2014

Peer-to-Peer Based Trading and File Distribution for Cloud Computing

Ping Yi
University of Kentucky, ypnmail@gmail.com

Peer-to-Peer based Trading and File Distribution for Cloud Computing

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Ping Yi

Lexington, Kentucky

Director: Zongming Fei, Ph.D,

Associate Professor of Computer Science

Lexington, Kentucky

2014

ABSTRACT OF DISSERTATION

Peer-to-Peer based Trading and File Distribution for Cloud Computing

In this dissertation we take a peer-to-peer approach to deal with two specific issues, fair trading and file distribution, arisen from data management for cloud computing.

In mobile cloud computing environment cloud providers may collaborate with each other and essentially organize some dedicated resources as a peer to peer sharing system. One well-known problem in such peer to peer systems with exchange of resources is free riding. Providing incentives for peers to contribute to the system is an important issue in peer to peer systems. We design a reputation-based fair trading mechanism that favors peers with higher reputation. Based on the definition of the reputation used in the system, we derive a fair trading policy. We evaluate the performance of reputation-based trading mechanisms and highlight the scenarios in which they can make a difference.

Distribution of data to the resources within a cloud or to different collaborating clouds efficiently is another issue in cloud computing. The delivery efficiency is dependent on the characteristics of the network links available among these network nodes and the mechanism that takes advantage of them. Our study is based on the Global Environment for Network Innovations (GENI), a testbed for researchers to build a virtual laboratory at scale to explore future Internets.

Our study consists of two parts. First, we characterize the links in the GENI network. Even though GENI has been used in many research and education projects, there is no systematic study about what we can expect from the GENI testbeds from a performance perspective. The goal is to characterize the links of the GENI networks and provide guidance for GENI experiments.

Second, we propose a peer to peer approach to file distribution for cloud computing. We develop a mechanism that uses multiple delivery trees as the distribution structure, which takes into consideration the measured performance information in the GENI network. Files are divided into chunks to improve parallelism among different delivery trees. With a strict scheduling mechanism for each chunk, we can reduce the overall time for getting the file to all relevant nodes. We evaluate the proposed mechanism and show that our mechanism can significantly reduce the overall delivery time.

KEYWORDS: Peer-to-Peer, Trading, File Distribution, GENI, Cloud Computing

Ping Yi

May 16, 2014

Peer-to-Peer based Trading and File Distribution for Cloud Computing

By

Ping Yi

ACKNOWLEDGEMENTS

It is a pleasure to thank those who made this thesis possible. I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family.

First of all, I would like to express my deepest gratitude to my advisor, Dr. Zongming Fei, for his excellent guidance, patience, caring, and leading me to do research in p2p networks applied in cloud computing. Dr. Fei has been a great mentor on every account, and his broad knowledge and constructive suggestions to this dissertation are sincerely appreciated.

I would like to thank the Director of Graduate Studies in the Department of Computer Science, Dr. Miroslaw Truszczynski and other faculty members of my Advisory Committee: Dr. Mukesh Singhal (Department of Computer Science), Dr. Dakshnamoorthy Manivannan (Department of Computer Science), Dr. Zhi Chen (Department of Electrical and Computer Engineering) and Dr. Yuan Liao (Department of Electrical and Computer Engineering) for their helpful comments on my dissertation.

Also, I would like to thank Dr. Hui Lu for her helpful comments on my dissertation.

Finally I would like to thank my family members.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data management is an important task in cloud computing. We target two aspects of the data management problem. One is the data storage and the other is the data delivery. For data storage, we specifically target the mobile cloud computing environment, in which data are typically stored close to their mobile users. To increase the coverage of geographical locations, different cloud providers may share and trade their computing and storage resources. They collaborate with each other and essentially organize some dedicated resources as a peer to peer sharing system. One well-known problem in such peer to peer systems with exchange of resources is free riding, which can lead to performance deterioration and even collapse of the whole system, if the total resource contributed by peers is less than that used by them. Providing incentives for peers to contribute to the system is an important issue in these systems. It can be complicated to design an incentive mechanism that is considered to be fair by peers. We design a reputation-based fair trading mechanism and develop a fair trading policy to provide incentives for collaborating clouds to contribute to the system.

Distribution of data to the resources within a cloud or to different collaborating clouds efficiently is another issue in cloud computing. The delivery efficiency is dependent on the characteristics of the network links available among these network nodes and the mechanism that takes advantage of them. Our study is based on the Global Environment for Network Innovations (GENI) [38, 39, 40], a testbed for researchers to build a virtual laboratory at scale to explore future Internets. Our study consists of two parts. First, we characterize the links in the GENI network. Even though GENI has been used in many research and education projects, there is no systematic study about what we can expect from the GENI testbeds from a performance perspective. The goal is to characterize the links of the GENI networks and provide guidance for GENI experiments. The information collected can be helpful for designing GENI experiments in selecting where resources should be reserved. Second, we propose a peer to peer approach to file distribution for cloud computing. Instead of delivering a file to all concerned nodes using the traditional client-server model, we develop a mechanism that uses multiple delivery trees as the distribution structure, which takes into consideration the measured performance information in the GENI network. Files are divided into chunks to improve parallelism among different delivery trees. With a strict scheduling mechanism for each chunk, we can reduce the overall time for getting the file to all relevant nodes.

1.1 Reputation and Incentives for Peer-to-Peer Networks

A peer to peer system relies on the cooperation of peers to accomplish tasks. It can distribute load to peers and get rid of the bottleneck typically existing at the server of a client-server system. We have seen many applications of the peer to peer paradigm, such as file sharing and multimedia streaming [1]. Instead of getting large media files from a central server, peers can download them from other peers and therefore achieve much better performance.

More recently, the peer to peer approach has been adopted for implementing efficient backup systems [2, 3, 4, 5, 6]. As the size of hard disk on a PC becomes larger, we have more space in the local disk than we need. While having multiple copies of the same file on the same disk does not improve the reliability of the data, we can use spare space to trade with other users and use them as backup for each other. There are two benefits with this peer to peer backup system. One is that with the peer to peer backup system, we can have multiple copies of important files on different machines of different clouds in different locations. If privacy is a concern, we can always encrypt the file stored at remote locations. In case of the machine crash or disk failure, we will not have a local copy of those important files. However, we still can obtain the remote backup copy from other peers. The replication of objects is also beneficial in another scenario. Consider that the data stored on the disk needs to be accessed remotely from a laptop when the user is on a trip. Because the local disk may not be one hundred percent on-line, it is possible that the user cannot get the data when needed. With the replication, the user can try to find the data at replicated peers. If any of the peers is on-line, the user will be able to get the data. This can greatly increase the availability of the useful data.

One well-known problem with peer to peer systems is free riding [7]. If peers just consume the resources of the system without contributing enough to the system, the overall resources of the whole system will gradually diminish. It will cause the deterioration of the performance of the system and can even lead to the collapse of the whole system. For example, in a peer to peer backup system, if the space contributed by a peer is less than the space used by it, the total available space of the system will decrease over time. Finally, it may become hard for a peer to find space to backup its files. The negative effect is that the peer will be less willing to contribute space. The vicious cycle will cause the system to collapse. Therefore, it is very important to design some mechanisms to provide incentives for peers to contribute in such systems. The goal is to maintain a healthy cooperation among peers so that everybody has enough space to use as backup when needed.

One common and simple way to design such an incentive mechanism is to require each peer to contribute at least as much storage space as it will use. While it is intuitively simple and in most cases effective, it does not consider quality aspect of the storage space. For example, one peer may be on-line almost all the time and provides very fast upload/download speed because of a high-speed Internet connection, while the other peer is connected to the Internet on and off with a slow connection. The quality of storage space on the first peer is considered higher than that on the second peer. It is unfair to the first peer if it is required to contribute the same amount of storage as the second peer.

In this dissertation, we propose a reputation-based fair trading mechanism for peer to peer backup systems. The quality of the storage space of a peer is observed by other peers and is measured as its reputation. We design a framework for peers to derive reputation about other peers through either direct observations of its own or indirect recommendations from others. When two peers want to exchange storage space for backup, the amount of storage space traded will be based on their reputation. Instead of trading equal amount of storage space between peers, the reputation-based trading mechanism favors peers with higher reputation. Based on the definition of the reputation used in the system, we derive a fair trading policy. It is interesting to notice that the optimal solution is not the same as the intuitive policy that requires a peer to get the storage space proportional to its reputation.

1.2 Characterizing the GENI Networks

The Global Environment for Network Innovations (GENI) is a project sponsored by National Science Foundation (NSF) with the aim to provide a collaborative environment to build a virtual laboratory for exploring future internets at scale [38, 40]. It has been transitioning from the development phase to the stage in which we pay more attention to deployment and adoption to provide support for research and educational experiments. It has attracted many universities and industrial partners to contribute their efforts towards developing a global federated network testbed. An experimenter can reserve both computing resources (such as PCs, virtual machines (VMs)), and networking resources (such as ION links, OpenFlow switches, VLANs, and GRE tunnels). They have full control of their slice and can install customized OS images on machines in the experiment. Recently, the GEMINI [57] and GIMI [58] projects developed instrumentation and measurement support, making it easy for experimenters to collect performance data about GENI experiments.

GENI consists of many aggregates, each of which manages a set of resources [41]. Typically, a GENI aggregate is administrated and controlled by an institution which can impose its own policies about the allocation of the resources. As more GENI racks are deployed on university campuses across the United States, GENI has grown to have tens of aggregates with resources available for network experiments [42]. The progress of GENI helps encourage researchers to use GENI as a testing environment for their research projects, as evidenced by new GENI projects on shakedown experiments [59].

The first step to design a GENI experiment is to set up a topology. There are a lot of choices when determining what and where GENI resources should be reserved for the experiment. An experiment can reserve all resources from a single aggregate. Alternatively, an experiment can use resources distributed over a wide geographical area. One decision that needs to be made in designing a GENI experiment is whether to use resources from one aggregate or from multiple aggregates. It depends on the types of experiments to be performed. Some experiments such as multimedia applications may have a strict end-to-end delay requirement that cannot be satisfied by nodes distributed over a wide area. They may have to get resources from a single aggregate. On the other hand, there are experiments that need to test the behavior of protocols on how they react to the cross traffic from the real world. It may be preferable to have resources from multiple aggregates. There is also a question about which aggregates to choose to put the experimental nodes.

While experimenters have a good understanding of what they want for their experiments, it is not so clear what they can get from the GENI networks. To make this decision, we need to have a good understanding of underlying networks. We focus on available bandwidth and measured latency of links in the network. For example, if an experimenter sets up a topology including nodes from Utah emulab, Kentucky emulab, and GENI racks from Wisconsin and Northwestern, what will be bandwidth and latency between two nodes from two different aggregates? They can be tested *after* the experiment has been set up, either by using some simple utilities (ping, iperf [61], etc), or with the help of instrumentation and measurement tools such as GEMINI [57] and GIMI [58]. It may take some time to finish the task. Further, we may want to know more about the bandwidth and latency. Do they change a lot over time? What kind of distribution do they follow? Are they aggregate dependent? What exactly can we get from links within an aggregate versus from multiple aggregates? How different are the bandwidth and latency of links within an aggregate versus from multiple aggregates? We collect and analyze the measurement data and try to answer these questions. We may want to have these questions answered or at least have a rough idea about them before we make a decision on how to set up an experiment.

We understand that the distinction between single aggregate and multiple aggregates is not absolute. In a single aggregate experiment, the links generally have lower latency and higher bandwidth. To make them suitable for an experiment that needs more realistic topology that has a wide variety of delays and bandwidths, we can add delay nodes in the middle of the topology to do traffic shaping, increasing the delay or reducing the bandwidth, or both. This adds an element of simulations/emulations, instead of pure experimentations. The resulting topology will have some characteristics of multi-aggregate experiments. On the flip side of the coin are experiments using multiple aggregates. For large network experiments, the number of nodes usually exceeds the number of aggregates available. We have to allocate multiple nodes within an aggregate. Thus, even in a multi-aggregate experiment, we may still have links within an aggregate. In either case, we need to have an idea about delays and bandwidth of both single-aggregate links and cross-aggregate links.

Performance measurement has been done on campus, regional and national backbone networks [60]. However, we have not seen a systematic study about the performance of GENI networks, especially from an experimenter's perspective. The goal of this study is to characterize the links of GENI networks and provide some insights for GENI experimenters into what they can expect to get from GENI networks. The information will be helpful to the decision making process for reserving resources from appropriate places to satisfy the need of experiments.

We present our study on performance of GENI networks and the tradeoff between single aggregate and multiple aggregates in the design of GENI experiments from the performance perspective. We will analyze how the links behave differently over a period of time. The data collected will shed some light on the design process for choosing where the nodes in the experiment should be located.

1.3 File Distribution in the Cloud

Cloud computing can provide infrastructure, platform and software as a service to meet the needs of users on demand. It has many attractive features, such as no up-front investment, no need to administrator a large number of machines, no worry about the upgrade/recycle outdated equipment, etc. We have witnessed that more users and enterprises are moving to the cloud to meet their computing needs and use the services provided by the cloud.

One category of applications especially suitable for the cloud environment is the parallel processing of big data, such as those using MapReduce. Due to the virtualization, an application can request variable numbers of virtual machines (VMs) to process the data file in parallel, depending on the performance goal of the users. We can increase the number of VMs requested if we want to get the results faster. On the other hand, we may reduce the cost by requesting less VMs if we can afford to get the results later. One of the tasks in these applications is to distribute the big data file to all the VMs processing them. It can become the bottleneck for reducing the turnaround time. In the GENI environment, we also see quite often that we need to install a certain package on all the experimental machines in a slice. We have to distribute the file to all the VMs involved.

The traditional method uses a client server model to send the file from the origin machine to all VMs in turn. The overall time can grow linearly with the number of VMs that need the file. In this dissertation, we adopt a peer to peer approach to deal with this problem. While the pure peer to peer method can reduce the delivery time from linear to logarithmic, we propose an approach that divides the origin file into blocks and can further reduce the the delivery time to a constant, no matter how many receivers there are in the system. This can significantly improve the overall performance for the applications that process big data file in the cloud computing environment.

1.4 Organization of this Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we review the related work regarding the topics of our research. In Chapter 3, we propose a reputation-based fair trading mechanism for peer-to-peer backup systems. Our approach provides incentives to encourage fair trading and improves system performance. In Chapter 4, we characterize the GENI networks and analyze the tradeoff between using single aggregate and multiple aggregates in designing GENI experiments. The analysis of data collected will shed light on the decision process for designing GENI experiments. In Chapter 5, we propse a block-based peer-to-peer file distribution in the cloud. We design a scheduling algorithm that can significantly reduce the distribution time in the cloud. Finally, we conclude the dissertation and outline our future work in Chapter 6.

Chapter 2

Related Work

In this chapter, we will discuss related work on peer-to-peer backup systems, reputations and incentives, GENI networks, and peer-to-peer based file distribution.

2.1 Peer-to-Peer Backup Systems

Peer-to-peer communication is an alternative approach to the traditional client-server model. Peer to peer networks can be divided into two categories, either unstructured or structured. Nodes in an unstructured peer-to-peer network pick their neighbors randomly. Examples of unstructured peer-to-peer networks include Napster [8], Gnutella [10], and KaZaa [9]. Query of data in an unstructured peer-to-peer networks is done through flooding. Typically, the request will be sent to a certain number (e.g., 7) of neighbors, which in turn forward the request to their own neighbors. The process will continue until the pre-set TTL expires. The node having the data will send a response back to the original node making the request. One potential problem with an unstructured peer-to-peer network is that it is possible that data cannot be found even if there is a copy in the network. Another drawback is that flooding may create a lot of traffic and it is not easy to determine what an appropriate TTL value should be set to.

Nodes in a structured peer-to-peer network are organized into a certain kind of structure, such as a ring, a mesh, butterfly, etc. If an object exists in a structured peer-to-peer network, it can always be found by a pre-determined number of steps. Performance measures used to distinguish different structured peer-to-peer networks include the number of neighbors for each node and the number of search steps to find the data. For a structured peer-to-peer network with n nodes, the number of neighbors can be $\sqrt{n}$, $\log(n)$, etc, and the number of search steps can be $\sqrt{n}$, $\log(n)$, or some constant number. Examples of structured peer-to-peer networks include Chord[12], CAN[15], Tapestry[16], Pastry[13], Koorde[17], Skipnet[18], EpiChord[19], and OpenDHT[20].

Peer-to-peer networks have been used as a structure to organize backup systems. Each peer contributes some storage space to the system. In exchange, it can use storage space from other peers to backup its data.

Here we present several representative projects on peer-to-peer backup systems, including pStore[2], PeerStore[3], Pastiche[4], and Cooperative Internet Backup Scheme[5].

(1) pStore

pStore[2] is a secure distributed backup system that makes use of unused hard drive space in PCs and supports CVS-like versioning system. pStores is built on a distributed hash table (DHT), which provides efficient retrieval of the backed up data. DHT is also used for retrieving the metadata that can be used to locate the backup data in the peer-to-peer network.

When a user wants to insert a file into the peer-to-peer backup system, pStore computes a namespace-filename identifier that is specific for that file and that user. It allocates a namespace for each user based on the private key. The namespace-filename identifier is a hash value of user's private key, pStore pathname, filename, and salt. So even if two users have the same file name, the resulting identifiers will be different. The file is encrypted and divided into blocks that are digitally signed. The meta-data indicating how to reassemble the blocks is signed and distributed together to the peer-to-peer backup system. A user can retrieve a file by providing the filename and the version. First the meta-data is retrieved and then all the blocks belonging to the file are retrieved and assembled.

The three primary design goals of pStore are reliability, security and resource efficiency. Reliability is provided through replication. Every block is replicated and have multiple copies stored on several different nodes in different locations. If some nodes become unavailable, they can still be retrieved from other nodes. Security is ensured by using encryption and content hashes. Data can only be decrypted by its owner, who can also verify the integrity by examining the digital signature. Only the owner can delete the data remotely. pStore achieves the resource efficiency by sharing stored data and exchanging data only when necessary.

(2) PeerStore

PeerStore[3] is a peer-to-peer backup system that decouples the meta-data management from the actual backup data storage. The meta-data layer is based on a DHT, which provides fast searching and duplicate detection. The actual data storage is based on a unstructured peer to peer system. It uses a symmetric trading scheme, requiring that a peer that wants to backup its file must also be willing to provide storage space for the peer storing its data. This decoupling design can reduce the management cost caused by maintaining the structure of the DHT. When a node joins or leaves the network, only the metadata needs to be migrated to other nodes. Actual data blocks do not need to be copied because of duplication. Since the metadata is relatively small, the maintenance cost is therefore reduced.

There are other benefits with this design. In addition of using different kinds of peer-to-peer systems to implement these two layers, the metadata layer and the storage layer can impose different strategies. For example, in PeerStore, the metadata layer adopts an aggressive strategy to keep the information up-to-date. At the same time, the storage layer introduces a fair trading policy that requires the storage space traded between partners are close.

Similar to pStore, files are also divided into blocks and the system will generate a unique identifier for each block. However, PeerStore uses a different method to create the ID for a block. It applies a cryptographic hash function twice to the content of the block. The hash value is also used as the symmetric key to encrypt the block. Only the owner of the block knows the hash value, so nobody else can decrypt the content.

(3) Pastiche

Pastiche[4] is a peer-to-peer backup system built on top of three enabling technologies: 1) Pastry[13], a structured peer-to-peer network that provides scalable, efficient routing for object location; 2) content-based indexing, a mechanism for finding common data among different files; 3) convergent encryption, an encryption technique allowing nodes to use the same encrypted representation for common data without sharing keys.

In Pastiche, when a node wants to make a backup of its data, it will first have to find a set of buddies. A buddy is defined as a node that shares a significant amount of data. In Pastiche, each node should maintain five buddies. The goal of using buddies rather than other nodes for backup is to reduce the storage space because the shared content only needs to be stored as one copy at each buddy using convergent encryption. A node can restore its data from any of its buddies whenever it wants.

Similar to other peer-to-peer backup systems, files in Pastiche are divided into chunks for backup. The difference lies in how the chunks are created. It uses the content-based indexing technique to identify the boundary regions called anchors [21] so that common shared data areas can be found among buddies. Anchors are determined using Rabin fingerprints[22].

When two nodes have identical chunks to backup, these chunks only need to be stored once. This is done with the help of convergent encryption. The content of the a chunk is hashed and the result is called the chunks handle, H_c. A secret encryption key, K_c, is generated from H_c. The content of the chunk is encrypted with K_c. To identify this chunk, another hash function is applied to H_c to get the public chunk ID, I_c. To enable every node that owns the chunk to decrypt the content, the handle will be included in the file's meta-data handle list. In addition, the meta-data also contains information about ownership, permission, creation and modification dates, etc. The meta-data is encrypted and written to the disk as actual data.

Pastiche uses Pastry for finding buddies of a node. A node sends out a signature consisting of a list of chunk IDs that describes a node's current file system. Those nodes with many common content will reply and be selected as buddies. In order to reduce the size of the signatures sent out, a node can send out a subset of its signature (called abstract) to other nodes.

