Algorithms for Reliable Peer-to-Peer Networks

Rita Hanna Wouhaybi

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Graduate School of Arts and Sciences

Columbia University

2006

©2006

Rita Hanna Wouhaybi

All Rights Reserved

ABSTRACT

Algorithms for Reliable Peer-to-Peer Networks

Rita Hanna Wouhaybi

Over the past several years, peer-to-peer systems have generated many headlines across several application domains. The increased popularity of these systems has led researchers to study their overall performance and their impact on the underlying Internet. The unanticipated growth in popularity of peer-to-peer systems has raised a number of significant problems. For example, network degradation can be observed as well as loss of connectivity between nodes in some cases, making the overlay application unusable. As a result many peer-to-peer systems can not offer sufficient reliability in support of their applications. This thesis addresses the problem of the lack of reliability in peer-to-peer networks, and proposes a number of algorithms that can provide reliability guarantees to peer-to-peer applications. Note that reliability in a peer-to-peer networking context is different from TCP type reliability. We define a reliable peer-to-peer as a network that is resilient to changes such as network dynamics, and can offer participating peers increased performance when possible. We make the following contributions to area of peer-to-peer reliability:

- we propose an algorithm that creates resilient low-diameter topologies that guarantee an upper bound on delays among nodes;
- we study parallel downloads in peer-to-peer networks and how they affect

nodes by looking at their utilities and the overall performance of the network; and

- we investigate network metrics relevant to peer-to-peer networks and their estimation using incomplete information. While we focus on latency and hop count as drivers for improving the performance of the peers, the proposed approach is more generally applicable to other network-wide metrics (e.g., bandwidth, loss).

Our research methodology encompasses simulations and analytical analysis to understand the behavior and properties of the proposed systems, and substantial experimentation, as practical proof of concept of our ideas, using the PlanetLab platform. The common overarching theme of the thesis is the design of new resilient network algorithms capable of offering high-performance to peers and their applications.

As more and more applications rely on underlying peer-to-peer topologies, the need for efficient and resilient infrastructure has become more pressing. A number of important classes of topologies have emerged over the last several years, all of which have various strengths and weaknesses. For example, the popular structured peer-to-peer topologies based on Distributed Hash Tables (DHTs) offer applications assured performance, but are not resilient to attacks and major disruptions that are likely in the overlay. In contrast, unstructured topologies where nodes create random connections among themselves on-the-fly, are resilient to attacks but can not offer performance assurances because they often create overlays with large diameters, making some nodes practically unreachable. In our first

contribution, we propose *Phenix*, an algorithm for building resilient low-diameter peer-to-peer topologies that can resist different types of organized and targeted malicious behavior. Phenix leverages the strengths of these existing approaches without inheriting their weaknesses and is capable of building topologies of nodes that follow a power-law while being fully distributed requiring no central server, thus, eliminating the possibility of a single point of failure in the system. We present the design and evaluation of the algorithm and show through extensive analysis, simulation, and experimental results obtained from an implementation on the PlanetLab testbed that Phenix is robust to network dynamics such as bootstrapping mechanisms, joins/leaves, node failure and large-scale network attacks, while maintaining low overhead when implemented in an experimental network.

A number of existing peer-to-peer systems such as Kazaa, Limewire and Overnet incorporate parallel downloads of files into their system design to improve the client's download performance and to offer better resilience to the sudden departure or failure of server nodes in the network. Under such a regime, a requested object is divided into chunks and downloaded in parallel to the client using multiple serving nodes. The implementation of parallel downloads in existing systems is, however, limited and non-adaptive to system dynamics (e.g., bandwidth bottlenecks, server load), resulting in far from optimal download performance and higher signaling cost. In order to capture the selfish and competitive nature of peer nodes, we formulate the utilities of serving and client nodes, and show that selfish users in such a system have incentives to cheat, impacting the overall performance of nodes participating in the overlay. To address this challenge, we design a set of strategies that drive client and server nodes into situations where they have to

be truthful when declaring their system resource needs. We propose a *Minimum-Signaling Maximum-Throughput (MSMT)* Bayesian algorithm that strives to increase the observed throughput for a client node, while maintaining a low number of signaling messages. We evaluate the behavior of two variants of the base MSMT algorithm (called the Simple and General MSMT algorithms) under different network conditions and discuss the effects of the proposed strategies using simulations, as well as experiments from an implementation of the system on a medium-scale parallel download PlanetLab overlay. Our results show that our strategies and algorithms offer robust and improved throughput for downloading clients while benefiting from a real network implementation that significantly reduces the signaling overhead in comparison to existing parallel download-based peer-to-peer systems.

Network architects and operators have used the knowledge about various network metrics such as latency, hop count, loss and bandwidth both for managing their networks and improving the performance of basic data delivery over the Internet. Overlay networks, grid networks, and p2p applications can also exploit similar knowledge to significantly boost performance. However, the size of the Internet makes that task of measuring these metrics immense, both in terms of infrastructure requirements as well as measurement traffic. Inference and estimation of network metrics based on partial measurements is a more scalable approach. In our third contribution, we propose a learning approach for scalable profiling and prediction of inter-node properties. Partial measurements are used to create signature-like profiles for the participating nodes. These signatures are then used as input to a trained Bayesian network module to estimate the different network

properties. As a first instantiation of these learning based techniques, we have designed a system for inferring the number of hops and latency among nodes. Nodes measure their performance metrics to known landmarks. Using the obtained results, nodes proceed to create anonymous signature-like profiles. These profiles are then used by a Bayesian network estimator in order to provide nodes with estimates of the proximity metrics to other nodes in the network. We present our proposed system and performance results using real network measurements obtained from the PlanetLab platform. We also study the sensitivity of the system to different parameters including training sets, measurement overhead, and size of the network. Though the focus is on proximity metrics, our approach is general enough to be applied to infer other metrics of interest, potentially benefiting a wide range of applications.

Contents

List of Figures

Acknowledgements

I would like to start by expressing my thanks and gratitude to my advisor, Andrew T. Campbell. His way of thinking inspired my critical thinking. Andrew's approach to research helped me in reaching my potential while maintaining my enthusiasm for the subject matter. His advice during the long Ph.D. process helped me in maintaining my focus and guiding me towards a better future. For all the times that Andrew encouraged me and told me I can make it while challenging my ideas, I express my gratitude. I am also grateful for his support during my stay at Columbia and for introducing me to many researchers in both academic and industrial circles.

I would like to thank Professor Aurel A. Lazar for introducing me to the world of scale-free networks, and game theory methods. The endless chats in the hallways with Professor Lazar, as well as his comments on many of my ideas were priceless. I express my thanks to Professor Edward G. Coffman for introducing me to many fields during the Comet Lab Coffee Hours. My gratitude to Professors Vishal Misra, Keith Ross, and Dan Rubenstein, as well as Jack Brassil for taking time from their busy schedules to sit on my committee.

During my stay at Columbia, I had the pleasure of spending two enriching in-

ternships at excellent research labs (Intel IT Research, and HP Labs). While doing so, two wonderful researchers mentored me. For that and their many advices and guidance, I wish to express my gratitude to John Vicente (Intel IT Research) and Sujata Banerjee (HP Labs). Without John and Sujata, my career decisions might have been very different. Thank you for introducing me to industry research and being frank and welcoming. I also want to thank the entire teams of Intel IT Research and NAPA group at HP Labs for their support and feedback on my work.

Many colleagues at Columbia's COMET Lab were not only great friends but excellent critiques of my work. For that and all the long late hours we spent discussing research over coffee, thanks. In fact, I was lucky to have met each and every one of you, and hope to keep hearing from you all.

On the personal level, I would like to thank my relatives back in Lebanon, and their support. Lots of thanks and appreciation to Sharon L. Middleton and her great presence in my life, her wonderful sense of humor when I most needed it, and her accommodation for my crazy work schedule. Last but not least, I wish to dedicate this work to the souls of Isabelle and Hanna. Mom and dad, thank you for teaching me to never stop dreaming. Without your love, dedication, and persistence, I would have been a very different person. I am grateful for you, eternally.

Chapter 1

Introduction

1.1 Overview

The phenomenal success and popularity of peer-to-peer (p2p) networks over the last decade took many providers, computer experts, as well as the public, by surprise. What started out initially as a modest software program for file sharing and downloading music (by the now "infamous" Napster system [58]) suddenly became a platform of great interest and importance to a wide range of user communities, in the home and the business enterprise. The freedom and flexibility offered by peer-to-peer networks results from the fact that the rigid communications model represented by the server-client relationship, which has been dominant for a number of years now, has collapsed offering peer-to-peer users total control over their communications patterns. This freedom comes, however, with a certain cost because end-users become responsible for providing and managing network and computing resources. As a matter of fact, applications and services using peer-

to-peer technology have become so popular, that service providers, software and hardware manufacturers, researchers, and even lawyers have dedicated a considerable amount of effort to try to influence and contribute toward the evolution of peer-to-peer networks and its technologies.

Technically speaking, a pure peer-to-peer network has the following characteristics [8]:

1. Each end-user (also called a peer) is a client and a server at the same time. Peer-to-peer applications often refer to an end-user as a servlet (a concatenation of the words "server" and "client").

2. The whole network is totally distributed, where there is no central authority that dictates roles or manages the network, in any way.

3. Routing is totally distributed and uses local information. Each end-user is, typically, connected to a small number of other end-users (also called nodes), resulting in a partial view or knowledge of the network topology, for any node.

Because these characteristics are in contrast to the server-client architecture, peer-to-peer networks present a number of new challenges and constraints. The distributed nature of peer-to-peer networks is a design choice not a hardened rule. For example, when Napster [58] appeared in 1999, it included a central server that managed the database of available files on the network, keeping track of the availability of files and their location. The server maintained the routing table for the whole network. However, as the system was shutdown due to copyright laws

infringement, the community realized the need for a fully-distributed network that has no single point of failure. As a result, Gnutella [37] came into existence in 2000, and became the predecessor to many more present day systems, such as KaZaA [78], Morpheus [82], and LimeWire [53], to name a few.

Peer-to-peer networks have evolved since the first appearance of Napster in 1999, and so have the challenges, problems and obstacles. Researchers are continuously challenged by an evolving problem space which includes, but is not limited to:

- Topologies of peer-to-peer networks [81] [69] [74] where researchers studied applying Distributed Hash Tables (DHTs) to distribute the database of the network, carrying information such as file and duplicate locations. Other systems [86] [78] tried to create a less rigid structure than DHTs while providing some bound on the number of hops between nodes.

- Security and attacks in peer-to-peer networks where researchers have studied many security problems that appeared and continue to appear in peer-to-peer networks. Solutions have been proposed for censorship resistance [86], anonymous connections [19], poisoning and polluting attacks [18], denial of service attacks [33], encryption [83], as well as other problems.

- Applications of peer-to-peer networks where researchers found innovative ways to provide improved performance to peers and higher availability [46] [25] of the overall network by exploiting topology. In fact, some problems were not possible to solve under current technology limitation had they not been adapted to a peer-to-peer topology [60].

- Incentives, Cooperation and Reputation in peer-to-peer networks where researchers dealt with solving the problem of free riders on the network (nodes that take benefit of the network by being only clients and do not serve anything in return) [45] [40] [91].

- Performance of peer-to-peer networks [17] [54] [20] [50] [4] where researchers have looked into various improvements for fault tolerance, content caching, replication, as well as other performance metrics.

As mentioned earlier, peer-to-peer networks changed the networking platform by moving from a traditional server-client environment to one where end-nodes have the freedom to communicate to any subset of nodes they deem appropriate and rely upon these nodes to provide their connections to the rest of the network. Such a major change in the topology requires, in our opinion, a different class of solutions that carries a higher degree of sophistication. At the same time, the solutions should be reliable and scalable facing the ever-changing nature of peer-to-peer networks. In order to address this, we have reached into other fields where computer scientists, typically, do not venture, to borrow appropriate solutions, for pressing problems in peer-to-peer systems. In each case, we looked for a solution in a field where "reliability" have been studied, achieved, and tested with success, while dealing with the unpredictability of other nodes and a dynamic system, whether such an area is social sciences, economics, or machine learning. Note, we define reliability (which is distinct from merely reliable communication as achieved by using a reliable transport protocol such as TCP) in a peer-to-peer networking context as a peer network that is resilient to changes (e.g., network

dynamics, attacks, etc.), and can offer peers increased performance when possible. In doing so, this thesis provides reliable algorithms for peer-to-peer networks, by empowering nodes with efficient yet simple techniques. We argue that existing peer-to-peer algorithms are often not scalable because developers have mainly tweaked client-server solutions without re-thinking the problems at hand. This thesis addresses a range of problems in peer-to-peer networks that limit the resilience and performance of the peer network, and proposes new scalable solutions.

With the absence of a central server or authority in peer-to-peer systems, reliability becomes a significant challenge, and even more so as the number of nodes increases in the system. Peer-to-peer networks are often criticized as not having sufficient a level of reliability for the prime-time business domain. Researchers have often tried to solve such problems by tweaking solutions devised for server-client networks. Because the peer-to-peer network paradigm is very different from client-server such solutions are not real remedies. Rather, they often have a breaking point that is easily achieved as the number of nodes on the network increases driving the complexity of the network.

In this thesis, we study peer-to-peer reliability as an overarching challenge and propose a solution that can be viewed along three axis. First of all, we argue that a reliable topology that has upper limits on its response time is essential for any peer-to-peer application. Such an upper bound should not sacrifice resilience for performance, thus, we study topologies that can provide a low-diameter topology while preserving the resilience of the network connectivity under the most severe dynamic conditions of nodes join and leave, as well as targeted attacks.

Second, we devise a system that allows a client node on a peer-to-peer network to take advantage of available resources provided by other server nodes in parallel, thus maximizing its benefit. Third, we propose an algorithm that can provide nodes with an estimation of metrics of other nodes, including round trip delay and node hops among others, providing nodes with information about the network as a whole. In doing so, we propose a flexible general framework that can be used for a number of different possible metrics depending on the needs of the overlaying applications and nodes. Such a system moves the functionalities into the end nodes which is in agreement of the whole end-to-end approach of peer-to-peer networks. We describe next the problems in existing peer-to-peer networks and how they affect their notion of reliability.

1.2 Technical Barriers

We now discuss the technical barriers facing the problems presented above and how they affect the system performance in a peer-to-peer network.

1.2.1 Low-Diameter Resilient Topologies

When Gnutella appeared, the main focus was to create a "resilient" topology, in the sense that there is no single point of failure, whose removal can bring the network down. Thus, each node in a Gnutella network [37] connects to a random subset of the existing nodes on the network creating a random graph topology [24]. Such a topology guarantees a resilient graph where shutting down the network, or at least disconnecting it into separate sub-graphs require the removal of a large

number of existing nodes. Keeping in mind that Gnutella came into existence after Napster was shutdown (simply by disconnecting the central server) Gnutella's focus was on creating resilience in terms of connectivity without paying attention to the effect of such a topology on the performance of the network as a whole.

As nodes join the network, running the Gnutella protocol, they connect to a random subset of existing nodes, creating what is mainly a random graph. The problem with a random graph is its high "diameter", where a diameter is defined as the average distance between any two nodes on the network in hops. As the number of nodes increases, the diameter increases linearly. Gnutella is mainly used for file exchange. After a node joins the network, it initiates one or more queries for specific objects. It forwards the queries to the nodes it connects to, typically referred to as "neighbors," which in their turn forward the queries to their neighbors, except if they carry the file requested themselves. This mechanism of forwarding queries is typically known as flooding, which can generate exponential traffic growth, if not limited by an upper bound for the number of forwards to be done, also known as TTL (Time To Live). Thus, each forwarding peer receiving a query decreases the TTL by 1. When the TTL reaches zero, the query is dropped and the file is declared unfound. Typically, the TTL is set to 7 in Gnutella.

At first, the number of nodes in Gnutella was under 100,000 [72], making most nodes reachable within the 7 hops enforced by the TTL. However, as the number of nodes started increasing, nodes faced a problem where they could not reach a considerable number of other existing nodes on the network due to the random topology. This translated into many queries failing despite the fact that nodes did carry the required files, but were more than 7 hops away from the requesting node.

As a result, nodes became restricted to the most common files on the network, as they were sufficiently replicated so that they can be found with such a flooding query. Thus, the network suffered from a large diameter that often was much bigger than 7 (the TTL). Because peer-to-peer networks rely on end-users, creating a scalable low-diameter topology raises a number of tecnhical challenges:

- Nodes have partial knowledge of the existing nodes and their interconnections. Thus, a node cannot calculate its list of optimal neighbors, and has to deal with incomplete information.

- Nodes are typically very dynamic, where some can join and leave the network in the order of seconds while other nodes stay for an extended period of time. Thus, any rigid structure, such as a tree, would be costly to maintain.

- Nodes can be malicious and should not be trusted. Thus, each node should be suspicious and any algorithm has to be adaptive to fast and aggressive attacks, otherwise, the resilience of the network will be compromised.

1.2.2 Optimizing the Use of Multiple Server Nodes

The first generation of peer-to-peer networks, as defined by Gnutella v0.4 [37], requires a node i to run a query for a needed object O by flooding. Once a node j carrying the object in question O is found, it returns an answer to i indicating the availability of O. Node i is then called the client node and node j the serving node, acting as a server for node i.

If object O is a popular object, then the probability of finding more than one serving node carrying it becomes higher. In Gnutella v0.6, a client node i takes advantage of this situation of multiple serving nodes, by dividing the object O into chunks and downloading these chunks in parallel from several serving nodes. Since end-users often have a higher download bandwidth then upload bandwidth, parallel downloads benefit node i by increasing its download throughput to an upper limit equal to the summation of the upload bandwidth of all serving nodes.

In peer-to-peer networks, nodes are often very dynamic, and might leave a network even if they were in the middle of serving an object to a client node. Thus, a client node i, downloading a certain object in parallel from several serving nodes, is enjoying a resilient service. In the event that one or more of the serving nodes disappear, node i does not have to restart the download of the entire object from another serving node. Rather, only the chunks whose downloads were interrupted are requested from the remaining serving nodes. This adds to the resilience of the object download as a whole.

Such a problem of multiple serving nodes is not new, as it was studied thoroughly in the area of Content Distribution Networks (CDN), where, by definition, multiple servers carry the same content whether it is web content or any other application. However, in sharp contrast to CDNs, where servers are well maintained by professional personnel, peer-to-peer networks tend to be very dynamic and the performance of nodes is quite often sporadic and unpredictable.

Thus, parallel downloads in peer-to-peer networks face many challenges:

- Serving nodes are often dynamic and their performance unpredictable. A

client node has to adapt to their changes in the absence of explicit knowledge about their behavior. Client nodes can only rely on their own observations.

- Client nodes are selfish and want to take advantage of the maximum available resources, a fact that might lead them to cheat and declare untruthful intents.

- Serving nodes are also selfish, and their behavior should be studied and taken into consideration when designing any parallel download algorithm.

1.2.3 Estimating Node Metrics Using Partial Information

Typically, a node has a limited and partial view of a peer-to-peer network. However, as the need for reliable services and applications increases, nodes require a more global knowledge of certain metrics on the network. For example, in a video streaming application, nodes value connecting to other nodes that can be reached within a short round trip delay. While in a disaster relief application, nodes might be more interested in connecting to nodes with the longest lifetime on the network.

Thus, depending on the application, nodes are often interested in a metric or a set of metrics, on a global scale covering all other nodes on the network. Considering that a network has N nodes, then if every node has to conduct its own measurements of such a metric, in order to determine its optimal deterministic connections, then the network performs $N(N-1)$ measurements. Add to that the dynamic nature of the nodes and their connectivity, resulting in repeating these

measurements quite often, we end up with a system generating traffic in the order of $O(N^2)$. Such a system is, at best, not scalable.

Thus, the challenges in determining network metrics are as follows:

- Nodes have to deal with partial knowledge of the network, and conduct a fraction of the complete set of measurements. Thus, the measurements should be well designed so that general behavior can be captured.
- Nodes have to predict changes in metrics in the future as well as correlate information collected, so that repeated measurements are less frequent.
- Any estimation mechanism should be general enough to be applied to several metrics and adaptive to many applications and their needs.

1.3 Thesis Outline

In this thesis, we propose a number of algorithms that can be used by applications to improve on the reliability and performance of peer-to-peer networks. We start by proposing low-diameter resilient topologies for peer-to-peer networks relying on partial information. We then present a formal model for parallel downloads in peer-to-peer networks and propose an algorithm that can achieve optimal performance for both client and server nodes. Finally, we devise a general scalable framework that nodes can use to estimate important metrics globally, using partial local information. We test our systems using the PlanetLab [66] platform, evaluating their usability and characteristics in an operational network.

1.3.1 Building Resilient Low-Diameter Peer-to-Peer Topologies

Unstructured networks, based on random connections are limited in the performance and node reachability they can offer to applications. In contrast, structured networks impose predetermined connectivity relationships between nodes in order to offer a guarantee on the diameter among nodes. We observe that neither structured nor unstructured networks can simultaneously offer both good performance and resilience in a single algorithm. To address this challenge, we propose Phenix, in Chapter 2, a peer-to-peer algorithm that constructs low-diameter resilient topologies. Phenix supports low diameter operations by creating a topology of nodes whose degree distribution follows a power-law, while the implementation of the underlying algorithm is fully distributed requiring no central server, thus, eliminating the possibility of a single point of failure in the system. We present the design and evaluation of the algorithm and show through analysis, simulation, and experimental results obtained from an implementation on the PlanetLab testbed [66] that Phenix is robust to network dynamics such as joins/leaves, node failure and large-scale network attacks, while maintaining low overhead when implemented in an experimental network.

1.3.2 Strategies and Algorithms for Parallel Downloads in Peer-to-Peer Networks

Chapter 3 starts by proposing an analytical model for parallel downloads in peer-to-peer networks. To address the challenges of such a system, we design a set of strategies that drive client and serving nodes into situations where they have to be

truthful when declaring their system resource needs. We propose the Minimum-Signaling Maximum-Throughput (MSMT) Bayesian algorithm that strives to increase the observed throughput for a client node, while maintaining a low number of signaling messages. We evaluate the behavior of two variants of the base MSMT algorithm (called the Simple and General MSMT algorithms) under different network conditions and discuss the effects of the proposed strategies using simulations, as well as experiments from an implementation of the system on a medium-scale parallel download PlanetLab overlay. Our results show that our strategies and algorithms offer robust and improved throughput to downloading clients while benefiting from a real network implementation that significantly reduces the signaling overhead in comparison to existing parallel download-based peer-to-peer systems.

1.3.3 A Learning Based Approach for Network Properties Inference

In Chapter 4, we propose a learning approach for scalable profiling and predicting inter-node properties. Partial measurements are used to create signature-like profiles for the participating nodes. These signatures are later used as input to a trained Bayesian network module to estimate the different network properties.

As a first instantiation of these learning based techniques, we have designed a system for inferring the number of hops and latency among nodes. Nodes measure their performance metrics to known landmarks. Using the obtained results, they proceed to create their anonymous signature-like profiles. These profiles are then

used by a Bayesian network estimator in order to provide nodes with estimates of the proximity metrics to other nodes on the network. In Chapter 4, we present our proposed system and performance results from real network measurements obtained from the PlanetLab platform. We also study the sensitivity of the system to different parameters including training set, measurement overhead, and size of network. Though the focus of this chapter is on proximity metrics, our approach is general enough to be applied to infer other metrics and benefit a wide range of applications. In fact, we argue through our results that our approach is very promising, as it makes use of anonymous profiles for nodes coupled with machine learning based estimation modules.

1.4 Thesis Contribution

In what follows, we summaries our contributions to reliable peer-to-peer networks presented in this thesis:

- We propose an algorithm that constructs low-diameter peer-to-peer topologies that do not sacrifice the resilience of the network as a whole, while achieving a diameter of the order $O(logN)$. We draw analogies to connections in social networks that have been widely studied and proven to provide reliability.

- We propose an analytical model for parallel downloads in peer-to-peer networks. We define the utilities of server and client nodes capturing the selfish behavior of nodes. We show the inefficiencies as well as the vulnerabilities

of existing systems implementing parallel downloads.

- We devise an algorithm for parallel downloads that can deal with the unpredictability of nodes using Bayes theorem in order to build profiles for serving nodes. We show how this algorithm can add to the reliability and performance of downloads by approximating optimal solutions.

- We define a general framework for predicting metrics in a peer-to-peer network. We propose algorithms for extracting the characteristic features of the collected measurements, creating anonymous profiles for nodes. We then use these profiles in a machine learning algorithm that can learn and adapt to nodes and network dynamics. Our work in this field includes collecting a large set of measurements on the PlanetLab platform in order to prove the validity of our proposed system. We also show that making profiles anonymous, a feature that sounds counter-intuitive, improves the estimation algorithm.

Chapter 2

Building Resilient Low-Diameter Peer-to-Peer Topologies

2.1 Introduction

Over the past several years, we have witnessed the rapid growth of peer-to-peer applications and the emergence of overlay infrastructure for Internet, however, many challenges remain as this new field matures. The work presented in this chapter addresses the outstanding problem of the construction of resilient peer-to-peer networks and their efficient performance in terms of faster response time and low-diameter operations for user queries. Low-diameter networks are often desirable because they offer a low average distance between nodes, often on the order of $O(logN)$. The two classes of peer-to-peer networks, found in the literature, either offer better resilience to node dynamics such as joins/leaves, node failure and service attacks, as in the case of unstructured networks [37] [78], or they offer

better performance as in the case of structured networks [69] [81] [94]. Because of the inherent tradeoffs in the design space of these different classes of peer-to-peer networks, it is difficult to simultaneously offer better performance and resilience without having to reconsider some of the fundamental design choices made to develop these network systems. We take one such alternative approach and propose a peer-to-peer algorithm that delivers both performance and resilience. The proposed algorithm builds a low-diameter resilient peer-to-peer network providing users with a high probability of reaching a large number of nodes in the system even under conditions such as node removal, node failure, and malicious system attacks. The algorithm does not impose structure on the network, rather, the established graph of network connections has the goal of creating some order from the total randomness found in resilient unstructured networks, such as Gnutella [37] and KaZaA [78].

Unstructured peer-to-peer networks, such as Gnutella, offer no guarantee on the diameter because nodes interconnect in a random manner, usually resulting in an inefficient topology. These unstructured systems are often criticized for their lack of scalability [72], which can lead to partitions in the network resulting in small islands of interconnected nodes that cannot reach each other. However, these same random connections offer the network a high degree of resiliency where the operation of the resulting network as a whole is tolerable to node removal and failure. In contrast, structured peer-to-peer networks based on Distributed Hashing Tables (DHTs), such as Chord [81] and CAN [69] have been designed to provide a bound on the diameter of the system, and as a result, on the response time for nodes to perform queries. However, these systems impose a relatively rigid struc-

ture on the overlay network, which is often the cause of degraded performance during node removals, requiring non-trivial node maintenance. This results in certain vulnerabilities (e.g., weak points) that attackers can target and exploit. Due to the design of DHTs, these structured topologies are also limited in providing applications with the flexibility of generic keyword searches because DHTs rely extensively on hashing the keys associated with objects [2] [16].

These observations motivate the work presented in this chapter. We propose Phenix, a scale-free algorithm that constructs low-diameter P2P topologies offering fast response times to users. An important attribute of Phenix is its built-in robustness and resilience to network dynamics, such as, operational nodes joining and leaving overlays, node failures, and importantly, malicious large-scale attacks on overlay nodes. The main design goals of Phenix can be summarized as follows: *to construct* low-diameter graphs that result in fast response times for users, where most nodes in the overlay network are within a small number of hops from each other; *to maintain* low-diameter topologies under normal operational conditions where nodes periodically join and leave the network, and under malicious conditions where nodes are systematically attacked and removed from the network; *to implement* support for low-diameter topologies in a fully distributed manner without the need of any central authority that might be a single point of failure, which would inevitably limit the robustness and resilience of peer-to-peer networks; and *to support* connectivity between peer nodes in a general and non-application specific manner so a wide-variety of applications can utilize the network overlay infrastructure. An important property of Phenix is that it constructs topologies based on power-law degree distributions with a built-in

mechanism that can achieve a high degree of resilience for the entire network. We show that even in the event of concerted and targeted attacks, nodes in a Phenix network continue to communicate with a low diameter where they efficiently and promptly rearrange their connectivity with little overall cost and disruption to the operation of the network as a whole. To the best of our knowledge Phenix represents one of the first algorithms that builds resilient low-diameter peer-to-peer topologies specifically targeted toward, and derived from, popular unstructured P2P network architectures, such as, Gnutella [37] and KaZaA [78].

In this chapter, we present the design of the Phenix algorithm and evaluate its performance using analysis, simulation, and experimentation. We make a number of observations and show the algorithm's responsiveness to various network dynamics including systematic and targeted attacks on the overlay infrastructure. We implement and evaluate Phenix using the PlanetLab testbed [66]. Experimental results from the testbed implementation quantify the algorithm's overhead and responsiveness to network dynamics for a number of PlanetLab nodes. The chapter is structured as follows. We discuss the related work in Section 3.2 and present the detailed design and operations of Phenix in Section 3.4. Section 2.4 presents a detailed evaluation of the algorithm's operation, followed by Section 3.6, which presents experimental results from the implementation of Phenix on the PlanetLab platform. Finally, we present a summary of the work in Section 3.7.

2.2 Related Work

Traditionally, low diameter networks tend to appear in social networks forming small-world topologies [5], while power-law behavior is often seen in many natural systems as well as man-made environments [1] [29] [43]. These observations led to a body of work related to analyzing and modeling of such networks [5] [10] [47] [49]. The contribution discussed in [9] on preferential attachment has been influential in our thinking. However, the idea of preferential attachment is used in Phenix as a basis to ensure resiliency in a fully distributed, dynamic peer-to-peer environment. The work on peer-to-peer networks presented in [27] makes use of small-world algorithms based on the proposition by Watts and Strogatz [87] on "rewiring" the network. In [27], the idea of rewiring is applied to a Chord [81] overlay. Pandurangan et.al. [63] [64] create a low-diameter peer-to-peer network but rely heavily on a central server that is needed to coordinate the connections between peers. This proposal creates a potential single point of failure in the overlay network. The authors also do not address the resilience of such a network in the event of targeted node removal, various attacks, or misbehaving nodes. Under such conditions the performance of the network would likely degrade and deviate from the low-diameter design goal.

A family of structured peer-to-peer topologies relying on DHTs, such as Chord [81], CAN [69] and Tapestry [94], has attracted considerable attention in the P2P/overlay community. However, such networks might be limited because they unduly restrict the queries that the users can initiate (e.g., keyword queries) due to the use of hashing tables to store objects at overlay nodes. These networks also

couple the application to the underlying infrastructure layer, which makes them attractive to specific applications, but the infrastructure may need to be revised to support changing needs of users. The idea of differentiating the rank of different overlay nodes (e.g., a super node over a regular node) in a peer-to-peer network has been used by a number of systems in order to achieve better performance. For example, KaZaA [78] uses the notion of "supernodes", and Guntella v.0.6 [37] uses "ultrapeers" [85] as supported by the Query Routing Protocol (QRP) [68]. KaZaA creates supernodes among peers by assigning an elevated ranking to nodes with a faster connectivity such as broadband Internet access. However, the implementation details of these popular P2P schemes are not open or published, which makes it difficult to make a comparative statement on the deployed algorithms. Ultrapeers are a standard feature of Gnutella v.0.6, constituting an essential element of QRP, as mentioned above. Ultrapeers differ from what we propose in Phenix in a number of ways. First, ultrapeers act as servers in a hierarchy that is widely known by all other nodes in the network. As a result of this predetermined hierarchy, ultrapeers create a number of vulnerabilities in the system. If ultrapeers were forcefully removed from the network by an attacker, the system would suffer considerably; potentially fragmenting the remaining nodes into disconnected smaller partitions. Another vulnerability arises when malicious nodes assume the role of ultrapeers and mislead other nodes into relying on them for services. An ultrapeer does not use lower level nodes (also called leaves) to relay traffic to other ultrapeers in the network, rather, ultrapeers interact directly with each other. Such reliance could create disconnected groups of nodes in the event that ultrapeers unexpectedly drop out of the network in an uncontrolled manner due to node failure

or forceful removal. Each ultrapeer also keeps state information related to the data held by leaf nodes that are connected to it. Creating such a hierarchy that is closely tied to the application level may call for a complete redesign in the event that the application's needs change or new applications need to be efficiently supported.

