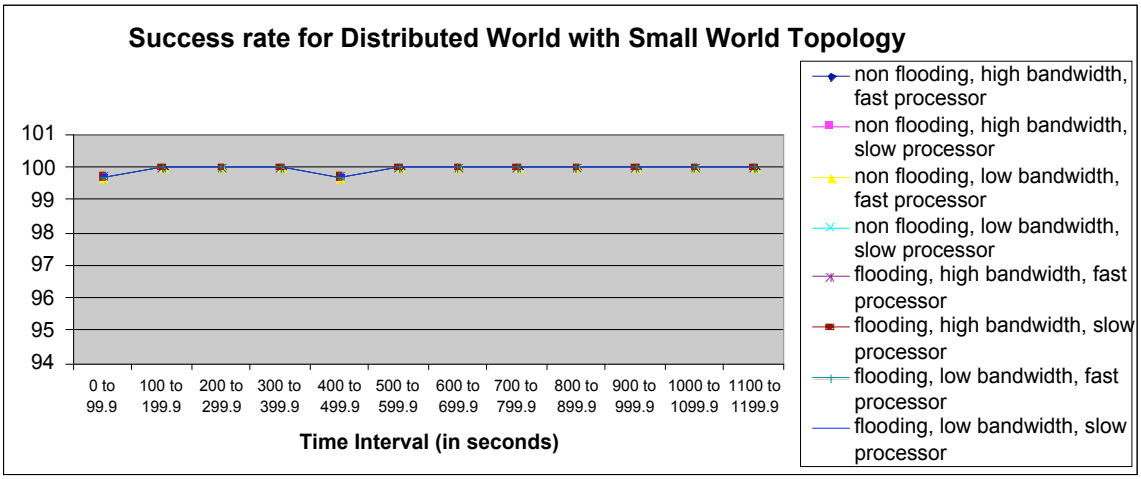

Figure B.21. Success Rate for Distributed World with Small World Topology.