(4) Cooperative Internet Backup Scheme

The Cooperative Internet Backup Scheme[5] developed at HP lab uses a decentralized peer-to-peer scheme to backup data on the hard drives of the participating computers. It relies on a centralized computer as a matchmaker to find partners. The centralized server also keeps track of nodes in the system. Nodes in the system register with matchmaker about their partners and other storage information. When a node needs to find partners for backup, it will send queries to the matchmaker. The matchmaker will recommend partners to the node. It is up to the node itself to contact potential partners to have an agreement as a backup for each other. It is also possible to break an existing partnership to enable a new node to have a partner. For example, node A wants to find a partner, but the matchmaker cannot find any node for it. However, we know there is an existing partnership between B and C. The matchmaker can recommend to break up the partnership between B and C and set up one partnership between A and B and another between A and C. While the number of partners of B and C does not change at all, A gets two new partners. The relation of partnership is symmetric, but not transitive.

Each participating node usually has multiple partners in diverse geographical locations to improve reliability. Different partners may have different agreements. The general rule of fairness between partners is ensured by equal exchange of disk space. The partnership is dynamic in the sense that the partners of a given node may change over time.

A logical disk consisting of spaces at the partners of a given node is the concept used in the backup system. It is based on Reed-Solomon error-correcting codes[23]. For any k data blocks, it generates m redundant blocks. The $k + m$ blocks are stored at partners and its own disk. As long as any k out of $k + m$ blocks can be retrieved, it will be able to recover the original k blocks. Each node can determine a reliability level it wants to achieve and decide an appropriate m.

To verify that a partner fulfills its obligation of up time, a node can periodically challenge its partner. If a node is not satisfied with the result, it can cancel the partnership after a grace period of two weeks. Then it can find new partners.

2.2 Reputation and Incentives for Peer-to-Peer Backup Systems

Trust and reputation have been studied for a long time. Earlier work on trust in computer science has focused on security, mainly for developing formal logic to analyze flaws in cryptographic protocols. Later a systematic model about the trust and reputation for distributed systems was established [24]. It gave a concrete definition of trust and reputation and how to distinguish them. It provided a framework to evaluate and combine recommendations to get an assessment of trustworthiness of a peer. This model and framework has been used in other application domains [25, 26, 27, 29].

Trust and reputation mechanisms have also been studied in peer-to-peer systems [31, 32, 33, 34]. For example, reputation was studied in the Gnutella peer-to-peer network [27]. The main goal was to prevent attacks and increase the P2P network security. A Bayesian network-based trust model was proposed to represent different aspects of trust in a peer-to-peer storage system [28]. It considered multiple performance aspects of a peer, such as upload speed, download speed, and file quality. The Bayesian network was used to derive the trustworthiness of a peer based on direct interactions of itself and recommendations from other peers. However, it did not discuss the fair trading policies for peer-to-peer storage systems.

2.3 GENI Networks

GENI has involved many universities and industry partners and grown significantly in recent years. It consists of multiple control frameworks [47, 48] and has resources mainly on university campuses in the United States and several sites in other countries. Figure 2.1 [1] shows the current GENI aggregates that are available for users to reserve network and computing resources for their experiments. It developed many tools supporting experimenters, such as Flack [46, 45] of ProtoGENI [47].

Several early GENI projects investigated performance measurement [69, 70, 71, 72, 73] in the GENI environment. They have different focuses and generally emphasize on developing tools to enable users to collect performance data. The GIMS project was an early measurement work, targeting at the capability of high-speed packet capture for GENI [70]. The LAMP project [71] was based on the earlier work of PerfSonar and intended to provide a common extensible format for data storage and exchange for measurement in GENI. The OnTimeMeasure project provided active measurement as an on-demand measurement service for fault analysis in GENI experiments [72]. The S3 project emphasized scalability of active measurements and provided a web interface to schedule their measurements of GENI experiments [73].

[1] Picture from http://portal.geni.net

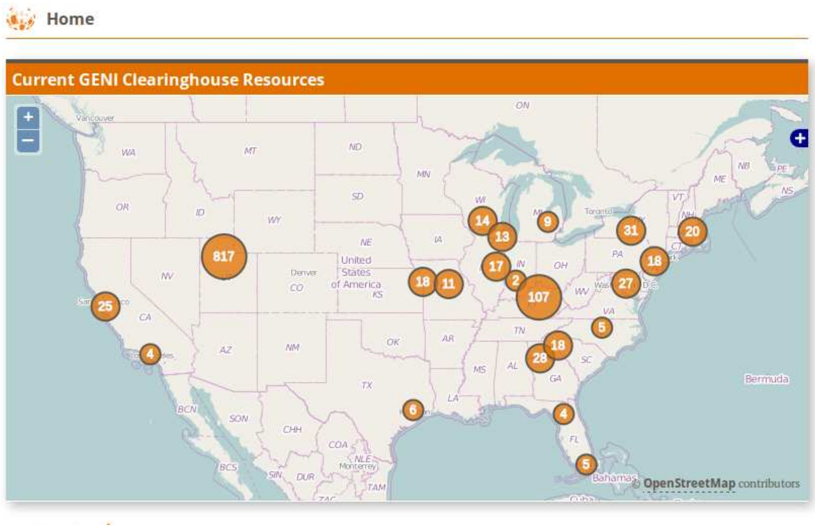

Home
Current GENI Clearinghouse Resources
817
107
31
28
27
25
20
18
18
18
17
14
13
11
9
6
5
5
4
4
2
© OpenStreetMap contributors

Figure 2.1: Current GENI aggregates

More recently, two major instrumentation and measurement efforts are under way in GENI. One is the Large-scale GENI Instrumentation and Measurement Infrastructure (GIMI) project [58], which makes use of OML library to instrument resources based on the ORBIT control framework. It can filter and process measurement flows, and consume measurement flows. It can also archive measurement data to iRODS for further processing. The other is the GENI Measurement and Instrumentation Infrastructure (GEMINI) project [57]. It is based on earlier INSTOOLS system [69] and perfSONAR system [74]. It started with supporting ProtoGENI, but can now support nodes from other control frameworks as well. All these GENI measurement systems emphasize on building tools to support users to collect measurement data *after* their experiments have been set up. In contrast, our work focuses on examining behaviors of different kinds of links in GENI networks and helps users in the design process of their experiments.

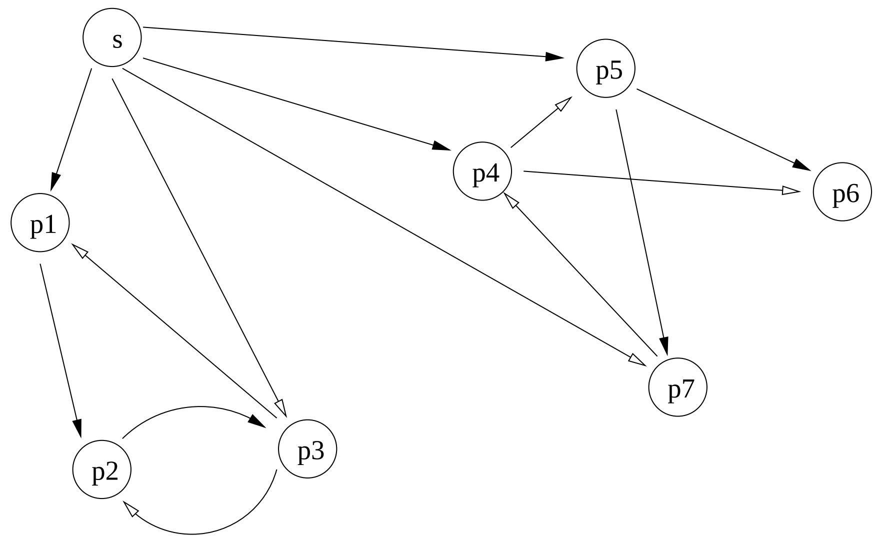


Figure 2.2: An Example of Tree-based Peer-to-Peer Multicasting

2.4 Peer-to-Peer based File Distribution

Peer to peer based file distribution has been used in multimedia streaming applications. There are two main categories of peer-to-peer based delivery mechanisms.

(1) Tree-based P2P Multicast Streaming

In tree-based P2P multicast, peers are first organized in an overlay network that contains a set of complementary multicast trees[76]. Figure 2.2 shows an example in which all nodes are connected by two complementary trees, one is by solid arrows and the other by hollow arrows. After the multicast trees are built, the server can split its files into different packet groups. Packets from each group only go to one tree. In Figure 2.2, the server separates its files into odd packets and even packets, and sends odd packets to one tree and even packets to the other tree. It is time consuming to build appropriate multicast trees. After building the multicast trees, the file transfer usually can be implemented easily.

(2) Mesh-based Multicast Streaming

Another category is mesh-based multicast streaming. Peers are also organized as an overlay network, but not necessarily in the form of trees. This offers more flexible ways for building the delivery structure. Since packets are not sent in trees, it is possible that duplicate packets are received by some nodes. One solution is that instead of sending packets from a sender to a receiver, senders can periodically broadcast their file list and let the receivers choose which packets they want to download. This clearly avoids packet duplications, but also makes the file transfer procedure more complicated.

Although the implementation of the mesh-based approach is more complex, its performance is significantly better than the tree-based method [77, 78]. It is also more fault-tolerant than the tree-based method because if a single node fails in the multicast tree, the streaming quality of all its descendants will be seriously affected. Another disadvantage of the tree-based approach is that the streaming speed is limited by the slowest peer in the multicast tree.

In addition to multimedia streaming, BitTorrent File Sharing is another application that uses peer-to-peer approach for file distribution [75]. The goal of a BitTorrent system is to deliver content as efficiently as possible in a peer-to-peer network. A special torrent file is created for each file that will be distributed using the BitTorrent system. A peer can join the torrent session after it downloads the torrent file from a website. A peer can download from multiple peers in parallel. Bittorent deploys the principle called *tit-for-tat*, implying the downloading speed of a peer is determined by how much you have uploaded to other peers. This principle encourages peers to contribute to the overall system.

Chapter 3

A Reputation-Based Fair Trading Mechanism for Peer-to-Peer Backup Systems

Peer-to-peer based backup systems can improve the reliability and availability of important files of the user by using spare disk space. We focus on the free riding problem that may cause the performance deterioration or system collapse. The goal is to develop a fair trading policy that takes into consideration of the reputations of users and provides incentives for users to contribute high quality service to the backup system.

3.1 The Basic Idea

In a peer-to-peer backup system, the basic operation is the trading of storage spaces between peers. Peer A allocates a certain amount of storage space for peer B to backup its files or exchange for storage space with other peers. In return, peer B allocates a certain amount of storage space for peer A. In traditional trading mechanisms, all peers exchange equal amount of storage spaces despite their differences in quality (such as on-line time, download/upload speeds) of the storage spaces they provide. It is obviously unfair to a peer with high quality storage space.

To deal with this problem, we propose a reputation-based fair trading mechanism that takes into account the quality of storage space. The observation of the quality of storage spaces of a peer is done by other peers and represented as its reputation. The basic idea behind the reputation-based fair trading is that the amount of storage space exchanged between peers should depend on their reputation. To that end, we design a framework for peers to build up reputation of other peers. Peers will use the direct observations of their own and the recommendations from others to derive the reputation values of other peers.

There are two approaches to reputation-based trading. A simple approach is that a peer will only trade with peers of the same reputation. This will eliminate the unfairness of trading mechanisms that do not consider reputation. A problem with this approach is that a peer can only find a limited number of other peers with the same reputation. To avoid this problem, we take a different approach that allows peers with different reputations to trade with each other. This will increase the pool of potential peers to trade with. Also the diversity of trading peers is beneficial to the overall health of the peer-to-peer system.

The key to designing such a trading mechanism is that given the reputations of peers, the trading policy should be considered to be fair by all peers. The design goal is to make sure that there is no performance penalty for a peer to trade with peers of different reputations. We will derive an optimal solution to the fair trading policy problem and analyze its relationship with other trading policies.

3.2 A Framework for Deriving Reputation of Peers

Reputation reflects the quality of a peer as observed by other peers. It can include many aspects, such as capabilities, honesty and reliability. In this dissertation, we focus on the metric called the on-line rate, which measures the percentage of the time during which a peer is connected with the Internet and provides the backup service.

A peer can derive the reputation of others in two ways. One is through direct observation. The other is to collect recommendations from other peers. We start with the first one, i.e., peer i can build its estimate of the reputation of peer j through its own experience. Suppose that peer i has stored some files at peer j. Then peer i can actively probe peer j to see whether the files are available, or passively record the interaction experience with peer j. If the total number of times peer i interacts with peer j is t_{ij} and the number of successful interactions is s_{ij}, then the reputation of j observed by peer i through direct observation is

$$d_{ij} = \frac{s_{ij}}{t_{ij}}. \tag{3.1}$$

Peer i may also ask recommendations from other peers to calculate the reputation of peer j. This is useful when peer i does not have enough experience with peer j, or peer i wants to get a more comprehensive picture about peer j. The recommendation from peer k about peer j comes in the form of $\langle d_{kj}, t_{kj} \rangle$. While d_{kj} is enough for a single value recommendation, t_{kj} gives the information about the total number of observations peer k has made of peer j. Peer i will summarize all the recommendations from a selected set of peers $\mathcal{S}$ together to determine the overall recommendation as follows.

$$e_{ij} = \frac{\sum\limits_{k \in \mathcal{S}} w_{ik} * d_{kj} * t_{kj}}{\sum\limits_{k \in \mathcal{S}} w_{ik} * t_{kj}}. \tag{3.2}$$

The w_{ik} is the weight that peer i gives peer k. It reflects how heavily peer i depends on peer k. Initially, all $w_{ik} = 1$ for all k. In which case, the above equation is equivalent to

$$e_{ij} = \frac{\sum\limits_{k \in \mathcal{S}} d_{kj} * t_{kj}}{\sum\limits_{k \in \mathcal{S}} t_{kj}}. \tag{3.3}$$

The e_{ij} is essentially the ratio of the number of successful interactions by all $k \in \mathcal{S}$ over the total number of interactions by all $k \in \mathcal{S}$.

As peer i gets more experience with the peers in $\mathcal{S}$, it will give different weights to these peers. They reflect the trustworthiness of these peers. For example, if peer i gives weight 1 to both peers k_1 and k_2 in $\mathcal{S}$ and gives weight 0 to all other peers. We will get $e_{ij} = \frac{d_{k_1 j} * t_{k_1 j} + d_{k_2 j} * t_{k_2 j}}{t_{k_1 j} + t_{k_2 j}}$.

Peer i may combine d_{ij} and e_{ij} by a weighted sum to determine the overall reputation value about peer j as follows.

$$r_{ij} = \alpha * d_{ij} + (1 - \alpha) * e_{ij}, \tag{3.4}$$

where $0 \leq \alpha \leq 1$. It is the weight that peer i puts on its own observation.

3.3 Fairness of Trading Policies

With the reputations established among peers, the next step is to develop trading policies based on the reputations. Ideally, these polices should be considered to be fair by all peers. However, fairness is a concept hard to define because it typically depends on the goal defined by some measures. In this section, we first define the measure of interest. Then we give a definition of fairness based on the measure and derive an optimal trading policy based on the definition.

To illustrate the fair trading policy, we assume that the reputation accurately reflects the on-line time of a peer. If a peer has reputation of 0.65, we assume that it will be on-line 65% of the time. Consequently, the probability that a document stored at it is available at a given time is 0.65. The measure of interest is *the availability* *of a document*, defined as the probability that the document is available, either at the original peer, or at other backup peers. In the derivation, we assume that a peer always keeps a local copy of a document even if it is replicated at other peers. So a document is not available only if neither the origin peer nor the backup peers are on-line.

Assume that peer A wants to trade with peer B, and their reputations are r_A and r_B, respectively. The question we want to answer is if A provides space s_A, how much space s_B peer A should get from peer B. The goal we want to achieve is that if A trades the same amount of space with a peer of exactly the same reputation, it should achieve the same availability.

If A trades space s_A with a peer of the same reputation, it will get s_A back from the peer. The probability that one peer is not available is $1 - r_A$. The probability that neither is available is $(1-r_A)^2$. So the availability of the document when trading with a peer of same reputation is

$$R_{same} = 1 - (1 - r_A)^2. \tag{3.5}$$

If A trades space s_A with peer B which has a lower reputation r_B, peer A is supposed to get more space as compensation. For example, we can assume that $s_B = k * s_A$ with $k \geq 1$. We describe the derivation treating k as a positive integer. After A gets $k * s_A$ space, it can trade it with k different peers of reputation r_B and get s_A space from each. So the availability of the document when trading with a peer of different reputations is

$$R_{diff} = 1 - (1 - r_A) * (1 - r_B)^k. \tag{3.6}$$

The goal we want to achieve is that the availability of the document will be the same no matter what the trading partner is. That is, we want $R_{same} = R_{diff}$. That is,

$$1 - (1 - r_A)^2 = 1 - (1 - r_A) * (1 - r_B)^k. \tag{3.7}$$

Therefore, we have

$$k = \log_{(1-r_B)}(1 - r_A). \tag{3.8}$$

So the optimal trading policy will let A get

$$s_B = s_A * \log_{(1-r_B)}(1 - r_A). \tag{3.9}$$

This policy will also be called *reputation-based fair trading* policy in the rest of this chapter. It is interesting to notice that this trading policy is not the same as the simple trading policy (called *proportional trading*) which lets each peer get a space proportional to its reputation. While we believe that it is a reasonable policy, it does not fulfill the purpose of keeping the availability of documents the same even when a peer trades space with other nodes of different reputations. To highlight the relationship between the reputation-based fair trading policy and the proportional trading policy, we make some approximation in the above derivation process. From equation (3.7), we get

$$1 - r_A = (1 - r_B)^k$$

The right-hand side can be approximated as

$$(1 - r_B)^k = 1 - \begin{pmatrix} k \\ 1 \end{pmatrix} r_B + \begin{pmatrix} k \\ 2 \end{pmatrix} r_B^2 - \ldots + (-1)^k r_B^k \approx 1 - k * r_B$$

So we have

$$1 - r_A \approx 1 - k * r_B$$

That is

$$k = r_A / r_B. \tag{3.10}$$

So we have the proportional trading policy as follows.

$$s_B = s_A * r_A / r_B. \tag{3.11}$$

It can be considered as an approximation of the optimal trading policy.

3.4 Trading Process

We use the deed concept introduced in [14] for describing the trading process. A deed represents the right of a peer to use space at other peers. Deeds can be used to store data, kept for future use, traded with other peers that need them, or split into smaller deeds. When a peer wants to replicate data to other peers in the P2P backup system, it will first find whether it holds deeds of other peers. If the space is large enough, it will use the deeds to replicate data on other peers. Otherwise, it will contact other peers and propose a trade of storage space. In order to get the deed with space large enough to store its data, it will contribute some of its local storage to the peer willing to store a copy of the data. If the contacted peer accepts, the trade is successful. Otherwise the node will try to find other peers. The deed that the peer gives to the other peer is determined by the trading policies. In particular, for the reputation-based fair trading, the sizes traded are determined by formula (3.9). Only both peers accept, the trade is successful. The peer that needs to store data may have to contact multiple peers before a successful trade can be made.

3.5 Performance Evaluation

We evaluate the effect of different trading policies on the availability of documents by letting a peer trade storage space with peers of different reputations. We compare the proposed reputation-based fair trading mechanism with the following trading policies: 1) *equal trading*, under which a peer trades equal amount of space with another peer

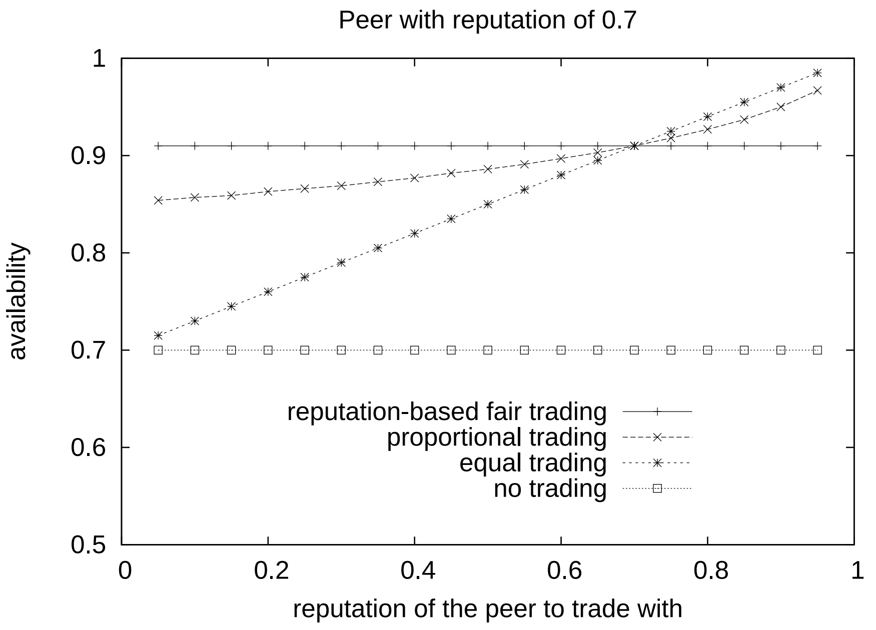


Figure 3.1: Comparison of trading policies by a peer with reputation 0.7

no matter what their reputations are; and 2) *proportional trading*, under which a peer gets space that is proportional to its reputation. We also include a *no trading* case as a baseline for comparison.

