2013

Reducing the cumulative file download time and variance in a P2P overlay via proximity based peer selection

Uriel J. Carasquilla
Nova Southeastern University, uriel.carrasquilla@mail.mcgill.ca

Reducing the cumulative file download time and variance in a P2P overlay via proximity based peer selection

By

Uriel Carrasquilla

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Information Systems

Graduate School of Computer and Information Sciences
Nova Southeastern University

2013

We hereby certify that this dissertation, submitted by Uriel Carrasquilla, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

_____ _____

Peixiang Liu, Ph.D. Date
Chairperson of Dissertation Committee

_____ _____

Sumitra Mukherjee, Ph.D. Date
Dissertation Committee Member

_____ _____

Greg Simco, Ph.D. Date
Dissertation Committee Member

Approved:

_____ _____

Eric Ackerman, Ph.D. Date
Dean, Graduate School of Computer and Information Sciences

Graduate School of Computer and Information Sciences
Nova Southeastern University

2013

Abstract

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Reducing the cumulative file download time and variance in a
P2P overlay via proximity based peer selection

By
Uriel Carrasquilla
April 2013

The time it takes to download a file in a peer-to-peer (P2P) overlay network is dependent on several factors. These factors include the quality of the network between peers (e.g. packet loss, latency, and link failures), distance, peer selection technique, and packet loss due to Internet Service Providers (ISPs) engaging in traffic shaping. Recent research shows that P2P download time is adversely impacted by the presence of distant peers, particularly when traffic goes across an ISP that could be engaging in P2P traffic throttling activities. It has also been observed that additional delays are introduced when distant candidate nodes for exchanging data are included during the formation of a P2P network overlay. Researchers have shifted their attention to the mechanism for peer selection. They started questioning the random technique because it ignores the location of nodes in the topology of the underlying physical network. Therefore, selecting nodes for interaction in a distributed system based on their position in the network continues to be an active area of research. The goal of this work was to reduce the cumulative file download time and variance for the majority of participating peers in a P2P network by using a peer selection mechanism that favors nearby nodes. In this proposed proximity strategy, the Internet address space is separated by IP blocks that belong to different Autonomous Systems (AS). IP blocks are further broken up into subsets named zones. Each zone is given a landmark (a.k.a. beacon), for example routers or DNS servers, with a known geographical location. At the time peers joined the network, peers were grouped into zones based on their geographical distance to the selected beacons. Peers that end up in the same zone were put at the top of the list of available nodes for interactions during the formation of the overlay. Experiments were conducted to compare the proposed proximity based peer selection strategy to the random peer selection strategy. The results indicate that the proximity technique outperforms the random approach for peer selection in a network with low packet loss and latency and also in a more realistic network subject to packet loss, traffic shaping and long distances. However, this improved performance came at the cost of additional memory (230 megabytes) and to a lesser extent some additional CPU cycles to run the additional subroutines needed to group peers into zones. The framework and algorithms developed for this work made it possible to implement a fully functioning prototype that implements the proximity strategy. This prototype enabled high fidelity testing with a real client implementation in real networks including the Internet. This made it possible to test without having to rely exclusively on event-driven simulations to prove the hypothesis.

Keywords: P2P networks, Content Distribution Networks (CDN), simulation, proximity based peer selection, random peer selection, node positioning system, zones, and landmarks.

Acknowledgements

This dissertation was made possible by the encouragement and support from my wife Susan, my advisor Dr. Liu, my dissertation committee and my employer Akamai Technologies.

My aspiration to complete the work came from my wife who never stopped believing I could do it. Without her, I would have faltered and left it as unfinished work.

I also wish to express my gratitude to Dr. Liu, my advisor and advocate, who challenged me every step of the way, kept me focused and guided me through all of the roadblocks to get here. With his enthusiastic supervision, constructive criticism, exceptional scientific knowledge and dedication inspired me to do the best I could be.

My appreciation is also due to the members of the supervisory committee, Dr. Simco and Dr. Mukherjee, whose knowledge and assistance to this study safeguarded the highest possible quality.

Last but not least, the faculty and students at Nova made my experience a very gratifying one. I look forward to crossing paths with most of the people I met again in what I hope is not a distant future.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Algorithms

List of Equations

Chapter 1

Introduction

Peer-to-peer (P2P) file sharing is a technology that can be used to download objects from the Internet. It is used mostly to download files that may contain documents, software packages, music, movies, or games. The peers are hardware devices capable of contacting each other via the Internet. Napster is an example of an early P2P implementation. It was designed on the concept of a central server. Later implementations included Gnutella and Kazaa. These later implementations moved away from the central server concept by distributing some of the functionality to the client software. A more recent P2P implementation is BitTorrent (BT). BT creates a new network known as an overlay for every file to be downloaded. This was a departure from the other implementations based on a single overlay.

Participating peers self-organize themselves to obtain a copy of a file. The focus is on a fast and efficient distribution of the file to all peers. There are no specific time targets set but each individual user makes a determination as to when the network is not performing. When dissatisfied, the user may abandon the download. The main issue to be discussed in this chapter is the observation made by researchers that the cumulative download time (CDT) in a P2P network is impacted when distant peers are present. A corollary also observed is that distant peers seem to relate to a high variance in download time for the population of peers participating in a file download.

In this work, a new peer selection framework is presented for selecting peers for interactions inside a P2P network with the objective of reducing unnecessary delays during a file download when distant peers are present. This chapter is organized as follows. First, the background leading to the problem is presented. This is followed by the goal established for this

dissertation. This chapter concludes with a discussion covering the barriers, issues, limitations, and delimitations of this study. The literature has a wide use of taxonomies and terms. For this work, the definitions of the terms to be employed can be found at the end of this chapter.

Background

This section contains an introduction to the concepts referenced in this research. In order to appreciate why distant peers have an impact on download times, it is relevant to discuss how the Internet is organized. Internetworking refers to the rules for connecting one network to another to form a larger network. The location where two separate networks exchange traffic is called a peering point. The aggregate of all the networks is called the Internet.

The Internet is a mix of private, public, academic, business, and government networks linked together via IP (Internet Protocol) for the exchange of packets. The Internet has become a global network owned and operated by Autonomous Systems (AS); a large number of interconnected corporations and Internet Service Providers (ISPs). The exchange of traffic between providers occurs through peering arrangements or inter-domain transit relationships at peering points. It is at peering points where Internet traffic is shifted between backbone operators and toward its destination. Peering points is also where a significant amount of packet congestion has been observed (Wang, 2007).

The top Internet access providers account for a small percentage of all the traffic. For instance, AT&T handles less than 5% of all the access traffic. It takes hundreds of network providers to handle half of the access traffic and thousands to handle all the traffic. This means that a centrally-hosted publisher making content available to end-users will require the traffic to travel over multiple access networks to reach its end-users. Also, it is a common practice for ISPs to offload received traffic not destined for their own network at the nearest (transit) peering

point. Since not every ISP is inter-connected with every other ISP, the offloaded traffic, that is traffic destined to another ISP, may end up circulating across multiple ISPs until it reaches its destination network. This nonessential tossing activity has an impact on the performance of the Internet by introducing delays in the delivery of packets. Since the owner of each of the networks is only interested in its own performance and not the overall Internet, this situation is unlikely to change any time soon.

A factor, not often mentioned, affecting delays in the Internet is BGP. Border Gateway Protocol (BGP), the protocol used for inter-ISP communication, uses hop counts for its route calculations regardless of the topologies, latencies, or real-time congestion of the underlying networks. Its main mission is to enforce networks' business agreements with each other rather than to provide good end-to-end performance. When routes stop working or connectivity degrades, BGP can be slow to converge on new routes. When routes are misconfigured, propagation of information can cause bloated paths and outages.

TCP can also contribute to delays. TCP, a widely used Internet protocol designed for reliability and congestion-avoidance, often results in poor performance in the presence of high latency or packet loss. TCP retransmissions can have a negative impact further slowing down communications. TCP may also become a bottleneck for large files because it requires receiver acknowledgements for every window of data packets sent. TCP throughput is inversely proportional to network latency. Thus, the RTT, using RTT as a measure of distance, between nodes can become the most significant bottleneck in download speeds.

Peers in P2P networks are only aware of the IP address of the other nodes. Therefore, measuring either geographical or RTT distance, a prerequisite to determining what may constitute a distant peer, is also a challenge. The concept of proximity, neighbors and

neighborhoods assumes an understanding of where nodes are positioned in the topology of the Internet so distances can be measured. But this understanding is difficult to obtain since the Internet is constantly in a state of flux and peering points can change without any prior notice making the topology of the Internet hidden from applications. In addition, geographical distances are not always an indication of either proximity or distance. Two nodes in the same city belonging to two different ISPs may be forced to exchange information via another far away city where the two ISPs have a meeting place (peering point) for the exchange of traffic.