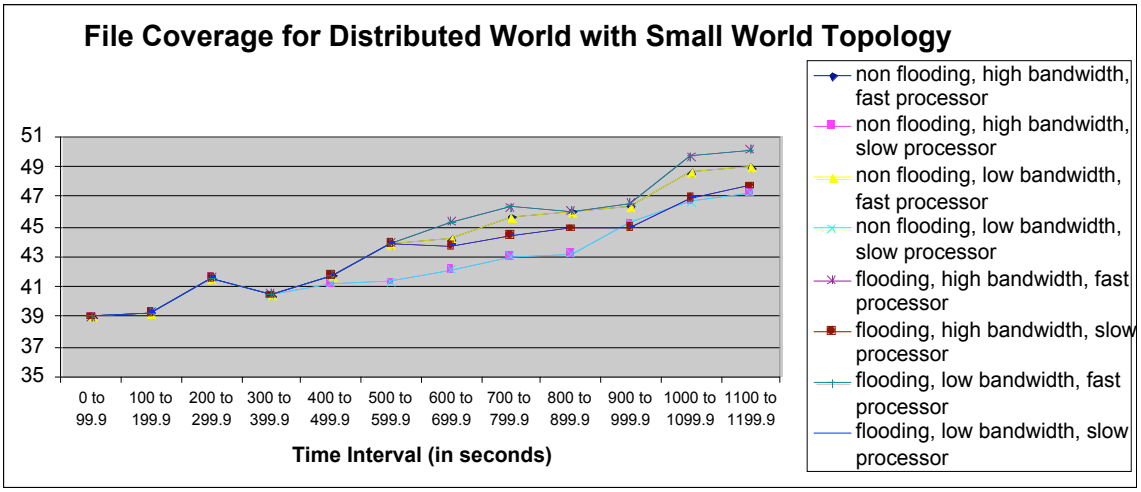

Figure B.22. File Coverage for Distributed World with Small World Topology

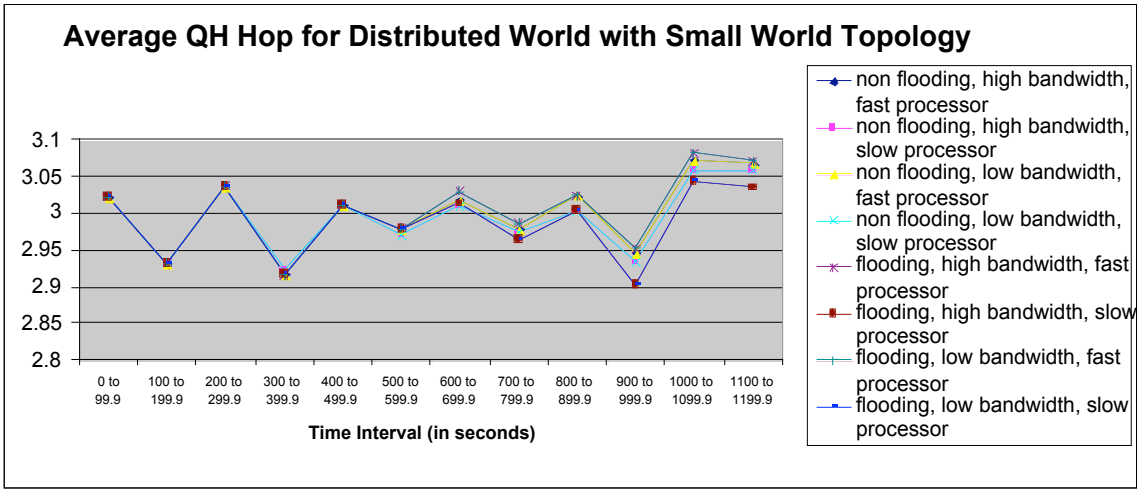

Figure B.23. Average QH Hop for Distributed World with Small World Topology.

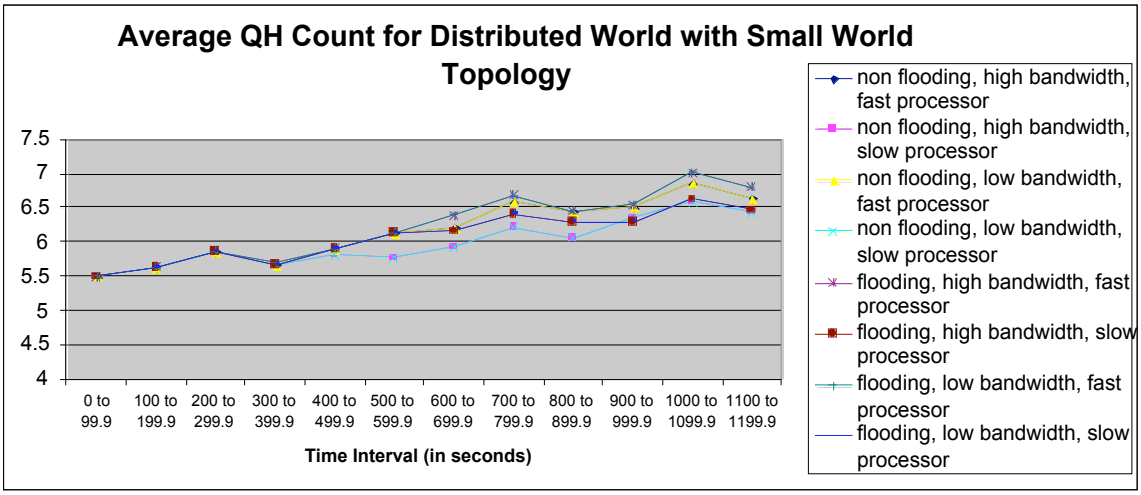

Figure B.24. Average QH Count for Distributed World with Small World Topology.