In our work, we make a distinction between the type of information carried by packets and the routing decisions that are made. RON [7] and i3 [3] have already been designed based on this approach, where a generic topology is proposed that is independent of the application that makes use of it. Such a topology would be an asset for smart search algorithms [2] [16] that direct queries instead of flooding the entire neighborhood of the requesting node. Finally, in the context of security, secure peer-to-peer and overlay networks have been proposed as policies to protect individual nodes against denial of service (DOS) attacks in the SOS [46] and Mayday [6] systems, but not in the context of an overall resilient P2P network architecture. Phenix addresses the resilience of the entire network and not the individual nodes.

2.3 Phenix Peer-To-Peer Networks

2.3.1 Power-Law Properties

The signature of a power-law or a scale-free network lies in its degree distribution, which is of the form presented in Equation (2.1).

$$p(K) \sim K^{-\gamma} \tag{2.1}$$

Many networks tend to have an exponent γ close to 2, for example, the Internet backbone connectivity distribution is a power law with an exponent $\gamma = 2.2 \pm 0.1$[29]. As a result of this distribution some nodes are highly connected and can act as hubs for the rest of the nodes. These nodes and their position in the network contribute to a highly desirable characteristic of these graphs: a low "almost constant" diameter, defined as, the average shortest path between two nodes in the graph. This graph is capable of growing while maintaining a low diameter hence the name scale-free networks. Typically, unstructured peer-to-peer networks suffer from a large diameter, which often causes the generation of more network traffic. This is inefficient because it requires nodes to either increase the radius of a search for an object, or opt for a low radius search, which would limit the probability of finding less popular objects in the network. These design trade offs result in increased signaling or degraded performance. In the light of these observations, it seems natural to construct a peer-to-peer topology that conforms to a power-law for its node degree distribution. However, for a proposed algorithm to be feasible, it must adhere to a number of design restrictions. First, the algorithm should be easy to implement and make few assumptions about the underlying network. Despite the problems associated with Gnutella, its deployment is widespread as a result of the simplicity of the underlying protocol [37]. Next, the algorithm should be fully distributed based on local control information, and not include any centralization of control, which might become a bottleneck or a target for attacks. Finally, the algorithm should be robust to node removal whether random or targeted. This means that the network should not be easily partitioned into smaller sub-networks and should be capable of maintaining a high level of

resiliency and low diameter in the face of node removal. The main motivation behind Phenix is to allow nodes in the network to "organically" emerge as special nodes (called preferred nodes) with a degree of connectivity higher than the average, so that a scale-free topology can be formed. In other words, we do not dictate special nodes or hierarchies in advance for the topology to emerge or the network to function. As shown in [9], such networks appear in nature due to preferential attachment, where newcomers tend to prefer connecting to nodes that already have a strong presence characterized by their high degree, and the dynamic nature of such networks involving growth. By examining social networks, we can observe the following; if someone joins a new social network, the first network of "friends" is pretty much random. However, most people, after seeing that a specific person has more acquaintances and is better connected to a larger number of members in that specific network, tend to acquire a connection to that person in order to gain better visibility. In fact, [9] shows that if a new node has knowledge of the states of all the existing nodes in the network and their interconnections, it can connect to the nodes with the highest degree giving it the highest visibility and putting it in a place where it is a few hops away from the rest of the network. This will guarantee that the resulting network has a degree distribution conforming to a power-law resulting in a low diameter. However, in a peer-to-peer network having such a global view is practically impossible, since most nodes typically can only see a small fraction of the network, and have to make decisions based solely on local information. We present the detail design of the Phenix Algorithm in the next section and show the emergence of a power-law topology through simulation and experimental results in Sections 2.4 and 3.6, respectively.

After presenting the detail design of the Phenix algorithm in the next section, we show through analysis that Phenix encourages the emergence of preferred nodes that follow power-laws in Section 2.3.4. We reinforce this observation through simulation and experimental results in Sections 2.4 and 3.6, respectively.

2.3.2 Phenix Algorithm Design

In what follows, we describe the Phenix algorithm for the simple case where nodes join the network. A node obtains a list of addresses using a rendezvous mechanism by either contacting a host cache server [35] or consulting its own cache from a previous session in a fashion similar to an initial connection, as described in Guntella v0.6 [37]. However, instead of establishing connections to "live" nodes from the returned list, the joining node divides these addresses into two subsets, as expressed in Equation (2.2): that is, random neighbors and *friends* that will be contacted in the next step.

$$G_{host,i} = [G_{random,i}, G_{friends,i}] \tag{2.2}$$

Then i initiates a request called a "ping message" to the nodes in the list $G_{friends,i}$, sending a message of the form:

$$M_0 = \langle source = i, type = ping, TTL = 1, hops = 0 \rangle \tag{2.3}$$

Each recipient node constructs a "pong message" as a reply containing the list of its own neighbors, increments the hops counter, decrements the TTL, and forwards a new ping message to its own neighbors, as follows:$M_0 = \langle source = i, type = ping, TTL = 0, hops = 1 \rangle$. Each node j receiving such a message

will send no pong message in reply, but instead add the node i to a special list called Γ_j for a period of time denoted by τ. Following this procedure, the node i obtains a new list of all the neighbors of nodes contained in $G_{friends,i}$ and constructs a new list denoted by $G_{candidates,i}$. Then i sorts this new set of nodes using the frequency of appearance in descending order, and uses the topmost nodes to create a new set that we denote as $G_{preferred,i}$, where $G_{preferred,i} \subseteq G_{candidates,i}$. Thus, the resulting set of neighbors to which i creates connections is $G_i = [G_{random,i}, G_{preferred,i}]$.

Node i opens a servent (server-client) connection to a node m (m is in the list $G_{preferred,i}$) where the word servent is a term denoting a peer-to-peer node, which is typically a server and a client at the same time as it accepts connections as well as initiates them. Then node m checks whether i is in its Γ_m list, and if this is the case, increments an internal counter c_m and compares it against a constant γ. If $c_m \geq \gamma$, then $c_m = c_m - \gamma$, a connection is created to node i, which we call a "backward connection", and the set of neighbors added as backward edges is updated, as follows: $G_{backward,m} = G_{backward_m} \bigcup \{i\}$. This backward connection creates an undirected edge between the two nodes i and m ($i \leftrightarrow m$)from the initial directed edge, as $i \rightarrow m$. In addition, γ ensures that a node does not add more connections than $d_{in,m}/\gamma$ where $d_{in,m}$ is the in-degree for node m, or the number of its incoming connections. When node i receives a backward connection from node m it will consider its choice of node m as a good one, and accordingly update its neighbors lists: $G_{preferred,i} = G_{preferred,i} - \{m\}$, and $G_{highly\_preferred,i} = G_{highly\_preferred,i} + \{m\}$. The final list of neighbors for node i is: $G_i = [G_{random,i}, G_{preferred,i}, G_{highly\_preferred,i}, G_{backward,i}]$.

A summary of this algorithm is presented in Figure 2.1, and an example of the creation of G_i is presented in Figure 2.2, for illustration purposes. In this particular scenario, the existing overlay network is shown in Figure 2.2 where the interconnections between nodes are shown with arrows, with the bold arrows representing connections that were created by preferential and backward formation. In the scenario, Node 8, wants to join the network and goes through the process shown in Figure 2.2. Node 8 starts by obtaining a list of hosts that are present in the network and then divides this list into two sub-lists where $G_{random} = [1, 3]$ and $G_{friends} = [5, 6]$. Then it contacts the nodes contained in $G_{friends}$ to obtain their lists of neighbors and constructs the following list $G_{candidates} = [7, 2, 4, 7]$. Sorting the nodes in descending order using their frequency of appearance yields $G_{preferred} = [7, 2]$. Then Node 8 constructs the final list $G = G_{preferred} \bigcup G_{random} = [7, 2, 1, 3]$ and connects to these nodes. Note, that as Node 8 starts its servent sessions with the resulting nodes in G then one or more of them might choose to create a backward connection to Node 8 depending on the values of their respective counters c.

2.3.3 Network Resiliency

According to the Webster Dictionary [57], the word resilience is defined as "an ability to recover from or adjust easily to misfortune or change." Networks with power-law degree distributions are often criticized in the literature for collapsing under targeted attacks. Under such conditions if a small fraction of the nodes with high degrees is removed from the network then the whole network suffers and

obtain G_{host} from web cache;

divide G_{host} into G_{random} and $G_{friends}$;

let s be the size of $G_{friends}$;

$G_{candidates} = \emptyset$;

for $(x = 0; x < s; x++)$

send M_0; where $M_0 = ping\langle i, G_{friends}[x], 1, 0\rangle$

$G_{candidates} = G_{candidates} \cup G_{G_{candidates}[x]}$;

$G_{preferred} = [g_1, g_2, ..., g_p] \subseteq$ (sorted)$(G_{candidates})$;

connect to all nodes in $G = G_{random} \cup G_{preferred}$;

if ((j connects back to i) && ($j \in G_{preferred}$))

$G_{preferred} = G_{preferred} - \{j\}$;

$G_{highly\_preferred} = G_{highly\_preferred} + \{j\}$;

Figure 2.1: Algorithm for *connect_to_network(i)*

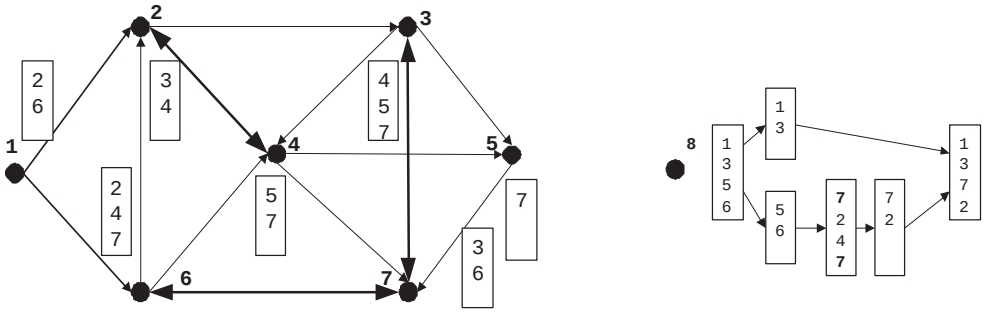

Figure 2.2: Example of Phenix Overlay Construction

often becomes disconnected into smaller partitioned fragments, also referred to as "islands" in the literature [9]. Phenix attempts to make connections resilient, protecting the well being of the entire network. We achieve this goal by following a set of guidelines that can be summarized, as follows. First, we attempt to hide the identity of highly connected nodes as much as possible, making the task of obtaining a comprehensive list that contains these nodes practically impossible. The second deterrent deals with neighbor updates, or what we call "node maintenance" (discussed below), where a network under attack can recover when existing nodes rearrange their connections and maintain connectivity. Note, that we assume that an attacker is powerful enough to force a node to drop out of the network, whether by denial of service attacks or by any other mechanism available, once an attacker acquires the IP address of such a node. In Phenix networks, resiliency implicitly means: the resilience of the whole network consisting of all "live" nodes where their connections form edges in a graph that is as close to a strongly connected graph as is possible, as we will show in Section 2.4.

Hiding Node Identities

In order to limit the likelihood of a malicious user obtaining a global view of the whole overlay graph (formed by the live nodes) of the network, Phenix supports three important mechanisms. First, a node receiving a ping message M_0 will respond with a pong message, and forward a ping message M_1 to its neighbors. All nodes receiving M_1 will add the originator to a list denoted by Γ_i. This list supports the notion of either "temporary blocking" or "black listing", where if the same originating node sends a ping message with the intent of "crawling" the

network to capture global or partial graph state information, such a message will be silently dropped with no answer/response sent back to the originating node. Black lists can be shared with higher layer protocols to isolate such malicious practices and can serve to isolate such nodes. A mechanism that detects a node crawling the network and silently discards queries will not stop a malicious user, but rather, slow its progress because the malicious node needs to obtain a new node ID (e.g., this would be similar to the Gnutella ID) to continue the crawl of the overlay, or wait for enough time for nodes to purge their black lists Γ_i. Peer-to-peer networks such as Guntella [37] have proposed including the MAC address as part of the node ID, making it even more difficult for an attacker to obtain a new and distinctly different node ID at a rate fast enough to continue the crawl. It is worth noting that if joins/leaves of an overlay network are dynamic enough then crawling at slower time scales will not yield an accurate view of the network state and topology. Even though such a scheme helps limit the impact that malicious nodes can have, it still does not fully eradicate potential attacks on the network. Next, Phenix also employs the policy of silently dropping any ping message, similar to the one shown in Equation (2.3), whose TTL value is greater than 1. A non-conforming node with malicious intent might generate such a message. Nodes drop these messages without responding to the originator or forwarding such a message to neighbors. This has the effect of eliminating crawling even if the originating node is not on the list Γ_i of the receiving node, in contrast to Gnutella where crawling is often practiced. Third, a node that establishes backward connections to other nodes in the network will not return these connections when it receives a ping in any of its pong reply messages. This policy is not meant

to protect the node's $G_{backward}$ sub-list of neighbors. Rather, it protects the identity of the node itself and any possible preferential status that the node may have, from an attacking node. If an attacker were to receive a long neighbors list from a node, it can infer that such a node is a highly connected node from the size of its neighbors' list. Thus, a node will only return the subset $G_{outside\_world}$ defined by Equation (2.4) in a pong message. In this case, this node does not need to forward M_1 to all of its neighbors. Rather, it only forwards M_1 to nodes in its $G_{outside\_world}$ subset since these are the nodes that might risk exposure to an attacker, where,

$$G_{outside\_world} = [G_{random}, G_{preferred}, G_{highly\_preferred}] \tag{2.4}$$

Node Maintenance Mechanism

In the event of an attack, the network needs to be responsive and able to rearrange connectivity in order to maintain strong connections between its nodes. In what follows, we propose a state probing mechanism that makes Phenix responsive to failed nodes or nodes that drop out of the overlay because of attacks. The number of neighbors of a node i, represented by h_i, is defined as the summation of the number of neighbors obtained through random, preferred and backward attachments; in other words, the out-degree of the node defined as the total number of outgoing connection for a node i. This total number is expressed as $h_i = h_i^r + h_i^p + h_i^b$, where $h_i^b = 0$, if $i \notin$ [preferred nodes]. h_i^r, h_i^p, and h_i^b represent the number of random, preferential (standard and highly), and backward neighbors, respectively. Nodes examine their neighbors' table in order to make sure that they are not disconnected from the network due to node departures, failures, or denial

of service attacks. If the following Inequality $h_i^r + h_i^p < threshold$ is satisfied, signaling a drop, then node i runs a node maintenance procedure, as described below.

If a node on the i's neighbors' list leaves the network gracefully, then it informs all the nodes connecting to it by closing the connections. However, if a node is forcefully removed or fails then node i will be informed of this fact only through probing where a message is sent to its neighbors, as follows: $M_2 = \langle source = i, type = ping, TTL = 0, hops = 0 \rangle$. In the case where no answer is received after a timeout (which is discussed in Section 3.6) then the neighboring node is declared down. The number of neighbors before node maintenance can be expressed as follows: $h_i^-(t_n) = h_i(t_{n-1}) - d_i^r(t_n) - d_i^p(t_n) - d_i^b(t_n)$, where, $h_i^-(t_n)$: current number of nodes (prior to the last maintenance run), and $d_i^r(t_n)$, $d_i^p(t_n)$, $d_i^b(t_n)$: the number of neighbors (random, preferential, and backward, respectively) lost since the last node maintenance. Following the node maintenance, we have:

$$h_i(t_n) = \begin{cases} h_i^-(t_n), & threshold < h_i^-(t_n) - h_i^b(t_n) \leq max \\ h_i^-(t_n) + u_i^p(t_n) + u_i^r(t_n), & otherwise \end{cases} \tag{2.5}$$

where, $h_i(t_n)$: the number of neighbors after the node maintenance and $u_i^p(t_n)$, $u_i^r(t_n)$: the number of new neighbors added preferentially and randomly, respectively. The ratio of preferential and random neighbors for a node i is presented in Equation (2.6).

$$\alpha_i(t_n) = \frac{h_i^r(t_n)}{h_i^p(t_n)}, \text{and} \frac{h_i^r(t_n)}{max - h_i^r(t_n)} \leq \alpha_i(t_n) \leq 1, \forall i, n \tag{2.6}$$

and the initial value of α is expressed by: $\alpha_i(t_0) = 1, \forall i$.

The update of neighbors is then performed according to Equations (3.4).

$$u_i^r(t_n) = d_i^r(t_n) \text{ and } u_i^p(t_n) = \begin{cases} \lceil \frac{\tau_i(t_n) - \mu^p}{\alpha_i(t_{n-1})} \rceil & , d_i^p(t_n) > 0 \\ 0 & , d_i^p(t_n) = 0 \end{cases} \tag{2.7}$$

where, $\tau_i(t_n) = \sum_{k=n-l+1} d_i^p(t_k)/l$. $\tau_i^r(t_n)$ is the average number of preferential neighbors that dropped out over the last l node maintenance cycles, measured at time t_n, mu^p is the expected value of the number of neighbors that disappeared in one node maintenance cycle. The symbol $\lceil\ \rceil$ rounds up the value to the next highest integer. Therefore, the final number of neighbors is:

$$h_i^p(t_n) = \begin{cases} h_i^p(t_0), & u_i^p(t_n) < h_i^p(t_0) - h_i^{-p}(t_n) \\ h_i^{-p}(t_n) + u_i^p(t_n), & u_i^p(t_n) < max - h_i^{-p}(t_n) - h_i^r(t_n) \\ max - h_i^r(t_n), & otherwise \end{cases} \tag{2.8}$$

For preferred nodes, we already have the following approximation: $h_i^b = \left\lceil \frac{n_i - \gamma}{\gamma} \right\rceil$, where n_i is the number of nodes pointing to node i. The preferred node updates its c_i counter, as follows: $c_i = c_i + (\gamma \times d_i^b(t_n))$, while no nodes are added in the backward set during the node maintenance process. Analysis of the effect of α on the network's behavior, particularly when faced with large-scale attacks is discussed in Section 2.4.

2.3.4 Preferential Nodes

We now show through analysis that Phenix encourages the emergence of nodes whose degree is higher than the average across the entire network, even if we initially start out with a completely random set of connections among nodes present

in the overlay network. In what follows, we analyze the emergence of nodes with a degree deviating from that of the average of the network. We call such nodes preferred nodes. Let us assume that we initially have a network of N nodes interconnected randomly. A new node i, running the Phenix algorithm wishes to connect to this network. So, i acquires a list of friends using a rendezvous or bootstrapping mechanism similar to the one used by many P2P systems. As described earlier, node i contacts these friends asking for their respective lists of neighbors. The summation of all answers constitutes the list of candidates. It follows that after node i acquires the list of $G_{candidates,i}$, the probability of connecting to a node on the list is directly proportional to the frequency of appearance of that node; that is to say, it is equal to the probability that a node will appear more than once in its list of candidates.

Let, μ be the average number of neighbors and N the number of nodes in the network. A new node i will connect to $\mu/2$ nodes randomly in $G_{random,i}$, since $\alpha_i(t_0) = 1, \forall i$, and will contact $\mu/2$ nodes requesting a list of their neighbors, which will become $G_{candidates,i}$. Thus, the resulting number of nodes on this latter list is an average of $\mu^2/2$.

Since we are interested in nodes appearing more than once on this list (which translates to a higher probability in initiating a connection to one of them), we calculate the probability of a node j appearing at least twice, which is expressed as the summation of the probabilities that j appears $2, 3, ...m$ times, where $m = \mu/2$. This upper bound of m comes from the fact that a node can appear at most once in each list returned by one node of the sub-list $G_{candidates,i}$. Thus the probability of a node appearing twice becomes the probability that it is on two of the lists of

nodes in $G_{candidates,i}$, and similarly, three appearances signifies the presence on three lists, and so on until m. The values of these probabilities are approximated by $(\mu/N)^2$, $(\mu/N)^3$, ..., $(\mu/N)^m$, respectively. Therefore, the probability that a node appears at least twice, encouraging a preferential attachment in a Phenix setup is given by the following equation:

$$P(X > 2) = \sum_{i=2}^{m} P(X = i) = \left(\frac{\mu}{N}\right)^2 + ... + \left(\frac{\mu}{N}\right)^m = \frac{1 - (\mu/N)^{m+1}}{1 - \mu/N} - 1 - \frac{\mu}{N} \tag{2.9}$$

since $\mu/N < 1$. Now that we know the value of the probability of a preferential attachment, we are interested in analyzing how fast such an attachment will take place (as the network grows) assuring the evolution of the network graph from a random network to one based on power-laws. Figure 2.3 plots the probability derived in Equation (2.9) versus the average number of neighbors for different values of N , the initial random network. We can observe that it is desirable for the initial network to be small so that preferential attachments start to form as early as possible; for example, given an initial Phenix network of 20 nodes, the probability of preferential attachment is around 0.117. This means that with the 9^{th} node joining the network, at least one preferential attachment is formed. It follows that after one preferential attachment forms, the probability of a second preferential attachment increases since the probability of this node appearing more than the others is already biased. Note that N is not the total number of nodes in the final overlay, but only the first initial nodes that come together in the network. Clearly, the overlay network can grow to encompass a much larger number of nodes, and at that time Equation (2.4) no longer holds because the connections

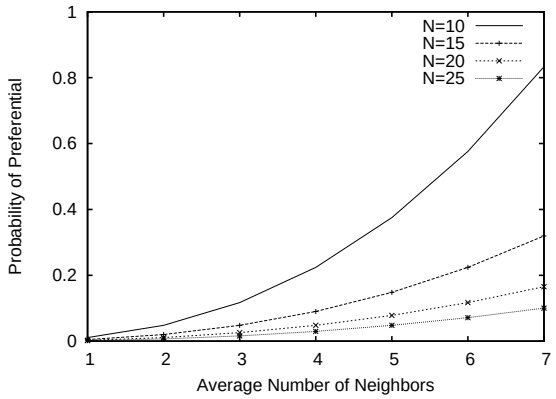

Figure 2.3: Probability that a Preferred Node Appears

among nodes is not random, but biased, forming a power-law, as we have just shown in this section.

2.4 Simulation

In what follows, we discuss the results obtained from implementing the Phenix algorithm in a simulation environment based on Java software. We start by examining the emergence of a power-law where nodes enjoy a low-diameter. We then study different types of attacks on an overlay network using the Phenix algorithm to measure the network's degree of resilience. Finally, we discuss the sensitivity of Phenix to different bootstrapping mechanisms.

2.4.1 Power-Law Analysis

Degree distributions following power-laws tend to appear in very large networks found in nature [9] [10]. However, we would like to have an algorithm where such

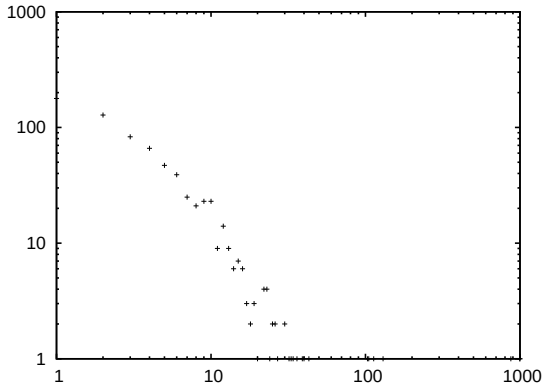

Figure 2.4: Degree Distribution for 1000 Nodes

a distribution will be present in networks of modest size. Such an algorithm might be useful in different situations for various applications where an assurance of a large number of nodes might not be feasible. We studied the effect of creating a network of pure joins in order to be guaranteed of the emergence of a power-law in such a simple scenario. The nodes join the network following a normal distribution at simulation intervals, by acquiring neighbors' connections based on the Phenix algorithm. Plotting the degree distribution for the resulting network of a 1000-node on a log-log scale shows a power-law emerging in Figure 2.4. This property is more clearly observed for a network of 100,000 nodes, as observed in Figure 2.5.

2.4.2 Attack Analysis

Next, we study more sophisticated networks where nodes join and leave the network using different scenarios. The attacks analyzed in this section are aggressive and to some extent extreme requiring additions of nodes to the network that prob-

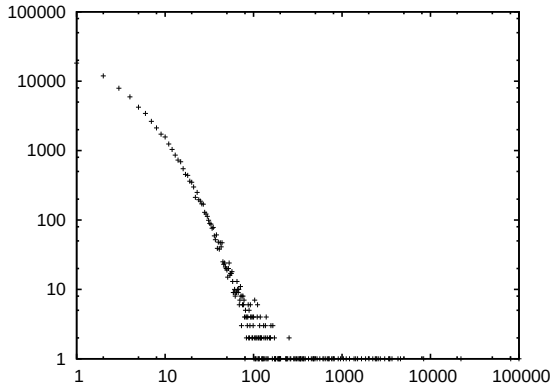

Figure 2.5: Degree Distribution for 100,000 Nodes

ably would not be typical of an attacker in a practical network. However, we chose to include such an analysis in order to test the limit at which the Phenix algorithm is capable of adapting, and the point beyond which the network does not serve its purpose anymore of interconnecting participants to each other.

We consider a number of attack scenarios where an attacker can perform one of three different types of distinct attacks on the network, or a combination of such attack scenarios. The first attack scenario consists of a user that acquires host cache information like a legitimate node might. The attacker contacts these acquired nodes with a M_0 message, getting the respective lists of their neighbors, and building his candidate's list, as a result. However, once the attacker has this information it will then attack the nodes appearing in this list more than once, removing them from the network. Such an attacker is limited in its capabilities and resources when compared to the two other scenarios discussed next, because the attacker attempts to target nodes that might have a node degree higher than the average without participating in the overall structure. However, such an attacker

has a level of sophistication because it is not removing nodes randomly. Rather, the attacker attempts to cause as much disruption as possible by maximizing the damage to the network in creating targeted attacks toward nodes that are important to the network performance, with as little investment as possible. The other two types of attacks are more organized from the attacker's perspective and require adding a large number of nodes to the network. Such an attack option is possible due to the fact that the network is open and welcomes any connection with no prior authentication or authorization. The first of these two additional attacks we denote as a "Group Type I" attack. This attack requires an attacker to add a number of nodes to the network that only point to each other, thus, increasing the probability that they will emerge as preferred nodes in the overlay network. The last type of attack, which we denote as a "Group Type II" attack, consists of adding a number of nodes to the network that would behave like normal nodes do. These last two types of attacks attempt to create anomalies in the network by introducing "false" nodes that remain connected for a prolonged period of time. Such a regime would ensure that other "true" nodes come to rely on these false malicious nodes due to the length of time that the false nodes are available in the network. Under such attack scenarios, these false nodes suddenly disconnect from the overlay network all at the same time with the intention of disconnecting and fragmenting the network into small islands of nodes. We also consider a hybrid attack scenario where the strategy dictates that some of the malicious nodes use the strategy of "Group Type I" and the others use "Group Type II" attacks.

The following simulation results are for an overlay network composed of 2000 nodes. Each node chooses a number of neighbors between 5 and 8, which repre-

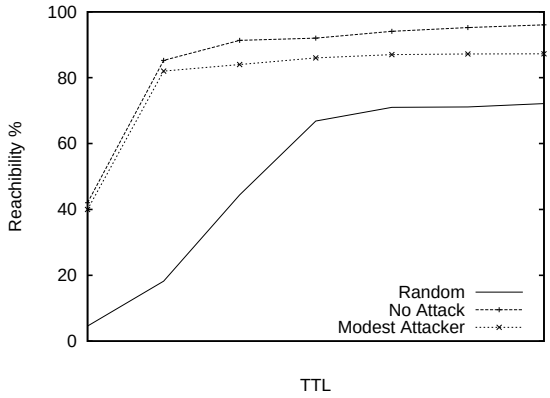

Figure 2.6: Modest Attacker

sents small numbers of nodes, if compared to Gnutella [37], denoted, respectively, by min and max, with equal probability while maintaining $\alpha_i(t_0) \leq 1, \forall i$, resulting in an average of $E(\alpha_i(t_0)) = 41/48$ for the whole network. However, this initial state for α will change as nodes join and, most importantly, leave the network, as we will discuss later. At each simulation time interval, the number of nodes joining the network is based on a normal distribution. For the case of nodes leaving the network, we consider three different cases: (i) the departure pattern is based on a normal distribution with a mean λ where nodes leaving are randomly selected from the overlay network. This scenario is equivalent to the case where the system faces no attacks, as shown in Figure 2.6; (ii) the departure pattern is based on a normal distribution, however, the nodes are removed by sending ping messages creating a sorted list of candidates, and removing preferred nodes from the network (this corresponds to the "modest attacker"); and (iii) represents group attacks as in the case of Group Type I, Group Type II, and hybrid of Group Type I/Group Type II attacks. In this case, a percentage of the nodes (note that different

values of this percentage are studied extensively later in this section) represent malicious nodes that conspire together to create the maximum possible damage to the whole structure of the network. The attack proceeds by having nodes at each interval leave the system as if there is no attack scenario until the malicious nodes suddenly drop out of the system, as described earlier. In each case of nodes leaving the system, we compare the performance of the network with a pure random network having the same average number of neighbors across all nodes, taking into consideration the min, max values, and backward connectivity from preferred nodes in a fashion similar to a topology created in the Gnutella network [37].

In all simulations, we start with a small number of nodes $n_{init} = 20$ that are interconnected randomly to each other with each node maintaining a number of neighbors $min \leq h_i \leq max$. The average rate of nodes arriving (i.e., issuing joins) is greater than the average departure rate, allowing the network to grow to the total number of nodes we would like to examine. In the case of Type I, Type II or hybrid group attacks, the process with which the network is formed starts by adding 50% of the legitimate or "true" nodes in incremental steps. At each step, the number of nodes added is drawn from a normal distribution, in a fashion similar to what would happen in a real P2P network. Following this, the malicious nodes are introduced in a single step giving them enough time to establish a strong presence in the network. We then add the next 50% of the legitimate nodes also in incremental steps. During all the steps, nodes continue to leave the network under a "no attack" situation. Eventually, we remove the malicious nodes, and study the effect on the remaining live nodes.

The metric measured for these networks consists of the percentage of unique reachable nodes in the network vs. the number of hops that we also denote by TTL. This measurement will give us an understanding of how many nodes can be reached when an application issues a query on top of the Phenix topology. Also note, that the same can be denoted as a radius because it starts with a node as the center and proceeds to try to cover as much of the network as possible. The figures represent this reachability metric in terms of the percentage of the total number of "live" nodes in the network. We compare the Phenix network under attack to a purely random network (as implemented by the Gnutella v0.6 [37]) because a random topology network is often cited to be the most tolerable to attacks [10]. Also, it is worth noting that the response of the network to various attacks is shown before the nodes run their node maintenance procedure (as described in Section 2.3.3) because the performance of a Phenix network will return back to the case of "no attacks" after a single neighbors maintenance is performed on each node.