Sherman, Nieh & Stein (2009) found that P2P performance is dependent on distance. The authors speculated that as packets transverse routers, delays are unavoidable. The explanation given was that a router is where congestion is likely to take place. The more routers a packet has to transverse, the more the chances packets may experience delays.

Peers may also be exposed to another kind of delays when going across ISPs. Additional delays may be experienced due to bottlenecks at the ISP peering points caused by either traffic congestions at peering points or ISPs engaging in P2P traffic throttling activities; that is dropping IP packets on purpose (Bindal et-al, 2006; and Laoutaris, Carra & Michiardi, 2008). For the case of TCP, lost packets must be re-transmitted, thus causing delays and waste in network capacity. The consequences from lost packets are longer network delays and a wider variance in the RTT as perceived by the nodes.

A solution to this traffic throttling practice would be for ISPs to abandon the practice. However, if P2P traffic remains high, then all ISPs would be required to invest more in routers and bandwidth to handle the extra traffic to mitigate delays. Since ISPs are not about to invest more capital to optimize the overall Internet and to handle additional P2P traffic from users given the fixed-fee structure in such arrangements, they justify their traffic shaping choice (i.e.

dropping P2P packets; thus freeing router's capacity) as necessary to avoid congestion, and thus prevent delays for their non-P2P traffic.

Researchers have also identified another suspect contributing to download delays. Peer selection is one of the key activities in the formation of a P2P overlay. In order to expedite the file download time, peers must find in the P2P network the needed pieces of the file being downloaded. Today, the de facto standard for P2P systems is to select peers at random without any concerns for the location/position of candidate nodes in the topology of the network (Sherman, Nieh & Stein, 2009). The approach for obtaining a list of candidate peers depends on the P2P implementation.

The P2P method to find candidate peers depends on the client software. For instance, in the Gnutella (2011) protocol, a client must find at least one other node for interaction. In some Gnutella implementations, a pre-existing IP address list of candidate peers is shipped with the client software. In some other versions, the Gnutella client contains the address of Gnutella Web Caches where the address of other active Gnutella nodes can be found. Once a client finds another active node, the client requests a list of active nodes (the IP addresses in particular) from the newly found active node. The Gnutella client selects nodes at random from its list of candidates and attempts to connect to each candidate until it reaches a certain quota. It connects to only as many nodes as defined in the quota, locally caching (i.e. a duplicate of the original data stored elsewhere in the system) the IP addresses it has not yet tried, and discards the addresses it tried but were no longer valid (e.g. the node may have left the system).

In the BT (BitTorrent, 2010) protocol, client nodes depend on the tracker to find each other. The tracker answer queries from client nodes for the list of IP addresses of other peers that are actively downloading a file. In BT, the term torrent is used to associate all those peers

downloading the same file. There is a separate tracker for each torrent and its job is to tell its clients, the participating peers in a torrent, where to find each other.

Therefore, as Chiu & Eun (2008) demonstrated, the number of hops for packets traveling between peers is directly related to the RTT. The RTT between two nodes can be thought of as a vector (with magnitude and direction) in the Internet Delay Space; in contrast to the concept of distance in Euclidian geometry based on the Cartesian coordinate system. This reasoning leads to the belief that the functioning and efficiency of the P2P system depends not only on finding peers that possess the needed pieces of the file but also on retrieving those pieces without unnecessary network delays.

Problem Statement

ISPs perceive P2P traffic as a threat to their service and engage in traffic shaping to free up their routers at peering points with other ISPs for non-P2P traffic (Laoutaris, Carra & Michiardi, 2008). Peering points often result in traffic bottlenecks and are subject to arrangements between ISPs with the details of such arrangements undisclosed to the public (Wang, 2007; Caida, 2011 and Liu et-al, 2010). Without the details of such arrangement, the topology of the Internet is hidden from applications for all practical purposes. The fundamental concern is not that ISPs engage in traffic shaping or the existence of possible congestion at peering points. The issue is the unnecessary delays that are introduced to the P2P network when candidate nodes for exchanging data are selected without taking into account their locality. This is a widespread issue given that the random peer selection is the most prevalent technique employed today by P2P client implementations. The random technique ignores the underlying physical network when a subset of nodes is selected from a large list of candidate nodes. More to the point, the random peer selection disregards Autonomous System (AS) membership and the

location/position of candidate peers in the topology of the Internet at the time when peers are selected for interactions (Laoutaris, Carra and Michiardi, 2008). Traffic shaping manifests to users in a P2P network as unpredictable network delays due to packets being dropped at ISP peering points (Laoutaris, Carra & Michiardi, 2008; Bindal et-al, 2006).

Bindal et-al (2006) studied this problem and observed that when traffic crosses ISP boundaries, many of the nodes cooperating to download a file were at risk of experiencing a significant higher download time. Laoutaris, Carra and Michiardi (2008) also studied this problem and they linked the delays and variance to the common practice of selecting peers at random without any consideration to their location in the topology of the underlying physical network. In a similar manner, Laoutaris, Carra and Michiardi, (2008) concluded that when distant peers are present, download times are negatively impacted. Their reasoning was that the delays were due to some packets having to transverse more routers with the consequences of an increased risk for packets to be dropped or for the packet to experience delays due to congestion along the way. Consequently, the mechanism for peer selection, the focus of this research, continues to be an active area of research.

The conjecture is that a bias for proximity during peer selection reduces both the cumulative and the variance in the download time in a P2P overlay. There is evidence supporting this conviction. First, it has been pointed out that reducing the amount of P2P traffic across ISPs will motivate ISPs to stop their P2P traffic shaping activities, thus reducing the cumulative download time (Biskupski, Cunningham & Meier, 2007). Second, Bindal et-al (2006) demonstrated via an event-driven simulation model that a reduction in the cumulative download time is conceivable when using a biased peer selection approach by giving preference

to nodes in the same neighborhood. However, the details of such an implementation were not sketched and were left as an exercise for future research.

The starting point for this research was to build upon the work of Bindal et-al (2006) by contributing a new proximity-based peer selection framework along with new algorithms capable of grouping nodes into neighborhoods. This biased selection reduces the chances of inter ISP traffic, the main source of congestion in the Internet, thus increasing the chances of improving the download time and variance for the majority of nodes participating in a file download. In the case where all the peers in a P2P are in the same AS and same location (e.g. same city), it is conceivable that the selection technique is irrelevant. But for those cases with widespread peers across the topology of the Internet, the more chances that the peer selection technique would have a positive impact on the cumulative download time in a P2P network.

The general guideline for a biased-based (e.g. proximity) peer selection technique is to build the overlay from the closest group of peers, closest in terms of location in the Internet topology, from a list of candidate peers (Cheng, Hutchinson & Ito, 2008; Dick, Pacitti & Kemme, 2009; Bindal et-al 2006; and Xie et-al, 2008). The challenge is that internal network information on which to base a biased selection such as the position of nodes in the network topology, bandwidth between nodes, RTT delays between nodes, number of hops, or packet loss rates, is not readily available to the peers (Han et-al, 2007). Therefore, before a biased peer selection solution could be developed, a peer positioning system was needed. For any biased/proximity solution to be widely accepted, it must produce a noticeable improvement compared to the random strategy, the baseline, in terms of the download time for the majority of the participating nodes. Also, to be of practical value, this new solution must have a small overhead for the levels of activity seen in today's P2P networks.

Goal

In the previous section it was speculated that it might be possible for peers to self-organize into subgroups based on proximity. The expectation is that peers in the same neighborhood could collaborate more efficiently, thus reducing their cumulative download time. This conjecture motivated the goal for this work. The established goal was to reduce the cumulative file download time and variance in a P2P overlay by employing a new proximity based peer selection mechanism that noticeably outperforms the incumbent random selection strategy. This goal has been pursued by other researchers (Bindal et-al, 2006). Missing from the literature were implementation details in the form of a framework or algorithms that could make it possible to build a software prototype for the purpose of conducting benchmarks to compare multiple peer selection techniques. Without an actual software implementation, high-fidelity simulations and benchmarks could not be conducted; a situation that so far has forced researchers to depend on analytical and event-driven simulation techniques when testing their hypothesis.