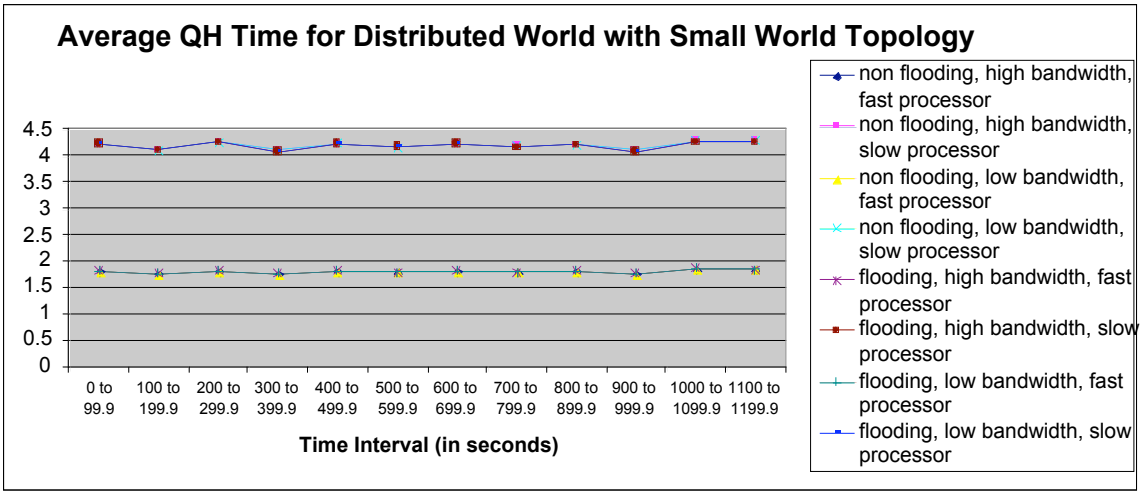

Figure B.25. Average QH Time for Distributed World with Small World Topology.

Without initial network topology

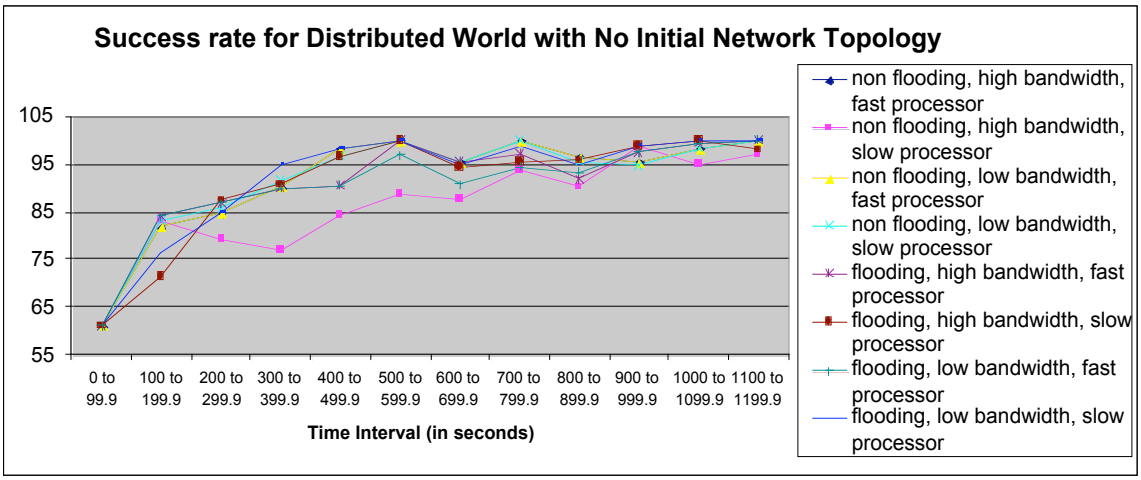

Figure B.26. Success Rate for Distributed World without Initial Network Topology.