Figure 3.1 illustrates the differences between different trading policies with regard to the availability. We have a peer with reputation of 0.7. It can trade with other peers with reputations ranging from 0.05 to 0.95. If it does not trade with any other peers, the availability of documents will be 0.7. If it uses equal trading policy to trade with other peers, the availability depends on the reputation of the peer it trades with. The availability has a slight improvement over no trading in the case of trading with peers with low reputations, and close to 1 when trading with peers having a reputation close to 1. So equal trading obviously favors the peers having a smaller reputation value. The reputation-based fair trading policy makes sure that the availability is the same as trading with peers of the same reputation. It always achieves the availability of 1-(1-0.7)*(1-0.7) = 0.91. The proportional trading policy

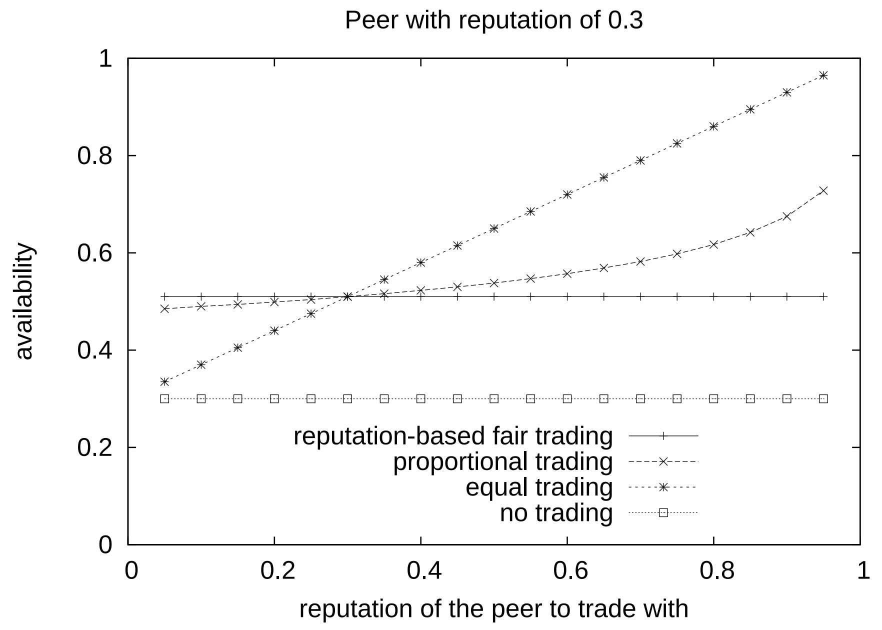


Figure 3.2: Comparison of trading policies by a peer with reputation 0.3

is the closest to the reputation-based fair trading policy. We notice that it is lower when trading with peers of a lower reputation and higher when trading with peers of a higher reputation than the reputation-based fair trading policy. So it still favors the peers with a lower reputation.

Figure 3.2 shows the availability of documents of a peer having reputation 0.3. When it trades with other peers with different reputations, it shows similar patterns. The only difference is that the equal trading and the proportional trading policies cross with the reputation-based fair trading policy at point of 0.3, instead of 0.7. This is because it is the point that determines whether the peers it trades with have a lower or higher reputation. Again the reputation-based fair trading makes sure that the availability of documents is the same no matter what peers it trades with, while proportional trading and equal trading always favors the peers with a lower reputation.

We will next show how trading can help improve the availability of documents for

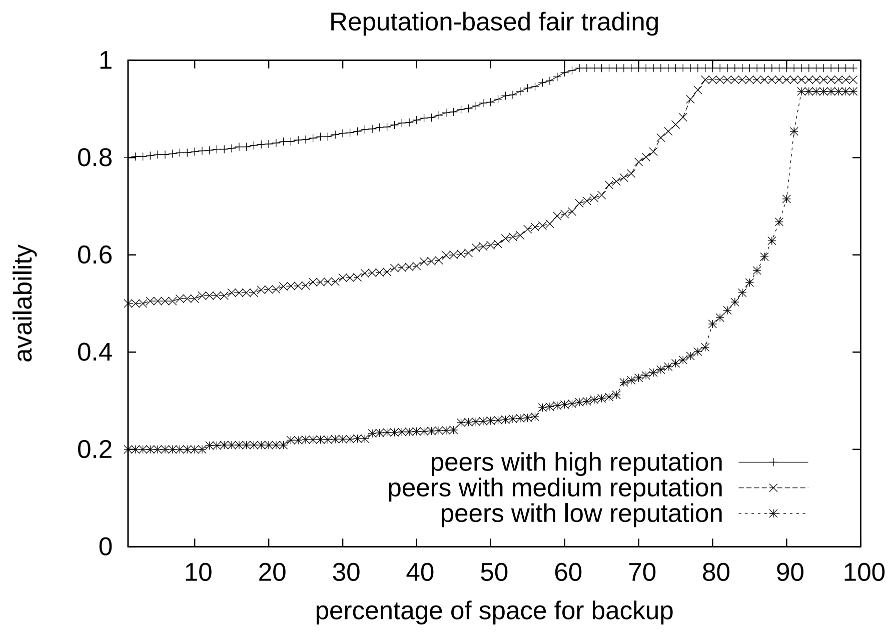


Figure 3.3: Reputation-based fair trading

a peer. We explore the cases in which different percentage of storage spaces can be used for trading. Figure 3.3 shows the availability of documents when the percentage of space for trading varies from 0% to 99%. We show three different types of peers, peers with high reputation (80% of time on line), peers with medium reputation (50% of time on line) and peers with low reputation (20% of time on line). We limit the number of backup copies to 3. When the percentage of space for trading is small, they do not have enough space for backup copies. So the availability is close to its on-line time. For example, for peers with high reputation, the availability is close to 0.8. When the percentage of space for trading increases, peers can trade more space and replicate their documents in other peers. This increases the availability of their documents. When they have a lot of space to trade, they can get enough backup space to replicate all their documents. So the availability increases close to 1. The observation is that the trading helps improve availability. The more space is used for backup, the higher the availability is.

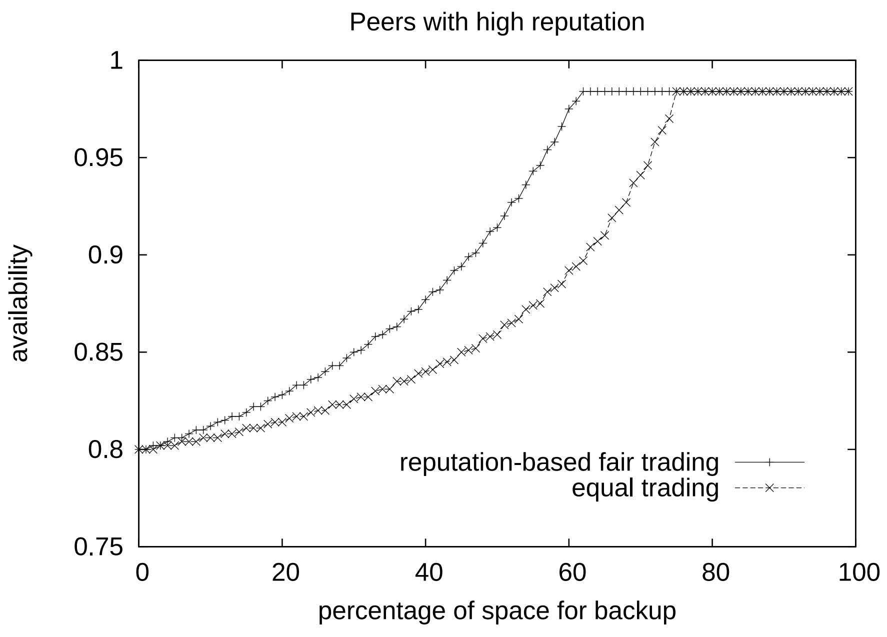


Figure 3.4: The availability at the peers with high reputation

Figure 3.4 compares reputation-based fair trading with equal trading for peers with high reputation. When the percentage of space for trading is very small, most of their documents will be stored locally. The availability of documents is close for the two policies when peers have very small percentage of space for trading. Both trading strategies have the same availability of 0.8 when the peers have 0% of space for trading. When they have a lot of space for trading, they can replicate all their files. Therefore, the availability of both policies is also the same. The interesting part is that when the space is limited, the reputation-based fair trading gives more space to peers with high reputation and therefore they have higher availability values than equal trading. This can become an incentive for peers to improve their on-line rate.

Figure 3.5 compares reputation-based fair trading with equal trading for peers with medium reputation. The difference between the two policies are much smaller. It can be explained by the fact that the effect of peers with higher reputation and the effect of peers with lower reputation cancel each other for reputation-based fair

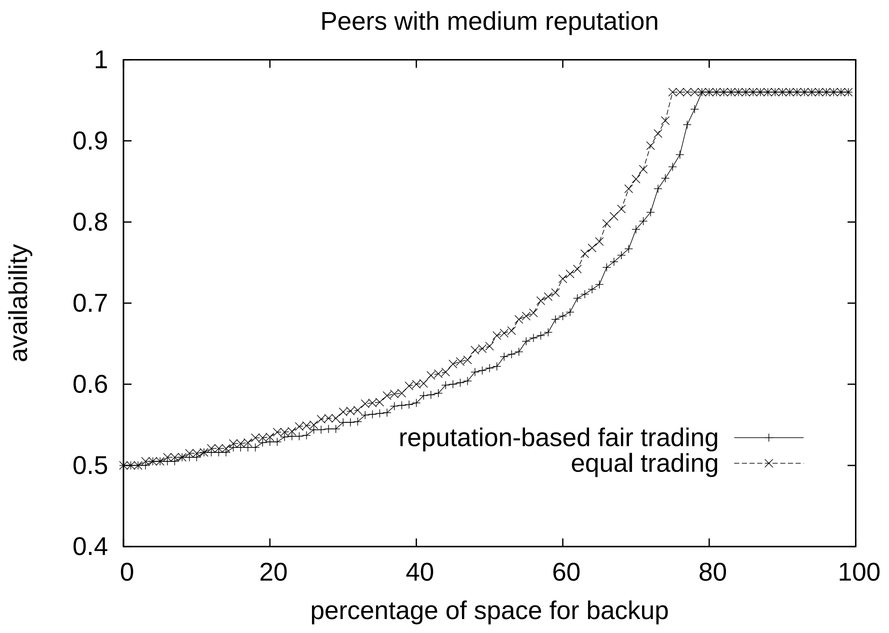


Figure 3.5: The availability at the peers with medium reputation

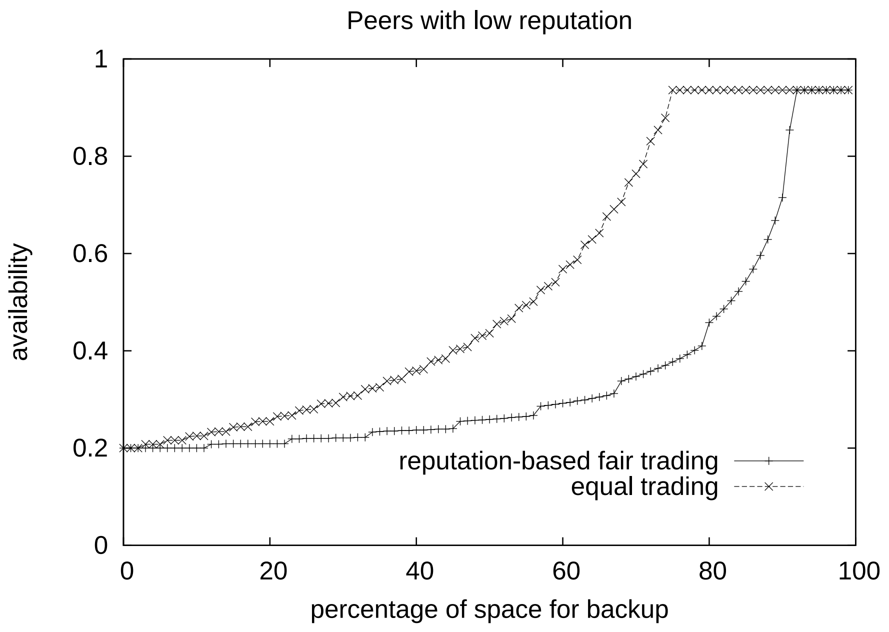


Figure 3.6: The availability at the peers with low reputation

trading. The space they get from other peers using the two different trading policies are almost the same. So the availability of documents is also almost the same.

Figure 3.6 shows the case for peers with low reputation. We can see that the availability is better than no trading (which is 20% availability). Again we see the pattern both policies have the similar availability at both ends, 0% of space for trading and the range of more than 92% of space for trading. In between, reputation-based fair trading gives them less space, and therefore, they have lower availability values than equal trading. Similarly, this can become an incentive for peers with low reputation to improve their on-line time and provide better service.

Chapter 4

Characterizing the GENI Networks and the Tradeoff between Single Aggregate and Multiple Aggregates

4.1 Introduction

We are interested in the performance of two kinds of links in GENI networks. One is the links that connect two nodes within a GENI aggregate. Typically, these nodes are located in the same room, or same GENI rack, or even the same (virtualized) physical machine. The other is the links that connect two machines (physical or virtual) located in two different GENI aggregates. The geographical distance between these two nodes can be as close as in the the same room, or as far as from east coast to west coast, or even located in different continents. They demonstrate a much wider variety. We are interested in observing their behavior over time and the tradeoff between using single aggregate links or cross-aggregate link in designing GENI experiments.

4.2 Designs and Methodologies

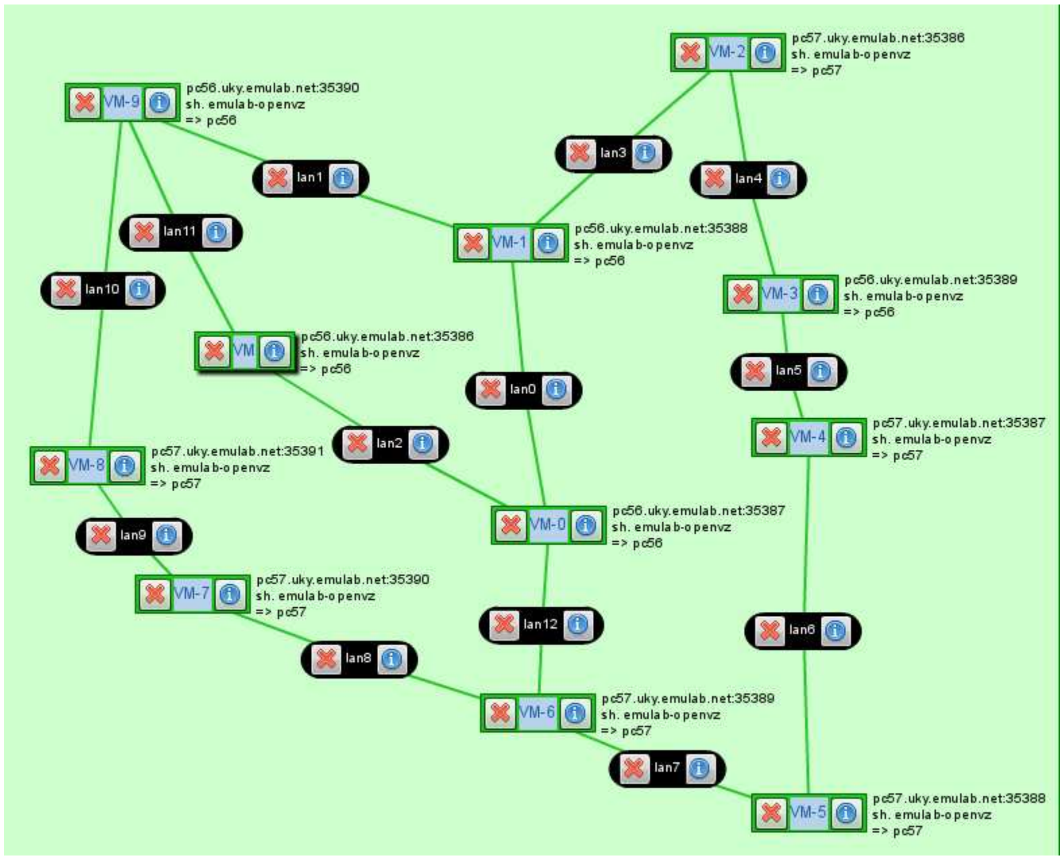


Figure 4.1: The single-aggregate experiment

To measure the performance of links within an aggregate, we design a 11-node topology as shown in Fig. 4.1. In GENI, multiple virtual machines (VMs) can be allocated from a single raw physical machine/computer (PC). We want to measure both the links that connect two VMs from the same physical machine and the links that connect two VMs from two different physical machines. Theoretically, three VMs are enough because we can have two VMs from the same physical machine and the other one from a different physical machine. We can create both kinds of links with these three machines. However, if we create a topology with three VMs, most likely we will end up with three VMs from the same physical machine due to the allocation algorithm used in GENI aggregates. Even though we can bind a VM to a specific physical machine, the submission through the GENI Flack interface is not well supported at the time of experimentation. Our strategy is to specify a topology as shown in Fig. 4.1 with enough number of nodes so that they have to be allocated to different physical machines. We understand that we do not have to measure all the links. Rather we select four links as representatives.

To measure the performance of links from different aggregates, we select 10 aggregates and set up a mesh topology as shown in Fig. 4.2. We have one VM from each of 10 aggregates. They are connected by 21 links (GRE tunnels) to form a mesh topology. We did not use any special layer-2 connections such as ION connections in the topology. We installed the `iperf` [61] on each virtual machine.

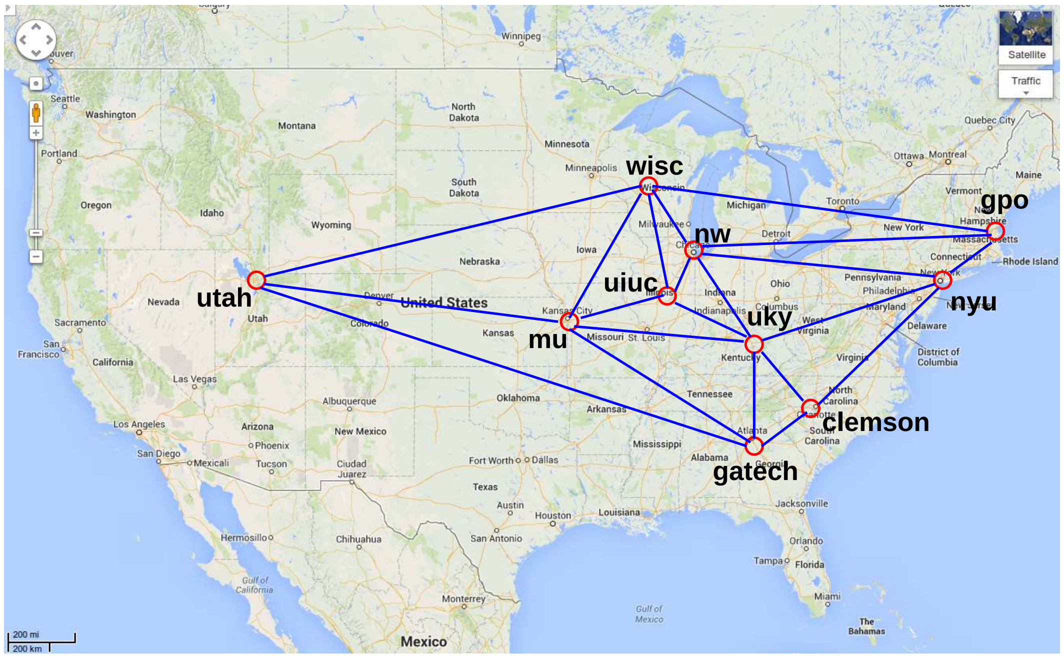


Figure 4.2: The multi-aggregate experiment

For each link, we collected two performance measures, bandwidth and latency. They are obtained by running `iperf` and `ping` tests on experimental nodes. To avoid generating too much traffic (mistaken as DoS attacks), we run the iperf test and the ping test once every hour. For the ping test, we limit the number of `ECHO_REQUESTs` to 10 for every test. After investigating the data collected, we found that the first `ECHO_REQUEST` takes substantialy longer time than other requests. One possible reason is due to the ARP request/reply time for the first one, while other requests can use the ARP cache to save time. In our calculation of the latency, we ignore the first value and calculate the average of last 9 values in the ping test. In the case of heavy traffic, `ECHO_REQUEST` can take a extremely long time. We may end up with only finishing less than 10 `ECHO_REQUEST`. Those cases are rare, but did happen several times during our test. In these situations, we just calculate the average time from whatever number of `ECHO_REQUEST/REPLY` finished.