Once the framework was formulated, a prototype was developed so high-fidelity simulations and benchmarks could be conducted to compare the random versus the biased peer selection strategies. Same as Bindal et-al (2006), the comparison metric was the cumulative download time for all participating peers. The download time has been identified as a valid performance metric (Lehrfeld and Simco, 2010) and delay, particularly RTT, is often used to evaluate the quality of routing. The use of this metric for comparing techniques requires awareness of the fact that the delay can be incurred not only by the length and bandwidth of the link, but also by other factors. These other factors may include traffic congestion at intermediate nodes along the path, time delays due hardware processing packets and the reliability of the links along the path.

Relevance and Significance

P2P technologies for downloading (data/music/movie) files account for a significant share of the Internet traffic and has been estimated to be as high as 60% (CacheLogic, 2011). BT (BitTorrent, 2011) is one of the most popular P2P technologies given its widespread use within the Internet (Sherman, Nieh & Stein, 2009) with an estimated 18% share of the Internet traffic (Almon, Tran & Abhari, 2008; Bindal et-la, 2006). In addition, ISPs don't have a method to unilaterally reduce P2P traffic (Choffness & Bustamante, 2010). Reducing P2P traffic lowers operational costs for ISPs and improves performance for other non-P2P traffic (Biskupski, Cunningham & Meier, 2007). A successful implementation of the proximity strategy will free up network capacity since packets would travel a shorter path, therefore, reducing time spent visiting routers (an activity that consumes bandwidth and processing capacity) along their path.

There have been many attempts to implement the proximity strategy but their success is not obvious considering that the random peer selection strategy remains the incumbent and dominant deployed solution (Ledlie, Gardner & Selzer, 2007). This work furthers the body-of-knowledge (BoK) by contributing a new framework for modeling the Internet Delay Space so it becomes feasible to establish neighborhoods where the identity of nearby peers can be uncovered. Considering the large P2P share of the Internet traffic, the contribution from this work should become apparent to the reader.

Barriers, Issues, Limitations and Delimitations

A barrier to collecting data from the field is the probable resistance by users employing P2P client software over the public Internet to share information. Users will resist the idea of letting the software they are using take measurements with the intention of reporting results to a

remote location. The term used for this kind of add-on software is "spyware" (Lipschutz, & Clyman, 2012).

In addition, even when the proposed algorithms proved correct, it does not mean that users will embrace this new solutions and that proof of the advantage of the new proposed alternative is final. Experiments in the public Internet are difficult to conduct and securing networks in the thousands of nodes is not realistic. These barriers explain why other researchers, such as Bindal et-al (2006), had to employ simulation techniques to make their points. The warning that comes with this approach is that simulations are just models and not the real world itself causing any derived conclusion to be unconvincing. Paradoxically, it is not possible to present undisputable proof until the proposed prototype is widely deployed in the real world but wide deployment may never happen because there is no proof.

Another issue is that the real world is not an ideal environment for a scientific set up where controls are expected to be in place to measure the outcome from repeatable experiments. As we saw earlier, the Internet is in a constant state of flux. Therefore, experimentation and high-fidelity simulations can answer the research questions but doubts might persist. Nonetheless, by running experiments and simulations using a real P2P client implementation under the public Internet, it is expected that a high level of reality will be introduced for the purpose of gaining confidence with the results to be achieved and presented in this dissertation.

Summary

Proximity, an alternative to the random method for selecting peers for interactions in P2P networks, was presented. The objective is to reduce both the cumulative download time and what recent research is showing to be a wide variance in download times as experienced by a large percentage of nodes participating in a file download. This dissertation was conceived

upon the promising results obtained from the event-driven simulations conducted by Bindal et-al (2006) and their finding that a proximity based selection strategy can outperform the random selection strategy. What these authors left out as pending work was the software specifications needed to build a P2P client implementation that could be used for testing both the random and biased peer selection strategies in the real world. This was the starting point for this work: to develop a framework for implementing a biased peer selection framework along with a software prototype based on a new set of algorithms. The prototype was a prerequisite before any attempt could be made to compare the time it takes all of the peers to download a file when using either the proximity or random peer selection techniques within the public Internet.

The next chapter contains a comprehensive review of the literature as it pertains to P2P networks with an emphasis on peer selection techniques. The review will be followed by a detailed description of the methodology in chapter 3 for conducting the benchmarks to compare both selection strategies along with the framework and specifications for the software routines needed to implement the proximity strategy into a software prototype. Chapter 4 shows the design and results obtained from testing the two competing selection strategies. Conclusions are presented in the last chapter.

Definitions of Terms

Nodes, Clients, Server, Peers and Neighbors: a node is a hardware device connected to the Internet and capable of running one or more P2P software implementations. When a node is requesting services from other nodes, it is known as a client. The same node could be servicing requests from other nodes and because of this different role it becomes a server. When a node joins a P2P network, it is known as a peer. Peers participating in a file download are said to be

part of an overlay network on top of the Internet. The peers in a P2P network that are grouped into the same zone based on the algorithms presented in this paper are known as neighbors.

RTT, Delay Space and ASN: packet delays are measured with a metric named RTT. An RTT (round-trip delay) measurement between nodes is an option often used in the Internet as a proxy for the measurement of distance. For an illustration of this proxy, suppose measurements are taken for the time it takes to walk between various locations in an office building with several floors. Each floor in this building can be modeled as a 2-dimensional space with a small number of gateways (the stairwells and elevators) connecting the different floors. It should be apparent that two arbitrary points located on the same side of the building but different floors are not necessarily in close proximity. Given the matrix of measurements for the time it takes to walk to some but not all of the different locations in the building, how can a model be built to predict the time between any two arbitrary points in the building? From this model, the predicted time between two arbitrary points would be required to tell their expected proximity; thus RTT is a proxy for distance.

In a similar manner to the building, the Internet Delay Space (IDS) is also composed of Autonomous Systems (AS) like the floors in the building which meet only at prescribed gateways (peering points in the Internet). A number is used to identify each AS and either corporations or ISPs may be in control of one or more AS numbers (ASN). Different from the building, the IDS is not readily available to Internet applications and taking RTT measurements between all hosts in the Internet is a complex problem.

Packets and Pieces of a file in P2P Networks: in a P2P network, packets are pieces of a file that are transferred around the Internet using multiple nodes for the purpose of copying (a.k.a. downloading) a file from one node (source) to another node (destination).

P2P Network Overlays: in the overlay, peers are fully connected via logical links that correspond to a path which can perhaps involve many physical links in the Internet. The bandwidth and processing capacity of the peers can be different. Peers are likely to be dispersed over a wide geographical area with their interactions requiring each packet to transverse many routers along their path from the source to the destination node.

BT Tracker: an HTTP/HTTPS service which responds to HTTP GET requests. The requests include metrics from clients that provides the tracker with information that can be used to report statistics about the peers participating in a torrent. The response from the tracker is a list of other peers participating in a torrent. Client nodes use this list to find other nodes participating in a torrent.

BT Client: software used to download a file. It maintains state information for each connection that it has with a remote peer such as whether the peer is choked or interested.

Publisher: used to describe a person or group who makes documents in digital format available online over the Internet. The publisher is concerned with creating content, distributing content to as few as possible locations but yet meet customer expectations in terms of download delays.

CDNs: refers to a network of servers that delivers WEB content to users based on the geographic locations of the user, the origin of the Web page and a content delivery server. A CDN copies the pages of a Web site to a network of servers that are dispersed at geographically different locations, caching the contents of the page. When a user requests a Web page that is part of a CDN, the CDN will redirect the request from the originating site's server to a server in the CDN that is closest to the user and deliver the cached content. The CDN will also communicate with the originating server to deliver any content that has not been previously cached. This service is

effective in speeding the delivery of content of Web sites with high traffic and Web sites that have global reach. The closer the CDN server is to the user geographically, the faster the content will be delivered to the user under normal conditions without the risk of failures. CDNs also provide some protection from large surges in traffic but very little for routing around failures. The process of bouncing through a CDN is nearly transparent to the user.

ISP: ISP is the abbreviation for Internet Service Provider, a company that provides access to the public Internet for businesses and consumers.

User: there is the user perspective who wants to get the content for a reasonable price for the kind of service received. In this paper, the user is the individual who uses the Internet to access content created by publishers after it has been fully developed and marketed. Users are clustered together when they originate traffic from the same LAN or Internet network.

BGP: the Border Gateway Protocol is the core routing protocol of the Internet. Its role is to keep track of the reachability among active AS.

Failures: when link failures occur along a path from source to destination or when the level of performance is so bad that even when the link may still be up it is so congested that packets are being lost very fast and packets suffer extensive time delays.