Each experiment ran 10 times to ensure that the results stem from the structure and properties of the Phenix algorithm. We then sampled 10% of the nodes and measured the reachability of each of the sampled nodes and calculated the averages for each result. All measurements deviated only a little from the averages presented, proving that the behavior of the distributed algorithm is indeed predictable and reliable.

Figure 2.6 shows a comparison of the performance for the first type of targeted attack discussed above, which we denote on the plot as the "modest attacker", versus the "no attack" and random network. We can see that in response to the targeted node removals, the performance of the network degrades but the loss is

quite tolerable and still offers a gain over the random topology. Thus, in this scenario, Phenix has the potential of offering the participating nodes a more efficient overall performance where a node can be reached even with a smaller TTL value.

Figure 2.7 shows four different attacks: 30% of both Group Type I and Group Type II attacks, and two hybrid combinations each resulting in a total of 30% malicious nodes in the overlay. In studying such a comparison we were interested in seeing which strategy might be more damaging in fragmenting the network and disconnecting the live nodes. We observed that Group Type I attacks create a larger fragments in the network when introduced as a small percentage, than the same number of nodes running in the Group Type II attack mode. In addition, when we have a smaller percentage of Group Type I nodes backed up by more nodes as Group Type II, the performance of the network degrades the most as the maximum number of nodes reachable drops, as shown in Figure 2.7. This is due to the fact that nodes in Group Type I attacks, point to each other, which means that if we increase their number beyond a certain threshold the probability that they will be chosen by legitimate users as preferential drops. However, Figure 2.7 also shows us that across all attack scenarios, the network does not collapse into small islands. A promising result shows the giant component, indicated by the maximum reachability, not dropping below 70% of the remaining "live" nodes under all attack conditions.

Figures 2.8 and 2.9 show the effect of Group Type I and Group Type II attacks on a Phenix network where the percentage of malicious nodes shown is actually the percentage from the final network. This means that if we have 10% malicious nodes in a 2000-node network then the number of legitimate nodes is 1800. This

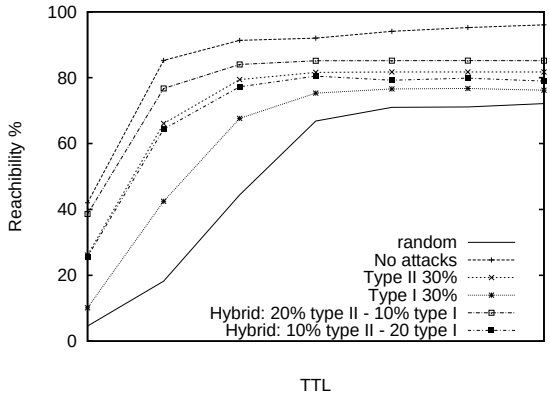

Figure 2.7: Comparison of Group Attacks

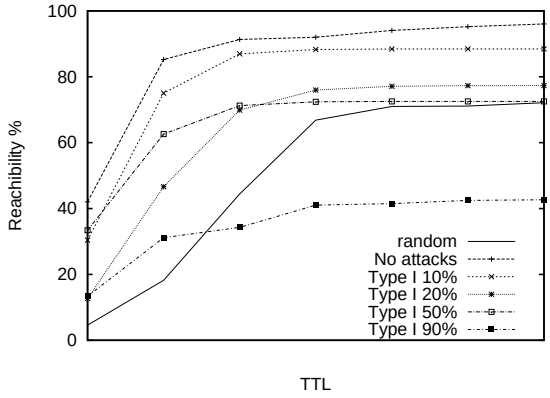

Figure 2.8: Type I Attacks

result implies that for an attacker to launch a 50% attack, he/she has to have the capability of introducing a number of malicious equal to the number of existing nodes in the network that he/she wishes to partition or harm.

In Figures 2.8 and 2.9, we can observe that a network under an attack of 50% malicious nodes scenario seems to provide a performance that is better than the 20% malicious nodes attack. This result seems counter-intuitive at first. However, it occurs because the number of nodes in the network becomes half the initial size,

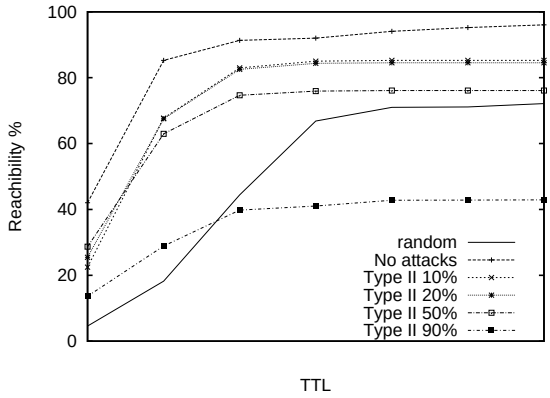

Figure 2.9: Type II Attacks

as the other half were malicious nodes that dropped out of the network, while the measured reachability is represented as a percentage of the total number of live nodes. Similarly, a network undergoing a 90% malicious node attack seems to reach a constant plateau with a lower TTL value than the initial network for the no attacks scenario, as shown in the figure. This is due to the fact that the structure of the network carries the signature of a power-law like distribution, offering a diameter in the order of $O(logN)$ where N is the total number of nodes participating in the network. As N drops to 10% of its initial size, the diameter follows by decreasing as well.

Measuring the giant component, which is the largest portion of the network that remains strongly connected, under different group attack scenarios is shown in Figure 2.10. If we consider, for example, the 20% attack for both Group Type I and Group Type II modes, we can observe that the giant component still amounts to around 80% of the total nodes of the network. At the same time, an 80% attack results in a giant component composed of 60% of the nodes. One can

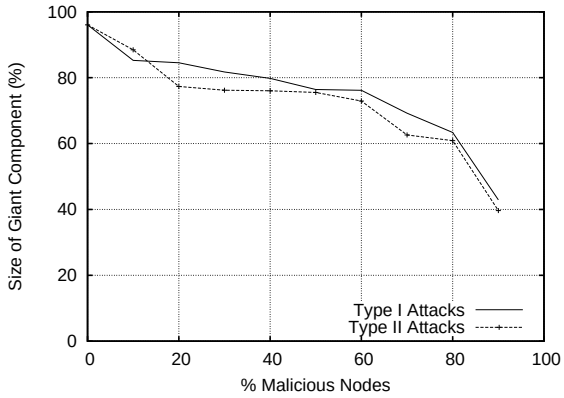

Figure 2.10: Giant Component

conclude that in order for a malicious attacker to divide a network of 400 nodes into half, then as many as 1600 nodes have to be introduced into the network for a considerable amount of time. This is a high price to pay to break such a network in two parts as the attacker is adding a number of nodes equal to 400% of the number of nodes in the initial targeted network. Add to this that the network recovers to a giant component in the order of 90% of the total number of nodes after performing one node maintenance interaction. This result looks very promising in terms of Phenix's ability to respond to such attacks.

We ran the same set of simulations where the total number of nodes is 20,000 instead of the 2,000 keeping all other parameters identical. In Figure 2.11, we present a summary for the hybrid attack discussed earlier. The behavior is very similar to that of the previous set of experiments showing that Phenix can provide a high degree of resiliency to the network independent of the total number of nodes in the network. Figure 2.11 also shows another signature of a power-law like distribution. A 20,000 node network reaches almost a stable plateau with a

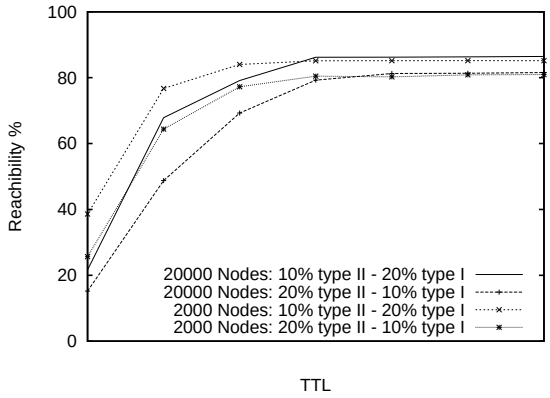

Figure 2.11: Hybrid Attacks in 2,000 and 20,000-node Networks

TTL larger by 1 hop than the 2000-node network, even though the total number of nodes is 10 times greater.

These plots indicate that increasing the TTL beyond a certain limit does not provide any significant benefit, as can be seen in Figure 2.7 and Figure 2.11. In fact, the number of reachable nodes seems to reach a maximum value beyond which increasing the TTL does not offer a wider variety of nodes reached. For example, it can be seen from Figure 2.8 that increasing the TTL from 4 to 5 in a 2000-node network with 10% malicious nodes of Group Type I will increase the reachability from 88.29% to 88.44%. This is a characteristic that can be exploited by applications where a query carrying a large TTL might have its hop decremented by more than 1 at a node receiving it because the gain of a larger TTL is not that significant. Such structure is beneficial in the sense that a reply can be returned to the originating node in a faster period of time because the number of hops is smaller than the random counterpart. An application sitting on top of such a topology might consider not to flood all of its neighbors limiting the generated

traffic. Rather, it can direct the search using a smart policy such as GIA [16], for example.

The α parameter introduced in Section 2.3.3 contributes to a fast recovery because most nodes will become quite aggressive in creating highly connected nodes after losing their preferred neighbors. This encourages the promotion of existing nodes to become highly connected nodes and assume the role of preferred nodes. We show the behavior of α in Figure 2.12. In this experiment, we use a hybrid attack of 10% Group Type I and 20% Group Type II. We can observe in Figure 2.12, that the initial value of the average of α across the entire network is close to 0.7 before introducing malicious nodes. However, when these nodes are added to the network (at time=60), they create a false sense of stability that can be seen in an increase and almost constant α despite the normal operation of the rest of the network where nodes are joining and leaving. Following the disappearance of the malicious nodes (at time=180), we observe a sudden drop in α across the entire network, as a sudden change is experienced by most legitimate live nodes. However, as the network goes back to normal operations, α starts to increase again, indicating that the network is in a stable state again. The choice of the α update influenced by Equations (3.4) ensures aggressiveness in decreasing it in order to respond as fast as possible to an attack, while the process of increasing it again is more conservative. We assumed any node can handle any traffic offered to it in the work presented, however, in practice this might not be the case and some nodes might refuse to have a higher in-degree than the average.

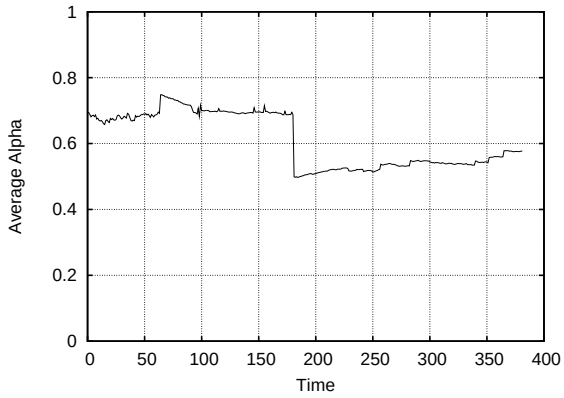

Figure 2.12: The Average of the Ratio of Preferred Nodes to Random Nodes Across all Nodes

2.4.3 Sensitivity to Bootstrapping Mechanisms

In this section, we test the sensitivity of the Phenix algorithm to the use of different bootstrapping mechanisms. We test mechanisms that are in use in existing peer-to-peer systems, and we compare them to the use of an ideal bootstrap server. We define an ideal bootstrap server as one that is able to return a list of nodes chosen randomly with equal probabilities from all the nodes present in the system, when contacted by a new node that needs to connect to the network. Note that in this section, we are not attempting to propose a scheme for a bootstrap server as it is beyond the scope of our research, however we are testing the dependence of Phenix on the different bootstrapping mechanisms.

We compare an ideal bootstrap server to a system where nodes on their first connection to the network obtain a list of existing nodes as in the case of the ideal bootstrap mechanism, however, we incorporate the idea of caching where a node i saves the addresses of its neighbors $G_i(t_0)$ that it acquired during a previ-

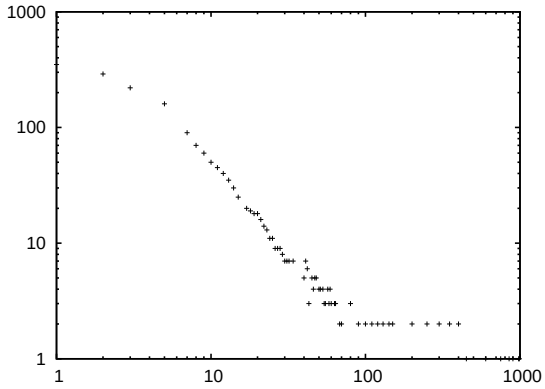

Figure 2.13: Degree Distribution While Using Caching

ous connection at time t_0. Node i favors connecting to the same set of neighbors at a later time t_n. This mechanism biases connections to be made to nodes that stay connected to the network for an extended period of time. By testing against this caching mechanism, we want to ensure that Phenix does not compromise its resilience in such a situation. We implement Phenix with 4,000 distinct nodes whose session lifetimes follow a distribution similar to observations of empirical data as reported by [75] and [77]. We measure the degree distribution of all nodes in the network, whenever the size of the network exceeds 2,000 nodes. The averaged results over 10 runs are shown in Figure 2.13. We can observe that the system still follows a power-law distribution preserving the desired characteristic of a low-diameter.

In order to measure the resilience of Phenix with such a bootstrapping mechanism, we repeat the experiment of Group Type I attacks, Group Type II attacks as well as Hybrid attacks. The results are shown in Figure 2.15. We can observe that such a bootstrapping mechanism does affect the performance but to a limited

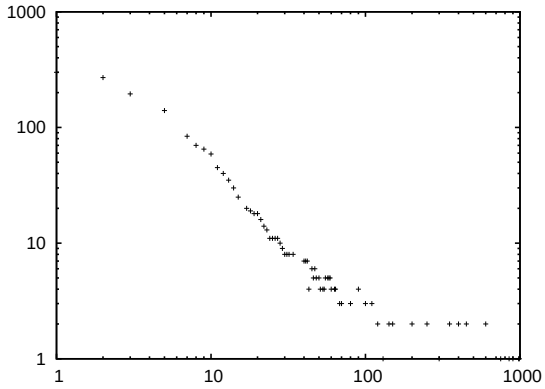

Figure 2.14: Degree Distribution With Partial Knowledge

extent in the sense that the reachability is lower than that for the case of an ideal random bootstrap server. However, these aggressive attacks did not succeed in dividing the network into separated islands. The reasoning behind this is that under the ideal random bootstrapping, nodes who emerged as preferred nodes were not necessarily the "oldest" in the system, since no caching is implemented. On the other hand, caching neighbors connections on client nodes changes the system by improving the chances of malicious nodes since they are staying in the system for a prolonged period of time and a returning node is more likely to connect to one of them than to a legitimate node. This adds to the effect of the simultaneous disappearance of malicious nodes helping them create a noticeable void in the overall presence of preferred nodes in the network, thus increasing the diameter. In addressing this void of preferred nodes, the remaining nodes are able to recover to a power-law distribution after one update of their list of neighbors, promoting existing nodes into a preferred status.

Another mechanism of bootstrapping that we test against is when the bootstrap

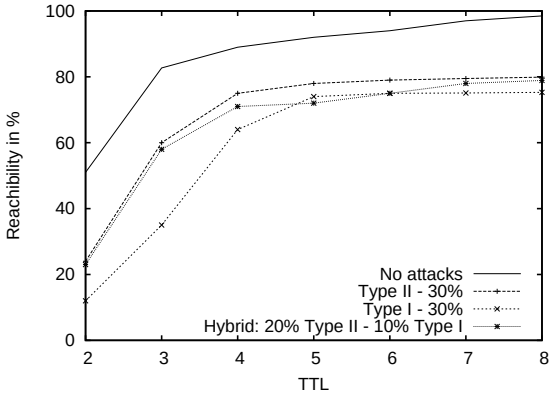

Figure 2.15: Group Attacks While Caching

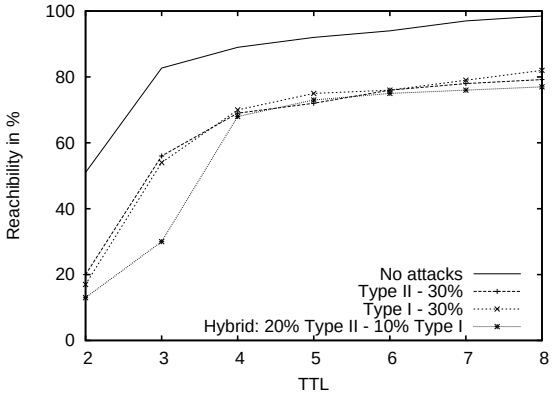

Figure 2.16: Group Attacks With Partial Knowledge

server does not know about all the nodes in the system, but instead has knowledge about a smaller subset that it chooses randomly from. The size of this subset is represented as a percentage of the total number of nodes that we denote by ρ. In such a scenario, the bootstrap server will still return a set of random nodes when contacted by new-coming nodes, however, this set is biased towards nodes that it knows about giving them a higher chance of being in control of which nodes become preferential. One might imagine that a bootstrap server should be able to know about a high percentage of nodes connected to the network since these nodes contact the bootstrap server before connecting to the network, allowing the server to add them to its list. However, this is often not the case due to the fact that nodes might use their cache from a previous session, as presented above, while they have a different IP DHCP-obtained; this will make the bootstrap server oblivious to their presence in the network. Another reason why a bootstrap server cannot obtain full knowledge is due to the use of distributed bootstrapping infrastructure on several servers, which typically do not exchange information among each other for scalability reasons; thus resulting in each bootstrap server having a partial view of the network.

We test Phenix using a network of 2000 nodes that operate with 5 distinct bootstrap servers. We assume that the initial subset of 20 nodes appearing in the network is known to all 5 of the bootstrap servers. However, any subsequent arriving node will pick a bootstrap server randomly with equal probabilities, and queries it for random nodes. At that instance, that specific server will add this new node to its list of known nodes. We observe that the degree distribution of this network is still powerlaw-like as seen in Figure 2.14. We test the reachabil-

ity of Phenix using such a mechanism under normal operation as well as under attacks for the same setup of 2000 nodes and 5 mutually independent bootstrap servers. The results are shown in Figure 2.16. Testing this algorithm against malicious attacks of Group Type I, Group Type II, and Hybrid shows that the network remains resilient under the first two cases of attacks, but seems to lose more under the Hybrid attack. Note that under the Hybrid attack the network does not get disconnected but instead its typical diameter increases deviating from a powerlaw behavior. The reason behind this is that with partial knowledge of nodes, malicious nodes constitute a set of preferred nodes and another set of nodes pointing to them. Thus, if we picture the network where the preferred nodes are in the center, the ones pointing directly to them constitute a circle around them. The Hybrid attack strategy puts malicious nodes in the center as well as a set of nodes around them. Thus, the topology becomes similar to a star topology. As the nodes in the center of the star and a big portion in the first layer disappear, as they are malicious, the network does not have sufficient connections to sustain the powerlaw distribution; consequently the diameter increases. Note that in our experiments, it took the nodes two rounds of the update mechanism to acquire a powerlaw distribution back, instead of the regular one round of updates that is sufficient under previous mechanisms and attacks scenarios.

Under such conditions, it seems necessary for the nodes to discover other nodes more aggressively instead of relying on the initial set. In order to alleviate this issue, we modify the Phenix algorithm by adding another mechanism that we call the discovery stage, which takes place during the initial connections stage. In the discovery stage, a node starts by connecting to one of the random nodes in

$G_{host,i}$, and sends a special ping message with $TTL = x$, where $x > 1$ and chosen randomly. Each node j receiving this special message will decrease the TTL by 1 and forward the message to only one of its neighbors also chosen randomly, as long as $TTL > 1$. If $TTL = 1$, then the receiving node j_x will send back a list of its neighbors (or a subset, if it is a preferred node) to the sender node i. This procedure introduces a more diverse sample that a node can use as a startup point to collect its final list of neighbors, while maintaining restricted crawling capabilities that a malicious node can abuse. In fact, this newly obtained list of neighbors from node j_x will be used by node i as G_i defined in Equation (2.2). Note that no matter how deep a node sends a ping message, it will stay in the "circle" of malicious nodes if it had already started with one of them forcing it to connect to the circle as its sole outbound connection to the rest of the network. However, the probability that a node will have all of its initial set of nodes belonging to the set of malicious nodes is quite low. Another mechanism to overcome such situations requires a new node to contact more than one bootstrap server adding to the diversity of its initial set. The results of simulating both of these mechanisms are presented in Figures 2.17 and 2.18. In the first technique, nodes send the initial discovery message with x chosen from the set $[2, 3, 4, 5]$ with equal probability. In the second technique, nodes contact two bootstrap servers chosen randomly from the set with equal probabilities. We can observe that the problem shown in Figure 2.14 is not replicated under these modifications.

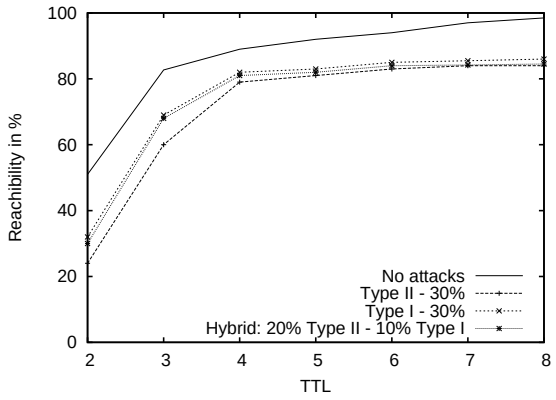

Figure 2.17: Group Attacks With Additional Discovery

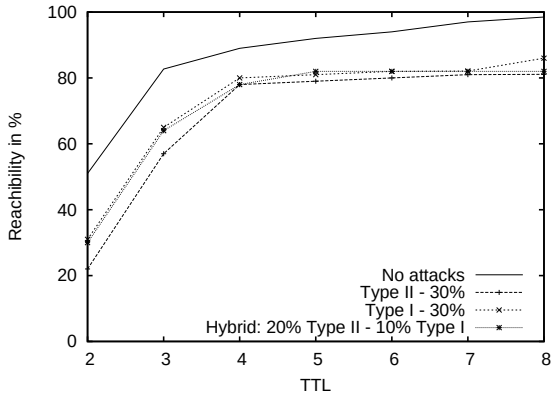

Figure 2.18: Group Attacks With Using 2 Bootstrap Servers

2.5 Experimental Testbed Results

We implemented Phenix in a real Internet-wide overlay environment running on the PlanetLab experimental testbed [66] for the purpose of measuring the overhead of the algorithm in the face of aggressive node removal scenarios. The code is built on the Open Source Jtella software system [44], a Java API for implementing the Gnutella protocol. We present our results from an implementation and experiment that ran on 81 PlanetLab nodes. We also measured the time needed for the network to recover from an attack targeted at highly connected nodes in the Phenix overlay running on PlanetLab.

2.5.1 Implementation

Each node in our implementation has two layers. The first layer being the Phenix algorithm composed of a servent (server and client) daemon responsible for incoming as well as outgoing connections. The node opens a socket connection waiting for incoming connections from other nodes either sending an M_0 (as described in Equation (2.2)), or nodes wishing to add this node to their neighbors' list. In terms of the graph, this connection receives and services all the incoming edges pointing to this node. The second type of connection constitutes all the connections that a node opens to other nodes, or the outgoing connections. As for the second layer, it is purely for experimental purposes, and opens a listening socket interacting with a central control server. The purpose of this latter layer is to be able to monitor the connections of a node in order to observe the progress of the network formation as well as the emerging topology. In addition, the control

server can send a stop signal to this layer asking it to remove the node from the overlay network; thus, emulating targeted node removal. The implementation is performed by modifying the JTella API which is a Java module based on Gnutella v0.6 [37]. The modifications are mainly in acquiring hosts and creating outgoing connections, making it conform to the Phenix algorithm, presented in Section 3.4, instead of the random Gnutella topology.

2.5.2 Degree Distributions Experiments

The Phenix overlay ran on the 81 PlanetLab nodes spread over 43 sites across 8 countries (Australia, Canada, Germany, Hong Kong, Sweden, Taiwan, UK, and US). The network started with $n_{init} = 10$ nodes interconnected randomly, in order to boot up the process of network formation. After that, nodes started joining at the rate of 2 nodes every 5 seconds by contacting the control server, which acts as a bootstrap server and provides the rendezvous mechanism by giving each node a list of 4 nodes that it can connect to. The generated list of nodes, given as a response for each request, is drawn randomly from nodes that have already joined the system with no bias given towards node location or proximity.

Thus, each starting node contacted the control server to get the initial G_{host} list, and applied the Phenix algorithm in making its decisions. In the following experiment, we chose the values of 3 and 4 for min and max (lower and upper bounds on the number of initial neighbors for a node, respectively), since the number of nodes (81 nodes) is a small number as compared to the growth of peer-to-peer systems in today's networks. Choosing higher values for min and max

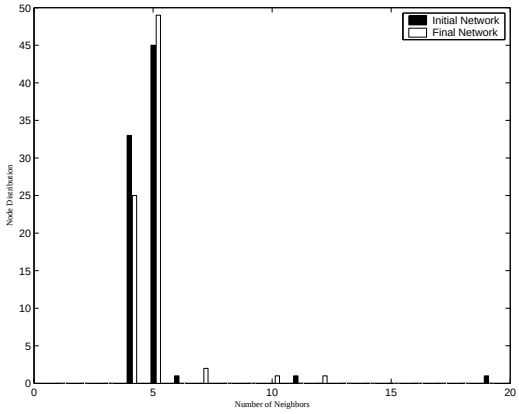

Figure 2.19: Out-Degree (number of neighbors) Distribution

would create a network that is closer to a mesh while lower values can easily result in situations where a node might find itself completely disconnected from the rest of the network with the removal of few nodes. Following the complete formation of the network and connections of all nodes, we took a snapshot of the resulting graph by examining the nodes' neighbors' list. Figure 2.19 presents the out-degree distribution (or number of formed outgoing connections) for the entire Phenix overlay network. The purpose behind this metric is to examine the number of nodes that emerged as preferential nodes and their respective degrees, as they acquired backward connections, thus, becoming hubs in the overlay network. We can see from the figure that the majority of nodes have between 3 and 4 neighbors, with the exception of 3 nodes with 5, 10, and 18 connections respectively. Before sending these 3 nodes the command to close their incoming and outgoing connections, we measured the rtt (round trip time) from the control server to every node in the network in order to see the diversity of the connections. Figure 2.20 shows the distribution of rtt for the overlay nodes. We can observe that although the

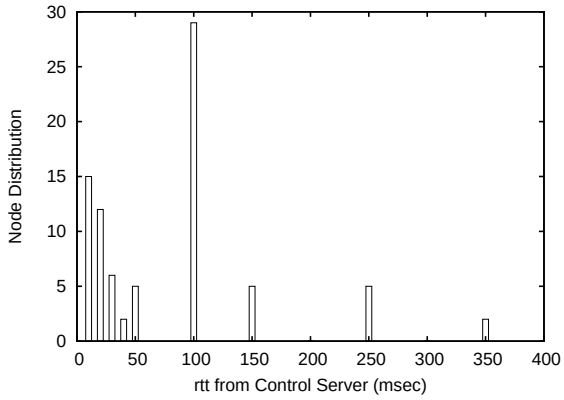

Figure 2.20: Round Trip Time (rtt) Distribution of Nodes in the Testbed

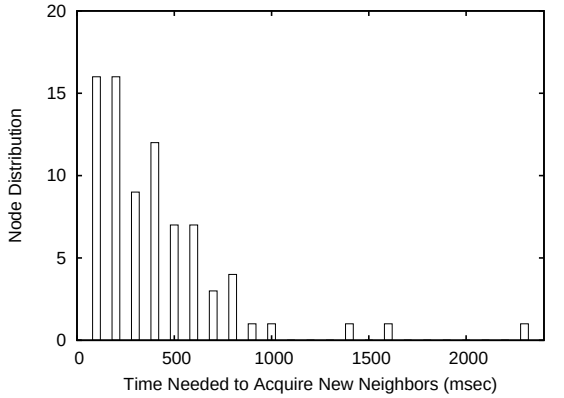

Figure 2.21: Node Maintenance Duration

majority of the nodes are within less than 100 msec reach from the control server, some offered a diversity in the network where their rtt reached higher values up to 350 msec, thus, offering a degree of heterogeneity for the experiment.

In this experiment we sent the 3 highly connected nodes (with 5, 10, and 18 connections) a stop signal through their control layer forcing them to close all of their connections. We then waited for the reaction of the rest of the nodes in the Phenix overlay, and measured how long it took them to rearrange their connections

and send their new state to the control server. Several factors enter into play when obtaining these results as presented by t_i: $t_i = rtt_j/2 + \zeta_i + rtt_i + \eta_i + rtt_i/2$. The total time needed for a node i to inform the control server that it performed the node maintenance, denoted by t_i, is the summation of five terms presented above. The first term is the time needed for the stop message to travel from the control server to the node to stop j, denoted by $rtt_j/2$. The second term is the time needed for the node i, in the case it is connected to node j, to realize that node j is no longer available (or the timeout of the connection, in this case we chose the value to be 1000 msec), denoted by ζ_i. The third term rtt_i is the time needed for node i to contact the control server requesting the address of one or more nodes that it can connect to, denoted by rtt_i. The fourth term, denoted by η_i, is the time needed to run the Phenix algorithm, which might involve contacting a friend node in the case of acquiring a preferential node. Finally, the fifth term $rtt_i/2$ is the time required to send the node maintenance outcome for the control server informing it of the change in the neighbors list. The distribution of time for each of the affected nodes to run this node maintenance mechanism is shown in Figure 2.21. We can observe that most nodes returned to a stable state by creating new connections in less than 1 second. Finally, Figure 2.19 shows a comparison of the resulting connectivity with the initial overlay graph, where we observe that 4 new highly connected nodes emerged ensuring the fast recovery of the Phenix overlay with a low-diameter topology.

2.6 Summary

We have presented a fully distributed algorithm called Phenix that creates low-diameter resilient peer-to-peer overlay networks. To the best of our knowledge Phenix represents one of the first contributions that simultaneously supports high performance in terms of low-diameter and fast response times, and is robust to attacks and resilient to various overlay dynamics and node failure scenarios. In this chapter, we have shown through analysis, simulation, and from results from an experimental implementation on the PlanetLab overlay that Phenix results in efficient connectivity, offering tolerance to various network dynamics including join/leaves and a wide variety of simple and more sophisticated node attacks. Because of the rise in number of security attacks and the growing creativity of attackers, the need for resilient overlays that can offer both performance and resilient properties will become necessary particularly for commercial reliable overlays. Phenix supports low diameter performance and resilience without sacrificing flexibility.

Chapter 3

Strategies and Algorithms for Parallel Downloads in Peer-to-Peer Networks

3.1 Introduction

Nodes joining peer-to-peer networks can benefit from initiating simultaneous requests for different parts of an object to different serving nodes carrying this object, or what is referred to as parallel downloads. The direct benefit of such requests is the increased total download bandwidth for the client nodes. Parallel downloads also offer increased resilience to the client node in the case where one or more of the serving nodes suddenly depart the network or fail. Serving nodes also benefit from parallel downloads because they do not have to serve a file in its entirety, sharing the responsibility with other serving nodes carrying the same

file. Dividing an object and downloading it in parallel is not as simple as it may seem since the client node wants to maximize its download bandwidth, maintains a low overhead of signaling messages, and be responsive to system dynamics such as sudden fluctuations in bandwidth offered by serving nodes or due to departing serving nodes. Existing implementations of peer-to-peer applications that employ parallel downloads do so in a naive fashion by either dividing the file into equal chunks and requesting each chunk from a different serving node, as is the case with Overnet [61] and eMule [28], or sending requests for small chunks frequently to serving nodes, as is the case with the implementation of Kazaa [78] and Limewire [53]. As a result many of these existing systems send a large number of signaling messages (e.g., object chunk requests), wasting a substantial amount of the available bandwidth that a downloading node could have taken advantage of.