We understand that there may be cross traffic from other applications on the Internet or other experiments of the GENI testbed. However, we try to prevent the tests in our own experiment from interfering with each other by shifting the starting time of the tests that may share nodes or network links. If a node has 4 neighbors, we can start the `iperf` and `ping` tests for these four neighbors at 0, 15, 30, and 45 minutes after the hour, respectively. We ran the tests for ten days and collected 240 data points for each test.

Links in these two experiments can be divided into three categories:

Category 1 (**Same PC**): the links connecting two VMs that are allocated from the same physical machine;

Category 2 (**Same Aggregate**): the links connecting two VMs that are allocated from two different physical machines located in the same aggregate; and

Category 3 (**Different Aggregates**): the links connecting two VMs that are allocated from two different physical machines located in two different aggregates.

The first experiment covers the first two kinds of links (category 1 and category 2), while the second experiment covers the third kind of links (category 3).

4.3 Performance Results

We collected both latency and bandwidth information from these two experiments. We first calculate the averages of latencies and bandwidths over the 10 day period for each link. The results are summarized in Table 4.3.

The links in the Same PC category have similar performance. So we only choose two links (from VM-0 to VM-1, and from VM-6 to VM-7) as representatives. For the same reason, we only choose two links (from VM-0 to VM-6, and from VM-3 to VM-4) as representatives for the Same Aggregate category. However, the performance of the links from the Different Aggregates category varies a lot. So we include the results for all the links in the second experiment in the table.

4.3.1 Latency

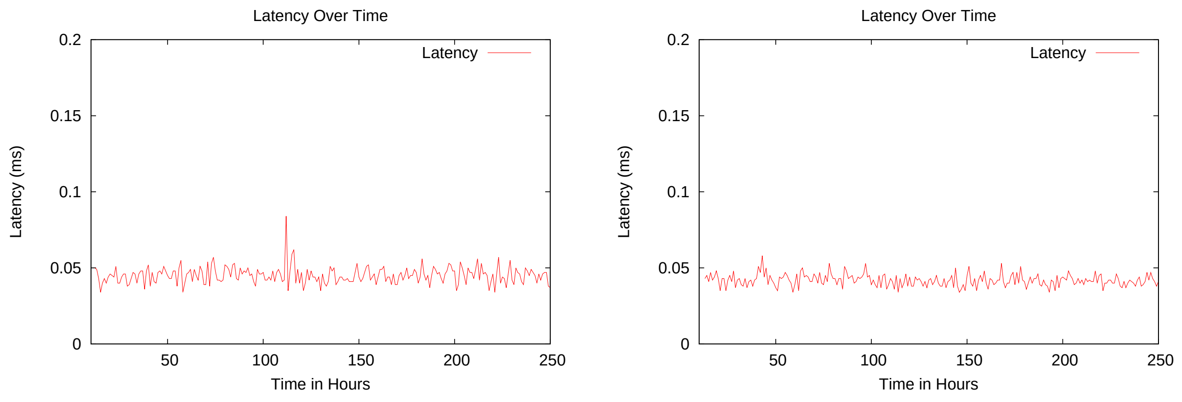
As expected, the average latencies for the links in the Same PC category are the smallest, measured at 0.042ms and 0.045ms. The latencies for the links in the Same Aggregate category are about 2.5 times as large, but still in the range of one tenth of a second. They are both much smaller than the links connecting VMs from two different aggregates. The lowest latency we got is the link connecting VMs from the Northwestern aggregate and the UIUC aggregate, measured at 3ms, which are 30 times as large as that of the links from the Same Aggregate category. We see a wide variety of latencies measured for different cross-aggregate links, ranging from 3ms to 60ms. When designing a GENI experiment, we may take the difference in latencies into consideration for reserving GENI resources.

While the average latencies give a general idea about the tradeoff between using nodes from a single aggregate versus from multiple aggregates, it is more interesting to observe how they change over time. Fig. 4.3(a) shows how the latency of the link from VM-0 to VM-1 in the first experiment change over the 10 day period. We can see that it always hovers around 0.045ms, with the highest at 0.084ms at one time

Table 4.1: Average latency and bandwidth

Category	link	Avg. Latency (ms)	Avg. Bandwidth (Mbits/second)
1. Same PC	VM-0 to VM-1	0.045	97.3
	VM-6 to VM-7	0.042	97.4
2. Same Aggregate	VM-0 to VM-6	0.115	474
	VM-3 to VM-4	0.116	469
3. Diff. Aggregates	Utah to Wisconsin	37	89
	Utah to Missouri	25	71
	Utah to Gatech	58	66
	Missouri to Wisconsin	32	90
	Missouri to Illinois	18	91
	Missouri to Kentucky	46	86
	Missouri to Gatech	53	82
	Wisconsin to GPO	41	34
	Wisconsin to Northwestern	17	90
	Wisconsin to Illinois	14	93
	Illinois to Northwestern	3	94
	Illinois to Kentucky	35	81
	Northwestern to GPO	31	39
	Northwestern to NYU	25	91
	Northwestern to Kentucky	44	85
	Kentucky to NYU	34	76
	Kentucky to Clemson	52	86
	Kentucky to Gatech	60	70
	Gatech to Clemson	20	90
	Clemson to NYU	25	92
	NYU to GPO	20	71

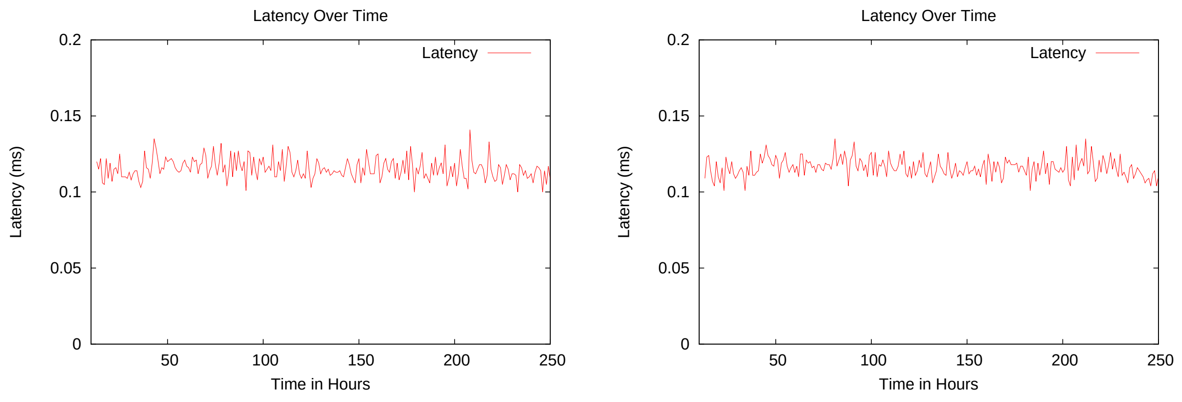
and with the lowest at 0.034ms three times. It is relatively stable and close to its average value. Fig. 4.3(b) shows that the link from VM-6 to VM-7 displays the similar pattern.



(a) from VM-0 to VM-1 (b) from VM-6 to VM-7

Figure 4.3: Latency of the links connecting two VMs from the same PC

The latencies for the links connecting two VMs from two different PCs within an aggregate are larger than that of category 1 links as shown in Fig. 4.4. Also larger is the range these latencies change. However, we still see a very stable pattern in terms how they change over time.



(a) from VM-0 to VM-6 (b) from VM-3 to VM-4

Figure 4.4: Latency of the links connecting two VMs from two PCs within an aggregate

The latencies for category 3 links demonstrate a wider variety of patterns. we first present the average latency calculated for each link in a different format in Fig. 4.5, so that we can better observe their relations. In general, the triangular inequality still holds for most triangles. There are several exceptions, for example, the triangles among NW, UIUC and UKY, among WISC, UIUC and NW, and among WISC. UIUC, and MU.

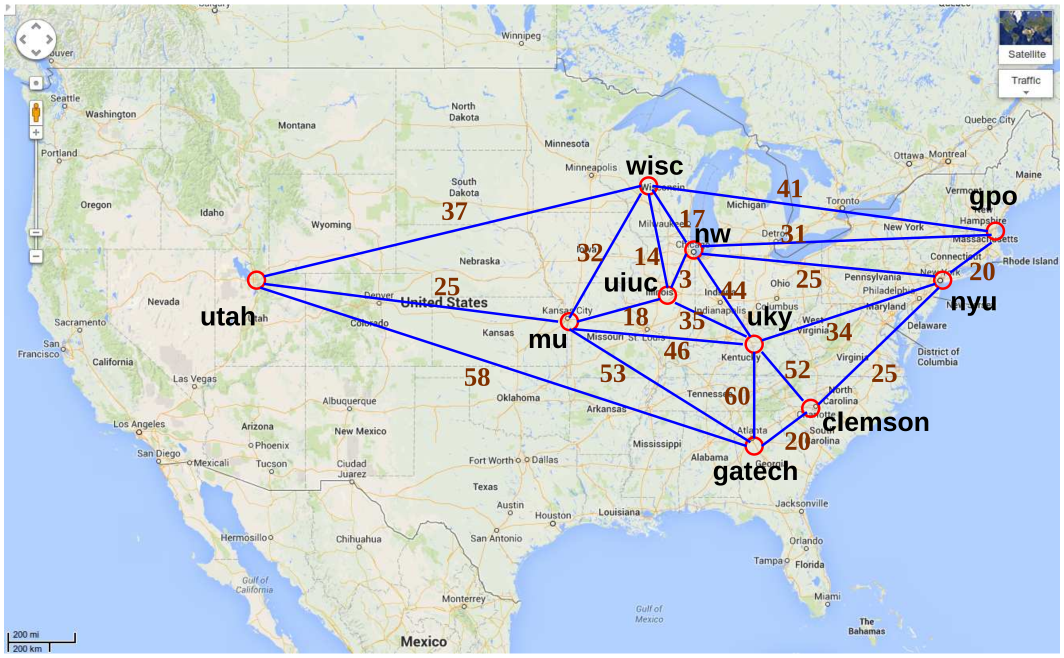
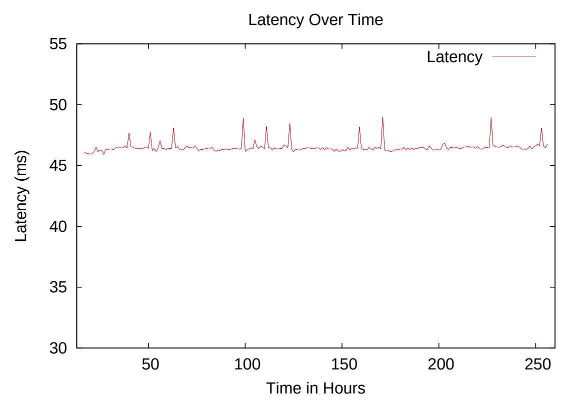


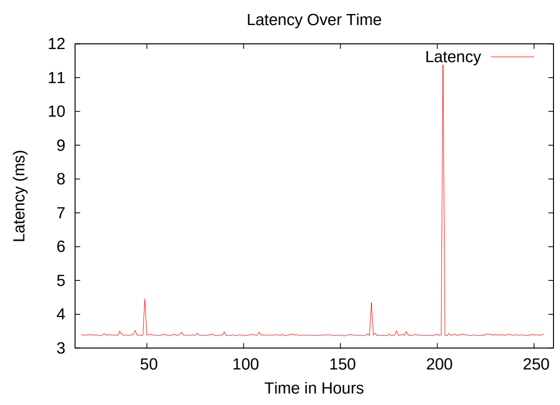
Figure 4.5: Average latency (ms)

The latency of cross-aggregate links shows different pattern. We can divide them into three groups. The first group demonstrates the behavior similar to what we observe in Fig. 4.6. In Fig. 4.6(a) we show how the latency of the link from Kentucky to Missouri [1] change over time. The absolute range of the change is larger than those links from categories 1 and 2. However, the percentage of the change is not large. Fig. 4.6(b) shows how the latency of the link between Northwestern and UIUC changes over time. We notice that the latency almost stays constant at 3.4 ms, except

[1]We use abbreviations here to indicate the VMs from a certain aggregate. "Kentucky" means the VM allocated from the University of Kentucky GENI aggregate. Similarly, "Missouri" means the VM allocated from the University of Missouri GENI aggregate. We use this convention for naming other VMs, too.



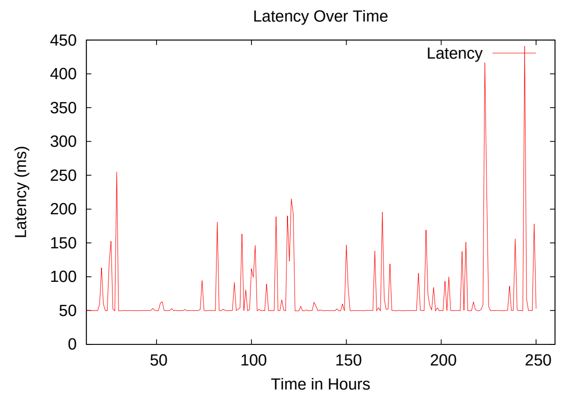
(a) from Kentucky to Missouri



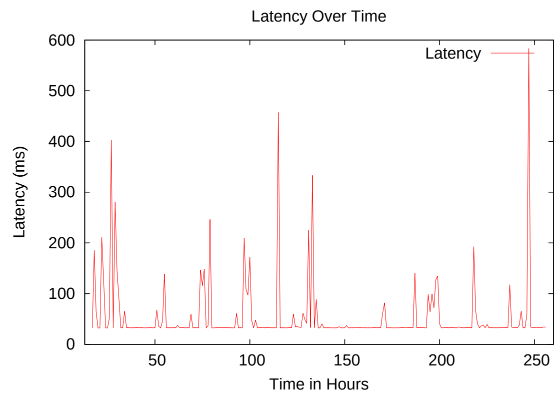
(b) from NW to UIUC

Figure 4.6: Latency of the links connecting two VMs from two different aggregates (group 1)

in a few cases it jumps to 4.5 ms and once up to 11.6 ms.



(a) from Utah to Gatech



(b) from Gatech to Missouri

Figure 4.7: Latency of the links connecting two VMs from two different aggregates (group 2)

The second group consists of those links demonstrating behavior similar to the links between Utah and Georgia Tech (Gatech) and between Gatech and Missouri. They are quite different from those links from group 1. Fig. 4.7 (a) shows the link from Utah to Gatech. Notice that the scales on y-axis in the figures are different. The range of the change in this case is almost 10 times as large as the average value. The link between Gatech and Missouri is shown in Fig. 4.7 (b). We notice that the latency varies significantly. This is probably due to the heavier traffic between the two sites. We found that about 78% of measured latencies are in the tight range from 32 ms to 38 ms. The rest are distributed in the range from 39 ms to the largest one at 583 ms. We can end up with a much more unpredictable behavior because the VMs are allocated from different aggregates.

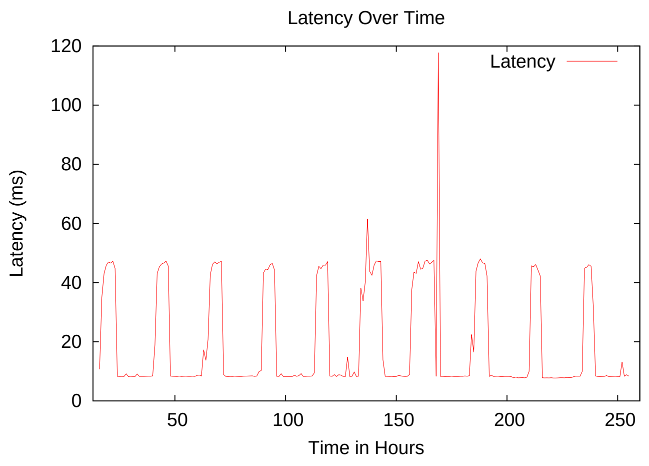
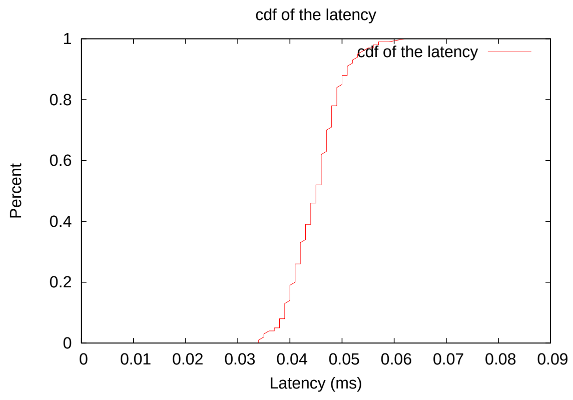


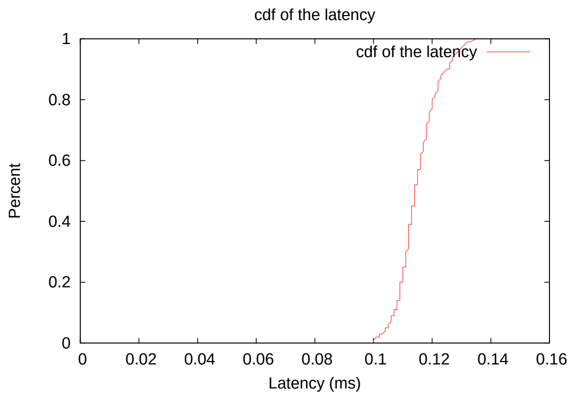
Figure 4.8: Latency from NYU to GPO over time (group 3)

A different pattern can be observed on the link between NYU and GPO in Fig. 4.8. We put it in the group 3. Instead of a few spikes, we can see that the latency stays at a higher level (around 45 ms) for a while before it goes back to the basic level at around 8 ms. It demonstrates a clear day and night pattern with 24 hours as a cycle. Over the 10 days, we can see 10 cycles. After discussing this with people from GPO, they observe similar pattern. One explanation is that it is more likely to be caused by using different Internet service providers at different times, rather than caused by the traffic. We found that about 63% latency values are in the range from 7.7 ms to 9 ms and 25% in the range from 42 ms to 48 ms.

To better understand the characteristics of the links from different categories, we plot the cumulative distribution function (cdf) of the latencies of these links. Since the two links from category 1 has similar behavior, we only include the cdf for the



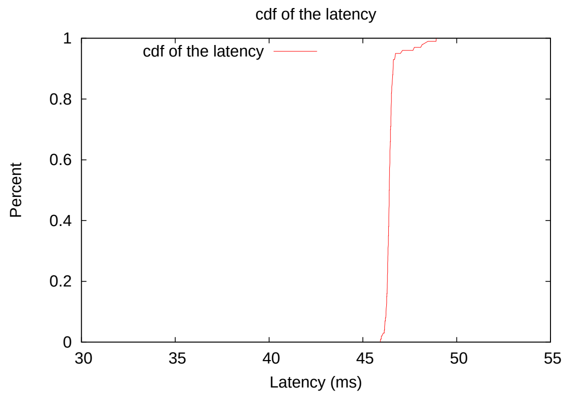
(a) Same Physical PC



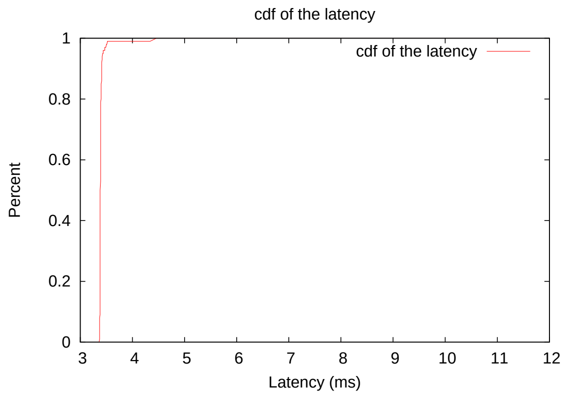
(b) Two Physical PCs

Figure 4.9: cdf of latencies for links with an aggregate

link from VM-0 to VM-1. We can see that most values are evenly distributed between 0.038ms and 0.05ms in Fig. 4.9(a). For the same reason, we only include the cdf for the link from VM-0 to VM-6 as the representative for category 2 links. We can see in Fig. 4.9(b) that most values are evenly distributed between 0.105ms and 0.125ms.