Chapter 2

Review of the Literature

This section summarizes relevant investigations upon which this work is built to address the issues identified in the introduction. P2P applications have become widespread in the (public) Internet. An estimate for P2P traffic is as high as 60% (CacheLogic, 2011) with a more conservative 40% share (Lehrfeld & Simco, 2010). Content Distribution Networks (CDNs) based on P2P technologies are also gaining Internet traffic share. BT (BitTorrent, 2010) is one of the most popular P2P technologies given its extensive manifestation within the Internet (Sherman, Nieh & Stein, 2009) with an estimated 18% share of the Internet traffic (Almon, Tran & Abhari, 2008; Bindal et-la, 2006). Reducing P2P traffic must be undertaken by the client software since ISPs don't have a method to unilaterally reduce it (Choffness & Bustamante, 2008). Reducing P2P traffic will reduce routing activities leading to lower operational costs for ISPs while improving performance for other non-P2P traffic (Biskupski, Cunningham & Meier, 2007). Guo et-al (2007) determined that BT torrent populations are heavily skewed with an average of 102 and a maximum of 10,000 peers. Ten out of the 550 busiest public Trackers had a population of 1,000 peers. One torrent is born every hour and 20 peers register per hour.

In the Androutsellis-Theotokis and Spinellis (2004) classification scheme, the BT P2P protocol can be classified as a structured, partially centralized system given that it has a centralized torrent publication, centralized system coordination via the tracker, and decentralized peer communication. It is structured because of the mapping between content (i.e. pieces of the file) and location (i.e. the peers). BT is a P2P content distribution system with millions of nodes and with hundreds of thousands overlays running P2P networks on a daily basis (Liu et-al, 2010). The most important BT algorithms are documented in Appendix A.

Lehrfeld and Simco (2010) grouped peer selection algorithms into two camps: random peer selection from a list of candidate nodes and peer selection using ranking based on predetermined metrics. Performance metrics can be based on RTT (round trip time) delays, bandwidth, Euclidian proximity between the nodes and other metrics. This grouping is consistent with the competing alternatives explored in this work. The incumbent is the random peer assignment strategy since it is the de facto implementation. The new approach is the proximity biased selection strategy that uses ranking as a criterion for selecting peers. It is new since biased-selection is missing in the list of the most popular BT implementations (Sherman, Nieh & Stein, 2009).

This chapter is organized as follows: first CDNs are introduced with a discussion on how P2P technologies are being absorbed into the traditional client-server CDNs. This discussion is followed by a review of the random strategy, the proximity strategy, a performance comparison between both peer selection strategies, and relevant research that address the issue of locality in the Internet, the cornerstone for any node positioning system. The published research in this review constitutes the state-of-the-art when it comes to modeling the Interned Delay Space; the basis for the proposed proximity-based framework to be discussed in chapter 3.

Content Distribution Networks (CDNs)

CDNs build their file distribution systems on the Internet with the intention to reduce latency in the delivery of Web content to users spread around the world. They can reduce latency by replicating Web in servers, called "caching servers" or "edge servers," located in geographically dispersed locations, as close as possible to the consumer of the Web content. CDNs rely on intelligent DNS (Domain Name Servers) when re-directing users to a nearby server. In that sense, CDNs have implemented the concept of proximity to reduce latency.

CDNs have evolved along two types of architectures: client-server and P2P. More recently, hybrid content delivery services combining features from both architectures have been introduced. In the case of CDNs employing P2P networks, the time it takes to download files is impacted by the peer selection algorithms in use by the client software. CDNs are a success story considering its 15% share of the Internet traffic, may be one of the most significant Internet applications (Salmon, Tran, & Abhari, 2008; Le & But, 2009). In this section, CDNs are described to show the evolution from client-server to P2P architectures.

Each CDN is a network of servers that delivers WEB content (e.g. a file) to a user based on the geographic location of the user, the location of the initial server where the Web content was published and the location of the content-delivery server that may have a cached (i.e. duplicated) version of the original content. A CDN copies the content of a Web site to a network of servers that are dispersed at geographically different locations, caching the content at strategic locations in the Internet.

When a user requests a Web page that is part of a CDN, the CDN will redirect the request from the originating site's server to a server in the CDN that is closest to the user and capable of delivering content that has been cached. The CDN will also communicate with the server where the original WEB content was stored for those cases where content has not been previously cached. Lately, P2P technologies are being explored to shift the CDN network cost to the users (Yin et-al, 2010).

Peer-to-peer CDNs (P2P-CDNs) consist of nodes running client software that enable file downloads to be completed by all participating nodes. Different from the traditional CDNs where surrogate servers provide persistent services, P2P-CDNs relies purely on voluntary peer participation.

Hybrid CDNs (HYBRID-CDNs) share the same characteristic with P2P-CDNs in that peers contribute uplink bandwidth capacity to other peers. But different from P2P-CDNs, HYBRID-CDNs share features with traditional CDNs such as "surrogate servers", which are added to P2P-CDNs in an attempt to improve download efficiencies. The random peer selection is the dominant implementation in P2P based CDNs such as in the Flower-CDN and LiveSky (Dick, Pacitti and Kemme, 2009; Yin et-al, 2010). The issues raised in the introduction apply to both P2P-CDNs and HYBRID-CDNs.

The Random Strategy

The objective for the random strategy is for each peer to select at random a subset of peers from a large list of candidate peers. A partial list of BT implementations using the random strategy include names such as BitComet, BitLet, BitLord, BitSpirit, BitThief, BitTornado, BitTorrent, BitTyrant, Deluge, FlashGet, Folx, FrostWire, KGet, KTorrent, Lftp, LimeWire, MiniGet, Miro, MLDonkey, µTorrent, OneSwarm, Opera, qBittorrent, rTorrent, Shareaza, SymTorrent, Tixati, TorrentFlux, Tribler, Vuze, Wyzo, Xtorrent, Xunlei and ZipTorrent. This section shows what other researchers have done to improve this strategy and what they have to say about non-random based peer selection strategies.

After reviewing the most active P2P client implementations based on the BT protocol, Sherman, Nieh & Stein (2009) found that the random peer selection strategy is used exclusively.

The theme about the superiority of the random selection was documented by Bindal et-al (2006). These authors found that most analytical and simulation studies attributed the success of BT to its near optimal algorithms and most researchers doubt that there was an option to the random technique.

Download times can be impacted by the service capacity of a peer (i.e. the capacity to upload packets to other peers) and by the fluctuations in the same service capacity of each peer over time (Chiu & Eun, 2006; Chiu & Eun, 2008; and Lehrfeld & Simco, 2010). These studies focused on improving the random strategy by evaluating the points in time as to when a new peer should be selected for interaction at random. These authors considered their options to be: never change peer, change after each piece of a file is downloaded, change after a period of time has elapsed, or download from several peers in parallel. These authors found that the different selection algorithms they considered are only sensitive to both fluctuations in service capacity and the capacity of peers. These authors neglected the possibility of congested peering points or ISP traffic throttling as an explanation to their observed fluctuations in service capacity. In other words, the impact from distant peers was not taken into account, even when wide variations in download times were present in their reported results.

In a separate study where the goal to improve the random strategy was successfully achieved, Lehrfeld and Simco (2010) proposed that when a client is dealing with a peer that starts to experience a degrading performance, the client should switch to a new peer to be selected at random. These authors were concerned that any rank based peer selection strategy would not guarantee of a bottleneck-free connection and indicated that proximity does not provide for the best possible download duration for clients. Lehrfeld and Simco made reference to other studies that had come to the same conclusion.

The results and comments from researchers in this section reinforce the belief that the proximity based peer selection would not improve the cumulative download times in a P2P network. However, if the reduction of distant peers in a P2P overlay was to result in an improved cumulative download time, there would be a role for the proximity strategy.

The Proximity Strategy

The objective for the proximity strategy is to build the overlay from the closest group of peers to a reference node from a large list of candidate peers (Cheng, Hutchinson & Ito, 2008; Dick, Pacitti & Kemme, 2009; and Xie et-al, 2008). The expectation is that a biased peer selection that gives preference to candidates in close proximity can reduce latency and network load.

Bindal et-al (2006) wondered if BT would still perform well under a different selection strategy. In other words, these authors were wondering if the random selection was a necessary condition for the success of the BT and near optimality of its algorithms and opened the possibility for a ranking based peer selection. These authors conducted benchmarks using an event-driven simulation tool to show that a biased neighbor selection based on proximity should maintain the near optimal performance of the BT networks; thus challenging the wisdom that random selection is a necessary condition for the optimal performance of P2P clients. The authors offered no guidelines on how a proximity-based technique could be implemented but a significant amount of a real P2P software code was incorporated into their simulation model; thus bringing a high level of credibility to their results.