We conjecture that in order to maximize the download performance of client nodes in a parallel download system, a more sophisticated and adaptive approach is needed; one that takes into consideration the competitive nature of nodes in the system and the network dynamics experienced by nodes in a real network implementation. To address this challenge, we propose a parallel download model for peer-to-peer networks based on game theoretic techniques that reflects the selfish, competitive and non-cooperative nature of peer nodes in the system. We model these nodes and solve the problem of dividing an object into chunks while maximizing download speed and minimizing signaling messages. We show that, as a result of this selfish behavior, the network can lack a Nash equilibrium. This lack of equilibrium basically translates into a situation where client nodes continue to include and omit certain serving nodes, which is detrimental to download speeds

and signaling costs. To counter this, we design a set of simple client and server strategies that minimize the effect of selfish nodes, lowering the risk of driving the network into oscillations due to the lack of Nash equilibrium in the system.

Because nodes do not have complete state information about peers, and instead rely on local observations, the optimal solution of object division cannot be realistically achieved in practice. Thus, we propose an estimation and prediction algorithm called the *Minimum-Signaling Maximum-Throughput (MSMT)* algorithm that is based on the Bayesian Theorem [42]. The purpose of the MSMT algorithm is to increase the observed throughput of the client node without adding an prohibitive amount of signaling messages into the network. We discuss two variants of the base MSMT algorithms called the *Simple and General MSMT* algorithms where the latter one is more responsive to different bottlenecks observed in the systems (e.g., at the client, network, and server). MSMT is a fully distributed algorithm that bases its decision-making on local state information only. We show the behavior of our proposed system and compare it to existing peer-to-peer systems (e.g., Limewire and eMule) using a combination of analysis, simulation, and experimentation from an implementation on a medium scale (102 node) on the Planetlab [66] overlay. Our results show that, with our proposed strategies and MSMT algorithm, nodes can achieve faster download speeds while incurring a lower number of signaling messages irrespective of changes in the file size, network traffic, and number of requests. We show the effect of different choices for important systems parameters on performance, such as, strategies for re-running queries during on-going downloads in order to discover new serving nodes, and changing the sizes of serving and wait queues on the serving node, as well as

network load and conditions.

The contributions of this chapter are:

- to model parallel downloads in peer-to-peer networks, using game theory techniques, by considering nodes as non-cooperative participants competing for the same resources;
- to investigate the existence of a Nash equilibrium in the system based on our model under different scenarios in the network, in order to better understand the effects of parallel download and to achieve more stable performance from the participating nodes' perspective;
- and finally to propose a smart adaptive algorithm that provides nodes with a near optimal performance for parallel downloads of objects.

In addition, our model assumes the following characteristics:

- Fully Distributed: we argue that in realistic systems, there is no central authority that can police the system. Thus, each node, whether server or client, has to deal with many parameters and uncertainty and come up with the best solution it can independently.
- Local Information: we assume that each node relies solely on local information based on what it is observing in terms of behavior from the other nodes interacting with it, and will not exchange any information with other nodes as such an exchange might provide an incentive for nodes to lie and cheat.

- Minimum Signaling: we propose an algorithm that has a main objective of maximizing speed without adding prohibitive cost in signaling. Thus, the proposed algorithm has "smart" components.

- Selfish Nodes: we assume each node is trying to maximize its utility which might lead it into cheating, if the need arises or if a gain can be accomplished as a result. Thus, our model assumes no cooperation among nodes and the proposed algorithm and strategies are designed in order to provide users with an environment where cheating would deteriorate their performance.

The structure of the chapter is as follows. We discuss the related work in Section 3.2. In Section 3.3, we describe our parallel download model for peer-to-peer networks and the necessary client and server strategies. We present a detailed description of the Simple and General MSMT algorithms in Section 3.4. Following this, we discuss our simulation results and the evaluation of the system deployed in a medium-scale Planetlab overlay, in Section 3.5 and Section 3.6, respectively. Section 3.7 presents a summary of the work.

3.2 Related Work

There is a growing body of work on parallel downloads found in the literature and deployed on the Internet. A number of popular peer-to-peer applications such as eMule [28], Kazaa [78], Limewire [53], and Overnet [61] use parallel download techniques, however, these applications either divide a file into equal sized chunks and request these chunks from different nodes, as is the case of Overnet

and eMule, or send requests for small chunks frequently to serving nodes as in the implementation of Kazaa and Limewire. Note that a detailed description of how Kazaa and Overnet work in practice is not publicly available and our observations on how objects are divided into chunks are based on our extensive monitoring of the behavior of these applications. However, we compare our proposed algorithms and strategies to eMule and Limewire in Section 3.6. Note that our model assumes a Gnutella-like protocol, thus we do not compare to BitTorrent [13] which relies on users forming groups and cooperating while downloading mutually exclusive chunks, only to exchange them later. Clearly, these existing Internet parallel downloads applications can have an adverse effect on the overall throughput of the network as a whole. This is because the client nodes are selfish and have the ultimate goal of increasing their own utility which is achieved by taking full advantage of all offered resources in the system. However, parallel downloads seem to be here to stay and therefore there is a need to develop new application protocols, control algorithms, and client/server strategies that can mitigate the adverse effects of parallel downloads. Minimizing the cost of parallel downloads on the network as a whole while maximizing the throughput achieved by clients constitutes the goal of our work presented in this chapter.

Interest in parallel downloads of online files has been an integral part of Content Distribution Networks (CDNs)[34], [48], [73]. However, the CDN environment is quite different from the peer-to-peer particularly when you consider the rate of arrival and departure of nodes in the system. In CDNs nodes are quite stable and remain online for extended periods of time, which contrasts with peer-to-peer networks where nodes are unpredictable and volatile.

On the other hand, in [12] the authors suggest the use of machine learning techniques to help a peer pick a serving node among the ones carrying its desired object, instead of aggregating all the bandwidth and taking advantage of all serving nodes. The focus of that paper is to find the most reliable node to download from in terms of its offered bandwidth and time spent in the network and does not take advantage of the aggregated bandwidth.

In addition, [15] and [32] study the benefits of using cooperative nodes in order to increase the storage capacity of the whole system, which is mainly targeted towards applications where nodes are cooperative. Research in [23], [51] and [52] show the benefit of using error correction codes in obtaining different chunks of an object from different serving nodes without targeting the actual division of such downloads. In fact, error correction codes can be used in order to compliment our study and offer better resilience in our model.

Finally, [14], [50], [71] and [88] discuss OceanStore, an infrastructure for sharing and serving resources. Even though these papers provide a lot of insight into such an infrastructure and assume the common use of parallel downloads among participating nodes for performance and redundancy, there is no mention of the actual divisions of these downloads and decision making involved.

Other researchers [67] [93] has investigated the performance of peer-to-peer systems but not for the case of parallel download scenario from a node's perspective. There has been little or no work on the analysis of parallel downloads for peer-to-peer networks. The closest work that relates to our study is [4]. In [4] the authors study parallel downloads determining the optimal peers to download from, minimizing the cost associated with the download, assuming guaranteed

bandwidth between clients and servers, and a cost for downloads directly proportional to the transfer from every serving node. However, we argue that in realistic peer-to-peer systems, nodes do not offer any guarantee in performance. In addition, client nodes typically pay a flat fee for their unlimited use of bandwidth and not usage-based fees.

3.3 Parallel Downloads Model and Client/Server Strategies

We first formulate a model for parallel downloads in peer-to-peer networks, and then present a set of recommended strategies to be implemented on the requesting (i.e., client) and serving nodes.

3.3.1 Parallel Downloads Model

The system contains a set of nodes, denoted by $\mathcal{A}$. Each node in $\mathcal{A}$ can initiate queries and if it carries objects can act as a server at the same time. So, our whole system can be expressed as follows:

$$\mathcal{A} = \mathcal{I} \bigcup \mathcal{N} \bigcup \mathcal{V} \tag{3.1}$$

where, $\mathcal{I}$, $\mathcal{N}$, and $\mathcal{V}$ are the subset of nodes initiating a request, serving an object, and idle, respectively. We also might have, $\mathcal{I} \bigcap \mathcal{N} \neq \emptyset$, accounting for the fact that a node i might be downloading an object and, at the same time, acting as a server where other nodes are downloading from it.

We denote the bandwidth that each node is using for its peer-to-peer application by [$\mathcal{B}_u$,$\mathcal{B}_d$], the upload and download bandwidth, respectively. We assume, initially, that congestion affecting $\mathcal{B}_u$ and $\mathcal{B}_d$ only happens on the last-mile of the node connection. This assumption helps us in formulating the model from an end-to-end perspective without taking into account the exact topology of the underlying network. However, we relax this assumption in Section 3.4.2 and propose the General MSMT algorithm that accounts for any change in the network's throughput.

A node i in the system sends a query for an object of size O_i. It hears back from a set of nodes $\mathcal{N}_i$, where $\mathcal{N}_i \subset \mathcal{N}$. It then initiates the game that is going to shape its strategy in dividing the object into chunks that can be downloaded in parallel from these nodes. At first, node i starts by downloading a set of small chunks which we denote by $O_{i,j}[0]$, where $j \in \mathcal{N}_i$ at time $t[0]$. Because the node is not aware of the usage on $\mathcal{B}_{u,j}[0], \forall j \in \mathcal{N}_i$, it will download equal small chunks of the file from all nodes. Thus, at time $t[0]$, we have $O_{i,j}[0] = O_{i,k}[0] \forall j,k \in \mathcal{N}_i$.

Now node i has a rough estimate of what to expect in terms of the available bandwidth from each serving node because it measures the download times from each serving node and computes a set of values that we denote by $\mathcal{B}^*[0]$, the set of observed bandwidth at time $t[0]$. The game becomes for node i to further divide the remainder of the object into chunks among the nodes and download these chunks. In general, we have $O_{i,j}[n]$ representing the chunk that node i is downloading from node j at time $t[n]$. The system is subject to the following

constraints:

$$\begin{aligned} &\textstyle\sum_{j\in\mathcal{N}_i}\sum_{n=0}^{T} O_{i,j}[n] = O_i, && \forall i \in \mathcal{I} \\ &\textstyle\sum_{j\in\mathcal{N}_i} B_{u,ji}[n] \leq B_{d,i}, && \forall i \in \mathcal{I} \\ &B_{u,ji}[n] \leq \mathcal{B}_{u,j}, && \forall j \in \mathcal{N}_i \end{aligned} \tag{3.2}$$

where T is the total time it takes to download the object. The intuition behind this set of constraints (3.2) is, respectively, all the downloaded chunks should add up to the object, the summation of all observed upload bandwidth from the nodes serving the file cannot exceed the download bandwidth of the node receiving the file, and finally, the requested upload bandwidth on any given node cannot exceed the bandwidth set aside to serve uploads in general.

3.3.2 Client Strategy

In this section, we define the utility of the client node and how it affects its behavior in dividing an object into chunks for parallel downloads from the serving nodes.

Dividing the Requests

When a node i obtains $\mathcal{N}_i$, it can proceed to send the requests for downloads of the chunks. At that point, i has to make a decision on how to divide these chunks and whether to use all the nodes in $\mathcal{N}_i$ or a subset. A whole spectrum of solutions exists. On one extreme, the node i can decide to have the smallest possible granularity in dividing the files, at the cost of generating a lot of signaling messages to the set of serving node in $\mathcal{N}_i$. On the other extreme, node i can decide to sacrifice efficiency by making one decision at first, generating one set

of requests and waiting for the downloads; this, of course, will not guarantee an optimal solution as far as speed is considered, as we will show next.

For the latter case, the number of messages that node i generates is equal to the number of nodes in $\mathcal{N}_i$. However, for the former case, the node divides the file into the smallest possible chunks and sends one request per chunk in a round-robin fashion to all the nodes in $\mathcal{N}_i$. This method generates s signaling messages where s is defined as:

$$s = \left\lceil \frac{O_i}{\xi} \right\rceil \tag{3.3}$$

where ξ is the minimum possible chunk size. s increases with the increase of file size O_i.

The problem consists now of finding the solution where the division is the closest to the solution with one set of requests, generating the least amount of signaling, while i has to rely on the knowledge provided by $\overline{B_{u,ji}}[n], \forall j \in \mathcal{N}_i$, (the average observed throughput) which changes with time.

Simple Game Setup

Under all the assumptions stated above, parallel download can be defined as a game where client nodes are competing to download their required objects. The game has the following characteristics:

- Non-cooperative: each node is acting in a selfish manner.
- Repetitive: each object is divided into chunks to achieve the highest throughput possible and the node can observe the download times of these chunks to adapt its strategy for the next set of chunks.

- With varying opponents: some nodes finish downloading their objects while others join at a later stage.

- With incomplete information: each node knows only its own action and can only see the outcome but has no explicit knowledge of the actions of the other active nodes in the network.

Analyzing the utility of each node, we know that a specific node has the sole objective of downloading its object as fast as possible, while minimizing signaling as it entails overhead that punishes download speeds. Thus, we define the utility u_i of node i to be:

$$\begin{aligned} u_i &= \alpha \max_j t_{ij} + \beta \sum_j s_{ij} \\ &= \alpha \max_j \left\{ \frac{O_{i,j}}{B_{u,ij}} \right\} + \beta \sum_j s_{ij}, j \in \mathcal{N}_i \end{aligned} \tag{3.4}$$

where α and β are normalizing factors, that represent how much a node i values fast downloads and minimal signaling, respectively. Node i wants to minimize (3.4).

Theorem 3.3.1. *The minimum for the first term of Eq (3.4) for a node is achieved when all download times are equal during any interval* $]t[n], t[n+1][$.

Proof. We have:

$$t_{i,j}[n] = \frac{O_{j,i}[n]}{B_{u,ji}[n]} \tag{3.5}$$

Let t_i be the solution where all the download times are equal.

$$t_i = t_{i,j_1} = t_{i,j_2}, \forall j_1, j_2 \in N_i \tag{3.6}$$

We want to prove that t_i is the optimal solution.

Let us assume that we have a better solution t_i^* so that $t_i^* < t_i$.

However, by definition and from Eq (6), t_i^* is a maximum, and since not all nodes finished at the same time then we have at least one node m finishing before t_i^*. So, we have:

$$\frac{O_{i,m}}{B_{u,mi}} < t_i^* \Rightarrow \frac{O_{i,m}}{B_{u,mi}} < \frac{O_i^*}{B_{u,i}^*}$$

Thus, if we take a small part of O_i^* that we denote by ϵ and download it using the node m, such that $O_{i,m} + \epsilon \leq O_i^* - \epsilon$ and since the respective throughput of the nodes did not change, we get the following $\frac{O_i^* - \epsilon}{B_{u,i}^*} < \frac{O_i^*}{B_{u,i}^*}$, or, in other words, we found a better solution than t^*, which contradicts our initial statement that it is optimal. □

After node i makes its initial test downloads at time $t[0]$, it can infer an expected value of the bandwidth that we denote by $\langle B_{u,ji} \rangle, \forall j \in N_i$, then using Eq (3.4) and Theorem 1, it can decide on $O_{i,j}[n]$ as well as whether it is going to use all of the serving nodes in $\mathcal{N}_i$.

Repetitive Game

Since the game is repetitive, node i can benefit from observing the outcomes of each step. However, the space of players is varying with time, where some nodes are no longer part of the game, as they finish downloading their objects, while other players might be introduced by initiating new queries. In fact, the whole space of players is changing as can be seen in Figure 3.1, where I_x is the subset of $\mathcal{A}$ at time t_x of nodes whose download activities has a direct effect on a certain

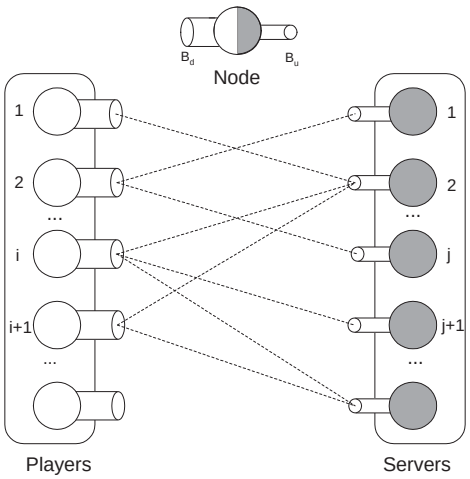

Figure 3.1: The System Setup

node i, as they are competing for the upload bandwidth of, at least, one common node j, where $j \in \mathcal{N}_i$.

Using the same argument, we can also deduce that the space of $\mathcal{N}_i$ might vary with time if the node initiates the query again and discovers other nodes carrying the object. This might be desirable in the case where the size of the object is above a certain threshold ($O_i > \gamma$) making the discovery of additional nodes beneficial. Also, some nodes might disappear from the network in the middle of the download, as they decide to leave the network. In this case, their available upload bandwidth $B_{u,ji}[n], \forall n$, from then on, will be considered equal to 0.

Note, that if a node is downloading a set of objects at the same time, it can start with the first one, check the bandwidth that this object is occupying and then use the rest of its available bandwidth to initiate the second game, and so on and

so forth. Or, as an another strategy, the node might decide to divide the bandwidth using some criterion among the different objects (equally, or proportionally to their respective sizes, for example), thus having several games in parallel for each of the objects. We do not tackle this problem, as, often, the user has his/her own priorities that are subjective and content-dependent.

Varying Bandwidth

Since bandwidth offered by serving nodes quite often varies with time, then downloads from the nodes won't proceed as expected. In fact, when a chunk $O_{ij}[n]$ finishes before the rest of the chunks, node i will have the incentive to redistribute the remainders of the rest of the chunks among all the nodes, to take advantage of all available resources. Thus, Eq (3.4) becomes:

$$u_i = \alpha \sum_{n=1}^{T} t_i[n] + \beta \sum_{j} s_{ij}, j \in \mathcal{N}_i \tag{3.7}$$

since i re-issues the requests after a chunk $O_{ij}[n]$ finishes. In this case, we have $t_i[n] = min_j t_{ij}[n]$, and $t_{ij}[n] = O_{ij}[n]/\overline{B_{u,ij}}[n]$.

Thus, in a realistic peer-to-peer network, a client node i, typically, has no knowledge of the change in available bandwidth offered by serving nodes. Thus, the problem reduces to estimating the expected value of the bandwidth that node i is going to experience when dealing with the nodes in the set $\mathcal{N}_i$. And, the "optimal solution" for object division into chunks should satisfy the following equation:

$$\frac{O_{i,j_1}}{O_{i,j_2}} = \frac{\mathbf{E}[B_{u,j_1 i}[n]]}{\mathbf{E}[B_{u,j_2 i}[n]]}, \forall j_1, j_2 \in \mathcal{N}_i \tag{3.8}$$

In Section 3.4, we detail the MSMT algorithm which attempts to provide an estimate of the expected download bandwidth to client nodes, under different network conditions.

3.3.3 Nash Equilibrium

We study now whether users in the system reach a Nash equilibrium. In fact, the existence of Nash equilibrium is dependent on the values of α and β, the two factors that determines how much a node i values speed and avoids signaling messages, respectively. These two factors play a major role in the decision a node makes at every $t_{ij}[n]$ on how to divide the chunks and to determine which nodes it will use as serving nodes.

Theorem 3.3.2. *The existence of a Nash equilibrium depends on the stability of $B_{u,ij}$ and on the choice of α and β.*

Proof. A node i has a utility defined by Eq (3.7), and it needs to minimize both parts of it. In addition, we have, from Theorem 1:

$$O_{ij}[n] = \frac{\overline{B_{u,ij}[n-1]}}{\sum_j \overline{B_{uij}[n-1]}} \tag{3.9}$$

$\overline{B_{u,ij}[n-1]}$ represent the average upload bandwidth observed by node i from all serving nodes in $\mathcal{N}_i$ at $t[n-1]$.

$$t_{ij}[n] = \frac{\overline{B_{u,ij}[n-1]}}{\overline{B_{u,ij}[n]}} \cdot \frac{1}{\sum_j \overline{B_{uij}[n-1]}} \tag{3.10}$$

Let $g_i \subset \mathcal{N}_i, g_i \neq \phi$ where,

$$\frac{|\overline{B_{u,ij}} - \langle B_{u,ij} \rangle|}{\overline{B_{u,ij}}} \gg 0, \forall j \in g_i$$

the subset of nodes that offered effective bandwidth substantially different than the expected value.

$$u_i^{-g_i} = \alpha \sum_{n=1}^{T^{-g_i}} t_i[n] + \beta \sum_j s_{ij}, j \in \mathcal{N}_i^{-g_i} \tag{3.11}$$

where $u_i^{-g_i}$ is the utility of node i while omitting g_i. By definition, all nodes in $\mathcal{N}_i^{-g_i}$ offer stable bandwidth, which can be expressed as:

$$\frac{\overline{B_{u,ij}[n-1]}}{\overline{B_{u,ij}[n]}} \approx 1$$

Eq (3.11) becomes:

$$u_i^{-g_i} = \alpha \sum_{n=1}^{T^{-g_i}} \frac{1}{\sum_j \overline{B_{uij}[n-1]}} + \beta \sum_j s_{ij}, j \in \mathcal{N}_i^{-g_i} \tag{3.12}$$

We already know from Theorem 1 that Eq (3.12) is the optimal solution for minimizing the second term. Thus, i will decide on $\mathcal{N}_i$ when,

$$\alpha \sum_{n=1}^{T^{-g_i}} t_i[n] + \beta \sum_j s_{ij} > \alpha \sum_{n=1}^{T} t_i'[n] + \beta \sum_j s_{ij}' \Rightarrow$$
$$\frac{\alpha}{\beta} > \frac{\sum_j s_{ij}' - \sum_j s_{ij}}{\sum_{n=1}^{T^{-g_i}} t_i[n] - \sum_{n=1}^{T} t_i'[n]}$$

and will omit g_i otherwise. The system will oscillate, lacking a Nash equilibrium, if for the same game setup, a node i has α/β chosen in a way that it continues to switch between omitting and including g_i. Such behavior will further deteriorate the system, as it is a typical tragedy of the commons phenomenon [41]. □

Intuitively, Theorem 2 states that if $\mathcal{N}_i$ contains serving nodes with stable upload bandwidth, then node i might be better off using just these nodes if it values

sending a low number of signaling messages. However, if α and β are chosen such that when the bandwidth offered by nodes in g_i is considerably significant, then node i will be tempted to include these nodes. When this happens, the bandwidth offered by g_i will decrease, especially if other client nodes in the system are accessing these nodes for the same reasons, causing node i to send more signaling messages. Then node i will start oscillating between including and omitting nodes in g_i and there is no Nash equilibrium.

A simple example of when such a situation tends to happen in reality is when there is a serving node j in the system that carries a multitude of files whose sizes follow a distribution with a large variance. In this case, client nodes downloading small objects will create an oscillation in the consumed bandwidth that will interfere with other clients downloading considerably larger objects. Another situation is when the serving node j carries a large number of objects making it a more likely serving node for a large number of client nodes. In addition, the situation will deteriorate even further if the objects are popular, simply because the demand for these objects is high.

3.3.4 Server Strategy

We now look at the perspective of the serving node and define its utility. For a serving node j the utility is of the form

$$u_j = \theta \max_i t_{ij} + \iota \sum_i s_{ij} \tag{3.13}$$

where θ and ι are normalizing factors. Also, we already know that t_{ij} is computed as follows:

$$t_{ij} = \frac{\sum_j O_{ij}(t)}{\overline{B_{u,ji}}} \tag{3.14}$$

Node j has the objective of minimizing Equation (3.13).

Theorem 3.3.3. *The minimum utility for a serving node j is achieved when it offers each client node the maximum bandwidth possible.*

Proof. Let node i aggregate all of its servers with the exception of j_1 as follows:

$$B_{u,ji}^{-j_1} = \sum_{j \neq j_1} B_{u,ji} \tag{3.15}$$

Assume that node j_1 decides to give node i a smaller allocation such as $\mathcal{B}_{u2,j_1 i} < \mathcal{B}_{u1,j_1 i}$. This will directly affect the decision of node i since the initial time was:

$$t_{1ij} = \frac{O_{1ij}}{\mathcal{B}_{u1,j_1 i}} = \frac{O_i}{B_{u1,i}} \tag{3.16}$$

where, $B_{u1,i} = \sum_j B_{u,ji} = \sum B_{u,ij}^{-j_1} + B_{u1,ij_1}$. Similarly, when node j_1 offers less bandwidth, node i responds with a new strategy that directly affects the time t_{2ij} as follows:

$$t_{2ij} = \frac{O_{2ij}}{\mathcal{B}_{u2,j_1 i}} = \frac{O_i}{B_{u2,i}}. \tag{3.17}$$

But

$$B_{u2,i} = \sum B_{u,ij}^{-j_1} + B_{u2,ij_1} \tag{3.18}$$

Since, $B_{u2,i} < B_{u1,i}$ then $t_{2ij} > t_{1ij}$ resulting in an increase to the utility of j_1 u_{j1} as expressed in (3.13). □

Thus, j has the incentive to offer each node i as much bandwidth as it can, of course, while omitting the obvious minimal solution of $B_{u,ij} = 0, \forall i$, or a free rider, that only acts as a client node without serving any objects.

To give insight to the serving node on how to divide its upload bandwidth among the client nodes, we notice the following.

Theorem 3.3.4. *The division of $B_{u,j}$ among client nodes has no effect on minimizing the utility u_j.*

Proof. If we aggregate all the requests that j receives that we denote by $O_j = \sum_i O_{i,j}$, then we need t_j time to serve the objects, where,

$$t_j = \frac{O_j}{B_{u,j}} \tag{3.19}$$

which is independent of the individual $B_{u,ji}, \forall i$. □

The recommendation for the serving node is to divide its bandwidth equally among clients irrespective of the size of the requested chunks, since any deviation from this might prompt selfish clients into abusing such a policy. For example, if the serving node offers small requests priority, then client nodes will tend to ask for a larger number of smaller chunks increasing the signaling (i.e., chunk requests) traffic, since a client node sends a signaling message for every chunk. This will increase the second terms in Equations (3.4) and (3.13). In contrast, if a serving node gives priority to larger chunks, then a client that needs a small chunk might request a larger chunk and drop the connection once it gets the smaller chunk that it initially wanted. Thus, we recommend that a serving node j offers client nodes equal portions of its upload bandwidth $B_{u,ji} = B_{u,j}/\mathbf{C}$, where $\mathbf{C} \in$

$\mathbb{N}, \mathbf{C} > 0$; in other words, $\mathbf{C}$ represents the number of clients that node j serves simultaneously or the size of its serving queue.

Next, we consider the choice of $\mathbf{C}$, the size of the serving queue. Because we want to minimize the second term of Equation (3.13), it is tempting to use $\mathbf{C} = 1$, and serve only one client at a time. In this case, the serving node j is offering its entire bandwidth which will minimize the download time (the first term in Equation (3.13)). However, this is not a desirable solution and can drive serving nodes to be untruthful in their declarations, by claiming less bandwidth than they can offer, as we show in the following theorem.

Theorem 3.3.5. *A serving node j should accept to serve at least 2 nodes in parallel, i.e.* $\mathbf{C} > 1$.

Proof. Let's assume that all nodes use $\mathbf{C} = 1$, also let us simplify the network into 2 downloads whose sizes are O_1 and O_2 ordered in increasing size. Let us assume in addition that $\mathcal{N}_1 \cap \mathcal{N}_2 = \{j\}$. Also, we define

$$\begin{aligned} B_1 &= \sum B_{u,l1}, \forall l \in \mathcal{N}_1, l \neq j \\ B_2 &= \sum B_{u,l2}, \forall l \in \mathcal{N}_2, l \neq j \end{aligned} \tag{3.20}$$

For simplicity, and only in this proof, we will denote $B_{u,j}$ as B_j. Now, we are going to show that Node j has the incentive to "cheat" by making $\mathbf{C} > 1$ under the condition that $\frac{O_2}{B_2} > \frac{O_1}{B_1+B_j}$. The same reasoning can be applied for the general case.

Scenario 1: All nodes including j assume $\mathbf{C} = 1$. The times needed to serve

O_1 and O_2 are denoted by t_1 and t_2, respectively.

$$\begin{aligned}
t_1 &= \frac{O_1}{B_1 + B_j} \\
t_2 &= t_1 + \frac{O_2 - t_1 B_2}{B_2 + B_j} \\
&= \frac{O_1}{B_1 + B_j}\left[1 - \frac{B_2}{B_2 + B_j}\right] + \frac{O_2}{B_2 + B_j} \\
&= \frac{1}{B_2 + B_j}\left[\frac{O_1 B_j}{B_1 + B_j} + O_2\right]
\end{aligned}$$

The time to serve both requests is: $t = t_2$.

Scenario 2: All nodes assume $\mathbf{C} = 1$, however node j "cheats" and assigns $\mathbf{C}$ the value of 2. The times needed to serve O_1 and O_2 become t_1^* and t_2^*, respectively.

$$\begin{aligned}
t_1^* &= \frac{O_1}{B_1 + \frac{B_j}{2}} \\
&= \frac{2O_1}{2B_1 + B_j} \\
t_2^* &= t_1 + \frac{O_2 - t_1(B_2 + B_j/2)}{B_2 + B_j} \\
&= \frac{2O_1}{2B_1 + B_j}\left[1 - \frac{B_2 + B_j/2}{B_2 + B_j}\right] + \frac{O_2}{B_2 + B_j} \\
&= \frac{2O_1}{2B_1 + B_j}\frac{B_j/2}{B_2 + B_j} + \frac{O_2}{B_2 + B_j} \\
&= \frac{1}{B_2 + B_j}\left[\frac{O_1 B_j}{2B_1 + B_j} + O_2\right]
\end{aligned}$$

In this case, the time to serve both requests becomes $t^* = t_2^*$. We can obviously see that $t_2^* < t_2$ giving node j an incentive of opting to strategy 2. However, this strategy is bad for all serving nodes other than j as $t_1^* > t_1$. □

Corollary 3.3.6. *It is desirable not to choose* $\mathbf{C}$ *as a uniform or guessable value since other serving nodes might exploit this knowledge, and again claim less than*

what they can offer. It is also desirable to keep $\mathbf{C}$ *small, otherwise the download speed as experienced by client nodes would change quite often, resulting in an increase in the second term of Equation (3.13).*

3.4 Minimum-Signaling Maximum-Throughput (MSMT) Bayesian Algorithm

We have shown in Section 3.3.2 that a node i needs to predict the upload bandwidth of its serving nodes in order to minimize Equation (3.4). In this section, we detail the MSMT algorithm for estimating the bandwidth that the serving nodes will offer a node i. This directly affect the "division" of an object into chunks at the client node among its serving nodes while dealing with no information from the network and the uncertainties arising from such a dynamic environment. Because node i has no knowledge of how the network will change over time, it needs to rely on some prediction mechanism in order to estimate Equation (3.8). MSMT is designed to be an adaptive/"smart" Bayesian algorithm in order to accurately estimate $\mathbf{E}[B_{u,ji}[n]], \forall j \in \mathcal{N}_i$ necessary for Equation (3.8), where n is the n-th round of the algorithm.