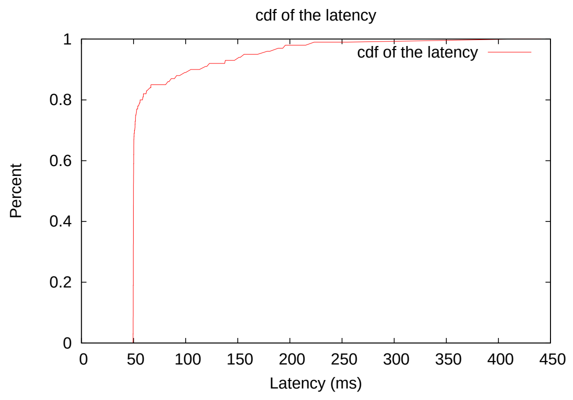
(a) From Kentucky to Missouri



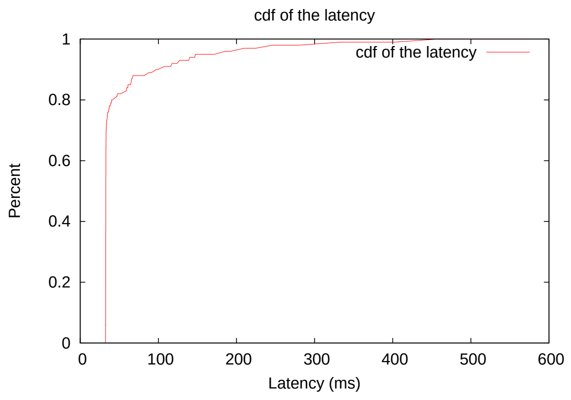
(b) From NW to UIUC

Figure 4.10: cdf of latencies of group 1 links

In contrary, the cross-aggregate links have a different distribution. For group 1 links, They have a lot of measured values close to a certain bottom value. In the case of the link from Kentucky to Missouri, more than 90% the latencies are between 46ms and 47ms, as shown in Fig. 4.10 (a). Similarly, the link between NW and UIUC has a lot of values close to 3.35 ms as shown in Fig. 4.10 (b). By looking at the data collected, we can see that the latency is in the range between 3.36 ms and 3.52 ms in more than 98% cases.



(a) From Utah to Gatech



(b) From Gatech to Missouri

Figure 4.11: cdf of latencies of group 2 links

For group 2 links, we see the latency values are distributed over a wider range. The latency of the link from Utah to Gatech is between 49.5ms and 52.5ms in more than 75% of the cases, as presented in Fig. 4.11(a). These two links also have a similar feature that the cdf of the latency of the link has a long tail because there are a significant number of values that are substantially larger than the average.

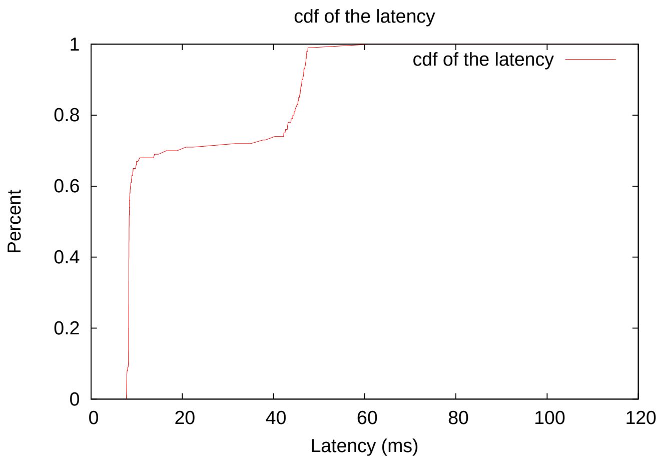


Figure 4.12: cdf of the latency from NYU to GPO (group 3)

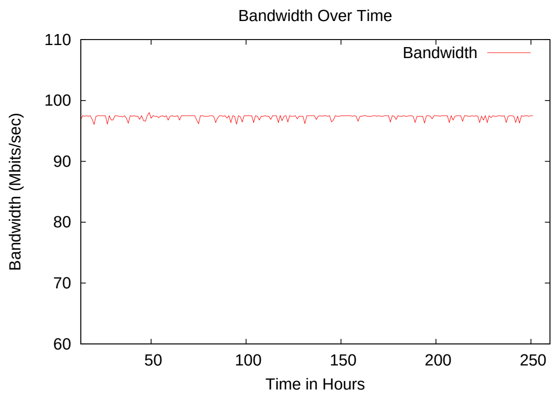
The cdf of the latency of the group 3 link shows a totally different pattern in Fig. 4.12. We observe two sharp increases, once at 8ms and another at 45ms.

4.3.2 Bandwidth

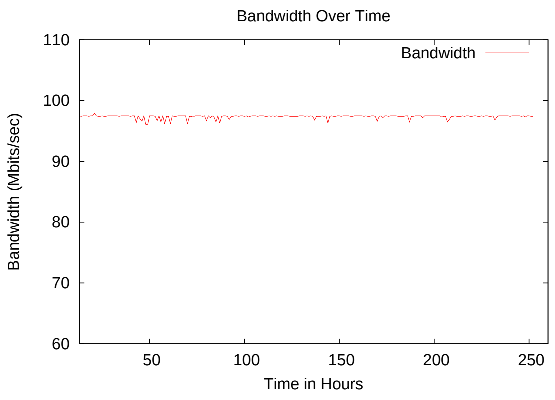
The latency of the links is only one factor to consider in designing GENI experiments. The other factor is the bandwidth of the links. If we investigate the bandwidth of the links closely, we can get a better idea what we can get from the network. From Table 4.3, we can see that category 1 links have a measured bandwidth of 97.3 Mbps and 97.4 Mbps. It can be higher because the two VMs these links attached to are located within the same physical machine. However, due to rate limit of the VMs, they are most likely capped at 100 Mbps. Fig. 4.13 (a) and (b) shows how the bandwidth of the link from VM-0 to VM-1 changes over time. Similar to the latency case, it stays close to the average level, appearing almost like a straight line.

Category 2 links achieve higher bandwidth, having average values at 474 Mbps and 469 Mbps. VMs in this case are connected with a gigabit switch. Because of the traffic from other experiments or load on the shared physical machines, the measured bandwidth is smaller than the maximal possible value. For the similar reason, we can see in Fig. 4.13 (c) and (d) that it oscillates quite a lot over time, ranging from 347 Mbps to 533 Mbps. However, the bandwidth of category 2 links is still much large than that of both category 1 links and category 3 links.

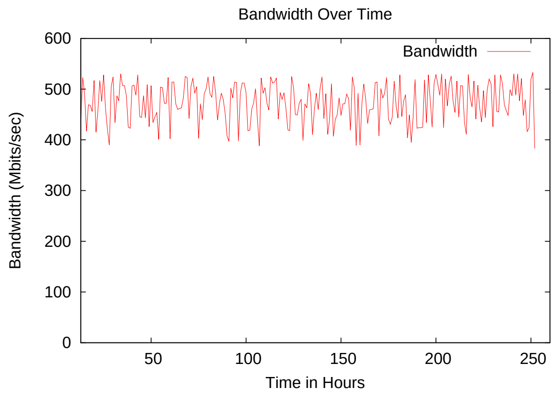
We get a totally different picture for the links connecting two VMs from different aggregates. Depending on the links, we can get an average bandwidth as low as 34 Mbps and as high as 94 Mbps. Similar to the latency case, we present the results in the topology in Fig. 4.14. We can see that the bandwidth varies from one to another. Many links have the available bandwidth at around 90 Mbps, close to the maximal possible bandwidth, which is 100 Mbps. However, there are several links that have significant lower bandwidth. For example, the link between Wisconsin and GPO is



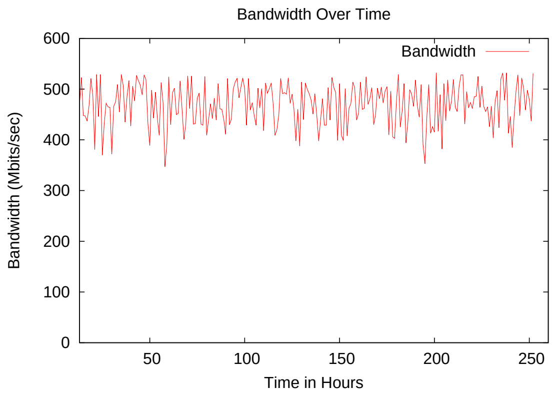
(a) from VM-0 to VM-1 (Same PC)



(b) from VM-6 to VM-7 (Same PC)



(c) from VM-0 to VM-6 (two different PCs)



(d) from VM-3 to VM-4 (two different PCs)

Figure 4.13: Bandwidth of the links connecting two VMs from the same Aggregate

34 Mbps and the link between Utah and Missouri is 71 Mbps.

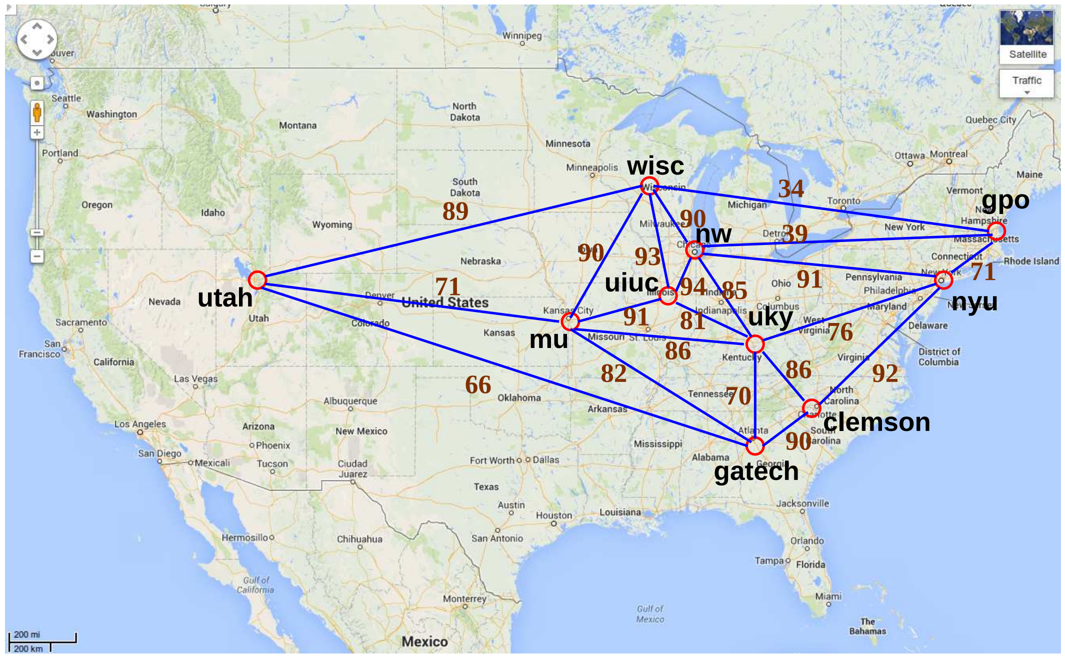


Figure 4.14: Average Bandwidth (Mbits/second)

Cross-aggregate links can also be divided into three groups based on the bandwidth. An example of group 1 link is shown in Fig. 4.15. We can see how the bandwidth of the link from a node at Clemson University to a node at the University of Kentucky changes over time. During the 10-day period, most measures are around 88 Mbps. There are a few cases in which the bandwidth drops to somewhere between 5 Mbps and 60 Mbps. This is most likely due to a burst of traffic from other applications or experiments competing available bandwidth with our tests.

The bandwidth of group 2 links changes more wildly over time, as shown in Fig. 4.16. This is because these links may compete with heavier traffic from other applications. Their behaviors are much more unpredictable than those links within a single aggregate. For the link from Utah to Gatech (Fig. 4.16 (a)), we can get a bandwidth measure as low as 8.5 Mbps and as high as 90.5 Mbps. Fig. 4.16 (b) shows the bandwidth of the link between the University of Missouri and Georgia Tech.

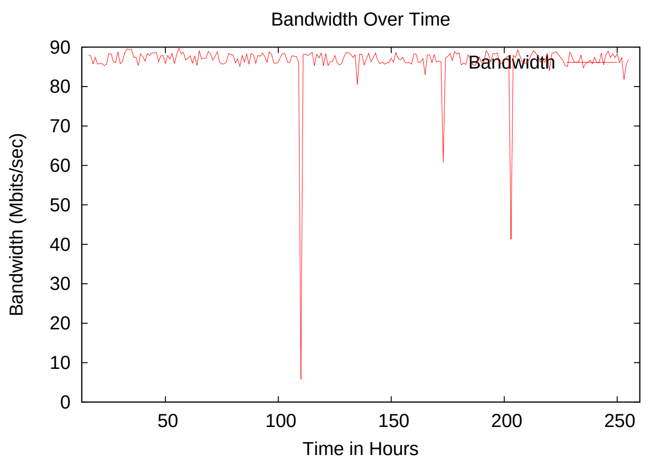
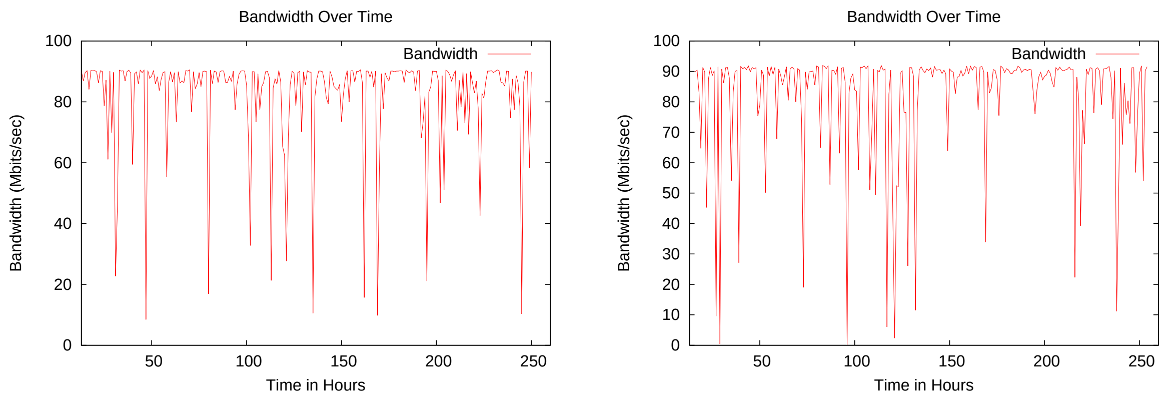


Figure 4.15: Bandwidth of group 1 link from Clemson to Kentucky changes over time

There is a clear top bandwidth measured at around 90 Mbps. However, there are a significant number of cases in which we get bandwidth way below this top level. It can be as low as close to 0 Mbps. One observation we made is that there are so many data points close to the top level bandwidth (90 Mbps) that they form a straight line on the top in the figure. Most links demonstrate this feature in our experiment.



(a) from Utah to Gatech (b) from Missouri to Gatech

Figure 4.16: Bandwidth of group 2 links connecting two VMs from two different aggregates

The two exceptions are the link between the University of Wisconsin and GPO and the link between Northwestern and GPO. They are categorized as group 3 links.

We show the first one in Fig. 4.17. The bandwidth is distributed wildly between 0 and 87 Mbps. No straight line can be drawn that connects data points on the top.

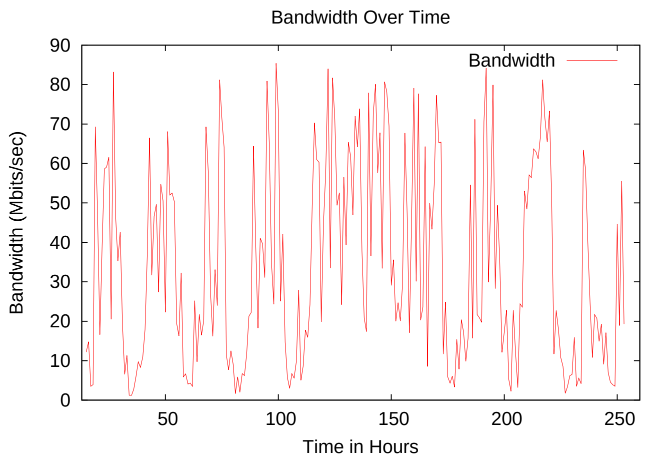
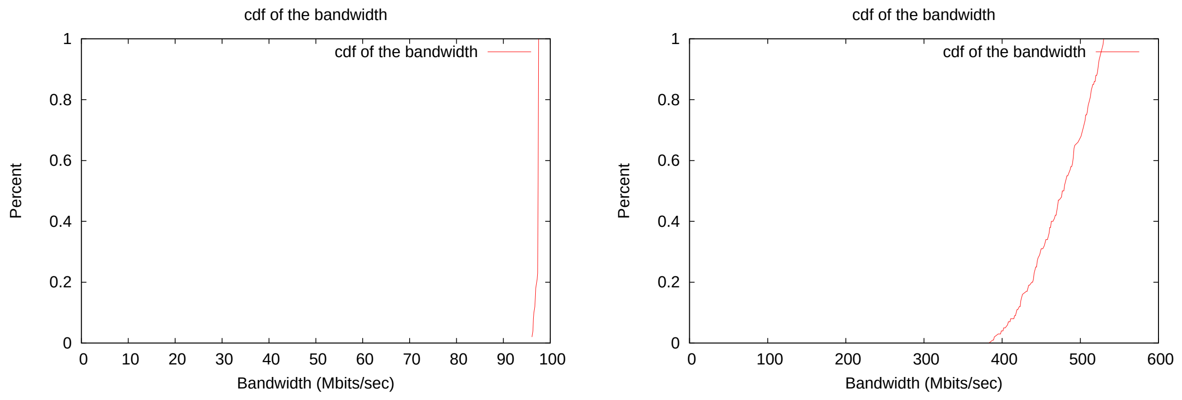


Figure 4.17: Bandwidth of the link from Wisconsin to GPO changes over time

To better understand the distribution of values of observed bandwidth, we draw the cdf of the bandwidth of these links.

In Figure 4.18, we use the link from VM-0 to VM-1 as the representative for category 1 links and the link from VM-0 to VM-6 as the representative for category 2 links. It is clear that the bandwidth of the links connecting two VMs from the same PC is distributed in a very narrow range, from 96Mbps to 98Mpbs, as shown in Figure 4.18(a). The bandwidth of category 2 links has a wider range, from 380Mbps to 530Mbps. However, it is still relatively concentrated, as shown in Figure 4.18(b).

The bandwidth for the links connecting VMs from different aggregates is distributed in a much wider range. To compare the characteristics of the measured bandwidth of the three group cross-aggregate links, we draw the cdfs of their bandwidth on the same plot in Fig. 4.19. For the link between Clemson and Kentucky, more than 96% of the measured values are distributed between 85 Mbps and 90 Mbps. Only less than 4% cases in which we got the bandwidth that is less than 85 Mbps. So we can say the bandwidth is pretty much constant. For the link between Missouri



(a) Same Physical PC (b) Two Physical PCs

Figure 4.18: cdf of bandwidth of links within an aggregate

and Georgia Tech, we got the bandwidth above 85 Mbps only 71% of the time. There are a significant portion (about 29%) of values that are somewhere between 1 and 86 Mbps. For the link between Wisconsin and GPO, half of measured values are below 25 Mbps and the other half above 25 Mbps. The values of bandwidth do not concentrate on any narrow range.

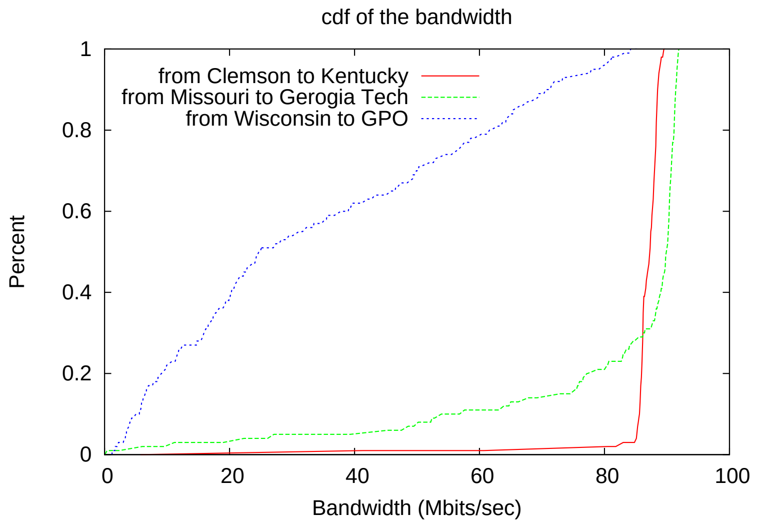


Figure 4.19: cdf's of the measured bandwidths

In summary, from the data we collected, we can see significant differences between single-aggregate links and cross-aggregate links in terms of latency and bandwidth.

Not only the average values are significantly different, but their behaviors over time can be quite different as well. When designing a GENI experiment, we can make use of performance data to decide where the nodes in the experiment should be located to meet the requirement.

Chapter 5

Block-Based Peer-to-Peer File Distribution in the Cloud

5.1 Introduction

Cloud computing is a new paradigm to meet the requirements of users by providing the computing, storage and networking services as demand from users arise. This pay-as-you-go model avoids big investment in the front and lets users start using the services provided by the cloud immediately. The users have greater flexibility and can easily handle unexpected load, data or computing requirements, because they can request more resources from the cloud as the need arises. Those applications that need to process a large amount of data in parallel are especially suitable for taking advantage of cloud computing.

One of the common issues encountered in cloud computing is to distribute big data files into processing nodes. For example, web indexing applications may have each machine to find a subset of keywords from a large file. We have to ship this file to all the machines so that each of them can process the file independently and in parallel. In GENI Desktop application, in order to initialize and instrumentize user experiments, we have to download and install customized software to all nodes involved in an experiment. All these applications need to distribute a large file to all the machines involved efficiently.