The proximity strategy has its own challenges that have proved difficult to overcome (Wong, Slivkins & Sirer, 2008; and Han et-al, 2007). According to these authors discovering nearby peers is a hard search problem given the efficiency and scalability expectations from the client software and the limited network information available to the client. These authors indicated that the P2P client software must have a small-complexity (execution time and memory space); which is lowest by simply selecting peers for interactions at random. This argument is

undisputable. Any approach that requires keeping information or history about other nodes is bound to increase complexity and overheads.

As pointed out in the introduction, the distances between nodes are not obvious given the fact that the topology of the Internet is distorted by the presence of transit links. Contributing to the challenge is the fact that the topology and transit links are constantly changing and very large with over 19 thousand Autonomous Systems (AS), about 8,800 ISPs, 44 thousand transit links and nodes in the millions (Caida, 2011; Cidr, 2011; and Liu et-al, 2010). The number of candidate peers can also be large, often in the thousands and spread across the Internet (Zhang et-al, 2010). Abrahao and Kleinberg (2008) concluded that a compact representation of the Internet is an NP-hard problem because of the transit links.

The uncovered challenges have lead researchers to focus their energies in building a compact model of the Internet Delay Space (IDS) that captures the nature of the Internet topology where latency is used instead of Euclidian distance as a metric; a topic to be reviewed later in the locality of the Internet section. Latency measurements are expected to contain some distortion (i.e. error) when taken between nodes but may prove more reliable as a proxy for proximity between nodes that the typical distance measurements using geographical latitudes and longitudes on the map.

Since most attempts to select peers based on proximity must deal with the fact that the Internet topology is hidden and cannot be directly measured, the concepts of hidden metric spaces and topology are presented before comparing the random and proximity strategy. At that point we will regress to review the cornerstone issue of any proximity based solution, the positioning of nodes in the Internet based on the concept of "locality in the Internet".

Hidden Metric Spaces in the Proximity Strategy

The geography of the Internet has a resemblance to the geography of the earth; it observes the borders between nations and follows the edges of continents, particularly those continents next to oceans where most inter-continental connections are laid. But this resemblance is only coincidental.

The Internet is a collection of interconnected Autonomous Systems (AS) where corporations and ISPs connect users to the rest of the Internet via wholesale sellers of network bandwidth. It is conceivable that corporations and ISPs control one or more AS (an AS is represented by a number and known as an ASN). It is known that cross-AS or cross-ISP traffic is exchanged at designated interconnection (i.e. peering) points via transit links (either peering or transit arrangements). These transit links have an impact on the geometry/topology of the Internet causing the map of the Internet to be in a state of flux. Therefore, geographical distance from latitude/longitude is not always the best indication of proximity between nodes and there is no easy method to determine the topology of the Internet. This situation is not specific to the Internet and has been the source of research.

The Cooperative Association for Internet Data Analysis (CAIDA, 2011) has been working on the idea of modeling the geometry of metric spaces that are not visible but can be inferred by other observable means (refer to Figure 1). They call it Hidden Metric Spaces (HMS) and the modeling is called "geometry-under-topology". The insight that can be gained from this work is that the HMS is akin to the Internet and the Internet topology could be revealed in the latency between nodes. The latency is a metric that can be obtained by measuring RTT latency between nodes in the Internet. The advantage of the RTT latency is that it is independent of the geographical location of the nodes but these two metrics may be correlated. These

assumptions and speculations are put forward for consideration since most of the research to be reviewed shortly in the 'locality of the Internet' section deals with this issue of hidden metric spaces. But before the issue of positioning nodes in the Internet based on locality can be elaborated, there is another science that proved relevant: topology.

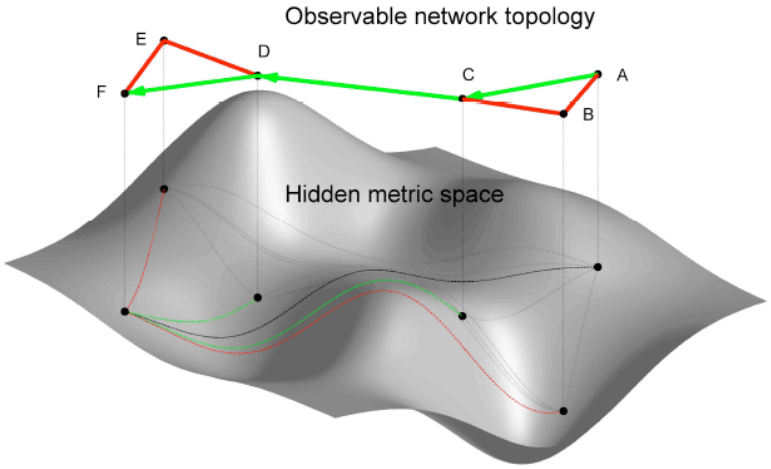

Figure 1. Hidden Metric Space

The topology of the Internet

The concepts of distance, neighbors, proximity and spaces have a mathematical reference. There are two branches of mathematics relevant to this study. First is geometry, a subject covered by Flegg (2001) and summarized in this section. Geometry is mostly concerned with the study of specific properties of figures in a given space. In one such space, the Euclidean space, the distance between two points (or nodes in a network) can be calculated by using the Cartesian co-ordinate system where the two points are placed and a distance formula applied. The distance is a metric that follows a set of rules. One such rule is that the distance when measuring from point A to point B should produce the same results as when measuring in the opposite direction. Euclidean geometry concerns itself with finding the different type of figures on a space. When two figures have one or more different properties then the figures are said to be different. Non-Euclidean geometries allow for more flexibility with variability in the property

of figures by relaxing rules of length, angles, shapes and permitting projections. In all these geometries regardless of the type of space, a straight line must remain a straight line for its axioms, rules and formulas to work.

When properties are relaxed beyond those that apply to geometry, topology, the second branch of mathematics, can be referenced. Topology deals with with spaces (Flegg, 2001) where one-one and bi-continuous transformations are permitted. In such transformations, each point is transformed to one-and-only-one point in the transformed figure with the characteristic that near points remain near in the transformed figure, that is, neighborhoods are preserved if the bi-continuous transformation is to hold true. Beyond transformations that are possible in geometry, figures can be stretched, bended, and twisted (elastic deformations). However, cutting is not allowed unless restored (repaired) in such a way to retain the "nearness" of the original points in the figure. Also, in topology, the measurement of distance is not involved; just the spatial relationships are considered. So how does topology relate to the Internet?

Applying the topology concepts in mathematics for vertices, points, regions, spaces, paths, maps, and networks, when routers/hosts (vertices or points in topology) are activated on the topological surface of the planet earth by a given AS along with links (connections or arcs in topology) between routers, a topological network comes into existence. When another AS is interconnected via peer-points (specific routers from both AS with a link between them), one network is over-imposed on top of the other network on the surface to create a new topological network. If each AS operated a separate network without a peering arrangement, and if only two networks existed on the planet, then two topological regions would exist. The nodes/points inside each of the topological region could not connect to nodes/points in the other topological region.

In the case of the Internet, once the two AS connect, then every node that uses the same IP scheme (e.g. v4) would connect to every other node via a topological path. A topological path consists of one or more links (arcs) that must be traveled from node to node to get from the source node to the destination node. The Internet has its own protocols to decide which nodes are selected along the path. Once you add more Autonomous Systems to the topological network, some of them may not be adjacent creating the situation where multiple "regions" will exist. The topological surface is no longer the geographical surface because there is an overlap in terms of geography (longitude/latitude) between two or more Autonomous Systems. Nonetheless, for each protocol in the Internet, every node must have a path to every other node even when the path transverses multiple regions. Topology and its mathematical terminology can be used to describe the topology of the Internet. Location and proximity can be discussed in the context of neighborhoods, without the need for a Euclidian-distance metric.

In a nutshell, the conditions for a metric space are as follows. If X is defined to be a non-empty set for which a non-negative real number d(x,y) is defined for all elements of "x" and "y" that belong to X that satisfy the following four conditions in Table 1, then "d" is termed a metric and X along with the metric "d" can be said to be a "metric space".

Table 1. Conditions for a metric space

1. Distance d(x,y) is greater than or equal to 0
2. Distance d(x,y) = 0 if and only if x=y
3. Distance d(x,y)=d(y,x)
4. Distance d(x,y) + d(y,z) is greater than or equal to d(x,z)

The problem with the Internet is that not all d-metrics can honor conditions 3 and 4 in Table 1. The RTT delay between nodes is bound to fail the last two tests in Table 1 because the time delay between node A and B is different when going in the opposite direction. Without such d-metric, the concept of nearby nodes, based on the idea that x and y are close to each other

if d(x,y) is less than a threshold to be defined arbitrarily, is not mathematically possible to calculate. But the concept of neighborhoods when using the IDS is relevant and can be used to discuss proximity between nodes in the Internet. Topology and IDS are essential concepts before a rank based peer selection strategy can be implemented and when looking for a framework (chapter 3) to solve the problem of positioning nodes in the topology of the Internet.