MSMT only operates at client nodes. We describe two versions of the MSMT algorithm. The Simple MSMT algorithm assumes no interfering background traffic between a client node and its serving nodes. Thus, a client node i assumes that the measured bandwidth from a serving node j is an integer fraction of j's upload bandwidth. We show that despite this simplistic assumption, the Simple MSMT

algorithm maximizes the download bandwidth while maintaining low signaling overhead. We then relax this simplistic assumption and extend the model with the General MSMT algorithm which can adapt to more challenging network conditions including congestion, losses, and node unreliability. Later in this chapter, we compare these two models in a medium-scale testbed implementation of the system using the PlanetLab platform under different network conditions.

Note that both algorithms need to predict the bandwidth obtained from serving nodes at the beginning of each round; a round is defined as the time when a node i needs to send new signaling messages (i.e., requests) to its serving nodes requesting a new set of chunks. Node i needs to undergo a new round whenever it finishes downloading at least one chunk from its serving nodes. Note that node i has no incentive to interrupt downloads and send a new set of signaling messages if the observed throughput from serving nodes change unless if at least one chunk finishes downloading. This feature is, in fact, quite beneficial since some oscillations in observed bandwidth can cancel each other, as we will see in experimental sections 3.5 and 3.6.

3.4.1 Simple MSMT Algorithm

A client node starts by building an initial probability distribution, for every serving node $j, \forall j \in \mathcal{N}_i$, also called "prior distribution". These distributions are used to extract the expected values of download bandwidth. Then the algorithm proceeds to bias these distributions at the end of every round according to the observed download bandwidth. The Simple MSMT Bayesian algorithm uses learning tech-

niques to calculate the maximum likelihood based on the observed measurements of the download bandwidth for each serving node.

A client node i that needs to download an object O_1 requests a set of initial chunks from each node in $\mathcal{N}_1$. These chunks constitute a small fraction of the total object size. This probe helps in having an estimate for $B_{u,ji}[0] \forall j \in \mathcal{N}_1$. The obtained results are used in order to build the "prior distributions" for the Simple MSMT Bayesian algorithm. Node i computes distinct prior distributions denoted by f_{ij} for each node $j \in \mathcal{N}_1$, that is, a normal distribution whose mean is $B_{u,ji}[0]$. The x-axis of the distribution is comprised of intervals, also called "clusters"; in other words, the x-axis is divided into r regions, and each region is assigned a mean b, that we refer to as a "center of gravity", and a probability. The center of gravity of each cluster is chosen such that it is scaled up or down from $B_{u,ji}[0]$ by an integer fraction. The centers of gravity are chosen in such a manner because we assume that $B_{u,ji}[0]$ is an integer fraction of $B_{u,j}[0]$. Each requesting client node i, running the Simple MSMT Bayesian algorithm, starts by estimating $\mathbf{C}_{ij}$, the size of the queue on a serving node j. A client node will then use $2.\mathbf{C}_{ij} + 1$ bins or clusters on the x-axis for its prior distribution. Now each cluster is given a specific bandwidth value for its center of gravity as follows:

$$b_{ij}[m] = \begin{cases} B_{u,ji}[0] \frac{1}{\mathbf{C}_{ij} - m + 1}, & 1 < m < \mathbf{C}_{ij} \\ B_{u,ji}[0] \frac{\mathbf{C}_{ij}}{2.\mathbf{C}_{ij} - m}, & \mathbf{C}_{ij} \leq m < (2.\mathbf{C}_{ij} - 1) \end{cases} \tag{3.21}$$

where m refers to the number of the cluster, and the set of all centers of gravity for a serving node j is denoted by $\mathbf{b}_{ij}$.

We start by assuming a normal distribution for the prior distributions, since we have no previous knowledge of the download bandwidth offered by the serv-

ing node j. When one of the chunks O_{ij} finishes downloading, node i uses the observed bandwidth $B_{u,ji}[n]$ in order to update the respective prior distributions of all serving nodes by biasing them towards the observed download bandwidth. $\mathbf{f}_{ij}$ is updated at time $t[n]$ as follows:

$$f_{ij}[\mathbf{b}_{ij}][n] = \frac{F_{ij}[\mathbf{b}_{ij}][n] + F_{ij}[\mathbf{b}_{ij}][0]}{L + seed_i}, \forall j \in \mathcal{N}_i \tag{3.22}$$

where $F_{ij}[\mathbf{b}_{ij}][n]$ is the number of times that the observed throughput offered to node i by node j is closer to the center of gravity b_{ij} up until time $t[n]$, L is the total number of observations that node i has seen for the object in question and $\frac{F_{ij}[\mathbf{b}_{ij}][0]}{seed_i}$ is the set of values used in constructing the prior distribution, $f_{ij}[0]$, that node i assumed when it started downloading. Intuitively, we are increasing the probability associated with the observed bandwidth and readjusting the distribution to keep the total equal to 1. Thus, these updates are adapting the predictions of the download bandwidth based on observed measurements.

Node i then computes the expected value for the bandwidth of each serving node as $M_{i,j} = \sum_n b_{ij}[n] f_{ij}[n], \forall j \in \mathcal{N}_i$. Then, each $B_{u,ji}$, where $j \in \mathcal{N}_i$, is estimated to be the element of $\mathbf{b}_{ij}$ closer to $M_{i,j}$ in terms of their cartesian distance. Then, Equation (3.8) is used to evaluate the new set of chunks O_{ij}. The Simple MSMT algorithm is detailed in Figure 3.2, and the state diagrams of an object download is presented in Figure 3.3.

We now present an example of the Simple MSMT algorithm. Let node 1 be the client node requesting an object of size $O_1 = 500Kb$ that is carried by 3 serving nodes; namely nodes 2, 3 and 4. Node 1 request a set of initial chunks each $10Kb$ in size from the three serving nodes. It measures the perceived upload

Simple MSMT Bayesian Prediction { // Node i for object O_i

// initialize

download $O_{ij}[0]$;

$B_{u,ij}[0] = O_{ij}[0]/t_{ij}[0], \forall j$;

decide on $\mathbf{C}_{ij}$;

calculate bins

$$b_{ij}[m] = \begin{cases} B_{u,ji}[0]\frac{1}{\mathbf{C}_{ij}-m+1}, & 1 < m < \mathbf{C}_{ij} \\ B_{u,ji}[0]\frac{\mathbf{C}_{ij}}{2.\mathbf{C}_{ij}-m}, & \mathbf{C}_{ij} \leq m < (2.\mathbf{C}_{ij}-1) \end{cases};$$

decide on the weight of prior distributions $seed_i$;

calculate $\mathbf{f}_{ij}[0], \forall j$ as normal with mean $B_{u,ji}[0]$;

// start downloading

$n = 1$;

calculate $O_{ij}[n] = B_{u,ji}[n-1]/\sum_j B_{u,ji}[n-1]$;

send requests for $O_{ij}[n], \forall j$;

// as long as O_i did not fail or finish

while ((O_i $\overline{Failed}$) AND (O_i $\overline{Finished}$)) {

if ($O_{ij}[n]$ $Done$) OR ($O_{ij}[n]$ $Wait \rightarrow Download$)

if ($O_{ij}[n], \forall j$ $Done$) then

O_i $Finished$;

else

$n = n + 1$;

locate m_j for each j s.t. $min_m|\overline{B_{u,ji}[n-1]} - b_{ij}[m]|$;

adjust distributions $\mathbf{f}_{ij}$ by biasing towards m_j;

calculate $M_{ij}[n] = \sum_m b_{ij}[m]f_{ij}[m]$;

$\langle B_{u,ji}[n] \rangle \leftarrow b_{ij}[m]$ s.t. $min_m|M_{ij}[n] - b_{ij}[m]|$;

calculate $O_{ij}[n] = \langle B_{u,ji}[n] \rangle / \sum_j \langle B_{u,ji}[n] \rangle$;

send requests for $O_{ij}[n], \forall j$;

}

}

Figure 3.2: Simple MSMT Bayesian Algorithm

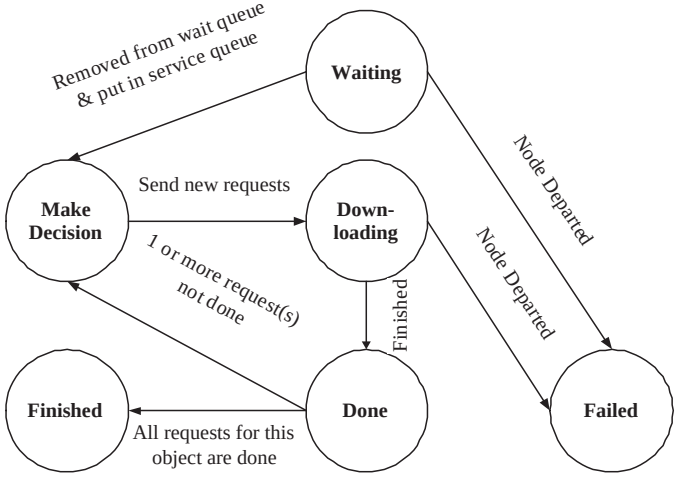

Figure 3.3: State Diagram of an Object Download

throughput as 10, 15, and 20 Kbps, respectively. Setting $C = 3$, node 1 builds 3 prior distributions for the 3 serving nodes and computes the next set of chunks for the remaining 470Kb of O_1. In this example, node 1 proceeds to request chunks whose sizes are 104Kb, 157Kb, and 209Kb, respectively from nodes 2, 3, and 4. The perceived throughput as observed by node 1 is depicted in Figure 3.4. In this example, the throughput of node 4 increases from 20 Kbps to 25 Kbps. On the other hand, node 2 maintains the same throughput until $t = 2sec$ when it increases from 10 Kbps to 20 Kbps. Likewise, node 3 maintains the same throughput of 15 Kbps, initially, however, at $t = 3.8sec$, the throughput drops to 10 Kbps. At $t = 6.3sec$ node 2 finishes downloading, thus node 1 has an incentive to re-divide the remainder of the chunks that are being downloaded from nodes 3 and 4, in order to take advantage of the offered bandwidth of node 2. Since, Node 1 is running the Simple MSMT algorithm, it biases its probability distributions and

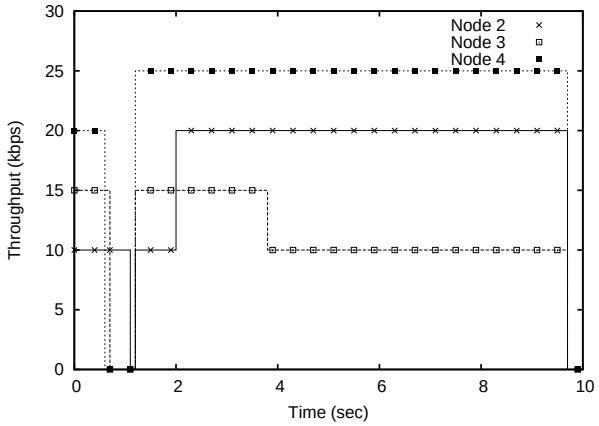

Figure 3.4: Throughput of Downloads

re-computes the expected values. As a result, it divides the chunks among nodes 2, 3, and 4 and sends requests with sizes 49Kb, 24Kb, and 61.5Kb, respectively. The downloads then proceed until they end at $t = 9.6sec$. The whole download finishes with 3 requests, including the initial measurement. Note that node 1 does not gain by re-issuing requests whenever the perceived throughput change, and is better off waiting until at least one chunk finishes downloading.

3.4.2 General MSMT

The General MSMT algorithm takes into consideration the existence of background traffic on the network between a client and a serving node. It does so by re-visiting Equation (3.21), where we define the centers of gravity. In Equation (3.21), we assume that the centers of gravity $b_{ij}[m]$ do not change with time and are directly dependent on $\mathbf{C}_{ij}$. We change this assumption in the General MSMT algorithm by adapting the values of $b_{ij}[m]$ every time we finish downloading a chunk. We start by assuming that the new centers of gravity $b_{ij}[m][0]$ are equal

to the $b_{ij}[m]$ as defined in Equation (3.21). However, the General MSMT updates these values after every download as follows:

$$b_{ij}[m][n] = \begin{cases} avg_n B_{u,ij}, & min_j |B_{u,ij} - b_{ij}[m][n-1]| \\ b_{ij}[m][n-1], & otherwise \end{cases} \tag{3.23}$$

Intuitively, Equation (3.23) can be explained as follows: for every serving node, locate the center of gravity closest to the perceived average bandwidth, change that center of gravity to the average measured bandwidth so far, for every serving node. For all other centers of gravity, do nothing.

We implement the General MSMT algorithm, test it in a real testbed subject to network fluctuations based on PlanetLab, comparing it to the Simple MSMT, and present the results in Section 3.6.

3.5 Simulation Results

In this section, we implement the client and server strategies, and the Simple MSMT algorithm in a Java-based simulator. Note, we do not consider the performance of the General MSMT in the simulator but do consider it during the PlanetLab experiments, (discussed in the next section) where its performance is more relevant to a real network with time-varying background traffic. In particular, we evaluate the impact of object size, dynamic networks (i.e., where serving nodes abruptly depart), the dependence on the size of the serving queue (i.e., number of nodes ($\mathbf{C}$) served simultaneously by a serving node), and the effect of clients re-issuing queries for downloads that are currently in progress in an attempt to gain better download performance. We first discuss our simulation setup and then

the specific experiments.

3.5.1 Simulation Design and Setup

We create a peer-to-peer network based on the Gnutella algorithm [37], where nodes join the network and establish random connections to existing nodes. We allow the network to grow to 2000 nodes. The simulator is designed to be dynamic where nodes can leave the network even if they were currently serving a request. The simulated network carries a set of files, each having an associated popularity. The popularity is drawn from an exponential distribution in order to reflect cases that are often seen in realistic networks, where some files are popular and in high demand, but, at the same time, cannot be represented by a Zipf distribution [38]. Each node carries a number of files that is greater than or equal to 0, in order to include free riders in the experiments. Any node can initiate one or more requests.

Nodes have upload and download bandwidth drawn from three different profiles where the values are typical of those observed in dial-up, broadband, and corporate settings, with bandwidth capacities of 56 Kbps, 300 Kbps, and 600 Kbps, respectively. A node i with a probability p_i initiates a request by sending a query to its neighbors with a $TTL = 5$. p_i regulates the arrival rate of requests in the system. The responding nodes constitute $\mathcal{N}_i$. At that point, the client runs the Simple MSMT algorithm and sends its requests to initiate the parallel downloads. Downloads follow the state diagram presented in Figure 3.3. Each serving node j has a serving queue of size $\mathbf{C}$ and a waiting queue of the same size $\mathbf{C}$. When a node j receives a request for a chunk $O_{i,j}$, it flags it as (i) downloading, if it has

less than $\mathbf{C}$ existing requests, (ii) waiting, if the number of existing requests is in $[\mathbf{C}, 2\mathbf{C}[$, or (iii) failed, if the number of existing requests is $2\mathbf{C}$. Whenever a chunk in the serving queue is complete, the serving node j waits for the client node i to send a new request for a chunk from the same object for a period of time before it times-out and assigns the empty slot to another request in its waiting queue, or divides its upload bandwidth among the remaining requests if the waiting queue is empty. A request also might fail when the serving node leaves the network. If all requests for an object are flagged as complete, the object is considered to be successfully downloaded and is flagged as done. However, if all requests are flagged as failed, then the object download failed and it is simply dropped.

In experiments discussed below, we compare the Simple MSMT algorithm to two other algorithms: (i) the "Last Observation" algorithm, which consists of using the last observed bandwidth as measured by the client node, as an estimate for dividing the objects into chunks; and (ii) the "Average" algorithm, which uses the average download bandwidth from a certain serving node up until the point in time where requests for a new set of chunks are needed. We compare the Simple MSMT algorithm against these two alternative, simple, and intuitive algorithms for estimating the bandwidth offered by serving nodes. For each experiment presented next, we run the same experiment five times and average over all the obtained results. The standard deviation for the five runs is very small leading us to conclude that the results are consistent.

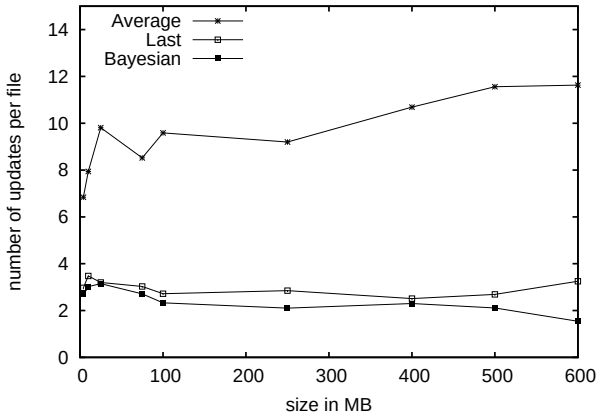

Figure 3.5: Number of Signaling Messages vs. Size of Object

3.5.2 Varying Object Size

We show, in this section, that the Simple MSMT Bayesian algorithm offers client nodes a gain in performance by decreasing the level of signaling messages irrespective of the size of the object being downloaded.

We start this experiment by populating the network with different objects of the same size. We repeat the experiment by changing the size of the objects and measure the number of signaling messages that a node sends until it finishes its download of a specific object. Only 10% of the nodes are allowed to depart during this experiment providing a fairly stable network. The results are depicted in Figure 3.5.

The "Average" algorithm shows the worst performance which is expected, since it is not able to grasp that when a change occurs that change might last for some time and averaging all the previous measurements does not provide a good prediction of future download bandwidth. In addition, as the size increases, the

number of signaling messages increases further, this is mainly due to the fact that total download time increases adding uncertainty and making the average of the past measured bandwidth even less appropriate for predicting the future behavior. The "Last Observation" algorithm offers a slightly better performance than the Simple MSMT when the object size is less than 10 MB, mainly due to the fact that the prior distribution was not always appropriate for capturing the behavior and the Simple MSMT needs some time to "learn" and adapt to the behavior of the serving nodes. However, as the size of the downloaded objects increases the Simple MSMT Bayesian requires less signaling messages and the gap between the signaling messages needed by the two algorithms increases. In this case, and for a live implementation, we propose that a node i keeps the biased distribution of a serving node j after it finishes the download of a certain object for a considerable period of time, since it has already adapted to the behavior of that serving node. Thus, re-using, in the future, the biased distribution of a serving node as the prior distribution might provide an additional benefit.

We repeat the same type of experiments where we have a mix of different object sizes in the network, which is typical of realistic networks. Figure 3.6 shows the results for the number of signaling messages versus the average of the size of objects in the network. Note that the distribution of object size is assumed to be normal. We can see that the Simple MSMT algorithm provides the best performance, outperforming this time both algorithms ("Last Observation" and "Average"), with an obvious advantage.

The two figures in this section show a behavior that looks at first counter-intuitive, where as the size of objects in the network increases, the number of

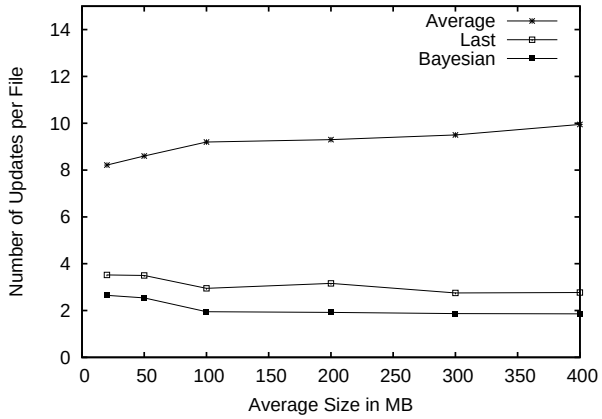

Figure 3.6: Number of Signaling Messages vs. Average Size of Objects

signaling messages decreases. This is mainly due to the fact that when the total download time is long, some oscillations in bandwidth seen in the network cancel each other; an artifact of our decision to keep downloading from serving nodes until one or more chunks finish, instead of wasting time and performance by re-issuing requests as the throughput fluctuates. Also, as we mentioned earlier, the MSMT Bayesian offers better performance as the size of the object increases, due to the fact that we are biasing the distribution with the perceived performance which provides a better fit than the initial normal distribution, as the Simple MSMT has more time to learn the exact behavior of the nodes. In fact, we can see from Figure 3.6 that the MSMT Bayesian offers a considerable gain of at least 30% over the "Last Observation" algorithm.

3.5.3 Dynamic Networks

In this experiment, we vary the departure rate of nodes in the network, in order to observe the impact on the performance of the downloads as the network becomes

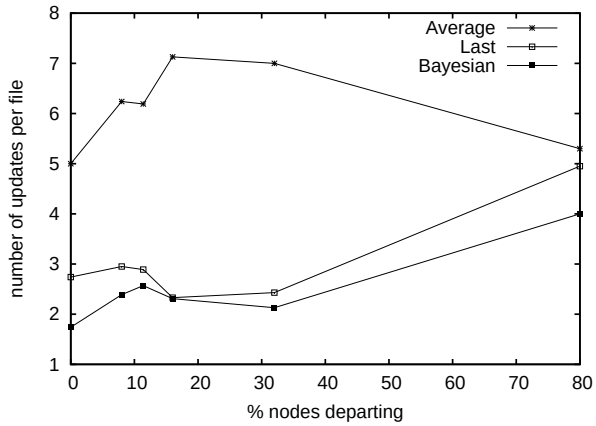

Figure 3.7: Number of Signaling Messages Per Object vs. % Nodes Departing

more dynamic, making the task of predicting future throughput challenging. The experiment initially considers a static network where no nodes depart, and then starts to increase the percentage of nodes leaving the network with 10% increments, up until we reach a point where the nodes are so volatile that 80% leave the network. The average size of objects is 50 MB. The result of the experiment is shown in Figure 3.7. The figure shows that the Simple MSMT algorithm offers the smallest overhead in terms of the number of signaling messages among the three algorithms under consideration. We can observe that "Last Observation" and Simple MSMT algorithms are quite responsive - as the number of departing nodes increases, more signaling messages are needed to continue the download.

We observe that the Simple MSMT algorithm generated the least amount of signaling messages meeting its design requirements. The "Average" algorithm, on the other hand, generated fewer signaling messages when the departure rate of nodes increased. The reason for this is mainly due to the fact that the "Average" algorithm gives equal weight to every past measurement of bandwidth when making

a future estimation. So, when the network is lightly to moderately loaded, changes in the measured bandwidth are typically not abrupt and giving more weight to more recent observations tend to match future bandwidth - a fact that the MSMT and "Last Observation" algorithms account for. On the other hand, as the load increases further, download bandwidth tends to have larger oscillations matching the blind averaging of the "Average" algorithm.

3.5.4 Varying the Size of the Serving Queue

This experiment studies the effect of the queue size of serving nodes (i.e., $\mathbf{C}$) on the system's performance. We have shown in Theorem 3 that it is desirable to have $\mathbf{C}$ strictly positive and as small as possible. In what follows, we provide some additional insights into the choice of this important system parameter. We run a set of experiments while only varying $\mathbf{C}$, which is used as the size of the serving queues and the wait queues on serving nodes as we had explained in Section 3.5.1. Figure 3.8 shows that as we increase $\mathbf{C}$, the number of signaling messages increases. However, in order to see the effect of $\mathbf{C}$ on the download bandwidth observed by the client nodes, we measure the average bandwidth per downloaded object for different values of $\mathbf{C}$. The results are shown in Figure 3.9. Even though the number of signaling messages per object increases with $\mathbf{C}$, the desired operating point seems to be somewhere between 5 and 7 as per Figure 3.9, where the observed download bandwidth reaches its maximum.

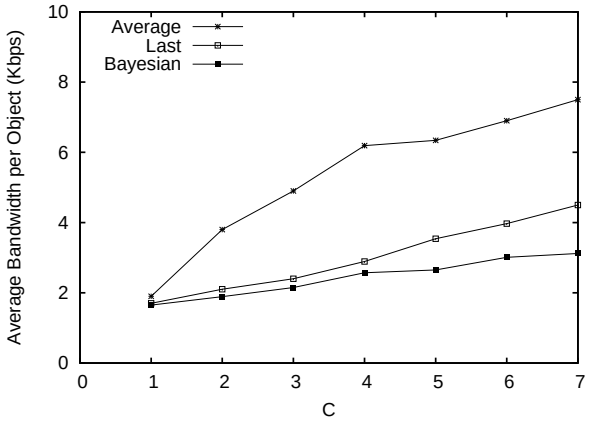

Figure 3.8: Number of Signaling Messages Per Object vs. C

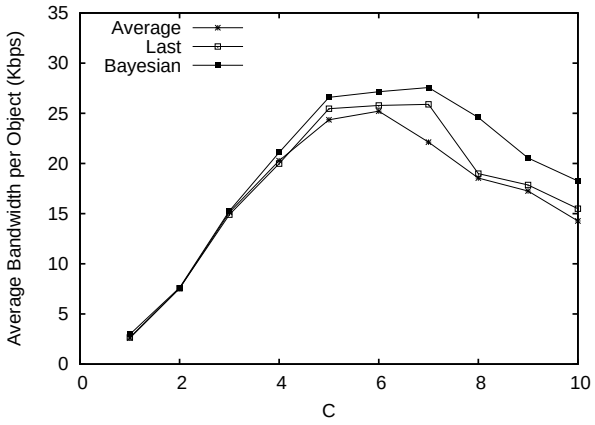

Figure 3.9: Average Bandwidth per Object vs. C

3.5.5 Re-running Queries

We assume so far that a query for an object is initiated at the beginning of a session, and then the download will start until the whole object is fully downloaded or all the serving nodes disappear before the end of the download resulting in a failed download. In this section, we investigate the case where client nodes can re-issue the query for an object currently being downloaded. We show that such a re-run of queries will help the client node, not only in getting higher total throughput in downloading its object, but also offering it a degree of resiliency in the case where all of its serving nodes depart from the network before it finishes the download. In this section, all nodes are running the Simple MSMT Bayesian algorithm.

We compare the naive strategy of no query re-runs during downloads to four alternative strategies, that are as follows:

- Periodic: the client node periodically re-runs the query.
- Lost 1 server: the client node re-runs the query as soon as it loses one or more of its serving nodes.
- Bandwidth drop by half: the client node re-runs the query when the observed total bandwidth drops by at least half of its original value.
- Servers drop by half: the client node re-runs the query when the number of serving nodes drop by at least half of its original value.

The first two methods described above (“periodic” and “lost 1 server”) are quite aggressive and greedy while the last two (“bandwidth drop by half” and “servers drop by half’) are more conservative. We run a set of experiments with a setup

similar to the one explained in Section IV.B, however, we fix the departure rate at 30% of the total number of nodes in the network. We start by plotting the distribution of the number of serving nodes per object download for the re-run strategies discussed above. The results are shown in Figure 3.10. We can see clearly that the "periodic" strategy changes the distribution of the number of serving nodes considerably increasing the number of serving nodes maintained during each object download. Intuitively, this increase will lead to more signaling messages in the network since the probability of having bandwidth fluctuations increases with the number of serving nodes. Note that the number of serving nodes per object download does not increase in an abrupt manner, in these experiments, mainly due to the fact that each serving node has a limit on the number of nodes it serves simultaneously represented by C (i.e., the serving queue), which acts as a bounding limit. The initial strategy of no re-runs has the lowest number of serving nodes as the distribution reaches its maximum at 10. The strategy of "lost 1 server" is, as we expect, quite aggressive. The remaining two strategies ("bandwidth drop by half" and "servers drop by half") offer comparable behavior to each other, as observed in Figure 3.10. This leads us to believe that a developer of a peer-to-peer application might want to opt for one of these two strategies that offer an acceptable compromise between increased throughput without incurring a large increase in the amount of signaling messages.

Next, we measure the observed bandwidth for the queries re-run strategies. The results are shown in Figure 3.11. We observe that all four strategies saturate at a higher bandwidth rate than the no re-runs policy. Thus, we can conclude that in terms of average throughput, re-running queries in general is useful and desired

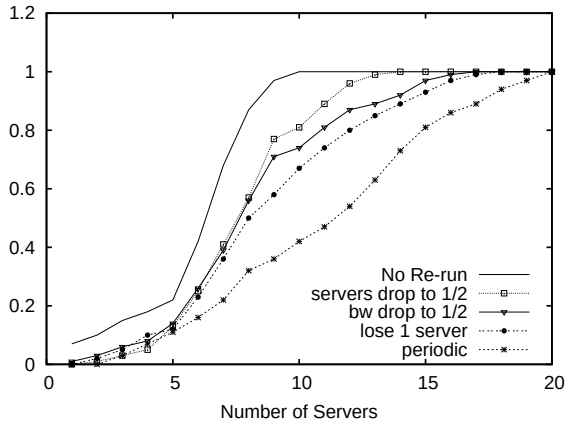

Figure 3.10: Cumulative Distribution of Number of Servers per Object

from the client nodes' perspective. In addition, the "bandwidth drop by half" and "servers drop by half" strategies offer higher throughput where some nodes experienced an average throughput of almost double that of the initial bandwidth. In contrast, the two aggressive strategies of "periodic" re-runs and re-run for "1 lost server" show some gain but do not offer a considerable change to justify the increase in signaling messages. The average number of requests sent in the case of the queries re-run strategies are 2.103, 3.521, 3.629, 8.921 and 9.735 for No Re-run, "servers drop by half", "bandwidth drop by half", "lost 1 server" and "periodic", respectively. We conclude that even though the two strategies that wait for a loss of half the bandwidth or half the servers increase the average number of signaling messages sent, they clearly make-up for this by offering considerable increases in download bandwidth to the clients. On the other hand, the two aggressive strategies of "periodic" and "lost 1 server" re-runs increase the number of signaling messages by several orders of magnitude while offering only a small increase in throughput, which might not be justifiable in most cases. Even though,

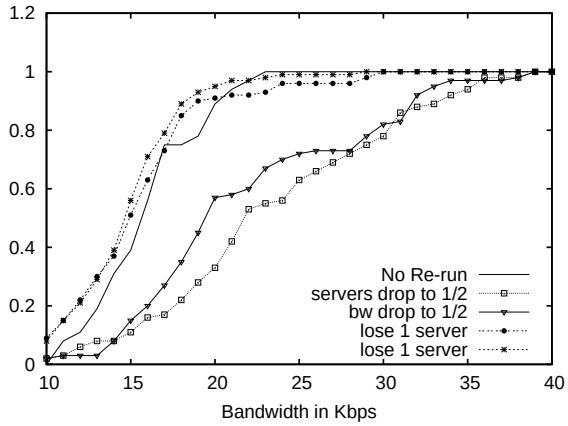

Figure 3.11: Cumulative Distribution of Average Throughput per Object

we cannot recommend a definite policy for re-runs as such a decision depends on the priorities of the user and the application in question, these experiments offer guidelines and insights into the expected behavior and outcome.

3.6 Implementation and Testbed Evaluation

In this section, we present results obtained from a testbed implementation of our system on the Planetlab platform [66]. We implement the client and server strategies, and the Simple and General MSMT Bayesian algorithms as a Java servlet (server and client) based on the open source JTella distribution [44]. We consider two sets of experiments for small (50 nodes) and larger-scale (102 node) overlays. We first deploy the code on a small number of PlanetLab nodes and measure the signaling on the network as well as the observed download bandwidth for the client nodes. We then increase the number of nodes in the second set of experiments and test the Simple and General MSMT Bayesian algorithms under

different network conditions by measuring the number of signaling messages as well as the percentage of correct prediction.