The traditional method will let the source to copy the file to all machines one by one until all the machines have a copy of the file. Assume the bottleneck link is the access link of the source node (also called origin node/server) of the file and its bandwidth is B. Further, we assume that the file size is F and we need to distribute to n machines, either physical machines or virtual machines. The naive sequential distribution will take $n * F/B$ time. It grows linearly with the number of nodes that need to receive the file. In this chapter, we will develop a block-based peer-to-peer distribution technique. It can significantly reduce the delivery time, compared with the traditional method. We will design scheduling algorithms to arrange the delivery to all receivers. The novel aspect of the algorithms is that they can achieve constant distribution time no matter how many receivers need to get the file.

5.2 Peer-to-Peer Based Distribution

The basic idea of our technique is based on the peer to peer distribution. Instead of letting the original source to send the file to all n receivers, we can let receivers to send among themselves. We use $r_1, r_2, \cdots, r_n$ to represent n receivers.

For example, in Figure 5.1, we show the delivery scheme with 8 receivers, where the original source (r_0) sends the file to r_1 first. After that, when r_0 sends the file to r_2, we can let r_1 send the file to r_3 at the same time. The vertical location represents the time. Similarly, in the next batch, we can let r_0, r_1, r_2 and r_3 send at the same time to r_7, r_5, r_6 and r_4, respectively. At last r_4 sends to r_8. We have four levels, so the total time is $4 * F/B$. If we let the original source send to all 8 receivers, the total time is $8 * F/B$. The peer-to-peer based distribution reduces the delivery time by half.

In general, we can build such a tree for an arbitrary n. The height of such a tree

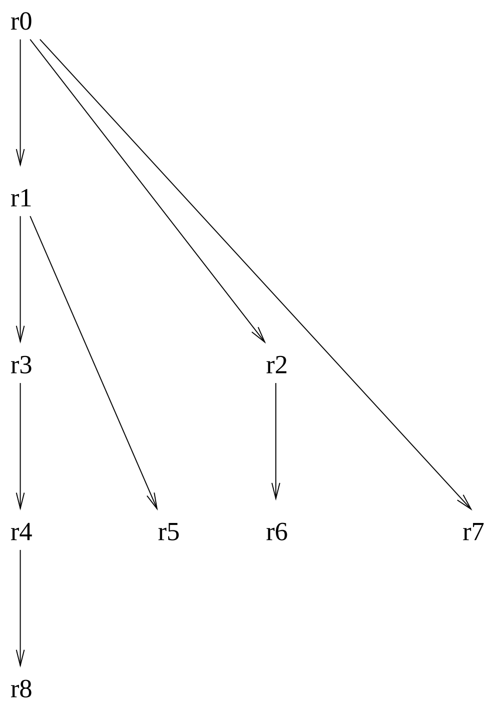


Figure 5.1: Peer-to-peer Delivery Tree

for n nodes is $\lceil \log_2(n+1) \rceil$. So the peer-to-peer distribution of the file will grow logarithmically with n. Figure 5.2 compares the time for distributing the file using the naive method and the peer to peer method. We can see that when the number of peers increases, the difference between the two methods will increase.

We can schedule the delivery among these nodes in different ways. However, the height of the tree cannot be reduced and it is the lower bound of the time to deliver a copy to all nodes.

5.3 Dividing a File into Blocks

To further reduce the distribution time of a file to all the nodes, we can divide the file into smaller blocks. Instead of using the whole file as the delivery unit, we can divide a file into m blocks. Assume they are $b_1, b_2, \cdots, b_m$. The size of each block is F/m.

When the whole file is the unit of delivery, the original node first delivers the

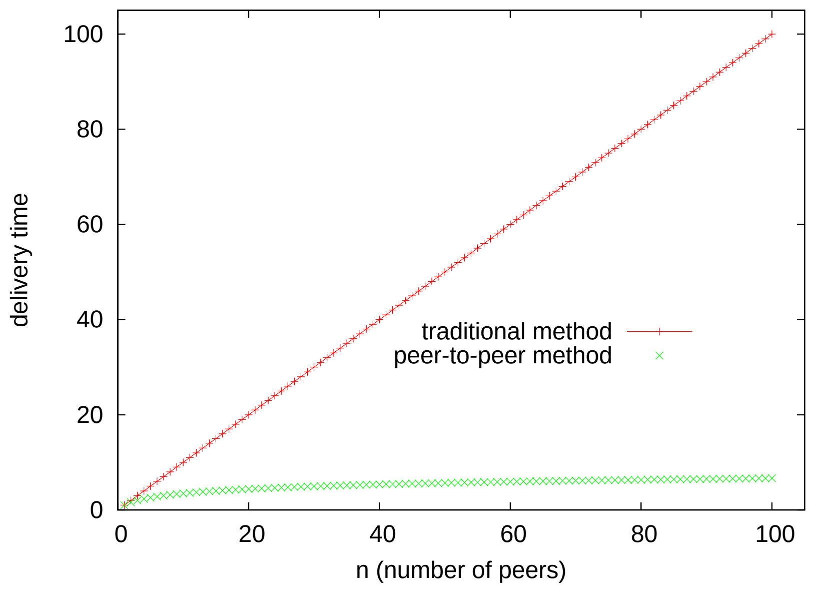


Figure 5.2: Comparing the Time for Distributing the File Using the Naive Method and the Peer-to-peer Method

whole file to the first node. The first node can deliver the file to other nodes while the original node is delivering the file to the second node. Notice that the first node can only start after it receives the whole file. This may take quite some time, especially for the case we are considering in which a file can be several megabyte or even several gigabyte large.

Instead, if we divide the file into blocks and use a block as a unit of delivery, we can significantly reduce the start time of the first node. After the origin node delivers one block to the first node, it can send the the block to other node while the origin node sends the same block or other blocks to a different node. Notice that we make an assumption that a node will not send a block and receive another block at the same time, because doing so will lengthen the delivery time. The tricky part is to make sure that all blocks will be finally received by all nodes in the system, so that each node can recover the original file. For a given node, it is also desirable to get different blocks from different nodes so that it will not totally depend on a single other node.

scheduling algorithm

```
1: $time\_slot = 0$;
2: while not(all receivers get $m$ blocks) do
3:    $time\_slot = time\_slot + 1$;
4:    $expect\_more = true$;
5:    mark all blocks as not inspected
6:    while $expect\_more$ do
7:       $block\_id = -1$;
8:       find the $block\_id$ that has not been inspected, does not have $n$ copies among
         receivers, and has the smallest number of copies among receivers
9:       if ($block\_id == -1$) then
10:         $expect\_more = false$;
11:      else
12:         find the $sender\_id$ that is neither a sender nor a receiver for this time slot,
            and has the block with $block\_id$
13:         find the $receiver\_id$ that is neither a sender nor a receiver for this time
            slot, and does not have the block with $block\_id$ and have the fewest number
            of blocks.
14:         if such sender or receiver cannot be found then
15:            mark $block\_id$ as inspected
16:         else
17:            record a schedule "AT $time\_slot$: FROM $sedner\_id$, TO: $receiver\_id$,
               SEND $block\_id$";
18:         end if
19:      end if
20:   end while
21: end while
```

Figure 5.3: Scheduling Algorithm

We design an algorithm to schedule when each is delivered to what node. To maximize parallel distribution, we want that a block to be delivered to some receiver can be further spread to other receivers. At any time slot, we should give priority to those blocks that have the smallest number of copies in the system. This is called the *rarest first* principle. On the other hand, when we need to decide whom should this block be sent to, we pick the node that has the smallest number of blocks. This encourages all nodes to participate in the system delivery. Otherwise, the delivery will be limited to a small number of the nodes having blocks. This is called the *fewest first* principle.

In Figure 5.3, we give the outline of the algorithm for the scheduling algorithm. After initializing the *time_slot* variable, the algorithm goes through a **while** loop until all receivers get all blocks they need. Each time, it increases the *time_slot* variable by 1 (Line 3), sets the initial value for *expect_more* variable (line 4), and marks all blocks as not inspected (line 5). The inner loop will continue as long as there are more blocks that can be delivered during this time slot (line 6). First find the block satisfying the condition stated in line 8. We use the rarest first principle in the selection. If we cannot find such a block, we are done with this time slot (lines 9 and 10). We should exit the inner loop. Otherwise, we will find the sender with the block and pick the receiver based on the fewest first principle (lines 12 and 13). Lines 14 to 19 do the bookkeeping work and record the schedule. Outer loop will exit when all blocks have been delivered to all receivers.

We ran the algorithm with 8 receivers with a file divided into 5 blocks. The result is shown in the following:

```
AT:     1: FROM r0, TO: r1, SEND b1

AT:     2: FROM r0, TO: r2, SEND b2

AT:     2: FROM r1, TO: r3, SEND b1

AT:     3: FROM r0, TO: r4, SEND b3

AT:     3: FROM r2, TO: r5, SEND b2

AT:     3: FROM r3, TO: r6, SEND b1

AT:     3: FROM r1, TO: r7, SEND b1

AT:     4: FROM r0, TO: r8, SEND b4

AT:     4: FROM r4, TO: r1, SEND b3

AT:     4: FROM r2, TO: r3, SEND b2

AT:     4: FROM r5, TO: r6, SEND b2

AT:     5: FROM r0, TO: r2, SEND b5

AT:     5: FROM r8, TO: r4, SEND b4

AT:     5: FROM r1, TO: r5, SEND b3
```

```
AT:     5: FROM r6, TO: r7, SEND b2

AT:     6: FROM r2, TO: r8, SEND b5

AT:     6: FROM r4, TO: r1, SEND b4

AT:     6: FROM r0, TO: r3, SEND b5

AT:     6: FROM r5, TO: r6, SEND b3

AT:     7: FROM r1, TO: r2, SEND b4

AT:     7: FROM r3, TO: r4, SEND b5

AT:     7: FROM r7, TO: r5, SEND b1

AT:     7: FROM r6, TO: r8, SEND b3

AT:     8: FROM r0, TO: r7, SEND b4

AT:     8: FROM r4, TO: r1, SEND b5

AT:     8: FROM r6, TO: r2, SEND b1

AT:     8: FROM r5, TO: r8, SEND b2

AT:     9: FROM r4, TO: r3, SEND b3

AT:     9: FROM r8, TO: r5, SEND b4

AT:     9: FROM r1, TO: r6, SEND b5

AT:     9: FROM r0, TO: r7, SEND b3

AT:    10: FROM r0, TO: r4, SEND b1

AT:    10: FROM r2, TO: r1, SEND b2

AT:    10: FROM r8, TO: r3, SEND b4

AT:    10: FROM r6, TO: r5, SEND b5

AT:    11: FROM r0, TO: r8, SEND b1

AT:    11: FROM r3, TO: r4, SEND b2

AT:    11: FROM r5, TO: r2, SEND b3

AT:    11: FROM r7, TO: r6, SEND b4

AT:    12: FROM r0, TO: r7, SEND b5
```

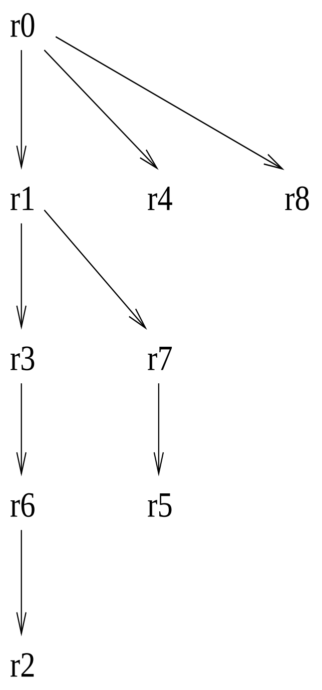
If we do not use the peer to peer approach, the delivery time will be $8 * F/B$ because we have 8 receivers. If we use peer to peer approach without dividing them into blocks, the delivery time will be $4 * F/B$ based on Figure 5.1. If we divide them into blocks, the total time is equal to the time of sending 12 blocks. Each block is one-fifth of the original file and needs $1/5 * F/B = F/(5 * B)$ time. So the total time is: $12 * F/(5 * B) = 2.4 * F/B$, which is smaller than the pure peer to peer approach.

The delivery tree for each block is shown in Figure 5.4. Notice that each receiver gets different blocks from a wide variety of other nodes. For example, receiver $r5$ gets block 1 from $r7$, block 2 from $r2$, block 3 from $r1$, block 4 from $r8$, and block 5 from $r6$, respectively. This is similar to the design of disjoint-parent tree for multimedia streaming using multiple multicast trees. This can reduce the level of dependency of one node on another so that the failure of a single node will not affect too many other nodes. We do want to point out that a node does not necessarily get blocks from all different nodes. For example, receiver $r6$ gets both block 2 and block 3 from $r5$.

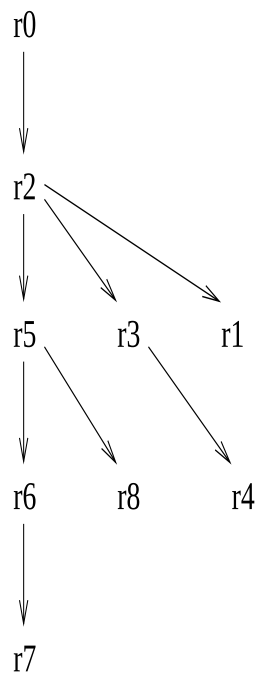
5.4 The Bandwidth Factor

The scheduling algorithm presented in the previous section assumes that the bandwidth between any two receivers and the bandwidth from the origin sender and any receiver are all the same. So it takes exactly the same time to send one block. In each time slot we schedule the transmissions, all will finish at the end of the time slot. However, the bandwidth from the origin sender to each receiver can be different from each other. They can be different from the bandwidth between different receivers. We will consider this factor in the scheduling algorithm.

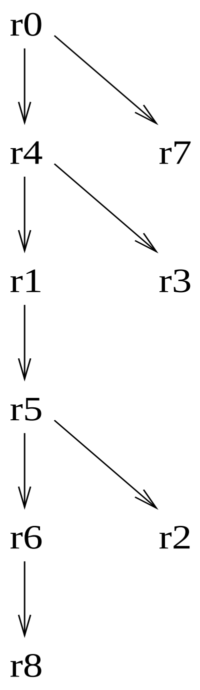
To simplify the problem, we discretize the bandwidth of all these links. Instead of using the continuous values of bandwidth, we divide all relevant bandwidth into groups. For example, if the bandwidth is distributed over the range from 0 Mbps to 100 Mbps. We can group all those links that have a bandwidth from 80 Mbps to 100 Mbps together. They will be able to finish the delivery of a block within the time represented by the size of a block divided by 80 Mbps. We assign a weight of 1 to



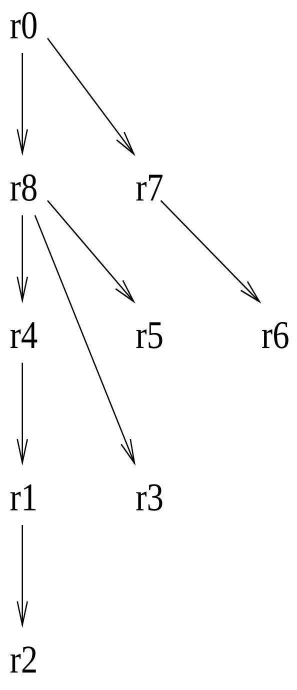
(a) for block 1



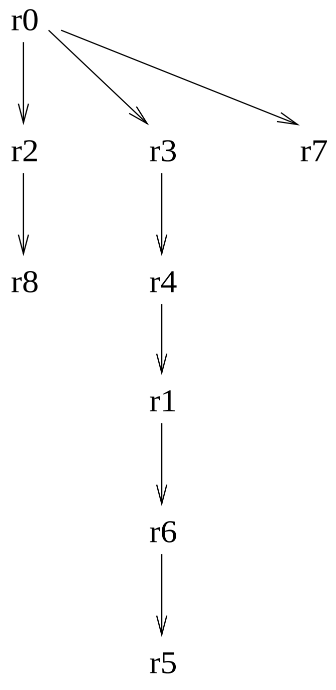
(b) for block 2



(c) for block 3



(d) for block 4



(e) for block 5

Figure 5.4: Delivery trees for each block

these links. For those links with bandwidth from 40 Mbps to 80 Mbps, it may take twice as long to send a block. We assign a weight of 2 to those links. Similarly, for links with bandwidth from 26.7 Mbps to 40 Mbps, the delivery time can be tripled. So we assign a weight of 3 to those links. We can continue this until we get to a certain threshold. Those links with a bandwidth smaller than the threshold will take so long to deliver a block that we just ignore them in the scheduling algorithm. For the rest of this section, we consider that all links are assigned a weight, representing the unit of time it takes to deliver a block over this link.

With weights assigned to all links, we need to consider this factor in the algorithm. There are two implications. One is that not all deliveries will finish in one time slot. For those links with weight greater than 1, it can take more than one time slot to finish. During these time slots, both the sender node and the receiver node cannot be selected to send or receive other blocks. The other implication is that when we select which sender to send and which receiver to receive a block, we will choose the link connecting a sender and a receiver with the smallest weight. If there is a tie, we will choose the receiver with a smaller total weight, which is calculated as the sum of the weights of the links adjacent to this receiver. We will call it the *smallest weight first* principle.

While we still stick with the rarest first principle and the fewest first principle, we need to put them in a correct order. In order to realize the potential of parallelism in which multiple nodes send to other nodes at the same time, we put the rarest first principle in the first place. We always determine which block should be transmitted first by finding a block with the fewest copies in the system. The second factor is the smallest weight first principle, which favors the sender and the receiver that can finish the current task fastest, or the receiver that can potentially send to other nodes quickly. The last factor we use is the fewest first principle that favors the receiver with the smallest number of blocks of the file.

weighted scheduling algorithm

```
1: $time\_slot = 0$;
2: while not(all receivers get $m$ blocks) do
3:   $time\_slot = time\_slot + 1$;
4:   $expect\_more = true$;
5:   mark all blocks as not inspected
6:   while $expect\_more$ do
7:     $block\_id = -1$;
8:     find the $block\_id$ that has not been inspected, does not have $n$ copies among
       receivers, and has the smallest number of copies among receivers
9:     if ($block\_id == -1$) then
10:      $expect\_more = false$;
11:    else
12:      find $sender\_id$ and $receiver\_id$ such that the weight of the link between
         $sender\_id$ and $receiver\_id$ is the smallest subject to the condition that they
         are not already a sender or a receiver in this time slot, the sender has the
         block with $block\_id$ and the receiver does not have the block with $block\_id$;
         in case there is a tie, choose $receiver\_id$ that has a smaller total weight;
         if there is still a tie, it will be broken by choosing the one with the fewest
         number of blocks;
13:      if such sender or receiver cannot be found then
14:        mark $block\_id$ as inspected
15:      else
16:        record a schedule "AT $time\_slot$ UNTIL $time\_slot + link\_weight - 1$:
           FROM $sedner\_id$, TO: $receiver\_id$, SEND $block\_id$";
17:      end if
18:    end if
19:  end while
20: end while
```

Figure 5.5: Weighted Scheduling Algorithm

The weighted scheduling algorithm is giving in Figure 5.5. The overall structure is the same as the unweighted algorithm. The main change is line 12, which deals with which node will be selected as the sender and which node will be selected as the receiver. The weight of the link is considered, not only for the current delivery, but also the potential of faster delivery from the receiver node to other nodes. The results of the scheduling algorithm is a little bit different in that it may need multiple time slots to send a block. The sender and the receiver are also considered busy during those time slots.

We run the algorithm with 8 receivers with a file divided into 5 blocks. We consider a scenario in which the links between the origin server and receivers are relatively slow, compared with the links between receivers. More specifically, we assume that the bandwidth of the links from the origin server to these receivers is one-third of the bandwidth of the links connecting these receivers. Based on the weight assignment scheme we discussed, we can let the weights from the origin server to all receivers to be 3, and the weight between any two receivers to be 1.