The random versus proximity performance comparison

This section presents the simulation results obtained by Bindal et-al (2006). The methodology and experiments from these researchers are for convenience summarized in this section.

These authors asked themselves if different peer selection mechanisms would have an impact on the time it takes to download a file as experienced by all participating peers. The authors found that the time needed for half the users to finish downloading the file is almost the same regardless of the peer selection mechanism. For the other half of the users, the authors observed a higher variance in download times when the random peer selection mechanism was employed.

The employed simulation demonstrated that the proximity based neighbor selection strategy maintained the nearly optimal performance of the random P2P peer selection mechanism while reducing the amount of cross-ISP traffic. The main evaluation metrics were download time and ISP traffic redundancy.

Measurement of download time included the cumulative distribution function (CDF), the 50th percentile value and the 95th percentile value. “ISP traffic redundancy” refers to the average number of times the blocks of the content file travels into the ISP until all peers inside each involved ISP complete their file download. What that means is that the lower the

redundancy, the lower the cross-ISP traffic is bound to be. The lowest redundancy was 1. The highest redundancy was N, where N is the number of peers inside the ISP.

The experiments examined two network settings that represent a typical P2P network: a homogeneous network consisting of 700 cable modem nodes spread among 14 ISPs, and a heterogeneous network consisting of a number of university nodes and 700 cable modem nodes. Since university nodes don't belong to an ISP, their inter-campus traffic is not expected to be exposed to traffic shaping. Bearing in mind that the owners of an ASN are either corporations or ISPs, this heterogeneous network is a good representation of the different types of AS. In both cases (ISP/university), the original seed (i.e. the publisher) is a separate node with variable bandwidth. The authors use the term "regular BT" and "biased BT" to mean the random peer selection and the proximity peer selection strategy respectively in BT type of P2P networks. Some of their results are duplicated here for convenience.

The normalized download time and traffic redundancy under ISP bandwidth throttle in a homogeneous network is shown in Table 2. The publisher is a separate node with 400 kilobit per second (Kbit/s) of uplink bandwidth. When looking at the left side of the table, the download time results are presented as a ratio between the download time and a base download time (i.e. normalized), which is the 50th percentile download time for the no bottleneck case at 5,321 seconds. It can be seen that the download time is impacted by the amount of throttling done by the ISP (i.e. the bottleneck in the table to the left refers to the amount of BW allocated for P2P traffic).

When looking at the right side of Table 2, the download time and traffic redundancy of random (i.e. Regular BT) versus proximity (i.e. Biased) is shown. The "k" parameter shown for the biased (i.e. proximity) strategy refers to the number of external neighbors (i.e. neighbors

belonging to a different ISP) as they are varied from 1 to 17 out of a maximum of 34. The simulation shows that the proximity strategy reduces the variation in the download time among all participating peers. An interesting finding was the similarity in the observed median for the download time between both strategies (5,313 and 5,220). But when the 95th percentile is considered, there is a significant reduction in the download time and since the objective is not the median (download time for half the users) but the overall experience of all involved peers, a drop in the cumulative download time is an encouraging result for the biased technique.

Table 2. Random versus Proximity Results (1 Mbit/s = 1 megabit per second)

Bottleneck	50th%	95th%	Redundancy
None	1	1.35	46.9
2.5 Mbit/S	1.43	1.59	31.76
1.5 Mbit/S	2.01	2.05	24.88
0.5 Mbit/S	3.33	3.53	21.65

Peer Selection	50th%	95th%	Redundancy
Regular BT	5,312	7,152	46.72
Biased k = 1	5,168	6,206	3.44
Biased k = 5	5,172	6,281	9.74
Biased k = 17	5,220	5,872	21.38

It should be emphasized that these experiments were based on an event driven mathematical modeling to simulate the two strategies. The methodology for conducting experiments and the metrics to use for measurements were adopted for this work.

Given the proximity advantages uncovered by the simulation and the fact that an Internet positioning system depends on being able to determine the location of nodes in the network, the focus of this review now shifts to see what other researchers have done to be able to assign nodes to neighborhoods or coordinates in the Internet; a pre-requisite to any ranked based strategy.

Locality in the Internet

A prerequisite to a proximity strategy is the capability to position nodes in the topology of the Internet so nodes from the same neighborhood can find each other to form an overlay. Researchers have come up with different ideas on how to position nodes but no such system is

widely available yet. This section will review such ideas and explain how they have contributed to the framework to be discussed in chapter 3.

Locality is not something the designers of the Internet had envisioned (Han et-al, 2007). The Internet's priorities are scalability, packet forwarding, and a best-effort strategy for delivering packets without warranties; neither topology nor distances or delays between nodes are mentioned. It has been observed that node performance and availability problems occur with some frequency in the Internet. The cause of most path failures has been router configuration mistakes, power outages and fiber line damages (Zhang et-al, 2004). Anomalies and failures cause increased BGP (Border Gateway Protocol) traffic, higher packet loss, and packet time delays. Since there is neither a central authority to monitor all Internet paths nor a widely accepted model of the Internet, detecting anomalies and failures are left to the applications.

TCP is a protocol which recovers by re-sending lost packets after a time threshold. Finding alternate routing paths from source to destination is complicated by the fact that many hosts lack multiple paths and most Internet failures occur near the destination host (Zhang et-al, 2004). The observed lack of symmetry in the Internet Delay Space (delay from A to B is not always the same as from B to A) is most likely due to failures, anomalies, congestions and recovery capabilities in some of the applications.

Measuring and keeping information such as latency between all routers/hosts participating in the Internet is not a good idea (Madhyastha et-al, 2009; Abrahao & Kleinberg, 2008). Madhyastha et-al proposed a different approach based on building a model of the Internet that could be hosted as an Internet service. The model could be used to make predictions about "link" properties between end-hosts; properties such as delay (RTT), packet loss and bandwidth. The original intention for this work was to incorporate this service into the prototype developed

for this work and if performance was acceptable, it could have been used to estimate dynamically the RTT between nodes.

The University of Washington is currently hosting such a service as described by Madhyastha et-al (2006) under the name of iPlane. The methodology employed in iPlane is as follows. A path composition technique is performed to make path predictions. Specifically, to predict the path from a source node to a destination node, the path composition technique composes two path segments that intersect with each other. The first segment of the path is from the source node to an arbitrary destination known as a vantage point. The second segment of the path is measured from one of iPlane's vantage points (a.k.a. landmarks) to the destination's prefix of the IP address. Depending on which intersecting pair of segments is chosen, the path obtained by composition is often similar to the actual route from source to destination. There is a cost to this approach due to the effort in building and keeping current an atlas of the vantage points. The atlas tends to grow very large as more and more vantage points are added, making it difficult to keep such a large atlas as data in each client node for routing decisions. This is a problem for a prototype requiring a small foot print.

To get around this limitation, instead of using an atlas of measured paths for all vantage points as done for iPlane, another similar service named iNano was introduced. In this service, an atlas of measured links is also introduced but is controlled for growth. The space required in iPlane is proportional to the number of vantage points while in the iNano system the representation of the vantage points, about 3,000, requires a storage space which is linear in the number of nodes and edges in the underlying Internet graph. Accordingly, the iNano's atlas fits in less than 7MB, almost three orders of magnitude smaller than atlas for iPlane; making it viable

for the iNano atlas to be distributed to lightly powered end-hosts. This amount of data would work for the case of keeping such information in the BT tracker.

The key challenge in making this approach work is to make accurate predictions about Internet path performance from an atlas of observed links. Setting up the query facility for iPlane on a P2P host proved easy. However, when testing for metrics between popular Internet hosts, the service would fail to provide any results. As a result, neither iPlane nor iNano could be used for this work. Nonetheless, the idea of vantage points to be known as landmarks is relevant for the development of the framework (chapter 3) given its reduced data footprint requirements.

In another research, a network-coordinates study using the Azureus (a.k.a. Vuze) BT client was conducted (Ledlie, Gardner & Seltzer, 2007). Coordinates were introduced to model latency. The authors observed that coordinate based systems show improvements over the default random assignment of peers. The coordinate system started with the geographical location and with a latency consideration. The concept of neighbor decay was introduced as a means to keep delay information obtained from the peers themselves as current as possible by paying more attention to the most recent delay measurements. Peers that were distorting their statistics had to be removed on the belief that they were outliers.