3.6.1 Experiment Set I

We deploy the code on 50 nodes located in 28 sites and spanning 11 countries (Australia, Canada, Denmark, France, Germany, Hong Kong, The Netherlands, Russia, Switzerland, UK, and USA) on the Planetlab platform [66]. We chose the sites so that they represent heterogenous environment in terms of bandwidth assigned to them where some of the sites offer high throughput typical of corporate and university networks, while others provide much lower throughput typical of home and dial-up users. The network is populated by 10 different files with sizes ranging from 4 MB to 250 MB. The distribution of the file size is 2, 3, 3, and 2 for files of 4 MB, 20 MB, 100 MB and 250 MB in size, respectively. Each participating node i with a probability p_{io} where, $1 \leq o \leq 10$ and $0 \leq p_{io} \leq 1$ decides to carry an object o. Note that if a node i has $p_{io} = 0, \forall o$, then this node is a free rider with no objects to offer to client nodes.

Signaling

After deploying these nodes, we initiate downloads where a client node forwards a query to its neighbors requesting a specific file with a $TTL = 2$. Small chunks of 50 KB (these are $O_{ij}[0]$) are initially downloaded from each of the serving nodes in order to determine their current available bandwidth. These measured throughput values are used by the client node to build the prior distributions, clusters, and

centers of gravity, as discussed in Section 3.4 for the Simple MSMT Bayesian algorithm. Division of chunks for the next round is determined at this point and the corresponding signaling messages are generated. The same experiment is repeated where client nodes use the "Last Observation" and "Average" algorithms.

We repeat the experiments while varying the number of requests in the system. Figure 3.12 shows the average time needed to download the files for the three different prediction algorithms ("Last Observation", "Average" and Simple MSMT). Note that as the number of requests increases congesting the overlay, we observe a decrease in the number of signaling messages. We can also observe how the Simple MSMT algorithm outperforms the other algorithms irrespective of the congestion in the overlay network and, on average, sends less signaling messages per downloaded object. Figure 3.13 shows that the download bandwidth decreases sharply as we increase the number of requests since we are increasing demand while supply is constant. However, the Simple MSMT provides better performance as downloads suffer from less jitters and the object finishes while maintaining relatively stable download speed versus the increased traffic in the network.

Effective Throughput

Next, we run another set of experiments on the same overlay network where we populated it with two objects O_1 and O_2 with respective sizes of 1 MB and 4 MB. Each participating nodes is either a free rider or a serving node carrying both objects. The ratios of both profiles are 30% and 70%, respectively, of the total number of nodes. We then choose a subset of nodes that we denote as $\mathcal{F}$, where

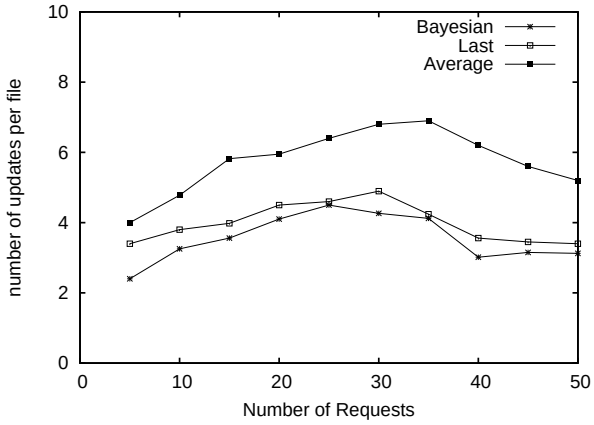

Figure 3.12: Signaling Messages per Object vs. Total Number of Requests

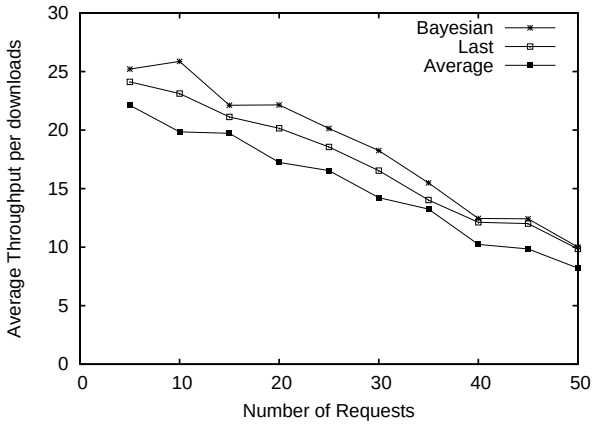

Figure 3.13: Average Download Bandwidth vs. Total Number of Requests

$\mathcal{F}$ consists of 10 free riders. We proceed by initiating a request from each node in $\mathcal{F}$ for O_1. After finishing downloading the object in question, each client node stays idle for 5 minutes before re-initiating the same request, creating background traffic in the overlay network. We then initiate a download from a free rider that we denote by Ω for O_2, where $\Omega \notin \mathcal{F}$. We repeat the same experiment from Ω for the "Last Observation", "Average" and Simple MSMT Bayesian algorithms running on all nodes. In addition, we rerun each experiment three times and report the averages in here.

The instantaneous download bandwidth measured by Ω for the different algorithms is shown in Figure 3.14. We can observe that the Simple MSMT algorithm not only offers better download bandwidth, but also a more stable allocation. The MSMT download suffers from only two dips: one at around 900 sec and another at around 1400 sec; these are due to sending a new set of signaling messages to its serving nodes due to the fact that one of its chunks has finished downloading. At the same time, both the "Last Observation" and "Average" algorithms suffer more dips and eventually take longer to finish their respective downloads. In fact, these dips, or short and sharp drops in bandwidth, are due to the fact that one or more of the parallel downloads of chunks finishes and the node runs its prediction algorithm (whether it is the "Last Observation", "Average" or MSMT Bayesian), and sends a new set of signaling messages to the serving nodes requesting different chunks. These dips also show the importance of the parameter β, the weight for minimizing the number of signaling messages, as it will directly affect the download bandwidth of the client node, and feeds-back to the first term of Equation (3.4).

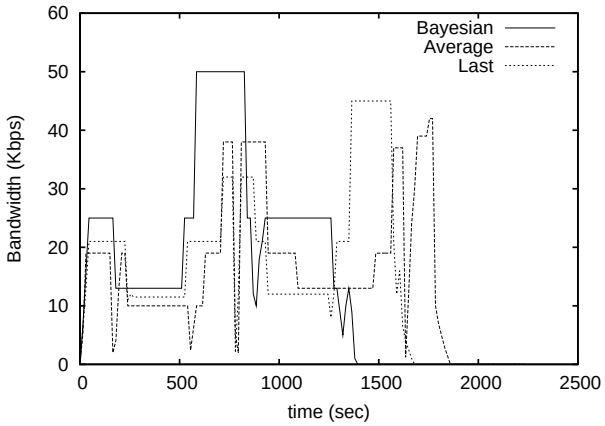

Figure 3.14: Throughput as Perceived by Ω

3.6.2 Experiment Set II

In this set of experiments, we study the system under loaded/stressful conditions, and increase the number of nodes participating in the overlay. We subject the system to congested scenarios typically found in peer-to-peer networks [38]. We deploy the code on 102 PlanetLab nodes located in 74 sites, spanning 24 countries (Australia, Brazil, Canada, China, Denmark, Finland, France, Germany, Greece, Iceland, India, Italy, Korea, Lebanon, The Netherlands, Norway, Poland, Russia, Spain, Sweden, Switzerland, Taiwan, UK and USA). We aimed to have the number of PlanetLab nodes sites as high as possible in order to minimize the cases where a node is downloading chunks from another node sitting on the same LAN. Our goal is to study more worst case scenarios where downloads have to cross WAN boundaries that are more likely to encounter congested bottlenecks or highly utilized links. We are also interested in including nodes that do not offer high bandwidth, unlike typical universities networks in North America, in order to

better represent users with low-speed access. We run the same set of experiments (presented below) several times while varying the time of day the experiment is run over a several days period. We populate the overlay with 15 files each 4 MB in size and 5 files each 600 MB in size; these files represent typical music and video files, respectively. The popularity of objects follows an exponential distribution. We start by deploying the Simple MSMT algorithm on the PlanetLab overlay and measure the number of signaling messages observed in the network. We then re-run the same experiments using the General MSMT algorithm under the same conditions in order to best measure the differences in the performance of the two algorithms. Each node issues a request with a certain probability that can vary in the same sense as the experiments discussed in in Section 3.6.1. We maintain the same ratios of 30% and 70% for free riders and active serving clients in the network, respectively.

Signaling

In this section, we report the number of signaling messages observed in the network for both MSMT algorithms and show the number of messages sent for both file sizes considered (4 MB, and 600 MB). Figure 3.15 shows the results of averaging the number of signaling messages over 5 different runs of the same experiment all under similar network conditions. Note, that even though we increase the number of requests, the load on the network before our experiment started is measured as unloaded and so are the nodes. We observe from the plot that the General MSMT outperforms the Simple MSMT. The performance difference is particularly pronounced when the number of requests increases, as more down-

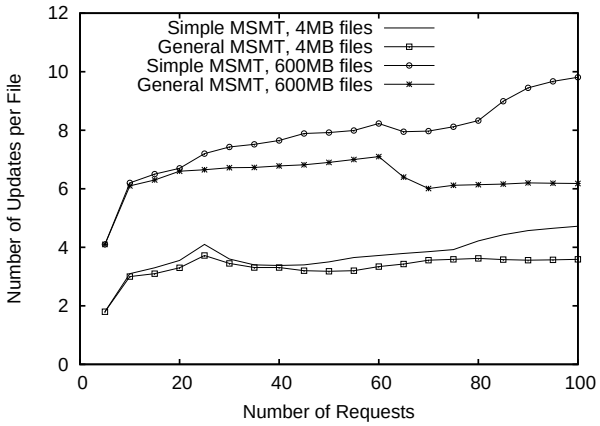

Figure 3.15: Update per Object vs. Number of Requests in the Network Under Light Conditions

loads generate increased traffic.

We repeat the same set of experiments this time under conditions where nodes and their traffic are under high load. The results are averaged and presented in Figure 3.16. We observe from the plot that the gap between the number of signaling messages associated with the Simple MSMT and General MSMT widens even more under these conditions, particularly, as the number of requests increases. Under heavy network load conditions, the download bandwidth for every serving node becomes harder to estimate since many factors influence it, such as, the load on the client/server machines, load on the providers network, and larger oscillations in available resources. As a result, the General MSMT updates its "centers of gravity" to reflect the expected download bandwidth that is used to calculate the chunks, as per Equation (3.8). The Simple MSMT algorithm continues to represent a simpler setup that assumes that the bandwidth measured is an integer fraction of the total bandwidth that its serving nodes offer. Such an estimation

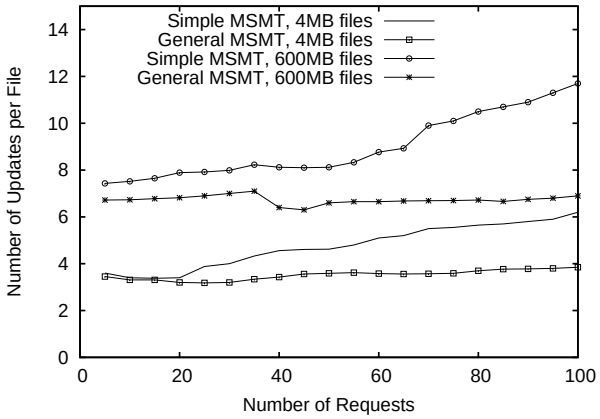

Figure 3.16: Update per Object vs. Number of Requests in the Network Under Loaded Conditions

does not take into consideration the background traffic, however. Thus, under these conditions, the Simple MSMT offers predictions that will often vary from the actual measured throughput, as is observed in Figure 3.16.

Note, that both the Simple and General MSMT algorithms require a substantially smaller number of signaling messages in comparison to any of the current peer-to-peer systems that are based on parallel downloads. For example, Overnet divides a file into equal chunks of 9.27 MB and issues at least 1 request for every chunk from any serving node. This translates to a minimum of 65 signaling messages for an object of 600 MB in size. While we can observe from figures 3.15 and 3.16 that the General MSMT algorithm requires a maximum of 28 (7 signaling messages x 4 serving nodes) messages for worst case scenario for a file of 600 MB in size, which represents a savings of 43% in signaling messages in comparison to Overnet [61].

Prediction

In this section, we attempt to measure how well our prediction mechanism performs by measuring the percentage of correct predictions. Throughout this section, we assume that the estimated download bandwidth at any step n is correct if it is within 5% of the measured average bandwidth from a specific serving node. We then report the total percentage of correct prediction that a node makes while downloading a certain object. We again consider two file sizes of 4 MB and 600 MB, separately.

In Figure 3.17, we show the percentage of correct predictions for low-load networks. We can see that the General MSMT algorithm predicts the average download bandwidth of nodes with a higher accuracy than the Simple MSMT algorithm. What is unexpected here is that the percentage of correct predictions does not deteriorate with the increase in the number of requests. This behavior is due to the fact that the General MSMT updates the centers of gravity and adapts to observed network changes. Figure 3.18 shows that the gap in correct predictions between the Simple MSMT and the General MSMT widens for small files of 4 MB in size. The reason for this is that for unloaded networks the Simple MSMT algorithm is able to provide better predictions than under loaded conditions. In the latter case, the estimates represented by the centers of gravity are based on the download of the first set of chunks $O_{ij}[0]$. Because the conditions of the network are quite stressed, the values of the initial estimates represented by the centers of gravity are no longer close to the actual download bandwidth. However, the General MSMT, by continuously updating these estimates, offers predictions that

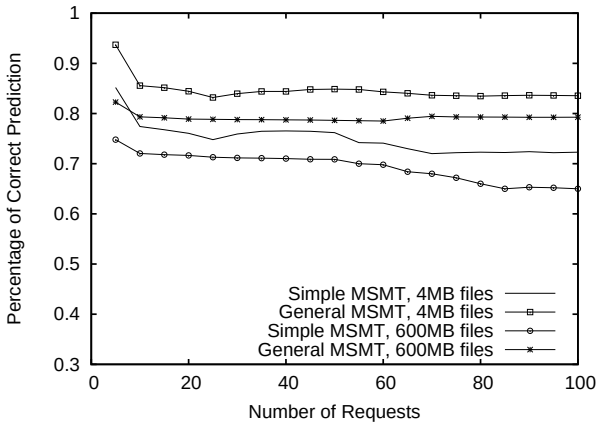

Figure 3.17: Correct Prediction vs. Number of Requests in the Network Under Light Conditions

are closer to the measured bandwidth.

3.6.3 Existing Systems

In this section, we compare the General MSMT algorithm to Limewire [53] and eMule [28]. We base our choice of Limewire and eMule on the fact that both are open source applications and can be adapted to our environment. While Limewire is a gnutella-like application, eMule is related to other popular applications such as Overnet [61], Kademlia [56], and eDonkey [26]. Thus, we conjecture that the benefits that the General MSMT provides nodes will hold true when compared to other applications as well.

Limewire classifies chunks as "black" when it has finished downloading, "grey" when it is being downloaded, and "white" as long as it has not started downloading yet. In addition, Limewire uses a "split/steal" swarming algorithm where it attempts to find a region of the file to download. Thus, if there is a "white" re-

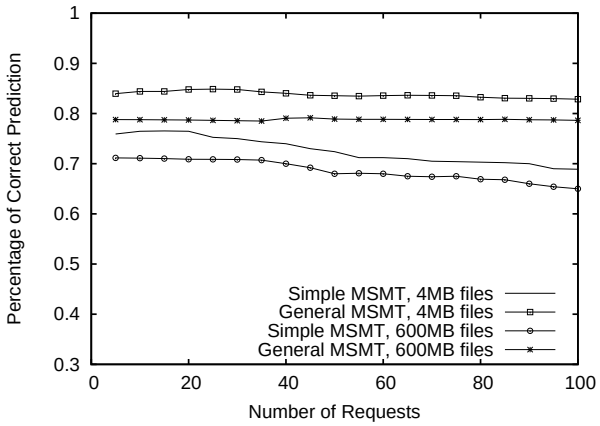

Figure 3.18: Correct Prediction vs. Number of Requests in the Network Under Loaded Conditions

gion, it sends a request for it, otherwise, it will "steal" part of a grey region from an on-going download and sends a request for it to another serving node. The documentation of Limewire [53] states that "if two threads A & B are swarming from uploaders at the same speed, the incomplete file will be downloaded in the order ABABAB... ". This means that the number of signaling messages sent is dependent on the smallest possible size that the swarm algorithm can steal. Inspecting the open source code that we downloaded on Oct 31, 2005 from http://www.limewire.org/, the smallest chunk that the swarming algorithm can "steal" is 16 KB. As for eMule, it basically divides a file into equal chunks and attempts downloads in a round robin fashion, where it waits until the download of a chunk ends to send a request for another chunk to the same serving node. According to the documentation of eMule v0.46c, a chunk is 9.28 MB.

In this set of experiments, we compare the signaling overhead of eMule, Limewire and the General MSMT algorithms, while the setup is the one detailed in Section

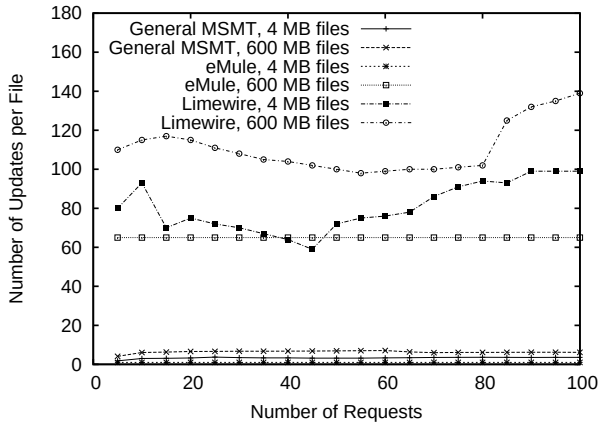

Figure 3.19: Comparing General MSMT to Existing Systems (Signaling Messages)

3.6.2. The results are depicted in Figure 3.19.

Since, eMule decides on a constant size of chunks up-front, we notice that the number of signaling messages sent does not change for the whole experiment, and increases linearly with the increase in file size. Note that the low number of signaling messages for small files (4 MB in our experiment) seems deceptively desirable. In fact, eMule downloaded the file from 1 single serving node instead of taking advantage of all nodes that carry the file, which contradicts the whole idea of parallel downloads.

Limewire, on the other hand, has a different behavior, where as the load in the system increases the number of signaling messages starts to decrease. However, after a certain point, the number of signaling messages increases at a fast rate. Also, note that for either file sizes (4 MB and 600 MB), Limewire seems to have the highest cost among the algorithms in terms of signaling messages.

Figure 3.20 shows the comparison among the algorithms for the average through-

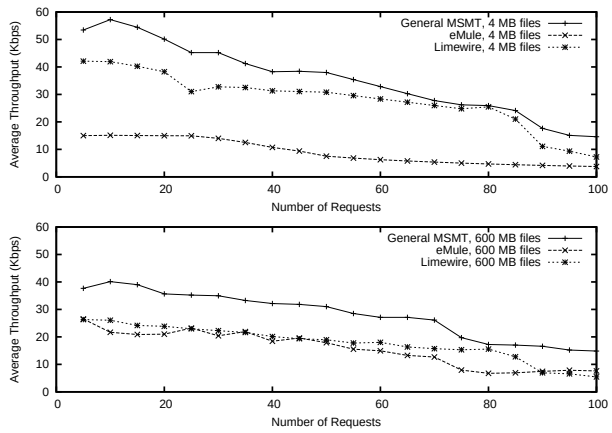

Figure 3.20: Comparing General MSMT to Existing Systems (Throughput)

put as perceived by the client nodes. Note that we measure the average throughput as the size of the file divided by the time to download all the chunks of that file. As expected, eMule offered the lowest average throughput, especially for small files since it is not taking advantage of the offered bandwidth of all serving nodes, which changes when the size of the file increases but remains far from the optimal case. Limewire offers download speeds slightly lower that those of the General MSMT, mainly due to the considerable higher amount of signaling messages and smaller sizes for the chunks. And, again the throughput of Limewire deteriorates as the load increases considerably due to the fact that the swarming algorithm is stealing smaller chunks and increasing the signaling overhead. In conclusion, the General MSMT seems to offer a balance between lower signaling messages and higher average throughput satisfying Equation (3.4).

3.7 Summary

In this chapter, we have shown that because of selfish nodes, the current implementations of parallel downloads in peer-to-peer networks provide far from optimal download performance. We formulated the optimal solution for the division of objects into chunks for simple networks with static nodes and uncongested connections. We discussed how such an optimal decision might lead nodes to untruthful declarations whether on the client or serving side. We defined a number of strategies to discourage nodes from such behavior and proposed the MSMT algorithm to provide nodes with a solution as close to the optimal division of objects into chunks, under realistic network conditions. We designed the MSMT algorithm to provide the maximum download speed to client nodes, by downloading objects as fast as possible. At the same time the MSMT algorithm maintains a low signaling overhead. We evaluated the effects of different parameter settings on individual nodes, as well as the network as whole, using simulations and results from an medium-scale experimental testbed running on the PlanetLab platform [66]. Our results show that our strategies and algorithms offer increased download performance and decreased signaling cost in comparison to other existing parallel download approaches.

Chapter 4

A Learning Based Approach for Network Properties Inference

4.1 Introduction

A number of emerging popular applications require the creation and maintenance of on-demand overlay networks of end systems. Such applications benefit from connecting to nodes that meet certain criteria, instead of choosing a random set of server nodes on the network. For example, a streaming media client would benefit by connecting to a media server that is lightly loaded and has high downstream available bandwidth and low latency. More sophisticated applications and services would use dynamic service composition in which the problem entails the computation of a service overlay path with the necessary service components, matching the required QoS criteria.

Finding the node or subset of nodes that meet some criteria of QoS metrics, us-

ing an exhaustive search, could translate to every node conducting measurements to every other node on the network. This approach is, at best, not scalable as the order of these extensive measurements is $O(N^2)$, where N is the number of nodes in the system. Previous work [22], [30], [31], [39], [59], [65], [80], [84], [89], [92] have looked at the problem of estimating one criterion, being the latency, and proposed conducting a smaller number of measurements, and estimating the closest node or subset of nodes to a specific node on the network, in terms of the round trip delay. All of these methods use some heuristics based model, and have different degrees of success in estimating network metrics, depending on the specifics of the network topology and structure. In fact, these existing system do not consider time as building component for their solution and instead force nodes into repeating their measurements continuously in order to adapt to the network dynamics with respect to changes over time. Thus, these methods lack the ability to learn and adapt to the changes in the underlying structure and dependencies between different components. A more general method that can adapt dynamically to the changes in network structure and provide high estimation accuracy is required.

In this chapter, we propose to apply a learning based Bayesian network approach to the problem of inferring network properties that is adaptive and does not depend on specific heuristics. The Bayesian approach is very powerful and has been applied in multiple technology domains with great success [21]. To assess viability of the proposed method, we present results related to node proximity presented by round trip delay and hop numbers. In the future, we plan to investigate other metrics such as uptime, bandwidth, interests (or communities), and

fluctuation in performance.

We require the presence of landmarks that all nodes conduct traceroute measurements to. Note that landmarks are defined as special nodes where each node in the system performs its measurements to these landmarks. Using the outcome of these measurements, we keep track of routers that appear more than once, which we denote as milestones, as was suggested in [92]. We use this information to further infer the topology of the network. We approach the problem by extracting signature-like profiles for nodes from the acquired information, including distances to milestones and landmarks. An important characteristic of these profiles is the fact that they can be anonymous making the system more scalable, as it uses a subset of nodes to generalize behavior and detect similar behavior among, otherwise, totally different nodes. When estimating the distance between two nodes in the system, we use their respective signatures to infer the answer. Our estimation approach then relies on probabilistic techniques based on Bayesian Networks [42]. Our obtained results are quite promising and provide a considerable gain when compared to existing systems.

Our contribution lies in dividing the node metrics estimation problem into two modules: profiling of nodes and then, accordingly, estimating the metrics in question. In doing so, we introduce the idea of signature-like anonymous profiles that make our system more scalable. In addition, we use machine learning techniques, more specifically Bayesian networks, in order to estimate the required metrics. We show through experimental results that such a probabilistic approach provides superior results when compared to existing systems.

This chapter proceeds with the Related Work presented in Section 4.2. We

then present the system in Section 4.3. In Sections 4.4 and 4.5, we discuss the data collected and the results obtained. We finally summarize in Section 4.6.

4.2 Related Work

Several schemes have been proposed to estimate Internet path properties. In this section, we review only the techniques to estimate network distances and proximity since we apply the learning based approach to estimate these metrics. Internet Distance Maps (IDMaps) [31] places tracers at key locations in the Internet. These tracers measure the latency among themselves and advertise the measured information to the clients. The distance between two clients A and B is estimated as the sum of the distance between A and its closest tracer A', the distance between B and its closest tracer B', and the distance between the tracers A' and B'.

M-coop [79] utilizes a network of nodes linked in a way that mimics the autonomous system (AS) graph extracted from BGP reports. Each node measures distances to a small set of peers. When an estimate between two IP addresses is required, several measurements are composed recursively to provide an estimate. King [39] takes advantage of the existing DNS architecture and uses the DNS servers as the measurement nodes.

King, M-coop, and IDMaps all require that the IP addresses of both the source and the destination are known at the time of measurement. Therefore, they cannot be used when the IP address of the target node is unknown.

There are schemes that use landmark techniques for network distance estimation. Landmark schemes [59, 70] use a node's distances to a common set of land-

mark nodes to estimate the node's physical position. In these schemes the nodes conduct measurements to every landmark node. The intuition behind such techniques is that if two nodes have similar latencies to the landmark nodes, they are likely to be close to each other. One such technique, called *Landmark ordering*, is used in topologically-aware Content Addressable Network (CAN) [70]. With landmark ordering, a node measures its round-trip time to a set of landmarks and sorts the landmark nodes in the order of increasing round-trip time (RTT). Therefore, each node has an associated order of landmarks. Nodes with the same (similar) landmark order(s) are considered to be close to each other. This technique however, cannot differentiate between nodes with the same landmark orders.

GNP (Global Network Positioning) [59] is another landmark based scheme. In this scheme, landmark nodes measure RTTs among themselves and use this information to compute the coordinates in a Cartesian space for each landmark node. These coordinates are then distributed to the clients. The client nodes measure RTTs to the landmark nodes and compute the coordinates for themselves, based on the RTT measurements and the coordinates of the landmark nodes it receives. The Euclidean distance between nodes in the Cartesian space is directly used as an estimation of the network distance.

GNP requires that all client nodes contact the same set of landmarks nodes, and the scheme may fail when some landmark nodes are not available at a given instant of time. To address this problem, Lighthouse [65] allows a new node wishing to join the network to use any subset of nodes that is already in the system (i.e., *lighthouses*) as landmarks to compute a global network coordinate based on measurements to these lighthouses.

Despite the variations, current landmark techniques share one major problem. They cause *false clustering* where nodes that have similar landmark vectors but are far away in network distance are clustered near each other.

Vivaldi [22] is another scheme that assigns a coordinate space for each host, but it does not require any landmarks. Instead of using probing packets to measure latencies, it relies on piggybacking when two hosts communicate with each other. With the information obtained from passively monitoring packets (e.g., RPC packets), each node adjusts its coordinates to minimize the difference between estimates and actual delay. Although Vivaldi is fully distributed, it takes time to converge, requires applications to sample all nodes at relatively same rate to ensure accuracy, and expects packets to add Vivaldi-specific fields.

Netvigator [92] is an attempt to leverage triangular inequality and improve the performance of landmark-based measurements. Instead of ping measurements, each node conducts traceroutes to selected landmark nodes. It performs triangular inequality based clustering heuristic, called *min_sum*, using the distance information not only between the nodes and landmarks but also between nodes and the intermediate routers. Hence, *Min_Sum* is an upper bound on the distance between the various nodes. While the performance results from PlanetLab measurements are promising, the tightness of this upper bound is dependent on the coverage of the underlying topology by the traceroute measurements. In this chapter, we use *Min_Sum* as a candidate for comparing the performance of our approach.

In addition, all of these techniques lack adaptability and require nodes to repeat their measurements, continuously, to ensure accurate results for the estimations.

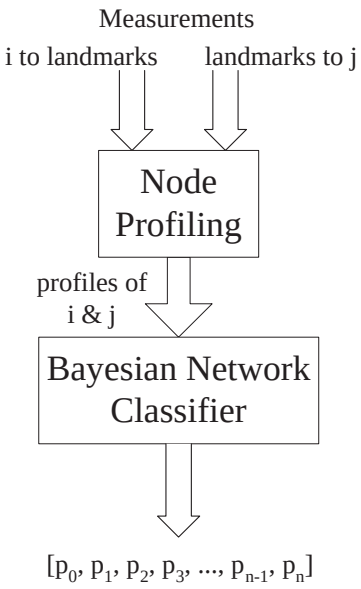

Figure 4.1: System Block Diagram

4.3 Profiling and Learning-based Estimation Techniques

In this chapter, we propose a new approach to infer and predict network properties based on machine learning techniques, such as Bayesian Networks. The goal of learning based prediction is to build a system that can learn from the profiles of nodes and, eventually, achieve a degree of "expertise" where changes in the metrics of existing nodes can be predicted. We believe that such a system will provide nodes with better predictions of changes in metrics and can achieve this goal in a scalable fashion.

Figure 4.1 shows the two basic components of our approach:

- **Profiler**: The profiler creates signature-like profiles for nodes, which basically capture the characteristics of the nodes, as well as the typological relationship between different nodes in the network. The profiling mechanism is primarily based on the knowledge about the *known* relationships between different nodes and how it might affect the metrics being estimated. As a rule, the signatures do not carry the explicit identity of the node in question. By doing so, we aim at creating an inference engine that scales with the dynamics of the network related to nodes joining and leaving, where signatures can sufficiently reflect nodes behavior without attaching an identity of a specific node to a profile, thus creating a general profile. This idea draws similarity from the approach used in detecting worms on the Internet by creating signatures of their behavior.

- **Learning-based Prediction Engine**: The profiles generated by the profiler are used as input to the prediction module. Initially, the prediction module undergoes a training period where a subset of true values of the metrics of interest are provided to the learning engine. In this chapter, we focus on Bayesian networks as a learning mechanism for the prediction engine. Based on the training, the prediction engine can learn about the *latent* dependencies in the system. A trained prediction engine, then, takes node profiles as input to provide a final estimate for the metrics in question.

Our proposed system can be used for estimating different parameters, however, in this chapter, we limit the metrics to the number of hops and latency among nodes. Studying and evaluating other metrics is part of on-going research. In the

example we show in Figure 4.1, the output is a vector, labeled $[p_0, ..., p_n]$, representing the probability distribution for the different classes, since the estimation in here is done using classification. For example, if we are targeting hop number estimation, the output is a probability distribution of the hop numbers between two nodes. The maximum number of hops is assumed to be 32 hops, thus, in the output vector p_{i-1} represents the probability of the number of hops being i.

Similar to landmark-based approaches, such as [92], in our system each node conducts traceroute measurements only to a set of selected landmarks. Before describing our algorithm, we proceed with a brief description of the Min-Sum [92] algorithm so as to introduce various terms. We then describe our profiling techniques and discuss our estimation algorithm based on Bayesian networks.

Many systems target latency estimation, as we describe in Section 4.2, however, we are not aware of any mechanism or algorithm proposed for hop number estimation. Thus, we modify the Min-Sum algorithm, used for latency estimation in [92], in order to estimate hop number in addition to latency and use it for comparison purposes. When we evaluate our algorithm for latency estimation, we compare it also to Vivaldi [22].