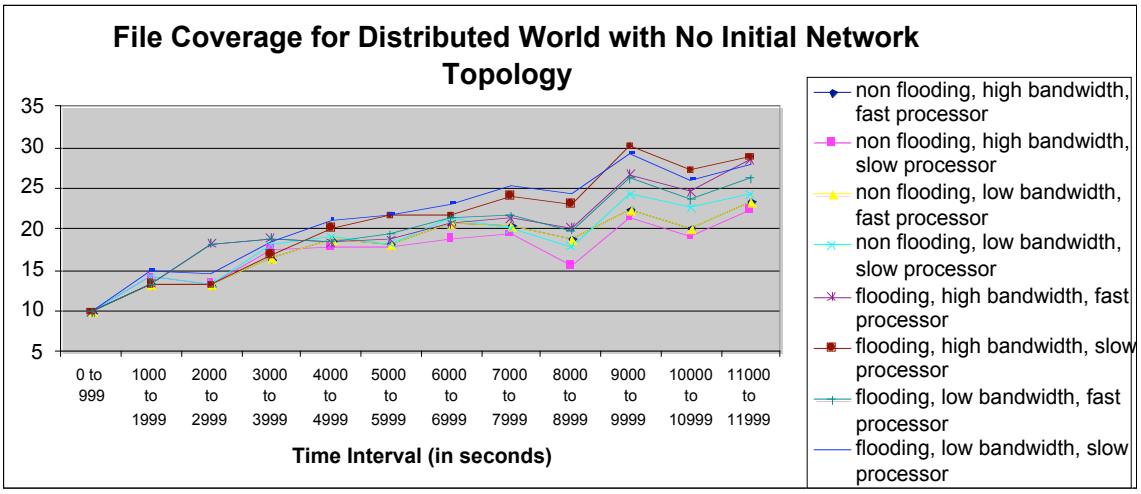

Figure B.27. File Coverage for Distributed World without Initial Network Topology.

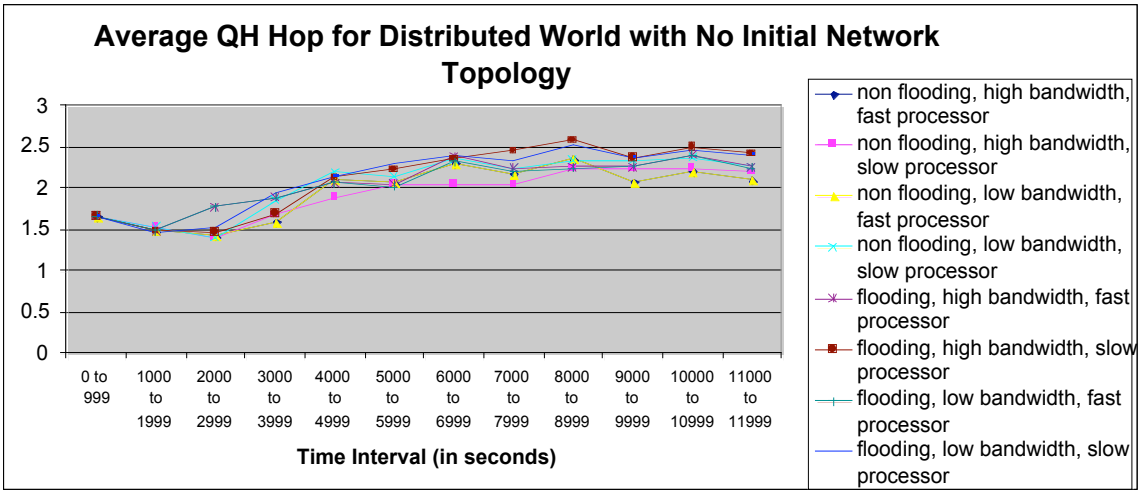

Figure B.28. Average QH Hop for Distributed World without Initial Network Topology.