Azureus presented challenges during the collection of data to create the models: chatty nodes, relative errors, stability of measurements, drifting centroids, intrinsic errors and variance. Chatty nodes caused application messages to skew measurements. By weighting recent communication more heavily they solved this issue. The relative error, that is the difference between the predicted and actual latencies experienced between two hosts, was not solved. The stability, that is how static the measurements were (unless the network was actually changing),

was not solved. Churn caused problems but they were able to deal with it by starting nodes at their last known coordinates. Drifting centroids occur when the coordinates started to move away from the origin. To get around this issue, gravity to pull the centroids back to the origin was introduced. Intrinsic errors were caused by some bad nodes that eventually had to be removed from the list of active nodes.

At the end, the variance captured by experiments using Azureus was found to be high making it difficult to decide if predictions were to be made regarding the median, mean or a statistical distribution. Today, Zaureus/Vuze employs the random peer selection technique. For this work, the concept of using fixed-coordinate systems was discarded but the idea of starting with the geographical location (latitude/longitude) and the concept of centroids proved to be relevant to the framework (chapter 3).

In a different study, also based on the coordinates approach for placing nodes on a map, Lafon and Lee (2006) looked into the issue of determining the intrinsic geometry of the Internet. As many other researchers have done, latency measurements between pre-selected nodes was collected and stored. These authors were looking for a system of coordinates that can be described from the collected data without reference to an outside structure (i.e. coordinates and locality to be derived from the data itself without any reference to any arbitrary external coordinate system). Their approach employed spectral properties of a pair-wise similarity matrix and graph theory to reduce the number of dimensions (geographical coordinates were two dimensions) to consider while preserving the relevant information needed to measure distance.

Computational complexity was reduced by working with a subset of the original graph created from the data. This approach for reducing complexity is similar to the concept of selecting well thought out nodes that can be used as landmarks from which the relative position

of nearby nodes can be estimated. This concept of reducing complexity by the use of landmarks to model the Internet based on the delays between the same landmarks was observed to be a common thread expressed in the literature.

Lafon and Lee (2006) demonstrated using analytical tools that computational complexity can be contained by reducing the number of nodes from millions to thousands and yet retain the characteristics of the Internet. These authors confirmed that global extrinsic distances such as geographical or Euclidean space are meaningless since they do not help explain the structure or geometry of the Internet. Since no prototype was made available and algorithms were not provided, the theories presented can only be used to define the framework to be used for positioning nodes in the Internet's topology. The idea of landmarks and relative positioning of nearby nodes with respect to those same landmarks was borrowed for the framework (chapter 3).

Also, supporting the idea of an Internet positioning system based on the concept of modeling the Internet Delay Space (IDS), Abrahao and Kleinberg (2008) indicated that a matrix "W", the matrix with distance measurements between Internet hosts, can be embedded with low distortion (i.e. low cumulative errors) between predicted and actual distances into a Euclidian space with multiple dimensions (besides dimensions such as geographical distance between hosts, other dimensions such as number of crossings at peering links, number of hops, number of ISP crossings, location in the network with a higher value when node is a user on a cable connection, packet loss rates, transmission medium such as via satellite, etc.). The objective was to explain the observed delays between nodes against the predicted delays (predictions from models to measure distance between nodes in the Internet). The "W" matrix consisting of RTT delays between nodes and geographical locations (landmarks) was the concept adopted for the framework (chapter 3).

Furthermore, Abrahao and Kleinberg (2008) found evidence (with statistical significance) that the presence of transit links do indeed have a direct influence on the delay space geometry of the Internet. These authors advised that modeling based solutions that are sensitive to the structural properties of the Internet's space geometry are bound to be superior in their predictions. These authors indicated that the AS topology is reflected in the IDS, that peers within the same AS require a lower number of variables (dimensions) to keep predictions within an acceptable amount of error, that the delay space exhibits a non-linear relation to its variables and the geographical location of nodes is a strong predictor/component of the IDS. The idea of a W matrix and that the delay space captures the topology of Autonomous Systems became the foundation for the framework (chapter 3).

Summary

There is no consensus on the criteria for peer selection but from this review it is clear that many researchers have found the biased approach to be a new alternative to the incumbent random technique. What remains an open problem for a possible implementation of a biased peer selection technique is the framework for modeling the concept of "positioning nodes by proximity in a neighborhood" based on a compact representation of the Internet Delay Space that accounts for the topology of the Internet so that nodes can be placed into neighborhoods. Once that framework and modeling is available, then the concept of finding nearby peers becomes an option for applications in the Internet.

Chapter 3

Methodology

Introduction

The research conducted by Bindal et-al (2006) demonstrated via an event-driven simulation that by introducing a new biased approach for peer selection in P2P networks it is conceivable to reduce the cumulative download time for the participating nodes. Bindal et-al proved that the biased peer selection performs as well as the random peer selection for the 50^{th} percentile of the participating peers and outperforms the random strategy for the 95^{th} percentile. The comparisons were based on the cumulative download function (CDF) for the time it takes participating nodes to download a file. The function is presented in Equation 1. In this equation, the observed download time from experiments determines the f(t) function. F(x) is the area under the curved from f(t) from zero to the value of "x".

Let x be the download time for a given peer.
F(x), the cumulative distribution function is

$$F(x) = \begin{cases} 0, \forall\ x\ < 0 \\ \int_0^x f(t)dt, \forall\ 0\ \leq x\ \leq max \\ 1, \forall\ x > max \end{cases}$$

Equation 1: CDF for Download Times

This work recreated the scenarios considered by Bindal et-al with some differences. A high-fidelity prototype was developed so benchmarks could be conducted over the public Internet instead of testing with event-driven simulations. The advantage with the prototype approach is that a computer based full functionality simulation could be conducted to include the entire client software as it is intended to work in the real world. Furthermore, once the prototype is

functional, it becomes possible to verify the design and functionality of the proximity based algorithms.

This high-fidelity testing approach improves the performance assessment when two selection strategies are benchmarked. Since the simulation can only approximate the real world experience, it became critical to run experiments using the public Internet to account for real world events. The algorithms, methods and procedures employed in this study are fully documented in this chapter to enable other researchers to reproduce this work.

This chapter is organized as follows. After introducing the model for the Internet Delay Space (IDS), a framework along with the algorithms for positioning nodes in the network is presented. The framework is followed in chapter 5 with the approach to be taken to prove that the algorithms work as intended. This proof can be obtained via a simulation with a very simplified P2P network and with the aid of a load generator (a driver) that is capable of simulating a multitude of nodes downloading a file.

Modeling the Internet Delay Space

The IP protocol defines the addressing of packets. An IP 32 binary digit is a number in a one dimensional space. IP addresses are displayed in notations such as 172.16.254.1 or A.B.C.D in general where each letter represents an IP class. There are 256 class-A IP ranges with some of them reserved, other set up for multicast and the remaining controlled by corporations (Iana, 2011). Since any discussion about topology requires a dimension higher than one, the first step was to convert this IP one dimensional space into two dimensions. This was possible to achieve with Hilbert's curves (Hilbert, 1891) and the fact that an IP address A.B.C.D is bound by four curves where each one ranges from 0 to 255. Therefore, a subset of IP addresses that belong to

an AS can be represented by a polygon in this two dimensional space. Still pending in this space were the spatial relationships.

A spatial relationship was introduced with the objective to simplify the entire Internet to a relatively small number of landmarks/beacons as done by Lafon and Lee (2006). In this space, nodes and landmarks were ranked by city, region and country in that order. Then, a node was associated with landmarks based on the ranking. The highest ranking gives the node the name of its closest landmark. Those nodes that share the same landmark are said to be members of the same zone. Thus landmarks could be used to position all other nodes into neighborhoods (i.e. the zones) one AS at a time. The network delay between landmarks (a.k.a. vantage points) was based on the work by Abrahao and Kleinberg (2008); particularly in reference to their proposed W matrix (RTT measurements between landmarks). Once the W matrix between landmarks was established, a model of the Internet Delay Space (IDS) became within reach.

In the IDS, landmarks/beacons in their roles as centroids of zones were used to cluster arbitrary nodes into neighborhoods. This was done one AS at a time and one node at a time based on geographical location of the nodes and landmarks. From a set of candidate landmarks, a node's closest landmark could be finalized based on the RTT taken a run time (using ping, a Linux command to measure RTT delays to another node) between the node and its geographically nearby landmarks. The IDS provides the foundations for the framework and algorithms in this research.

Framework and Algorithms

Bindal et-al (2006) demonstrated that the proximity based peer selection approach is promising for removing the unpredictability of the download time for the majority of the participating peers regardless of their location in the topology of the Internet. In addition,

Abrahao and Kleinberg (2008) established that the topology of the Internet is reflected in the Internet Delay Space (IDS) and also that the geographical location of a node is a strong component of the IDS. These findings constitute the foundations for the framework described in this section.