4.3.1 Min-Sum Algorithm

As mentioned earlier, the Min-Sum algorithm proposes estimating network latencies among nodes using heuristics based on triangular inequality. In here, we provide a short summary of its operation.

In a system with N nodes and L landmarks, each node conducts traceroute

measurements to every landmark. We refer to these measurements as the discovery of the uplink routes. In addition, if we are considering the asymmetric Min-Sum algorithm where routes on the network can be asymmetric, then each landmark will also conduct traceroute measurements to every node on the network. We refer to this set of measurements as the discovery of the downlink routes. The result is $2 * N * L$ measurements. Every time a router is encountered more than once, then its status is "promoted" to milestone. Note that the definition of a router includes the landmarks themselves, even if they are, physically, servers or end-nodes, thus all landmarks are milestones by definition. We denote the set of common milestones encountered on the uplink routes from node i and the downlink routes to node j as $L(i, j)$. The min-sum algorithm then estimates the distance between a node i and a node j as:

$$min(dist(i, l) + dist(l, j)), \forall l \in L(i, j) \tag{4.1}$$

In fact, considering the intuition of triangular inequality, the min-sum algorithm provides an upper-bound estimate for network latency among nodes.

4.3.2 Profiling Techniques

In here, we present the four profiling techniques that we explored. Based on the results of comparing the performance of the four techniques, detailed in Section 4.5, the Node Histogram provides the best performance. Hence for sake of brevity, we only provide extensive results for the Node Histogram Profiling Algorithm, in this chapter. We now describe the operation of the algorithms, a summary of their pseudocode is presented in Figure 4.2.

Calculate m-Closest Profile { // from i to j
obtain $\mathcal{M}_{i,up}$ & $\mathcal{M}_{j,down}$;
calculate distances $\mathcal{D}_{i,up}$ from i to $\mathcal{M}_{i,up}$;
calculate distances $\mathcal{D}_{j,down}$ from $\mathcal{M}_{j,down}$ to j;
$P_{i,j} = [\mathcal{D}_{i,up}[1..m], \mathcal{D}_{j,down}[1..m]]$;
}
Calculate m-Closest with Counter Profile { // from i to j
obtain $\mathcal{M}_{i,up}$ & $\mathcal{M}_{j,down}$;
calculate distances $\mathcal{D}_{i,up}$ from i to $\mathcal{M}_{i,up}$;
calculate distances $\mathcal{D}_{j,down}$ from $\mathcal{M}_{j,down}$ to j;
$\mathcal{M}_{i,j} = \mathcal{M}_{i,up}[1..m] \cap \mathcal{M}_{j,down}[1..m]$;
$C =\mid \mathcal{M}_{i,j} \mid$;
$P_{i,j} = [\mathcal{D}_{i,up}[1..m], \mathcal{D}_{j,down}[1..m], C]$;
}
Calculate Node Histogram Profile { // from i to j
obtain $\mathcal{M}_{i,up}$ & $\mathcal{M}_{j,down}$;
calculate distances $\mathcal{D}_{i,up}$ from i to $\mathcal{M}_{i,up}$;
calculate distances $\mathcal{D}_{j,down}$ from $\mathcal{M}_{j,down}$ to j;
map $\mathcal{D}_{i,up}$ to a histogram $\mathcal{H}_{i,up}$;
map $\mathcal{D}_{j,down}$ to a histogram $\mathcal{H}_{j,down}$;
$P_{i,j} = [\mathcal{H}_{i,up}, \mathcal{H}_{j,down}]$;
}
Calculate Milestone Histogram Profile { // from i to j
obtain $\mathcal{M}_{i,up}$ & $\mathcal{M}_{j,down}$;
$\mathcal{M}_{i,j} = \mathcal{M}_{i,up} \cap \mathcal{M}_{j,down}$;
$P_{i,j} = \phi$;
for every $ms \in \mathcal{M}_{i,j}$
obtain $I_{ms,up}$ (nodes that pass ms on uplink);
calculate distances $\mathcal{D}_{ms,up}$ from $I_{ms,up}$ to ms;
obtain $I_{ms,down}$ (nodes that pass ms on downlink);
calculate distances $\mathcal{D}_{ms,down}$ from ms to $I_{ms,up}$;
map $\mathcal{D}_{ms,up}$ to a histogram $\mathcal{H}_{ms,up}$;
map $\mathcal{D}_{ms,down}$ to a histogram $\mathcal{H}_{ms,down}$;
$P_{i,j} = [P_{i,j}; \mathcal{H}_{ms,up}, \mathcal{H}_{ms,down}]$;
}

Figure 4.2: Bayesian Profiling Algorithms Pseudocode

m-Closest

In order to estimate the distance in terms of number of hops from node i to node j using the m-Closest profiling algorithm, a node starts with the set of milestones that it encounters when running traceroute measurements to the landmarks. Of course, this set includes the landmarks themselves. We denote this set of milestones by $\mathcal{M}_{i,up}$. The profiling module then builds a vector that we denote by $\mathcal{D}_{i,up}$ that contains the distances from node i to every milestone $ms_i \in \mathcal{M}_{i,up}$. The profiling module then sorts the vector $\mathcal{D}_{i,up}$ in ascending order, and truncates the first m values. Thus, the signature-like profile of a node i becomes the distances from i to the m-Closest milestones that it encounters. Similarly, for the destination node j, the profiling module considers the traceroutes from the landmarks to j, extracts the encountered milestones that we denote by $\mathcal{M}_{j,down}$, builds the distances vector $\mathcal{D}_{j,down}$ of the milestones to j in ascending order, and truncates the first m values. The resulting vector that feeds into the Bayesian network has a dimension of $2m$.

m-Closest with Counter

The m-Closest with Counter algorithm operates in a similar fashion to the m-Closest algorithm. The profiling module builds the same vector as the m-Closest consisting of distances to the m-Closest milestones to the nodes in question. In addition, a counter is added that represents the number of common milestones whose distances are included in the m-truncated vectors.

We present an example of the operation of the m-Closest and m-Closest with

Counter algorithms. Assume that we have the case presented in Figure 4.3 with three nodes and three landmarks. We would like to estimate the distance in terms of number of hops from Node 1 to Node 2. Inspecting the traceroute measurements from Node 1 to the three landmarks reveals that two milestones were discovered along the routes, namely Milestone 1 and Milestone 2, with distances from Node 1 of 3 and 2 hops, respectively. Similarly, analyzing the traceroute measurements on the downlink from the landmarks to Node 2, we encounter three milestones, namely Milestone 1, 2 and 3 with distances of 5, 4 and 2 hops, respectively. Applying the m-Closest algorithm with m=2, we obtain the following vector profiles representing the distances to the 2-closest milestones for each node:
Node 1: [2, 3]
Node 2: [2, 4]
Thus, the input to the Bayesian module becomes the concatenation of these 2 profiles: [2, 3, 2, 4].

As for the m-Closest with Counter, we add a counter, that we denote by C, indicating how many of the milestones whose distances are presented in the m-Closest vector are in common. In our example of Figure 4.3, we only have 1 milestone in common (Milestone 2), so we set $C = 1$. The input to the Bayesian module becomes: [2, 3, 2, 4, 1].

With the m-Closest and the m-Closest with Counter algorithms, we create, as desired, anonymous profiles for the nodes, that do not hold the specific identities of the nodes. The signature-like profiles for the nodes created by these algorithms capture the connectivity of the nodes by registering number of milestones at different hops numbers from the node. As the computation overhead of the prediction

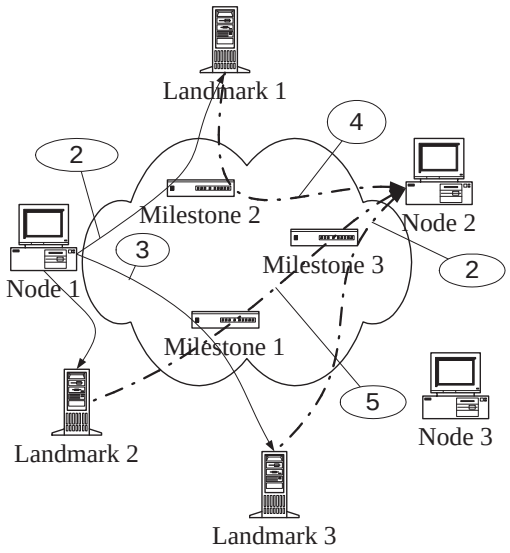

Figure 4.3: Example of m-Closest Algorithms

module depends on the length of the input vectors, both of these algorithms truncate information about the milestones to consider only the m-Closest milestones.

Node Histogram

The Node Histogram profiling algorithm is designed to retain topological information about the position of nodes with respect to all milestones encountered with traceroute measurements. When conducting measurements to landmarks, a node i encounters a set of milestones that we denote by $\mathcal{M}_{i,up}$. The distances to these milestones is represented by the vector $\mathcal{D}_{i,up}$. Node i converts $\mathcal{D}_{i,up}$ into a histogram that we denote by $\mathcal{H}_{i,up}$. As an example of this, consider Node x with the following distances to milestones vector $\mathcal{D}_{i,up} = [2, 2, 3, 5, 6, 6, 6, 8, 10]$. Mapping this vector into a 12-dimensional histogram, we obtain $\mathcal{H}_{i,up} = [0, 2, 1, 0, 1, 3, 0, 1, 0,$

$1, 0, 0]$. Note that the histogram starts with 1 as the minimum distance. In the above example, since we had no milestone that is 1 hop away from Node x, we set the first value to 0. However, we have two milestones that are each 2 hops away, thus we set the second value to 2, and so on. Note that in our implementation, $\mathcal{H}_{i,up}$ is a 32-dimensional vector, representing the maximum number of hops as defines in traceroute measurements. Similarly, a histogram is built for the downlink measurements for every node denoting the distances from the milestones to the node. We denote the downlink histogram vector by $\mathcal{H}_{i,down}$. Thus, the input to the Bayesian module consists of $[\mathcal{H}_{i,up}, \mathcal{H}_{j,down}]$ when estimating the distance from node i to node j.

Visualizing the Node Histogram profiling algorithm, if a node sits in the center, the algorithm builds a vector that includes all milestone information for a node. It aggregates these milestones as concentric circles. The circles have increasing order radii and different "intensities" corresponding to the number of milestones that are at a certain distance from the node. Just like the m-Closest algorithms, the Node Histogram algorithm generates an anonymous profile that does not carry the specific node's identity.

As a reminder, the network comprises of N nodes and L landmarks. Every node i conducts traceroute measurements to every landmark discovering the uplink routes, and every landmark l conducts traceroute measurements to every node in the system discovering the downlink routes. As a result, and whenever a router appears on at least one uplink and one downlink route, then it is considered a milestone. In addition, all landmarks, by default, are considered milestones. Thus, after collecting all of these measurements, the network discovers M milestones.

Milestone Histogram

The Milestone Histogram profiling algorithm looks at the network from the milestones' perspective. In fact, this algorithm is similar to the Node Histogram in the sense that it builds the circle around a specific node. However, instead of using an end-node as the center, it builds the circle around a milestone.

Every milestone ms in the system is encountered by a set of nodes on the uplink measurements that we denote by $N_{ms,up}$ and a set of nodes on the downlink measurements denoted by $N_{ms,down}$. The distance vectors from the nodes in $N_{ms,up}$ to ms is denoted by $\mathcal{D}_{ms,up}$, while the distance vector from the milestone ms to all nodes in N_{ms_down} is $\mathcal{D}_{ms,down}$. Similarly to the Node Histogram, we map the distance vectors $\mathcal{D}_{ms,up}$ and $\mathcal{D}_{ms,down}$ into distance histograms $\mathcal{H}_{ms,up}$ and $\mathcal{H}_{ms,down}$, respectively.

When estimating the distance from a node i to a node j, the Milestone Histogram algorithm will inspect the traceroute measurements from node i to the landmarks and those from the landmarks to node j. The first set of measurements yields a set of milestones $M_{i,u}$, while the second set reveals a set of milestones $M_{j,down}$. We define $M_{i,j} = M_{i,up} \cap M_{j,down}$ as the set of common milestones. Then, we use the uplink and downlink histograms $[\mathcal{H}_{ms,up}, \mathcal{H}_{ms,down}]$ of every milestone $ms_x \in M_{i,j}$. This means that the Bayesian module is going to be queried for an estimation $\mid M_{i,j} \mid$ times corresponding to every milestone in $M_{i,j}$. In the evaluation, we present in Section 4.5, we average all the estimates in order to obtain one final estimate of the distance in terms of number of hops between node i and node j. Note that the Milestone Histogram algorithm is more computa-

tionally intensive than the Node Histogram algorithm, since for every estimation we are querying all milestones. Also note that the set of landmarks is included in every $M_{i,j} \forall i, j$ following the definition of a milestone that includes all landmarks in the system. This basically means that any pair of nodes have at least L (the set of landmarks) milestones in common, which means will have to run at least L different estimations.

4.3.3 Bayesian Techniques

The block diagram of our proposed estimation Bayesian algorithm is depicted in Figure 4.4. In describing the Bayesian algorithm, with a slight abuse of notation, we are going to refer to the Bayesian network nodes as components in order to avoid confusion with the use of the word node to denote participating machines on the physical network. Thus, expanding the Bayesian network, as shown in Figure 4.4, Block 3 has the profiles of the nodes as input, and is a continuous Gaussian component. In addition, Block 2 is a hidden binary component, and Block 1 is the output component acting as a T-class classifier. Thus, the output of the Bayesian network is a T-dimensional vector representing the probability distribution of the T different classes. In the case of hop numbers estimation $T = 32$ corresponding to the hop numbers between the two input nodes. Note that this Bayesian network structure is quite simple where we have one component for each of the input and output and one hidden node. The goal of hidden component is to capture the latent relationships. We also experiment with a more complex structure.

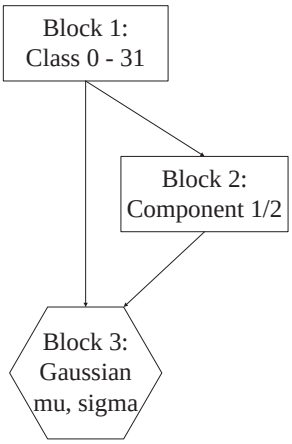

Figure 4.4: Simple Bayesian Network Structure

For example, if we need to estimate the distance between node i and node j, we use the measurements from i to the landmarks and those from the landmarks to j as an input to the profiling module. This first module will create the respective profiles of i and j to feed into the Bayesian estimation algorithm, and the second module of our system will output a decision vector. We choose to use the median of the output probability distribution as the value of the estimation. Thus, the estimated distance is actually the position (or index) of the maximum value in the T-dimensional output vector.

We also modify the Bayesian network as presented in Figure 4.5 where we divide the input into two vectors corresponding to the profiles of the two nodes in question. We also add another hidden block for the newly introduced input node. This modification of the structure of the Bayesian network takes into consideration the fact that the input consists of two independent vectors, being the two profiles of the two nodes in question. We compare the performance of both Bayesian network

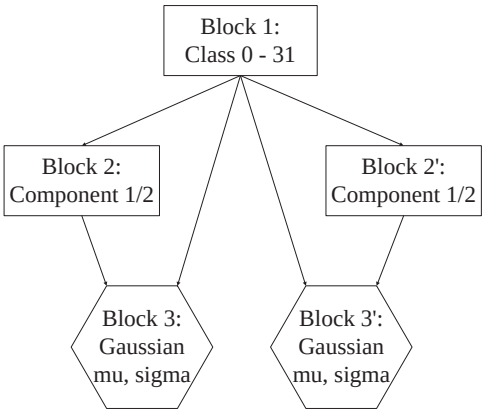

Figure 4.5: Modified Bayesian Network Structure

structures in Section 4.5. In our implementation of these Bayesian networks, we used the Bayes Net Toolbox (BNT) [11] on Matlab 7.0.1 [55].

4.4 Measurement Setup

In order to test our proposed algorithms, we collected measurements on the PlanetLab platform that involved all 580 machines participating in the network as of August 2005. We deployed a modified version of the scriptroute suite of tools [76], where we removed the restrictions on the number of simultaneous measurements that exist in the default distribution. Our measurements engine, on every node, runs once every 8 hours collecting information using 3 tools, namely ping, traceroute and rockettrace (a modified version of traceroute that ships with scriptroute), targeted towards all other nodes on PlanetLab. Collecting such data is essential for testing the correctness of the estimates that the algorithms will provide in a real system. We also used a very small subset of these measurements for

training the Bayesian network. The engine collected data from August 1 through August 10, 2005. While conducting these measurements, each engine on every node, independently chooses a random starting point from the list of the PlanetLab nodes; this was essential so that our massive measurements will not be mistaken for a DDOS (Distributed Denial of Service) attack and ensure that nodes are not in sync when sending their probing packets to any specific node.

When considering hop numbers, the data that we collected turned out to be time-insensitive, where, with few exceptions, the number of hops between pairs of nodes did not vary over time. The few exceptions included nodes that were not responsive either due to a problem on the node itself or due to a restart, where traceroutes to these nodes were unsuccessful. Other exceptions seemed due to some loops in the network or to other strange behavior where the final destination of a traceroute seems to repeat few times before the measurement ends. These problems were mainly apparent when one of the end nodes was an alpha PlanetLab node (an alpha node means a node that is still under development and unreliable). Thus, when presenting and testing the proposed algorithms below, we do not include any time component in our studies of hop numbers estimation.

In addition, the results presented test the validity of the system for different subsets of the collected data in terms of number of nodes as well as landmarks. We also study the sensitivity of the system to different parameters including training set, measurement overhead, and size of network.

4.5 Evaluation

In this section, we present the results of evaluating and tuning the parameters of our system. Our system comprises of the profiling module and the Bayesian Network estimator. We use the estimation accuracy of a given metric as the primary parameter to evaluate the performance of our system. We first define this metric, which we refer to as *Accuracy* in the rest of the chapter. We then present the results of the implementation of our system to estimate two metrics: (1) number of hops and (2) latency between any two nodes in the network.

The importance of accurately and efficiently estimating locality of services and computing network distances between different nodes has significantly increased due to proliferation of p2p networks and is also evident from the abundance of latency estimation schemes. Similar to applications' use of network latency to improve the download performance, the number of hops between nodes can be potentially used as a measure for path reliability.

4.5.1 Accuracy

The accuracy metric captures how well the system can rank nodes in terms of their proximity (either using number of hops or latency) to a specific node. Assuming that an algorithm returns a set of k nodes as the closest estimates (we use the term "closest" when dealing with latency or hop number proximity) for a certain node i that we denote by S_k^i. Let the closest node to node i be node j. Thus, the accuracy is 1 if $j \in S_k^i$ and 0 otherwise.

The k-accuracy of an algorithm is computed as the presence of the closest

node j to a certain node i in the set of the k closest nodes as returned by the estimation system. More formally, it is defined as follows:

$$a(i) = \begin{cases} 1, & j \in S_k^i \\ 0, & otherwise \end{cases} \tag{4.2}$$

The accuracy metric measures how well a system ranks nodes in terms of respective distances from a certain node. It is a valid measure, since, in many practical applications, nodes are interested in the closest candidate(s) to them rather than the actual number of hops or the actual latency.

Note that in many practical situations, a node i will query the estimation system for the k closest nodes. Then, node i will perform its own measurements to this set of nodes. The reasoning behind this is that the estimation mechanism is basically providing the k possible candidates of closest nodes and it is up to the node i to perform its own measurements to determine the actual closest among this set. Thus, it is essential for the estimation system to provide the querying node i with its actual closest node among the returned k nodes while maintaining $k << N$. Note that if k is comparable in magnitude to N then the whole purpose of an estimation system is defeated since the node i is launching k additional measurement on the network and the system cannot scale. Note that as k increases, by definition, the accuracy increases. The goal is to achieve as high an accuracy with as low k as possible.

4.5.2 Estimation of Number of Hops

In this section, we compare the accuracy obtained by our proposed system comprised of the profiling and estimation modules, as presented in Section 4.3, to the min-sum algorithm for the hop number estimation between node pairs. We start by choosing a subset of our PlanetLab measurements consisting of 113 nodes and 11 landmarks distributed as follows: 2 in Europe, 2 in Asia, 1 in South America, 4 on the East coast, 1 on the West coast and 1 in the Middle of the US. We also use the simple Bayesian network structure presented in Figure 4.4. We use a modest number of measurements for training the Bayesian network, corresponding to 500 sample random measurements, which adds up to 3.95% of all possible measurements of $N(N-1)$ for $N = 113$, in order to keep the overhead for measurements at a minimum. Note that 113 nodes represents a high percentage of all PlanetLab sites. We want to test the effect of using a heterogenous and diverse set of nodes, thus we start with these 113 nodes and increase the number to include all PlanetLab nodes.

We start by evaluating the different profiling algorithms presented in Section 4.3.2 and compare them to the Min-Sum algorithm. We present the accuracy in Figure 4.6. We observe that for only $K = 2$, the accuracy of the Node Histogram algorithm reaches 81.25%. With this superior performance of the Node Histogram, we pursue to evaluate only the Node Histogram algorithm, and study its behavior as we tune and test against the different parameters of the system.

We also plot the cumulative distribution of the absolute error as presented in Figure 4.7. At a first glance, Figure 4.7 seems to suggest that the min-sum algo-

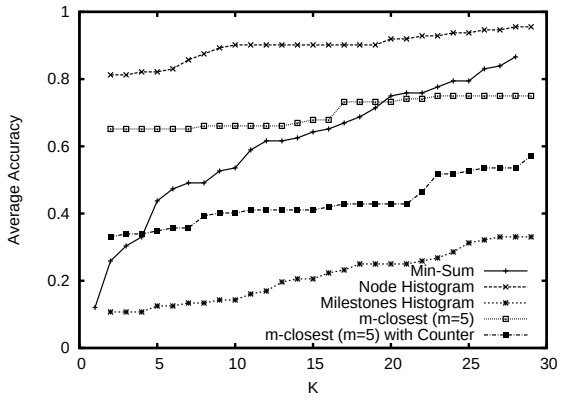

Figure 4.6: Average Accuracy for the Different Profiling Algorithms

rithm offers a better estimate with a lower absolute error then the Node Histogram profiling algorithm with the Simple Bayes Network Estimation module. In fact, the figure shows that only around 20% of the estimation output had an error of less than 10 hops using the Node Histogram profiling algorithm and the Bayesian Network estimation module. At the same time, Min-Sum had around 58% of the estimation output with 10 hops or less in terms of absolute error. However, Figure 4.6 tells a different story which looks counter-intuitive, as the Node Histogram algorithm shows a superior performance. The reason behind this result is due to the fact that the Node Histogram algorithm orders the nodes correctly in terms of their hop number distances from a specific node; a trait captured by the accuracy metric. However, the actual estimations were shifted by a constant, as can be seen in the absolute error. Looking closer into this shift, it averages, in this example, at 15.498.

Next, we evaluate the dependence of the Node Histogram and the Bayesian Network Estimation system on various parameters; namely the number of nodes,

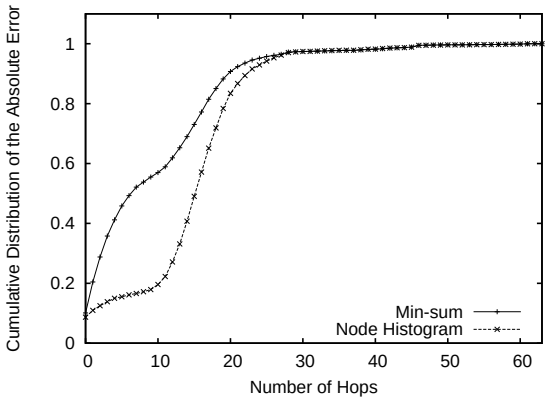

Figure 4.7: Cumulative Distribution of the Absolute Error

the number of landmarks, the use of the two proposed Bayesian network structures, and the amount of training used in the Bayesian module.

We study now the effect of the number of landmarks over the performance. We increase the number of landmarks while maintaining the same number of nodes. The results presented in Figure 4.8 show an interesting behavior. First of all, as we increase the number of landmarks from 11 to 13, we notice a slight improvement in the accuracy for smaller value of k; namely for $k < 5$. However, this improvement does not seem to be consistent for larger values of k. Looking at the cause of this behavior, we observe that, sometimes, an increase in the number of landmarks does not necessarily results in an increase in the number of milestones, thus no increase in the information provided in the histograms of the nodes. However, the distances to the landmarks themselves get incorporated in the histograms of the nodes, since we assume that a landmark is also a milestone, by definition. This additional information (the distances to the newly added landmarks) does not always translate into more information that the Bayesian network classifier

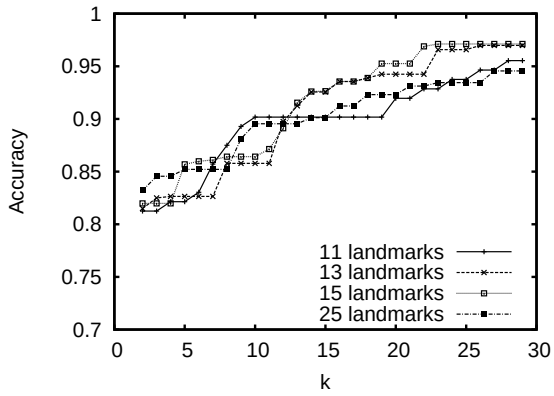

Figure 4.8: Accuracy vs. Number of Landmarks

can use for more accurate results. Later in this section, we will re-visit the idea of increasing the number of landmarks as the number of nodes increases.

When we switch from the simple Baysian Network classifier presented in Figure 4.4 to the Modified Bayesian Network classifier of Figure 4.5, we notice that the accuracy improves. In fact, looking at Figure 4.9, we can see how the Modified Bayesian Network is able to characterize the nodes with a higher accuracy. The reason behind this lies in the fact that the two input histograms represent two different nodes and treating them as separate input variables makes it easier for the Bayesian network classifier to characterize them.

As in any learning-based system, we need to train the Bayesian Network classifier. This training is quite costly in terms of computation resources, and requires end-to-end measurements to be used for training. Thus, for the system to be scalable, we need to keep this training to a minimum versus the dynamics of the network as a whole such as the addition of nodes to the system, since we want a system that does not need to be re-trained every time a node joins or leaves.

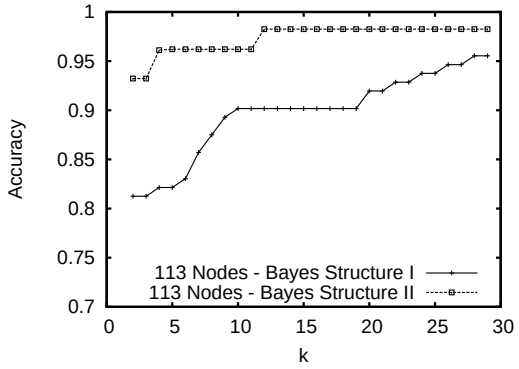

Figure 4.9: Effect of Bayesian Network Structure on Accuracy

In what follows, we study the effect of increasing the network size on the accuracy. We consider two scenarios: at first, we increase the number of nodes and measure the accuracy of the system, then we re-train the system in order to include the newly added nodes and compare the results. Figure 4.10 depicts the accuracy of the tested networks for both scenarios of re-train and no re-train. We observe that re-training indeed does improve the accuracy. However, as we will see next, this is mainly due to the fact that the network that we used for the initial training (113 nodes) was too small to yield information that can be used for other nodes. As the initial network size that is used for training increases further, we can see that we can continue to use the obtained Bayesian Network classifier for larger networks, since the data was enough to capture the specifics of the topology of the network as a whole.

Note that as nodes are added to the network, new milestones might emerge. These can be either routers that never appeared before or routers that had appeared only once before the new addition of nodes, thus did not qualify prior to this addition to become milestones. In this case, we update the histograms of the

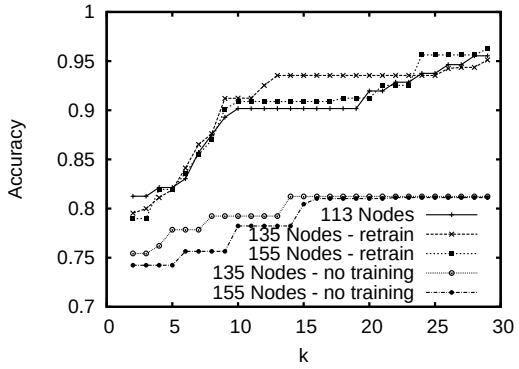

Figure 4.10: Effect of Initial Training Set and Number of Nodes on Accuracy

affected nodes to reflect the new milestones, despite the fact that we may have used the old histograms of these nodes for the training of the Bayesian Network classifier. In fact, we argue in here that this change does not affect the classifier since the signature-like profiles of our system does not contain the identity of the respective nodes and is meant to capture a snapshot of the network characteristics; in the case of the hop number, the characteristics, we are interested in, describe the topology of the network.

In this set of experiments, we start with a subset of the network of 200 nodes, 15 landmarks, and the Modified Bayesian Network classifier. We extract the signatures of the nodes and use 2000 samples for training the Bayesian Network classifier. Note that we increase the number of samples used for training as we increase the number of nodes, however, the percentage is still modest compared to the full N^2 measurements of 40000. Figure 4.11 shows the accuracy of the classifier versus k. Then we increase the number of nodes in the network and re-measure the accuracy of the classifier without re-training the classifier. We show the results in Figure 4.11. We also show the accuracy for the nodes that were

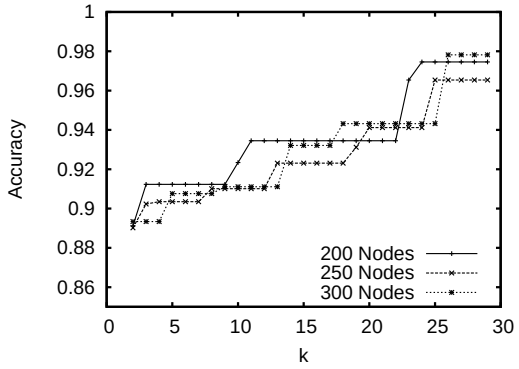

Figure 4.11: Accuracy for the Same Initial Set of 200 Nodes

added in each experiment to the initial network of 200 nodes. By measuring the accuracy of these nodes, we are, actually, testing how well the Bayesian Network classifier is able to generalize rules from the initial observed data (i.e. that of the initial 200 nodes) and use these observations to predict the behavior of other nodes.

We now increase the number of nodes in our set and re-train for every set of experiments. We plot the results of the accuracy in Figure 4.12 showing that the accuracy does not deteriorate as we increase N and with a slight increase in the number of landmarks L the percentage of correct classification depicted in the accuracy remains in the same range showing that the algorithm is able to characterize the topology correctly.