The result after running the algorithm is shown in the following:

```
AT     1 UNTIL     3: FROM r0, TO: r1, SEND b1
AT     4 UNTIL     6: FROM r0, TO: r4, SEND b2
AT     4 UNTIL     4: FROM r1, TO: r7, SEND b1
AT     5 UNTIL     5: FROM r1, TO: r2, SEND b1
AT     5 UNTIL     5: FROM r7, TO: r8, SEND b1
AT     6 UNTIL     6: FROM r7, TO: r3, SEND b1
AT     6 UNTIL     6: FROM r8, TO: r6, SEND b1
AT     6 UNTIL     6: FROM r1, TO: r5, SEND b1
AT     7 UNTIL     9: FROM r0, TO: r3, SEND b3
AT     7 UNTIL     7: FROM r4, TO: r1, SEND b2
AT     8 UNTIL     8: FROM r4, TO: r6, SEND b2
AT     8 UNTIL     8: FROM r1, TO: r7, SEND b2
AT     9 UNTIL     9: FROM r6, TO: r8, SEND b2

AT     9 UNTIL     9: FROM r7, TO: r5, SEND b2

AT     9 UNTIL     9: FROM r1, TO: r2, SEND b2

AT    10 UNTIL    12: FROM r0, TO: r4, SEND b4

AT    10 UNTIL    10: FROM r3, TO: r6, SEND b3

AT    11 UNTIL    11: FROM r6, TO: r7, SEND b3

AT    11 UNTIL    11: FROM r3, TO: r8, SEND b3

AT    12 UNTIL    12: FROM r8, TO: r5, SEND b3

AT    12 UNTIL    12: FROM r3, TO: r1, SEND b3

AT    12 UNTIL    12: FROM r6, TO: r2, SEND b3

AT    13 UNTIL    15: FROM r0, TO: r3, SEND b5

AT    13 UNTIL    13: FROM r4, TO: r7, SEND b4

AT    14 UNTIL    14: FROM r7, TO: r8, SEND b4

AT    14 UNTIL    14: FROM r4, TO: r2, SEND b4

AT    15 UNTIL    15: FROM r4, TO: r5, SEND b4

AT    15 UNTIL    15: FROM r8, TO: r1, SEND b4

AT    15 UNTIL    15: FROM r2, TO: r6, SEND b4

AT    16 UNTIL    16: FROM r3, TO: r4, SEND b5

AT    16 UNTIL    18: FROM r0, TO: r6, SEND b5

AT    17 UNTIL    17: FROM r4, TO: r1, SEND b5

AT    17 UNTIL    17: FROM r3, TO: r2, SEND b5

AT    18 UNTIL    18: FROM r4, TO: r8, SEND b5

AT    18 UNTIL    18: FROM r3, TO: r5, SEND b5

AT    18 UNTIL    18: FROM r1, TO: r7, SEND b5

AT    19 UNTIL    19: FROM r8, TO: r4, SEND b1

AT    19 UNTIL    19: FROM r1, TO: r3, SEND b2

AT    20 UNTIL    20: FROM r2, TO: r4, SEND b3

AT    20 UNTIL    20: FROM r7, TO: r3, SEND b4
```

Assume the bandwidth between the origin server and the receiver is B. If we do not use the peer to peer approach, the delivery time will be $8F/B$ because we have 8 receivers. If we use peer to peer approach without dividing them into blocks, the delivery time will be $4F/B$. If we divide them into 5 blocks, each block takes 3 time units to be sent from the origin server to a receiver, i.e., $(F/5)/B = 3$ time units. So each time unit is equal to $F/(15B)$. The total time is equal to 20 time units, i.e, $20 * F/(15B) = 1.33F/B$, which is smaller than the pure peer to peer approach.

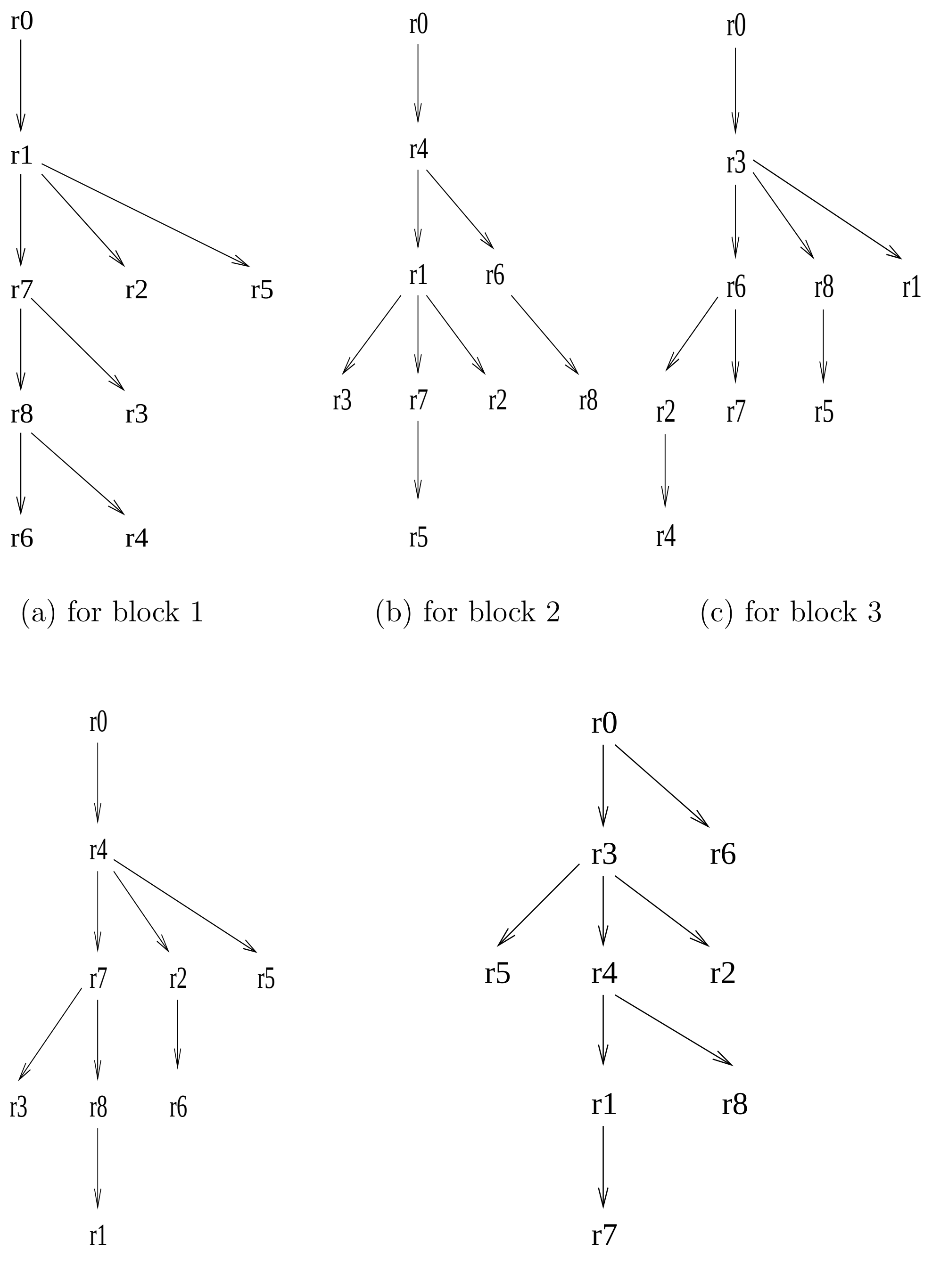
We can draw the delivery tree for each block as shown in Figure 5.6. In the unweighted case in Figure 5.4, the origin server can be scheduled to send to multiple receivers in each tree. Notice in the weighted case, the origin server only sends to one receiver in the first four trees and sends to two receivers in the fifth tree. This is because the links between the origin server and the receivers are slower and should be avoided, if possible. The algorithm prefers the delivery among the receivers themselves.

5.5 Performance Evaluation

5.5.1 Experiment Setup

We evaluate the performance of the proposed methods in this section. The number of receivers varies from 1 to 128. The file can be divided into 1 to 128 blocks. The metric we use is the total time from the start until the time every receiver has a copy of the original file. The unit of time is the time of sending one file (of size F) from the original server to one receiver. If we assume the bandwidth is B, the time unit will be F/B.

We compare our method with two other methods. One is the *traditional method* that lets the origin server to send the file to all the receivers individually. The other method is the *pure peer to peer method* that lets those receivers having a copy of the file send to other receivers, but the file is sent as a whole without being divided into blocks.



(a) for block 1 (b) for block 2 (c) for block 3

(d) for block 4 (e) for block 5

Figure 5.6: Delivery tree for each block in the weighted case

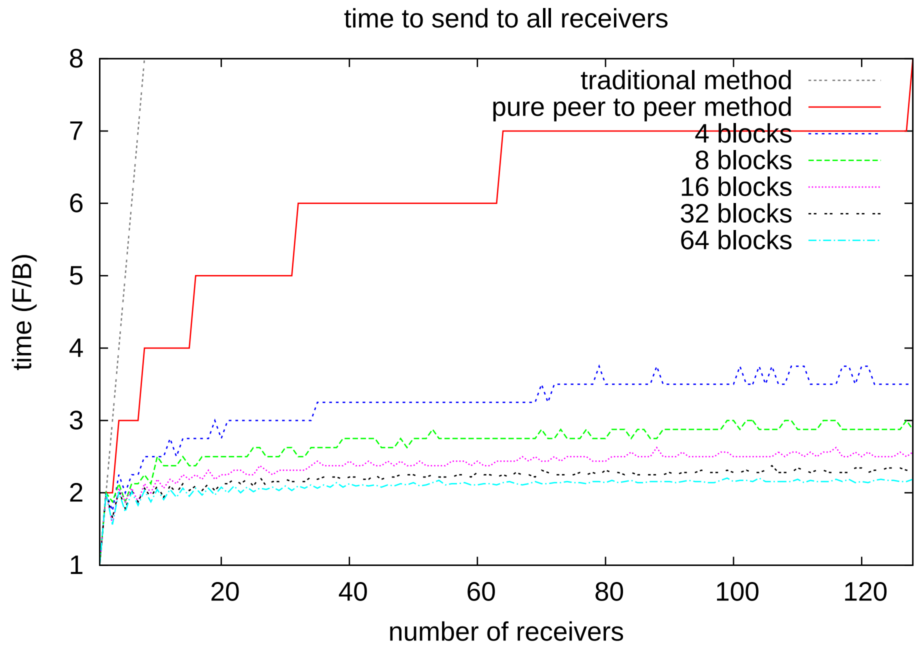


Figure 5.7: The effect of different methods

5.5.2 Unweighted Case

We start with the unweighted case in which all the bandwidths are considered the same. In Figure 5.7, we divide a file into 4, 8, 16, 32, and 64 blocks and deliver the file to all receivers. With a given number of blocks, (e.g., 32 blocks), the time increases when the number of receivers increases from 1 to 128. The smaller the number of blocks, the faster the time increases. This is because when the number of receivers is large and the number of blocks is relatively small, we do not have enough number of blocks to fully realize the benefit of parallel distribution. Another observation is that given a fixed number of blocks, the time to deliver the file to all receivers grow slower than both the traditional method and the pure peer-to-peer method.

Given a number of receivers, we would like to know how many blocks we should divide the file to be delivered. We investigate 128, 64, 32, 16, 8, and 4 receivers, respectively in Figure 5.8. We can take a look at the case with 32 receivers. When the number of blocks is 1, the delivery time is 6. When the number of blocks increases

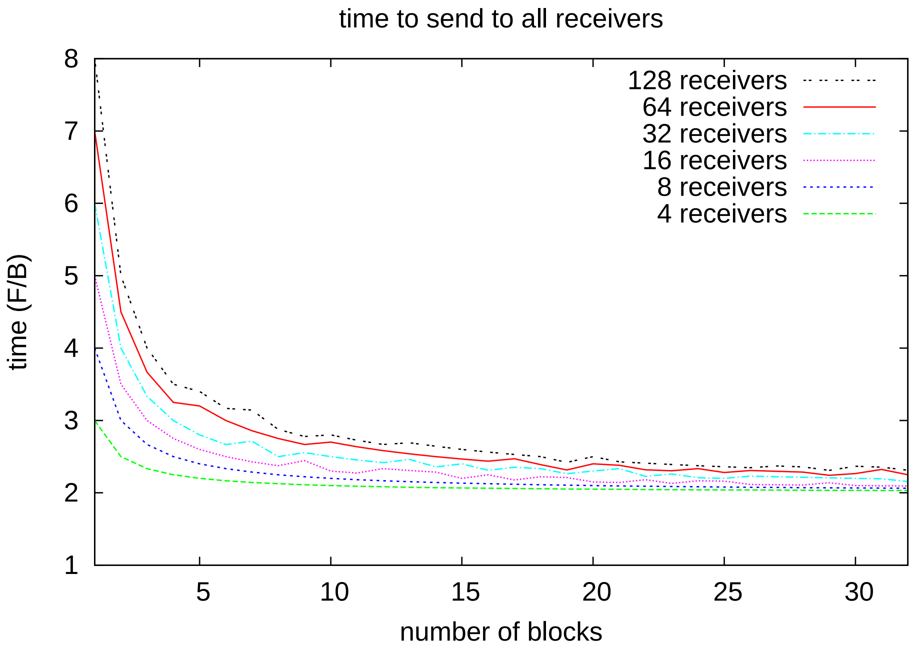


Figure 5.8: The effect of the number of blocks

to 5, the delivery time is reduced to 2.8 units. When the number of blocks further increases to 16, the time is further reduced to 2.3 units. When the number of blocks is equal to the number of receivers, the time is 2.16 units. When the number blocks is further increased, we do not see much further improvement. When the number of receivers is 128, 64, 16, 8 and 4, we observe similar behaviors. One rule of thumb for choosing the number of blocks is that we can have the number of blocks get a value somewhere greater than half the number of receivers and smaller than the number of receivers.

In Figure 5.9, we compare our method with the pure peer-to-peer method, which grows logarithmically and is significantly better than the traditional method. We let the number of receivers vary from 1 to 256, which is bit larger range than that in the previous experiments. We use the conclusion from the previous figure to determine the number of blocks for each case. More specifically, for a given number of receivers, we divide the file into blocks. The number of blocks is equal to two-thirds of the

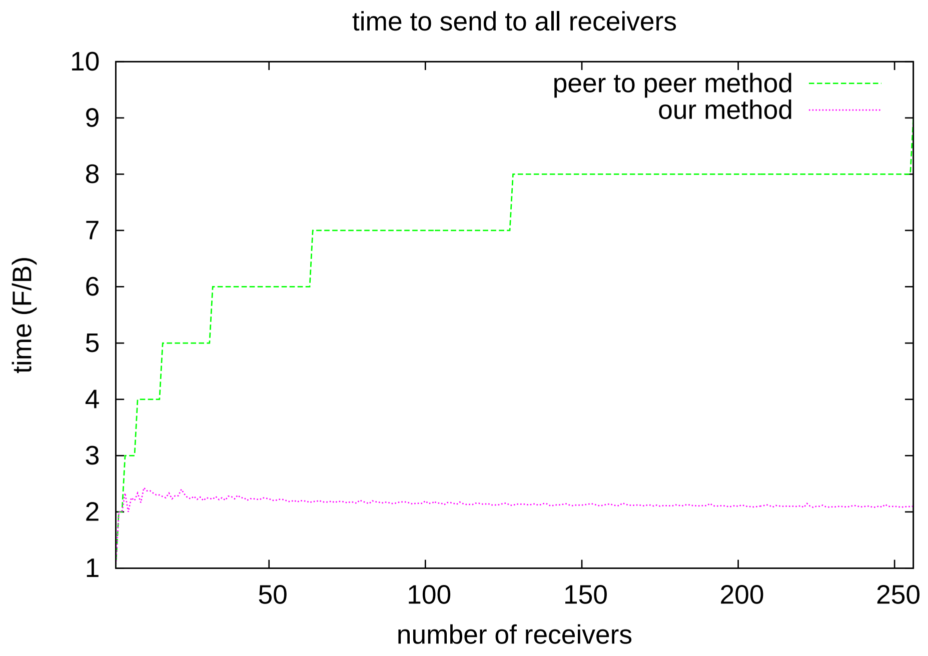


Figure 5.9: The comparison of three methods

number of receivers. We can see that for the pure peer-to-peer method, the delivery time increases from 1 to 9 when the number of receivers increases from 1 to 256. Even though it is much slower than the linear growth of the traditional method, the delivery time is unbounded when the number of receivers increases. In contrary, with an appropriate number of blocks chosen for a given number of receivers, we can see that the delivery time for our method almost stays constant around 2.3 units. Even when the number of receivers increases further, we do not see any significant increase of delivery time.

5.5.3 Weighted Case

In the weighted case, we assume that the bandwidth between the origin sender and all the receivers in the system is half of the bandwidth between two receivers. In other words, the weight of the links between the origin sender and all the receivers is 2 while the links between two receivers is 1. Similar to unweighted case, we compare our

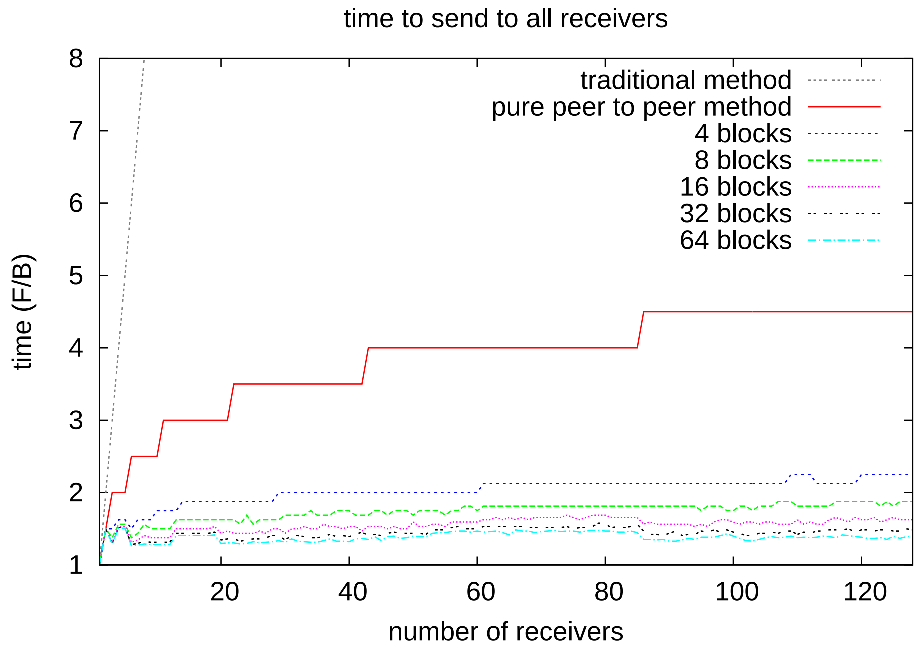


Figure 5.10: The effect of different methods in the weighted case

method with the traditional method and the pure peer to peer method, as shown in Figure 5.10. The traditional method grows linearly with the number of the receivers. The pure peer to peer method is significantly better than the traditional method and grows logarithmically. We notice that the pure peer to peer method performs better in the weighted case than in the unweighted case. In the setup of our experiment, the peer to peer transmission between receivers is twice as fast as the transmission between the origin server and the receivers. Even though the height of the peer to peer delivery tree is the same, but the time for each level is shorter because we can design the delivery in such a way at certain levels delivery only happens between receivers. For the similar reason, our methods with different numbers of blocks also perform better than their counterparts in the unweighted case. We also observe that given a certain number of blocks, the time generally becomes larger when the number of receivers increases.

In Figure 5.11, we examine the effect of the number of blocks on the performance

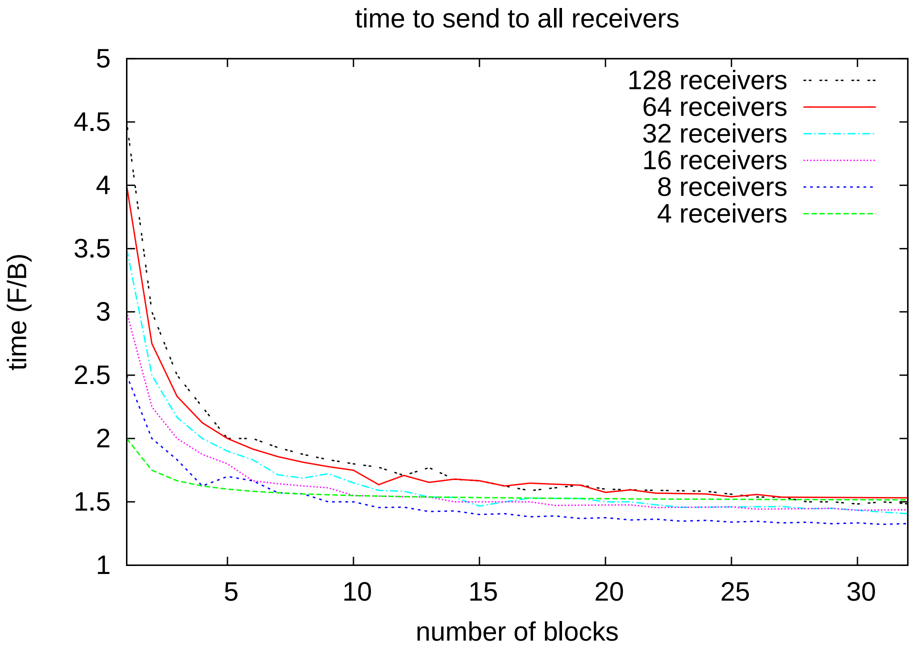


Figure 5.11: The effect of the number of blocks in the weighted case

given a certain number of receivers. In general, when the number of blocks increases, the delivery time decreases. After a certain threshold value, we do not see much decrease further. For a given number of receivers, we can choose the number of blocks based on the the number of receivers.