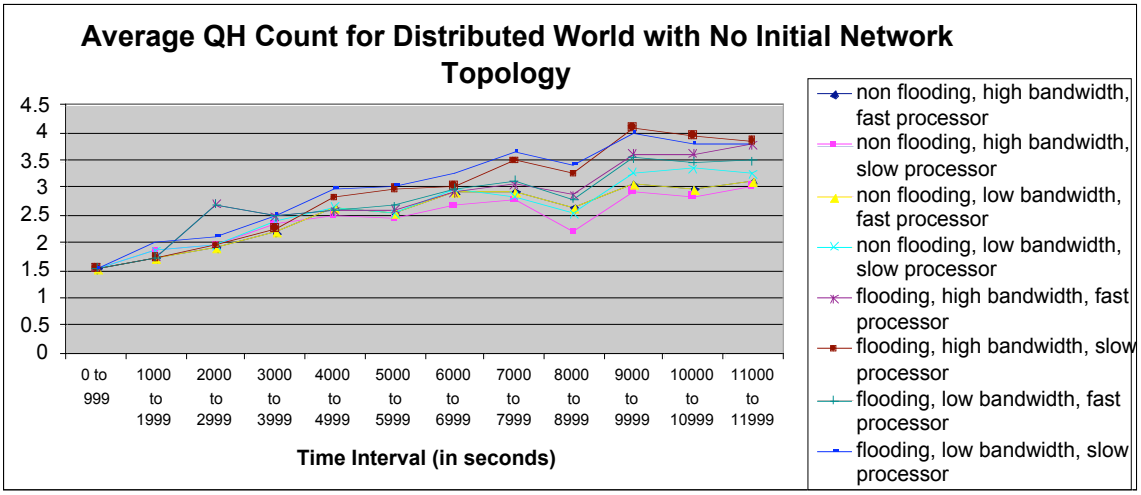

Figure B.29. Average QH Count for Distributed World without Initial Network Topology.

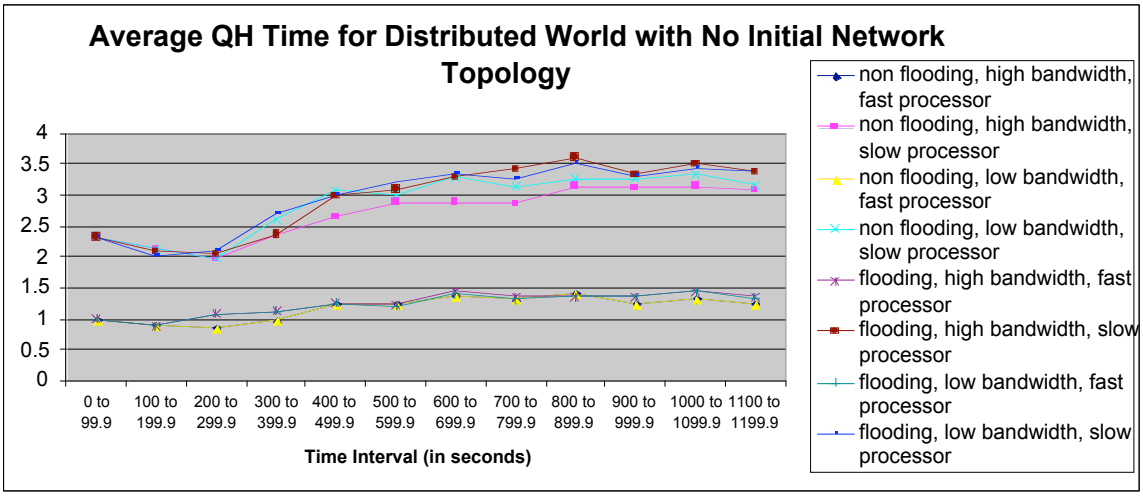

Figure B.30. Average QH Time for Distributed World without Initial Network Topology.