By definition, the Bayesian Network algorithm relies on likelihood maximization leading to the use of iterative approximation techniques [42]. We test the performance of the whole system of profiling and estimation as we change the number of iterations allowed during the training stage of the Bayesian Network estimator. Figure 4.13 shows the accuracy plotted for the different values of k as

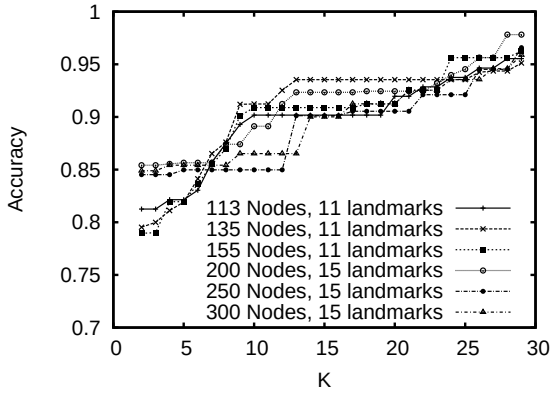

Figure 4.12: Accuracy vs. Number of Nodes in the system

we vary the number of iterations. The network used for this experiment consists of 552 nodes and 22 landmarks. We evaluate the accuracy for 2, 4, 8 and 15 iterations during the training stage. We observe that for this larger set of nodes, a small number of iterations does not provide a high accuracy for a small value of k. In fact, the accuracy for $k = 2$ was below 15% for the 2, 4 and 8 iterations. However, as we increase the number of iterations to 15, the accuracy jumped to around 80%, a major improvement. What happens in here is due to the fact that Bayesian Network maximum likelihood is trying to maximize its function and, just like any other learning mechanism, uses these iterations to refine its parameters. Note that this behavior is not an artifact of our proposed system, but is a normal behavior of any system that relies on Bayesian Networks.

4.5.3 Latency Estimation

When it comes to latency, defining the histogram of nodes requires us to take a closer look into the data as the measurements are not discrete values as was the

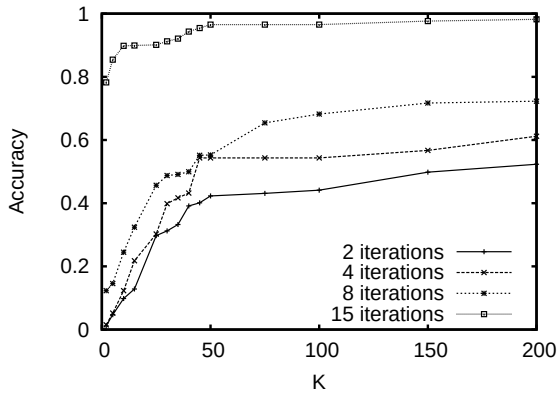

Figure 4.13: Accuracy vs. Number of Iterations During Training

case of the hop numbers. Plotting the distribution of the latencies from nodes to routers and from routers to nodes obtained from our studied system of 113 nodes presented in Section 4.5.2, is presented in Figure 4.14. Figure 4.14 shows that the latencies can be grouped into 3 groups; less than 50 msec or very close by routers, between 50 msec and 500 msec or moderately close routers, and larger than 500 msec or far off routers. For the first range (less than 50 msec), we use a granularity of 1 msec among the different intervals of the histogram. While, between 50 msec and 500 msec, the step becomes 10 msec and over 500 msec, it becomes 50 msec with a maximum of 1200 msec. This results in a vector whose dimension is 111 points. Note that deciding on each group and its granularity is tunable and can be modified if the application requires so.

Comparing the accuracy for the Node Histogram profiling algorithm and the Modified Bayesian Estimation module to Min-Sum and Vivaldi, we observe the results in Figure 4.15. It is, in here, worth noting that the subset of 113 nodes and 11 landmarks that we used was not ideal. In other words, some of the nodes

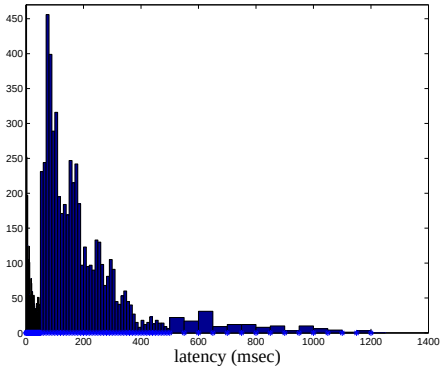

Figure 4.14: Distribution of Latencies

were not responsive most of the time, if not always. Such a situation is typical of PlanetLab as the nodes are often under heavy load and sometimes sporadically disconnected from the network or remain unusable for an extended period of time. For Vivaldi and Min-Sum, we disregard these unreliable nodes and omit them altogether from the analysis. By doing this, we assume that there is a filtering mechanism that analyzes the data before submitting it to Vivaldi or Min-Sum and throws away unreliable data. However, we do not offer the same filtering for the Bayesian Network estimator, since we assume that this system is able to recognize such nodes on its own. This hypothesis is tested in this experiment.

The results shown in Figure 4.15 demonstrate clearly that the Bayesian Network estimator is able to predict distance among nodes and pick closest nodes much more precisely than Vivaldi and Min-Sum. In fact, for a small value of $k = 1$, the Bayesian Network system provides an accuracy of more than 70%, while Vivaldi is at 1.6% and Min-Sum at 13%; a clear advantage of the Bayesian Network system. In addition, for $k = 10$, the Bayesian Network accuracy is

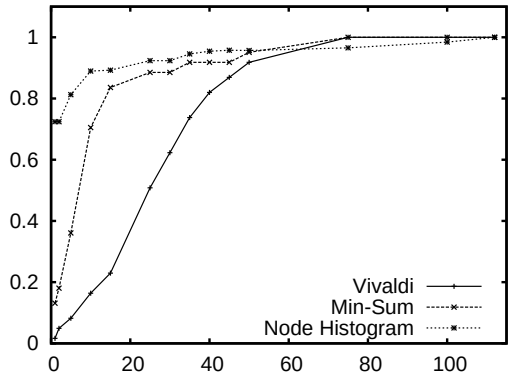

Figure 4.15: Comparison of the Algorithms for Latency Estimation

at 88.9% compared to 16.4% for Vivaldi and 70.5% for Min-Sum. One point, though, worth noting, is that as k goes over 50, this advantage seems to switch and the Bayesian Network system seems to behave the worst among the three algorithms. This is due to the fact of the advantage we gave Min-Sum and Vivaldi by performing the filtering described earlier. However, we argue that, for most practical applications, choosing a high value of k such as 50 or more is not desirable, since the list returned to node i of possible candidates will be too long to provide a useful answer and will force node i to conduct a high number of measurements; thus an over-use of network resources.

When dealing with latency, we notice that the measurements show clear variations with time. Thus, we expand the profiling vectors of nodes to include two flags: the first indicating whether the day of that specific measurement was a week day or a weekend day, the second indicating the time period when the measurement was taken as morning, afternoon, or night. This translates into an expanded profile vector of 113 values corresponding to the 111 vector, presented above, and

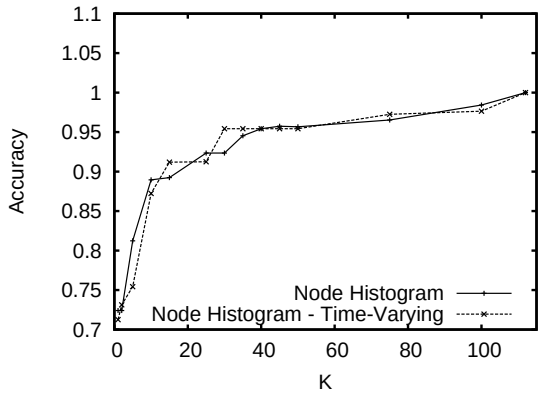

Figure 4.16: Predicting Latencies Over Time

the 2 flags. Since the first flag is mainly binary and the second one can take one of three possible values, we end up with 6 combinations. We repeat the training of the Bayesian Network estimation module using 3000 samples. We start with the same set of 500 samples used in the experiment where time variations were not considered and use six measurements corresponding to the six different combinations. We then test the estimation for the whole network by studying the accuracy. The results in Figure 4.16 show that our Bayesian Network estimator with the help of the Node Histogram can estimate latencies and predict their changes with time, with a high accuracy, a feature that other latency and distance estimators do not consider. Note that the flags can be different and can include further details of the latency changes such as hourly, if the need be.

4.5.4 Scalability and Other Practical Considerations

In this section, we look at the practical consideration of implementing a real system based on the proposed approach. The primary focus is on the computation

and measurement overhead needed when a new node joins a network of N existing nodes. We believe that this kind of scalability is essential for the usage of such prediction and estimation system.

Assuming a network of N initial nodes, a system that has complete information requiring each node to make its own measurements to every other possible node on the network, requires $N(N-1)$ measurements; thus is in the order of $O(N^2)$. However, our system assumes that we have L landmarks where $L << N$ and requires $2NL$ measurements (note that the factor 2 is added since we assume asymmetric links and require each node to make measurements to every landmark and every landmark to make measurements to every node). This measurement overhead is same as the overhead incurred by other landmark-based proximity estimation techniques. Also, our system requires an additional θ random measurements to be used for the initial training data.

In addition, when we consider the addition of nodes to the network, we note that in a system that relies on actual complete measurements, $2N$ measurements are required for every new node: N measurements from the new node to every existing node and N measurements from every existing node to the new node. On the other hand, assuming no re-training, our proposed estimation system requires $2L$ measurements: L measurements from the new node to the landmarks and L measurements from the landmarks to the new node. This considerable decrease in the required measurements makes such an estimation mechanism quite attractive for applications.

The inference mechanisms should only incur incremental overheads when nodes join or leave the system. It is important to only consider profiling mech-

anisms that do not require recomputation of signatures of already existing nodes and complete re-training of the Bayesian network as nodes join or leave the system. Similary, the known properties of different metrics should be leveraged to restrict the dimensionality of the signatures. For instance, the dimensions of profiles for hop count inference was set to 32 based on the diameter of the network. In case of latency, the distribution of the latency between different nodes was used as a guide for marking the bins for Node Histogram algorithm. Similarly some knowledge about the underlying network might be used to tune the value of m in m-Closest algorithm.

4.6 Future Work & Summary

In this chapter, we have presented a learning based estimation approach for network and node metrics that relies on probabilistic techniques, more specifically Bayesian Networks. Our approach creates signature-like profiles for nodes that help presenting and defining their characteristics. We evaluated our approach for two network metrics (number of hops and latency) using data collected from the PlanetLab platform as an initial proof of concept. However, we would like to test out our system on a bigger set of data in order to support the claim of feasibility of its implementation, as part of our future work.

In addition, we would like to study more metrics than the ones presented here (namely hop numbers and latency), such as available bandwidth, uptime, network connectivity and interest communities. Our results are encouraging and motivate us to investigate further features, such re-enforcement learning in the system,

where after being presented with an estimate, nodes can feed-back into the system in case of an error. This will help tune the system as time goes and can contribute to more accurate estimates.

Chapter 5

Conclusion

Peer-to-peer networks break the classical networking architecture of client-server relationship. By eliminating the server, or in general, the central point of authority, reliability in the system becomes a major challenge. In this thesis, we presented our contributions by adding reliable components and features to peer-to-peer networks. The algorithms described attempt to address several issues in peer-to-peer networks including topologies, throughput, and network metrics. The problems that we addressed are of complex nature, requiring us to reach into different areas for possible solutions with satisfactory results.

In typical peer-to-peer networks, end nodes have no guarantee in terms of connectivity. This often translates into the forming of "islands" where sub-networks start to form that are highly connected, however, nodes within a sub-network, typically, are restricted to their immediate neighbors within the same sub-network. In Chapter 2, we address this issue by proposing algorithms that can provide low-diameter connectivity to the participating nodes. By doing so, however, we main-

tain the resilience of the network where an attacker has to invest a huge number of nodes and resources in order to break the network into totally disconnected sub-networks. Our algorithm, Phenix, borrows from the area of social networks, where resilience and connectivity have been studied, defined and proven.

Phenix leverages the strengths of existing unstructured peer-to-peer networks without inheriting their weaknesses and is capable of building topologies of nodes that follow a power-law while being fully distributed requiring no central server, thus, eliminating the possibility of a single point of failure in the system. We presented the design and evaluation of the algorithm and showed through extensive analysis, simulation, and experimental results obtained from an implementation on the PlanetLab testbed that Phenix is robust to network dynamics such as bootstrapping mechanisms, joins/leaves, node failure and large-scale network attacks, while maintaining low overhead when implemented in an experimental network.

From the application-level perspective, end-nodes often are involved in downloading objects or accessing resources. In Chapter 3, we optimize this download process by taking advantage of the availability of multiple serving nodes. Our contributions lie in looking at the problem from a game theory perspective, an essential tool for defining the competitive nature of peer-to-peer nodes. We define the utility of the client nodes and the serving nodes. We show the lack of Nash equilibrium, which has the negative effect of driving the network into oscillation. We then propose a set of strategies for the client and serving nodes designed to maximize their respective utilities, while at the same time offering incentives for nodes to be truthful.

In addition, and in order to provide stable and reliable throughput for client

nodes, we propose an algorithm based on Bayesian theorem [42] that would optimize throughput based on the uncertainties of the network. We show the increase in performance provided by our algorithm when compared to existing peer-to-peer systems. In fact, our algorithm (labeled MSMT) provides reliability facing changes in the networks as well as the dynamic nature of nodes. We achieve such a behavior by building probabilistic profiles for nodes that get updated based on previous observations. Such profiles are efficient, in terms of computational resources, and sufficient when it comes to overall performance.

Since peer-to-peer networks lack a central point of authority by definition, end-nodes have to rely on local information based on their partial view of the network. However, in order to create reliable connections to their peer nodes, it is often quite a complex problem, for nodes, to decide which subset of existing nodes meet their requirements for reliability. Thus, the final contribution of this thesis, as presented in Chapter 4, looked into providing estimates of network metrics in peer-to-peer networks.

Since networks are quite complex, we argue that estimating any metric related to them, such as hop numbers or latency, cannot be carried on with a deterministic approach. Thus, we propose a learning approach for scalable profiling and predicting node metrics. Partial measurements are used to create anonymous signature-like profiles for the participating nodes. These signatures are later used as input to a trained Bayesian network module to estimate the different network properties.

As a proof of concept for our proposed learning based techniques, we designed a system for inferring the number of hops and latency among nodes. Each

node conducts measurements of their performance metrics to known pre-defined landmarks. These measurements are typical of existing estimation techniques and algorithms. However, our contribution to the field was two-fold. First, we used the obtained measurements in order to create an anonymous signature-like profile for each node. We showed that these profiles capture the behavior and characteristics of the nodes and can be used to infer metrics. This, basically, allows us to use these profiles by a Bayesian network estimator in order to provide nodes with estimates of the proximity metrics to other nodes on the network. Our approach for estimation constitutes an additional novel contribution to the field.

In Chapter 4, we presented our proposed system and performance results from real network measurements obtained from the PlanetLab platform. We also studied the sensitivity of the system to different parameters including training sets, measurements overhead, and network dynamics. Though the focus was mainly on proximity metrics, our approach is general enough to be applied to infer other metrics and benefit a wide range of applications.

Last but not least, in proposing all the above mentioned systems and algorithms, we relied heavily on testing our ideas on a realistic environment, in order to ensure their validity. In order to achieve this, we implemented them on the PlanetLab platform [66]. As a result, we dealt with the errors, uncertainties, and failures of PlanetLab, demonstrating that the proposed systems will be able to deal with such realistic environments, and leading us to conclude that we have contributed to improving the reliability of peer-to-peer networks where our algorithms can work and have been studied under realistic conditions.

Chapter 6

My Publications as a Ph.D. Candidate

In here, I list my publications during my years at Columbia University. The list includes as well collaborations with industry researchers that either took place or started during my internships.

6.1 Patents

- Rita H. Wouhaybi and John Vicente. Cognitive Peers. Intel Corporation.
- Puneet Sharma, Rita H. Wouhaybi and Sujata Banerjee. Bayesian Network Metric Estimation. Hewlett Packard Company.

6.2 Journal Papers

- Rita H. Wouhaybi R. H., and Andrew T. Campbell, ”Building Resilient Low-Diameter Peer-to-Peer Topologies,” Under submission to IEEE JSAC.
- Rita H. Wouhaybi, and Andrew T. Campbell, A Minimum-Signaling, Maximum-Throughput Algorithm for Parallel Downloads in P2P Networks, Under submission.
- Jeff Sedayao, John Vicente, Rita H. Wouhaybi, Hong Li, Manish Dave, Sanjay Rugta, and Stacy Purcell, ”PlanetLab and its Applicability to the Proactive Enterprise,” Intel Technical Journal (ITJ), Volume 8, Issue 4, November 2004.
- R. R.-F. Liao, Rita H. Wouhaybi, and Andrew T. Campbell, ”Incentive Engineering in Wireless LAN Based Access Networks”, IEEE Journal of Selected Areas in Communications (JSAC), Special Issue on Recent Advances in Multimedia Wireless, Vol 21, No. 10, December 2003.

6.3 Conference Papers

- Rita H. Wouhaybi, Puneet Sharma, Sujata Banerjee, and Andrew T. Campbell, A Learning Based Approach for Network Properties Inference, Under submission.
- Rita H. Wouhaybi, and Andrew T. Campbell, ”Phenix: Supporting Resilient Low-Diameter Peer-to-Peer Topologies”, IEEE Infocom 2004, Hong Kong,

March 7-11, 2004.

- R. R.-F. Liao, Rita H. Wouhaybi and Andrew T. Campbell. Incentive Engineering in Wireless LAN Based Access Networks, Proc. 10th International Conference on Network Protocols (ICNP 2002), Paris, France, November 12-15, 2002.

6.4 Workshops, Panels and Technical Reports

- Rita H. Wouhaybi, and Andrew T. Campbell, ”Building Resilient Low-Diameter Peer-to-Peer Topologies,” Technical Report, December 2005.

- Panel: Knowledge Plane: Hype or Breakthrough in Managing Internet Networks, David Clark, Simon Crosby, Bob Briscoe, John Strassner, Bob Braden, Dave Lewis, and Rita H. Wouhaybi, MMNS 2004, San Diego, October 3-6, 2004.

- Rita H. Wouhaybi, and Andrew T. Campbell, Keypeer: A Scalable, Resilient Distributed Public-Key System Using Chord, Technical Report.

- Jonathan Clemens, Rita H. Wouhaybi, and Hong Li, ”The Internet as a Network of Fully-Connected Networks,” Adaptive and Resilient Computing Security (ARCS) Workshop, Santa Fe Institute, November 2004.

- Workshop Presentation: Rita H. Wouhaybi, ”Incentive Engineering in Wireless LAN-based Access Networks”, Dagstuhl Seminar on Quality of Ser-

vice in Networks and Distributed Systems, Dagstuhl, Germany, October 2002.

Bibliography

[1] L. A. Adamic. “The small world web,” Proceedings of the 3rd European Conf. On Digital Libraries, vol. 1696 of Lecture notes in Computer Science, Springer, 1999, pp. 443-452.

[2] L. A. Adamic, R. M. Lukose, and B. A. Huberman, “Local search in unstructured networks,” Review chapter to appear in Handbook of Graphs and Networks: From the Genome to the Internet, S. Bornholdt and H.G. Schuster (eds.), Wiley-VCH, Berlin, 2003.

[3] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica, “Towards a more functional and secure network infrastructure,” UCB Technical Report No. UCB/CSD-03-1242.

[4] M. Adler, R. Kumar, K. Ross, D. Rubenstein, D. Turner, D. Yao. “Optimal Peer Selection in a Free-Market Peer-Resource Economy,” Second Workshop on the Economics of Peer-to-Peer Systems (P2P ECON), Cambridge, Massachusetts, June 2004.

[5] L. A. N. Amaral, A. Scala, M. Barthelemy, and M. Stanley, "Classes of small-world networks," Proceedings of the National Academy of Sciences, vol. 97, no. 21, October 2000.

[6] D. G. Andersen, "Mayday: distributed filtering for Internet services," Proceedings of 4th Usenix Symposium on Internet Technologies and Systems, Seattle, WA, 2003.

[7] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), 2001.

[8] S. Androutsellis-Theotokis and D. Spinellis. "A survey of peer-to-peer content distribution technologies," ACM Computing Surveys, 36(4):335371, December 2004.

[9] A-L Barabsi, and R. Albert, "Emergence of scaling in random networks," Science, 286:509, 1999.

[10] A-L Barabsi, and R. Albert, "Statistical mechanics of complex networks," Center for Self-Organizing Networks, University of Notre Dame, Notre Dame, Indiana.

[11] Bayes Net Toolbox (BNT). http://bnt.sourceforge.net/.

[12] D. S. Bernsteing, Z. Feng, B. N. Levine, and S. Zilberstein. "Adaptive Peer Selection," Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, California, February 2003.

[13] BitTorrent. http://www.bittorrent.com/.

[14] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. “Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs,” ACM SIGMETRICS 2000.

[15] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. “Informed Content Delivery Across Adaptive Overlay Networks,” ACM SIGCOMM 2002.

[16] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making Gnutella-like P2P systems scalable,” Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (ACM Sigcomm 2003), pp. 407-418, 2003.

[17] Y. Chen, R. H. Katz and J. D. Kubiatowicz. “Dynamic Replica Placement for Scalable Content Delivery.” In Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002.

[18] N. Christin, A. S. Weigend, J. Chuang, “Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks,” ACM Conference on Electronic Commerce 2005: 68-77.

[19] I. Clarke, O. Sandberg, and B. Wiley. “Freenet: A distributed anonymous information storage and /etrieval system.” In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, June 2000.

[20] E. Cohen and S. Shenker. “Replication Strategies in Unstructured Peer-to-Peer Networks.” Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (ACM Sigcomm 2002), pp. 61-72, 2002.

[21] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, J. Chase, “Correlating instrumentation data to system states: A building block for automated diagnosis and control,” Operating Systems Design and Implementation (OSDI), San Francisco, December 2004.

[22] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. “Vivaldi: A Decentralized Network Coordinate System,” In the Proceedings of the ACM SIGCOMM ’04 Conference, Portland, Oregon, August 2004.

[23] L. Dairaine, L. Lancerica, and J. Lacan. “Enhancing Peer to Peer Parallel Data Access with PeerFecT,” Networked Group Communication 2003: 254-261.

[24] R. Diestel, *Graph Theory.* Springer 2000.

[25] R. Dornfest, “Email: A P2P Enabler?” O’Reilly OpenP2P, http://www.oreillynet.com/pub/wlg/42.

[26] eDonkey. http://www.edonkey2000.com/.

[27] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, “Efficient broadcast in structured P2P networks,” 2nd International Workshop on Peer-to-Peer Systems (IPTPS ’03), Berkeley, CA, February 2003.

[28] eMule. http://www.emule.org/.

[29] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," Proceedings of the 1999 conference on Applications, technologies, architectures, and protocols for computer communications (ACM Sigcomm 1999), pp. 251-262, 1999.

[30] R. Fonseca, P. Sharma, S. Banerjee, S.J. Lee, S. Basu, "Distributed Querying of Internet Distance Information," IEEE Global Internet Symposium (in conjunction with InfoCom 2005), Miami, Florida March 2005.

[31] P. Francis, S. Jamin, C. Jin,, D. Raz, Y. Shavitt, L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," IEEE/ACM Trans. on Networking, Oct. 2001.

[32] A. C. Fuqua, T. Ngan, and D. S. Wallach. "Economic Behavior of Peer-to-Peer Storage Networks," Workshop on Economics of Peer-to-Peer Systems (Berkeley, California), June 2003.

[33] T.J. Giuli, P. Maniatis, M. Baker, D. S. H. Rosenthal, and M. Roussopoulos, "Attrition Defenses for a Peer-to-Peer Digital Preservation System." Proceedings of the USENIX Annual Technical Conference, Anaheim, CA, USA, April 2005.

[34] C. Gkantsidis, M. Ammar, and E. Zegura. "On the Effect of Large-Scale Deployment of Parallel Downloading," IEEE Workshop on Internet Applications (WIAPP'03), 2003.

[35] Gnucleus. The Gnutella Web Caching System. http://gnucleus.sourceforge.net/.

[36] Gnutella Development Group. http://groups.yahoo.com/group/ gnutella-dev/.

[37] The Gnutella RFC. http://rfc-gnutella.sourceforge.net/.

[38] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J Zahorjan. "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19), Bolton Landing, NY, USA, October 2003.

[39] K. P. Gummadi, S. Saroiu, S. D. Gribble., "King: Estimating latency between arbitrary Internet end hosts," Proceedings of SIGCOMM IMW 2002, November 2002, Marseille, France.

[40] M. Gupta, P. Judge, and M. Ammar. "A Reputation System for Peer-to-Peer Networks." In Proceedings of the NOSSDAV'03 Conference, Monterey, CA, June 1-3 2003.

[41] G. Hardin. "The Tragedy of the Commons," Science 162, 1243-1248 (1968).

[42] G. R. Iversen, *Bayesian Statistical Inference.* Sage University Papers Series, Quantitative Applications in the Social Sciences ; No. 07-043. Beverly Hills, Calif. Sage Publications, Inc., 1984.

[43] M. Jovanovic, Modeling Large-scale Peer-to-Peer Networks and a Case Study of Gnutella. Master's thesis, University of Cincinnati, 2001.

[44] JTella. http://jtella.sourceforge.net/

[45] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. "The Eigentrust Algorithm for Reputation Management in p2p Networks." In Proceedings of the twelfth international conference on World Wide Web, pages 640-651. ACM Press, 2003.

[46] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: secure overlay services," Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (ACM Sigcomm 2002), pp. 61-72, 2002.

[47] B. J. Kim, C. N. Yoon, S. K. Han, and H. Jeong "Path finding strategies in scale-free networks," Phys. Rev. E., 65:027103, 2002.

[48] S. G. M. Koo, C. Rosenberg, and D. Xu. "Analysis of Parallel Downloading for Large File Distribution," Proceedings of IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2003), San Juan, PR, May 2003.

[49] P. L. Krapivsky, G. J. Rodgers, and S. Redner, "Degree distributions of growing random networks," Phys. Rev. Lett., 86:5401, 2001.

[50] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. "OceanStore: An Architecture for Global-Scale Persistent Storage," Proceedings of the Ninth international Conference on Architectural Support for

Programming Languages and Operating Systems (ASPLOS 2000), November 2000.

[51] J. Lacan, L. Lancérica, and L. Dairaine. “Speedup of Data Access Using Error Correcting Codes in Peer-to-Peer Networks,” Proceedings of IEEE International Symposium on Information Theory (ISIT-2003), p. 471, Yokohama, Japan, June 2003

[52] J. Lacan, L. Lancérica, and L. Dairaine. “When FEC Speed up Data Access in P2P Networks,” IDMS/PROMS 2002: 26-36.

[53] Lime Wire LLC. LimeWire. http://www.limewire.com/.

[54] Q. Lv, S. Ratnasamy and S. Shenker. “Can Heterogeneity Make Gnutella Scalable?” In Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002.

[55] Matlab. http://www.mathworks.com/products/ matlab/.

[56] P. Maymounkov and D. Mazières. “Kademlia: A Peer-to-peer Information System Based on the XOR Metric,” Proceedings of 1st International Workshop on Peer-to-peer Systems, Cambridge, Massachusetts, March 2002.

[57] Merriam-Webster online. http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=resilience

[58] Napster Inc. (Formerly Roxio, Inc.). Napster. http://www.napster.com/.

[59] T. S. E. Ng, H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches", Proceedings of IEEE INFOCOM'02, New York, June 2002.

[60] A. Oram (Ed), *Peer-to-Peer:Harnessing the Power of Disruptive Technologies.* Oreilly 2001.

[61] Overnet. http://www.overnet.com/.

[62] V. Padmanabhan, L. Qiu, and H. Wang, "Server-based Inference of Internet Link Lossiness," In Proceedings of IEEE INFOCOM'03, San Francisco, CA, USA, April 2003.

[63] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter P2P networks," IEEE Journal on Selected Areas in Communications, Vol. 21, pp. 995-1002, Aug. 2003.

[64] G. Pandurangan, P. Raghavan, and E. Upfal, "Building P2P networks with good topological properties," Technical Report, 2001.

[65] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, S. Bhatti, "Lighthouses for Scalable Distributed Location," IPTPS '03.

[66] PlanetLab. http://www.planet-lab.org/

[67] D. Qiu and R. Srikant. "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," Proceedings of ACM SIGCOMM, Portland, Oregon, September 2004.

[68] Query Routing for the Gnutella Network, Version 1.0, http://www.limewire.com/developer/query_routing/keyword %20routing.htm

[69] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (ACM Sigcomm 2001), pp. 161-172, 2001.

[70] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," Proceedings of Infocom 2002.

[71] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. "Maintenance-Free Global Data Storage," IEEE Internet Computing, pp. 40-49, 2001.

[72] J. Ritter, "Why gnutella can't scale. no, really," http://www.darkridge.com/ jpr5/doc/gnutella.html, 2001.

[73] P. Rodriguez, A. Kirpal, and E. W. Biersack. "Parallel-Access for Mirror Sites in the Internet," IEEE Infocom 2000, March 2000.

[74] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems." In proceedings Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms. Heidelberg, Germany, November 12-16, 2001. Lecture Notes in Computer Science, Volume 2218, Jan 2001, Page 329.

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

[76] Scriptroute. http://www.cs.washington.edu/ research/networking/scriptroute/.

[77] S. Sen, and J. Wang, "Analyzing peer-to-peer traffic across large networks," Proceedings of the second ACM SIGCOMM Workshop on Internet measurement workshop, Marseille, France, pp. 137-150, 2002.

[78] Sharman Networks LTD. KaZaA Media Desktop. http://www.kazaa.com/.

[79] S. Srinivasan and E. Zegura, "M-coop:A Scalable Infrastructure for Network Measurement," Third IEEE Workshop on Internet Applications (WIAPP '03).

[80] S. Srinivasan, and E. Zegura, "Network Measurement as a Cooperative Enterprise," IPTPS '02.

[81] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (ACM Sigcomm 2001), pp. 149-160, 2001.

[82] StreamCast. Morpheus. http://www.morpheus.com/.

[83] T. Sundsted, "The practice of peer-to-peer computing: Trust and security in peer-to-peer networks," IBM DeveloperWorks, http://www-128.ibm.com/developerworks/java/library/j-p2ptrust/.

[84] L. Tang, and M. Crovella, "Virtual Landmarks for the Internet," Internet Measurement Conference Oct 2003.

[85] Ultrapeers: Another Step Towards Gnutella Scalability. http://groups.yahoo.com/group/the_gdf/files/Proposals/Ultrapeer/ Ultrapeers_1.0.htm

[86] M. Waldman, A. D. Rubin, and L. F. Cranor, "Publius: A robust, tamper-evident, censorship-resistant web publishing system," In Proceedings of the 9th USENIX Security Symposium, August 2000.

[87] D. J. Watts, and S. H. Strogatz, "Collective dynamics of 'small-world' networks," Nature 393, 440-442, 1998.

[88] H. Weatherspoon, and J. Kubiatowicz. "Erasure Coding vs. Replication: A Quantitative Comparison," Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002.

[89] B. Wong, A. Slivkins, and E.G. Sirer, "Meridian: A Lightweight Network Location Service Without Virtual Corrdinates," In the Proceedings of the ACM SIGCOMM '05 Conference, Philadelphia, Pennsylvania, August 2005.

[90] R. H. Wouhaybi, and A. T. Campbell, “Phenix: Supporting Resilient Low-Diameter Peer-to-Peer Topologies,” IEEE INFOCOM’2004, Hong Kong, China, March 7-11, 2004.

[91] L. Xiong and L. Liu. “Building Trust in Decentralized Peer-to-Peer Communities.” In Proceedings of the International Conference on Electronic Commerce Research, October 2002.

[92] Z. Xu, P. Sharma, S.J. Lee and S. Banerjee, “Netvigator: Scalable Network Proximity Estimation,” HP Labs Technical Report, HPL-2004-28.

[93] X. Yang, and G. de Veciana. “Service Capacity of Peer to Peer Networks,” IEEE Infocom 2004, Hong Kong, China, March 2004.

[94] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz, “Tapestry: A Resilient Global-scale Overlay for Service Deployment,” IEEE Journal on Selected Areas in Communications, Vol. 22, pp. 41-53, Jan 2004.