We compare our method in the weighted case with the pure peer-to-peer method in Figure 5.12. The number of receivers changes from 1 to 256. We let the number of blocks of the file be two-thirds of the number of receivers. The pure peer-to-peer method grows logarithmically and there is no up limit when the number of receivers grows. The delivery time of the file to all receivers using our method dividing the file into blocks remain constant, even when the number of receivers keeps increasing.

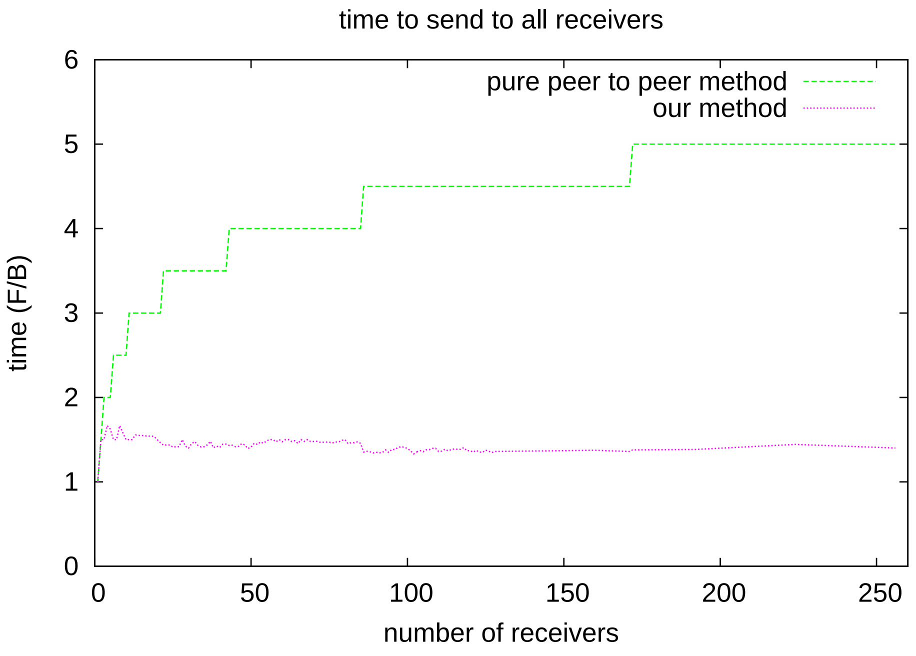


Figure 5.12: The comparison with pure peer to peer method in the weighted case

Chapter 6

Conclusion and Future Work

This dissertation presents my research work on developing a framework for providing incentives for peers to contribute to a peer-to-peer based storage backup system, characterizing GENI networks to provide guidance to GENI experimenters where to reserve resources for their experiments, and designing a block-based peer-to-peer file distribution mechanism for efficient transfer of big data files in the cloud environment. In this chapter, I summarize my dissertation work and present my future research plans.

6.1 Research Accomplishments

Peer-to-peer backup systems rely on the cooperation of their users to provide storage space. Unfortunately users of these systems can behave selfishly when left to their own. Providing incentives for peers to contribute more to the system will be beneficial to the overall health of the system. We design a framework for peers to derive reputations of other peers and propose a reputation-based trading policy for peers to exchange storage space. The difficult task is to design policies that are considered to be fair by all peers. We develop a fair trading policy based on the performance measure of the availability of documents. We find that the intuitive proportional trading policy is not optimal. The performance evaluation shows that the reputation-based fair trading can provide incentives for peers to improve the quality of service they provide.

Understanding the GENI networks is an important step in making a good design for GENI experiments. We focus on the performance aspect of the GENI networks by collecting latency and bandwidth data from two experiments. Our results are only a snapshot of the GENI networks over a short period of time. It gives us an idea what we can get from different kinds of links (within a physical machine, within an aggregate, across different pairs of aggregates). The observed behaviors and the collected performance data of the links from different categories provide helpful information for GENI experimenters. The information provides hints to experimenters on where they should reserve resources from ever-growing GENI aggregates. As more researchers and educators use the GENI network testbed, there is a growing need to better understand all aspects of GENI.

It is a common and important task to transfer big data files from an origin server outside of a cloud environment to a set of machines that will process these data in the cloud. The efficiency of these transfers can affect the overall performance of cloud applications. We explored the peer-to-peer approach to distributing big data files. While the pure peer-to-peer approach can reduce the delivery time from linear to logarithmic, we proposed a block-based approach that can further reduce the delivery time from logarithmic to a constant. We developed a scheduling algorithm for arranging how each block of a file is transferred between the origin server to nodes in the cloud and among these nodes themselves. We took into consideration of the difference in bandwidth of different links and designed a weighted version of the algorithm. The performance study showed that the delivery time of our mechanisms does not increase as the number of nodes in the system increases.

6.2 Future Work

Providing incentives is one approach to encourage cooperations. It relied on the fair trading policy to award those providing better quality service. Fairness is a hard issue and has been studied in different contexts, such as fairness in heterogeneous mulitcast communication, fairness in quality of service routing. We will continue to explore the fairness issue in the peer-to-peer backup system and extend the framework to the cloud computing environment.

Our future work on measuring GENI networks will increase the scale and scope of our data collection, such as collecting data in a longer period of time from more GENI aggregates and including layer 2 connections (such as ION) of the GENI networks in our study. We will do more statistical analyses to provide deeper insights into the characteristics of the links in the GENI networks. Another direction we will pursue is to use tools such as OnTimeMeasure to do the anomaly event analysis and study the prediction accuracy.

With a better understanding of the GENI networks, we plan to use GENI as a testbed for cloud experiments. We will request resources from multiple appropriate GENI aggregates and use GENI machines as cloud resources. First we will test our file distribution mechanisms to see how the algorithms perform in a real world of the Internet environment. With the large geographical presence of GENI resources, we can even test in a large collaborating cloud settings. Second, we will investigate the distributed clouds. The computing resources supported by cloud service providers in a data center is massive. Even with careful reliability consideration with redundant resources, we still see the report of outages of big data centers. Google, Amazon, and RackSpace have all experienced outages due to various causes, such as hardware failure, power failure, and even files. Therefore, a service supported by a single cloud (from a single cloud service provider) will not meet the requirement of applications that have high expectation of reliability. We envision that future highly reliable applications will need resources from multiple clouds. We will explore other issues, such as data replication, consistency, and transfer, arising from the collaborations from this distributed cloud environment, using GENI as the testbed to study the performance of new protocols and algorithms.

Bibliography

[1] O. Abboud, K. Pussep, A. Kovacevic, K. Mohr, S. Kaune and R. Stein- metz, Enabling resilient p2p video streaming: survey and analysis, *Multimedia Systems*, Vol. 17, No. 3, P177-197, 2011

[2] C. Batten, K. Barr, A. Saraf, and S. Treptin, pStore: A secure peer-to-peer backup system, *Technical Memo MIT-LCS-TM-632, MIT Laboratory for Computer Science*, December 2001.

[3] M. Landers, H. Zhang, K-L. Tan, PeerStore: Better Performance by Relaxing in Peer-to-Peer Backup, *Fourth International Conference on Peer-to-Peer Computing (P2P04)*, 2004.

[4] L. P. Cox and B. D. Noble, Pastiche: Making backup cheap and easy, *In Proceedings of Fifth ACM/USENIX Symposium on Operating Systems Design and Implementation*, December 2002.

[5] M. Lillibridge, S. Elnikety, A. Birrel, M. Burrows, and M. Isard, A Cooperative Internet Backup Scheme, *In Proceedings of the 2003 Usenix Annual Technical Conference*, P29-41, 2003.

[6] F. Morcos, T. Chantem, P. Little, T. Gasiba, and D. Thain, idibs: An improved distributed backup system, *In Proceedings of the 12th International Conference on Parallel and Distributed Systems*, July 2006.

[7] M. Karakaya, I. Korpeoglu, and zgr Ulusoy, Free riding in peer-to-peer networks, *IEEE Internet Computing*, Vol. 13, No. 2, P92-98, 2009.

[8] S. Saroiu, K. Gummadi, and S. Gribble, A measurement study of peer-to-peer file sharing systems, *In Proceedings of Multimedia Computing and Networking 2002*, San Jose, CA, USA, January 2002.

[9] Nathaniel S. Good and Aaron Krekelberg Usability and privacy: a study of Kazaa P2P file-sharing *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, April 2003.

[10] The Gnutella Protocol Specification, http://dss.clip2.com/GnutellaProtocol04.pdf

[11] L. O. Alima, S. El-Ansary, P. Brand and S. Haridi, DKS(N, k, f) A family of Low-Communication,Scalable and Fault-tolerant Infrastructures for P2P applications, *The 3rd International workshop on Global and P2P Computing on Large Scale Distributed Systems (CCGRID 2003)*, May 2003.

[12] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, *In Proceedings of the ACM SIGCOMM*, August 2001.

[13] P. Druschel and A. Rowstron, Pastry: Scalable, distributed object location and routing for large-scalepeer-to-peer systems, *In IFIP/ACM International Conference on Distributed Systems Platforms*, P329-350, November 2001.

[14] B. F. Cooper and H. Garcia-Molina, Peer-to-peer data trading to preserve information, *ACM Transactions on Information Systems*, pp. 133-130, vol. 20, no. 2, April 2002.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, A Scalable Content-Addressable Network, *In Proceedings of the ACM SIGCOMM 2001 Symposium on Communication, Architecture, and Protocols*, P161-172, August 2001.

[16] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz, Tapestry: A Global-scale Overlay for Rapid Service Deployment, *IEEE J-SAC*, Vol. 22, No. 01, P41-53, January 2004.

[17] M. F. Kaashoek and D. R. Karger, Koorde: A Simple Degree-optimal Distributed Hash Table, *In The 2nd Interational Workshop on Peer-to- Peer Systems (IPTPS03)*, 2003.

[18] Nicholas J.A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman, Skipnet: A Scalable Overlay Network with Practical Locality Properties, *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS03)*, March 2003.

[19] B. Leong, B. Liskov, and E. Demaine, EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management, *In 12th International Conference on Networks (ICON04)*, November 2004.

[20] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, OpenDHT: a public DHT service and its uses, *In Proceedings of the ACM SIGCOMM 2005 Symposium on Communication, Architecture, and Protocols*, P73-84, 2005.

[21] U. Manber, Finding similar files in a large file system, *In Proceedings of the USENIX Winter 1994 Conference*, P1-10, January 1994.

[22] M. O. Rabin, Fingerprinting by random polynomials, *Technical report TR-15-81, Center for Research in Computing Technology, Harvard University*, 1981.

[23] J. S. Plank, A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems, *Software,Practice and Experience*, Vol. 27, No. 09, P995-1012, 1997.

[24] Abdul-Rahman A. and Hailes S., Supporting trust in virtual communities, *In Proceedings of the Hawaii International Conference on System Sciences*, Jan 2000.

[25] Azzedin F. and Maheswaran M., Evolving and Managing Trust in Grid Computing Systems, *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE '02)*, May 2002.

[26] Qing Fang and Jie Gao and Leonidas Guibas, Reputation Formalization for an Information-Sharing Multi-Peer System, *Mobile Networks and Applications*, Vol. 18, No. 04, P515-534, November 2002.

[27] Cornelli F. and Damiani E., Implementing a Reputation-Aware Gnutella Servent, *In Proceedings of the International Workshop on Peer-to-Peer Computing*, May, 2002.

[28] Wang Y. and Vassileva J., Trust and Reputation Model in Peer-to-Peer Networks, *In Proceedings of Third International Conference on Peer-to-Peer Computing*, Sept. 2003.

[29] Montaner M. and Lopez B., Opinion based filtering through trust, *In Proceedings of the 6th International Workshop on Cooperative Information Peers (CIA02)*, September, 2002.

[30] Sabater J. and Sierra C., Regret: a reputation model for gregarious societies, *In 4thWorkshop on Deception, Fraud and Trust in Peer Societies*, 2001.

[31] Michael Piatek, Tomas Isdal, Arvind Krishnamurthy and Thomas Anderson., One hop reputations for peer to peer file sharing workloads, *In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, April 2008.

[32] Cristiano Costa and Jussara Almeida, Reputation Systems for Fighting Pollution in Peer-to-Peer File Sharing Systems, *In Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing*, September 2007.

[33] Zhuhua Cai, Ruichuan Chen, Jianqiao Feng, Cong Tang, Zhong Chen and Jianbin Hu, A holistic mechanism against file pollution in peer-to-peer networks, *In Proceedings of the 2009 ACM symposium on Applied Computing*, March 2009.

[34] Giancarlo Ruffo and Rossano Schifanella, A peer-to-peer recommender system based on spontaneous affinities, *Transactions on Internet Technology (TOIT)*, February 2009.

[35] A. Rowstron and P. Druschel, Storage Management And Caching In PAST, A Large-Scale, Persistent Peer-To-Peer Storage Utility, *ACM SOSP*, 2001.

[36] J. Ioanndis, S. Ioanndis, A. D. Keromytis, and V. Prevelakis, Fileteller: Paying and getting paid for file storage, *In Proceedings of the Sixth Annual Conference on Financial Cryptography*, P282-299, 2002.

[37] T.-W.J. Ngan, D. S. Wallach, and P.Druschel, Enforcing fair sharing of peer-to-peer resources, *In Proceedings of the Second International Workshop on Peer-to-peer Systems*, February 2003.

[38] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, GENI: a federated testbed for innovative network experiments, *Computer Networks*, January 2014, http://dx.doi.org/10.1016/j.bjp.2013.12.037.

[39] The GENI Project Office, GENI System Overview, *http://www.geni.net/docs/GENISysOvrvw092908.pdf.*

[40] GENI Concepts, *http://groups.geni.net/geni/wiki/GENIConcepts.*

[41] GENI Glossary, *http://groups.geni.net/geni/wiki/GENIGlossary.*

[42] GENI Aggregates, *http://groups.geni.net/geni/wiki/GeniAggregate.*

[43] Robert Ricci and Ted Faber, GENI RSpec, *http://groups.geni.net /geni/attachment/wiki/GeniRspec/rspec-draft-v0.5.doc.*

[44] RSpec Reference Guide, *http://www.protogeni.net/ProtoGeni/wiki/RSpecReference2.*

[45] J. Duerig and R. Ricci and L. Stoller and M. Strum and G. Wong and C. Carpenter and Z. Fei and J. Griffioen and H. Nasir and J. Reed ans X. Wu, Getting Started with GENI: A User Tutorial, *ACM SIGCOMM Computer Communication Review (CCR)*, Vol. 42, No. 01, P72-77, 2012.

[46] The Flack GUI, *http://www.protogeni.net*, 2012.

[47] ProtoGENI., *http://www.protogeni.net.*

[48] ORCA, *https://geni-orca.renci.org/trac/.*

[49] Ellen W. Zegura and Ken Calvert and Samrat Bhattacharjee, How to Model an Internetwork, *Proceedings of INFOCOM'96*, P594 -602, San Francisco, CA, 1996.

[50] Ken Calvert and Matt Doar and Ellen W. Zegura, Modeling Internet Topology, *IEEE Communications Magazine*, Vol. 35, No. 05, P160-163, June 1996.

[51] Ellen W. Zegura and Kenneth L. Calvet and Michael J. Donahoo, A quantitative comparison of graph-based Internet topology, *IEEE/ACM Transactions on Networking*, Vol. 06, No. 05, P770-783, December 1997.

[52] Bernard M. Waxman, Routing of Multipoint Connections, *IEEE Journal of Selected Areas in Communications*, Vol. 06, No. 09, P1617-22, December 1988.

[53] Jared Winick and Sugih Jamin, INET-3.0: Internet Topology Generator, *IEEE Journal of Selected Areas in Communications*, University of Michigan, http://www.eecs.umich.edu /techreports/cse/02/CSE-TR-456-02.pdf, 2002.

[54] Alberto Medina and Ibrahim Matta and John Byers, BRITE: A Flexible Generator of Internet Topologies, *Boston University*, BU-CS-TR-2000-005, http://www.cs.bu.edu/faculty/matta/Research/BRITE/.

[55] Omni, *http://trac.gpolab.bbn.com/gcf/wiki/Omni.*

[56] GENI Portal, *http://portal.geni.net.*

[57] GEMINI: A GENI Measurement and Instrumentation Infrastructure, *http://groups.geni.net/ geni/wiki/GEMINI.*

[58] GIMI: Large-scale GENI Instrumentation and Measurement Infrastructure, *http://groups.geni.net/geni/wiki/GIMI.*

[59] New GENI Shakedown Experiments, *http://groups.geni.net/geni/wiki/ GEC18Agenda/NewDocAndExpts.*

[60] P. Calyam, D. Krymskiy, M. Sridharan, and P. Schopis, Active and passive measurements on campus, regional and national network backbone paths, *Proceedings of ICCCN'05*, October 2005, pp. 537–542, san Diego, CA.

[61] iperf - TCP and UDP bandwidth performance measurement tool, *http://code.google.com/ p/iperf/.*

[62] Document Object Model (DOM), *http://www.w3.org/DOM/.*

[63] C.Jin,Q.Chen and S.Jamin, Gene, *Inet: Internet topology generator,Tech.rep.cse-tr-433-00*, University of Michigan EECS Dept, 2000.

[64] Jeannie Albrecht and Danny Yuxing Huang, Managing Distributed Applications using Gush, *Inet: Internet topology generator,Tech.rep.cse-tr-433-00*, Proceedings of the Sixth International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, Testbeds Practices Session (TridentCom), May 2010.

[65] Avraham Leff and James T. Rayfield, Web-Application Development Using the Model/View/Controller Design Pattern, *Proceedings of the IEEE Enterprise Distributed Object Computing Conference*, P118-127, September, 2001.

[66] The ns Simulator, *http://www.nsnam.org/.*

[67] Scalable Simulation Framework (SSF), *http://www.ssfnet.org/.*

[68] OMNeT++ Simulator, *http://www.omnetpp.org/.*

[69] James Griffioen and Zongming Fei and Hussanmuddin Nasir and Xiongqi Wu and Jeremy Reed and Charles Carpenter, The Design of an Instrumentation System for Federated and Virtualized Network Testbeds, *Proc. of the First IEEE Workshop on Algoirthms and Operating Procedures of Federated Virtualized Networks (FEDNET)*, Maui, Hawaii, April 2012.

[70] GIMS: High-Speed Packet Capture for GENI, *http://gims.wail.wisc.edu/docs/Tutorial.html*, 2005.

[71] Leveraging and Abstracting Measurements with perfSONAR(LAMP), *http://groups.geni.net/geni/wiki/LAMP*, 2011.

[72] Prasad Calyam and Paul Schopis, OnTimeMeasure: Centralized and Distributed Measurement Orchestration Software, *http://groups.geni.net/geni/wiki /OnTimeMeasure*, 2012.

[73] Sonia Fahmy and Puneet Sharma, Scalable, Extensible, and Safe Monitoring of GENI Clusters, *http://groups.geni.net/geni/attachment/wiki/ScalableMonitoring/design.pdf*, 2010.

[74] PerfSONAR, *http://www.perfsonar.net/*.

[75] J.A. Pouwelse, P. Garbacki, D.H.J. Epema and H.J. Sips The Bittorrent P2P File-sharing System: Measurements and Analysis *4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, Feb 2005.

[76] P. Baccichet, T. Schierl, T. Wiegand and B. Girod Low-delay peer-to-peer streaming using scalable video coding *Packet Video 2007*, Vovember 2007.

[77] N. Magharei, R. Rejaie and Y. Guo Mesh or multiple-tree: A comparative study of live p2p streaming approaches *INFOCOM 2007*, 2007.

[78] N. Magharei and R. Rejaie PRIME: Peer-to-peer receiverdriven mesh- based streaming *INFOCOM 2007*, 2007.

Vita

Personal Data:

Name: Ping Yi

Gender: Male

Educational Background:

- Masters of Science in Computer Science, University of Kentucky, 2011.
- Bachelor of Engineering in Electrical Engineering, Harbin Engineering University, 1999.

Research Experience:

- Ph.D. Research Field: Computer Networks, 2006 - Present.

  The Laboratory for Advanced Networking,

  Department of Computer Science, University of Kentucky, Lexington, KY, USA.

Teaching Experience:

- Teaching Assistant, 08/2006 - 05/2011.

  Department of Computer Science, University of Kentucky, Lexington, KY, USA.

Work Experience:

- Senior Web Developer, 01/2014 - Present.

  Eastern Kentucky University, Richmond, KY.

- Web Developer, 10/2012 - 12/2013.
  Able Engine, Lexington, KY.
- Software Developer, 05/2012 - 09/2012.
  Lexmark International Inc., Lexington, KY.

Publications:

- Zongming Fei, Ping Yi and Jianjun Yang, The Tradeoff between Single Aggregate and Multiple Aggregates in Designing GENI Experiments, *9th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, Guangzhou, China, 2014.
- Ping Yi and Zongming Fei, Characterizing the GENI Networks, *The 3rd GENI Research and Educational Experiment Workshop*, Atlanta, USA, 2014.
- Ping Yi, Jianning Liu and Zongming Fei, A Reputation-Based Fair Trading Mechanism for Peer-to-Peer Backup Systems, *The 17th IEEE International Conference on Networks*, Singapore, 2011.