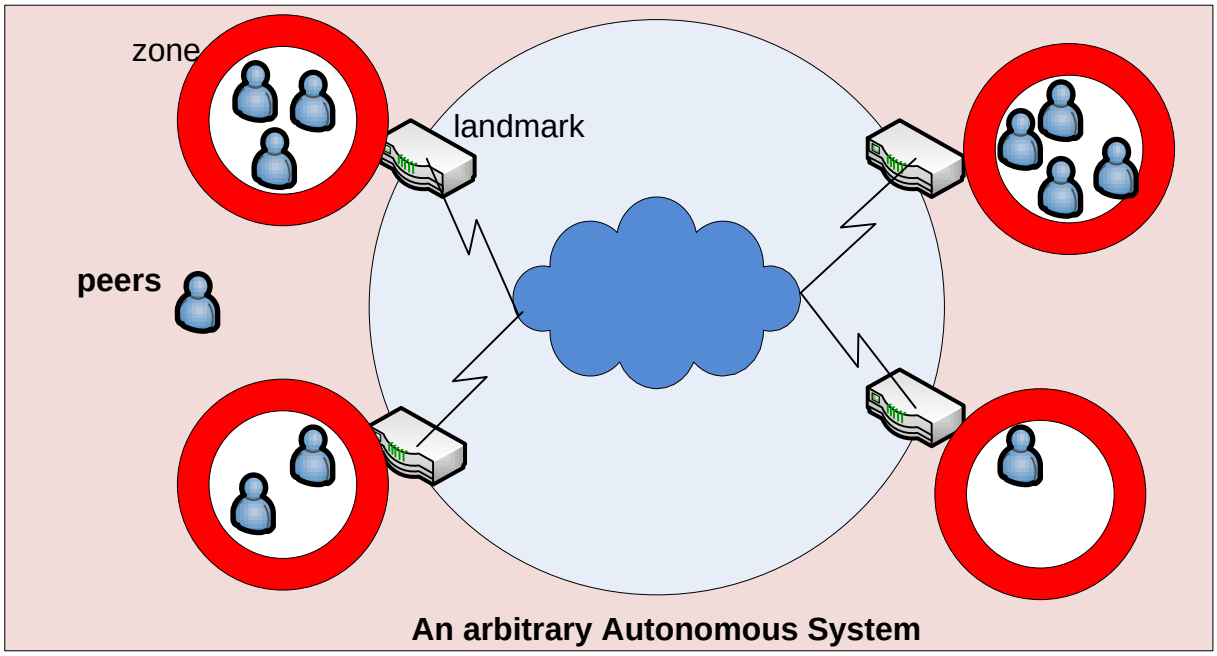

Figure 2. Pre-selected Landmarks and Zones for one AS

The framework is as follows. Since an AS has a subset of all valid IP addresses, the Internet IP address space can be split into multiple (mutually exclusive) subsets, one per AS. The nodes in each AS (based on the IP address of the node) are confined within a topological figure that looks like a polygon (e.g. a square) in a two dimensional topological space. Depending on the geographical diversity of each AS, pre-selected beacons known as landmarks (routers or DNS servers in the Internet) based on their IP addresses are injected into the polygon. In this figure, only selected landmarks are considered and they represent a small subset of all available routers in the AS network. Furthermore, each landmark in an AS is to represent a zone (e.g. a circle in the square) within the identified polygons, one polygon per AS. For instance, referring to Figure

2 , assume an AS (the square) contains multiple landmarks, where each landmark represents a zone (the circle). Nodes close to a landmark in terms of distance (RTT or geographical) delays are said to be members of the same zone.

Considering that delay-latency measurements either already exist or can be taken between landmarks, and therefore between zones if the landmark is considered like a centroid of a zone, the latency quantities can be embedded within the polygons to enable RTT-based measurements between the centroids of the zones (i.e. between landmarks). The number of landmarks should reflect the breadth of locations within each AS. The advantage of this approach is that once the landmarks are pre-selected, the search space depends on only the number of pre-selected landmarks regardless of the number of nodes involved. This insight is what converts this hard-search challenge into a polynomial-search problem.

Part of the heuristic when using landmarks is that peers are assigned their membership to zones based on their AS's assigned IP and geographical (latitude/longitude) proximity to nearby landmarks (keeping in mind that both the topology of the Internet is reflected in the IDS and that geographical location of the node is a strong component of the IDS). In the topological terminology, zones are like neighborhoods and landmarks are the reference points to the neighborhoods. This heuristic is what allows the algorithms to get nodes close enough to a subset of landmarks from which only one of the landmarks is selected based on RTT measurements to be taken at execution time using ping (a Linux command).

From the literature review (section dealing with the "locality in the Internet"), it was clear that any heuristic for assigning nodes to zones using geographical locations has a high possibility of producing errors while positioning nodes into zones. To improve the heuristic based on geographical location, the RTT between a client and nearby landmarks is obtained at execution

time via the ping or traceroute command to make the final determination for membership to a zone. Ping and traceroute are computer network utilities that can be used to measure the round-trip time for messages sent from the originating node to a destination node. For instance, referring to Figure 2, when a peer arrives, its geographical location (longitude/latitude) can be compared with the landmarks for each given AS and before final membership into a zone. Once a subset of nearby landmarks is identified, the node can issue a ping command to say 3 of the nearby landmarks to decide which one is closest in terms of RTT. The closest of the 3 would become the nearest landmark. Nodes sharing the same nearest landmark are said to belong to the same zone. This approach hinges on the assertions made by Abrahao and Kleinberg (2008) indicating that the geographical location (latitude/longitude) within an AS is a strong component of the IDS.

Number of landmarks per AS

The number of landmarks per AS is an issue that required some further consideration. The number of ASes is greater than 16,000 (Caida, 2011). The number of landmarks per AS could be further optimized but for research purposes, all available landmarks per AS were included. The RTT measurements between landmarks were collected from measurements taken by Caida (2011) and the data is publicly available. The tradeoff is that too many landmarks per AS can result in unnecessary processing overheads without contributing any value to the objective of selecting peers by proximity. If not enough landmarks are selected per AS, or they are not geographically dispersed, then the issue would be one of not enough diversity that may impact the quality of the peer selection as implemented by the algorithms. Finding the appropriate number of landmarks per AS can be an assignment in itself to further improve the algorithms. However, for this work there was no need to further optimize the algorithms after all of the

available landmarks per AS were included at the expense of some processing overhead. The world can be represented with about 3,000 landmarks as done with iNano (Madhyastha et-al, 2006).

Design and specifications for the new algorithms

The objective for the proximity algorithms is to return a list of nearby (neighbor) nodes to the tracker for each requesting client (numwant is the parameter for the maximum) based on their location in the Internet topology. The location of nodes can be represented in a two dimensional space based on longitudes and latitudes. Inside this new space, the set of IP addresses that belong to an AS can be represented as points inside a figure that looks like a polygon. Depending on the geographic diversity of each AS, landmarks are selected a priori as part of this investigation to represent zones within the polygons. Once a zone is populated with member nodes, the zones become neighborhood and nodes inside a zone are called neighbors.

Considering that latency measurements can be taken between landmarks (i.e. the centroids of the zones), the latency measurements were embedded within the space. The number of landmarks to be selected a priori based on this study reflected the breadth of locations within each AS. Peers were assigned their membership to zones based on their AS membership and proximity to nearby landmarks. The return list of candidate peers was built by selecting those peers found in the same zone as the requesting client. If more peers were needed, then the adjacent zones (based on proximity to the next landmark) were returned. This process was repeated until the requested number of peers was satisfied.

The algorithms were implemented in a prototype named FriendTorr, highlighted in Figure 3. FriendTorr works in conjunction with the BT Tracker by creating a list of candidate peers based on proximity. FriendTorr can be in one of two states: initiation or listening. For

every request from the client, a thread to run the tracker computer code is created. In a similar manner, a new FriendTorr thread is created for every request from the tracker. Depending on the experiment, the FriendTorr thread can be created, at the experimenter's discretion, for either the random or proximity peer selection techniques. When the tracker is first initiated, a FriendTorr thread is created to establish the needed matrixes during the initiation state. All the required matrixes for the algorithms are built during the initiation phase. The matrixes are needed to convert from the IP addresses of the nodes to latitudes/longitudes, to assign the AS for the node and to provide information about the landmarks such as their location in terms of geography (city, region and country). Initiation can be invoked during the listening state to refresh the required matrixes but for the scope of this work, it was not necessary since it was more convenient to stop and start the process as needed.

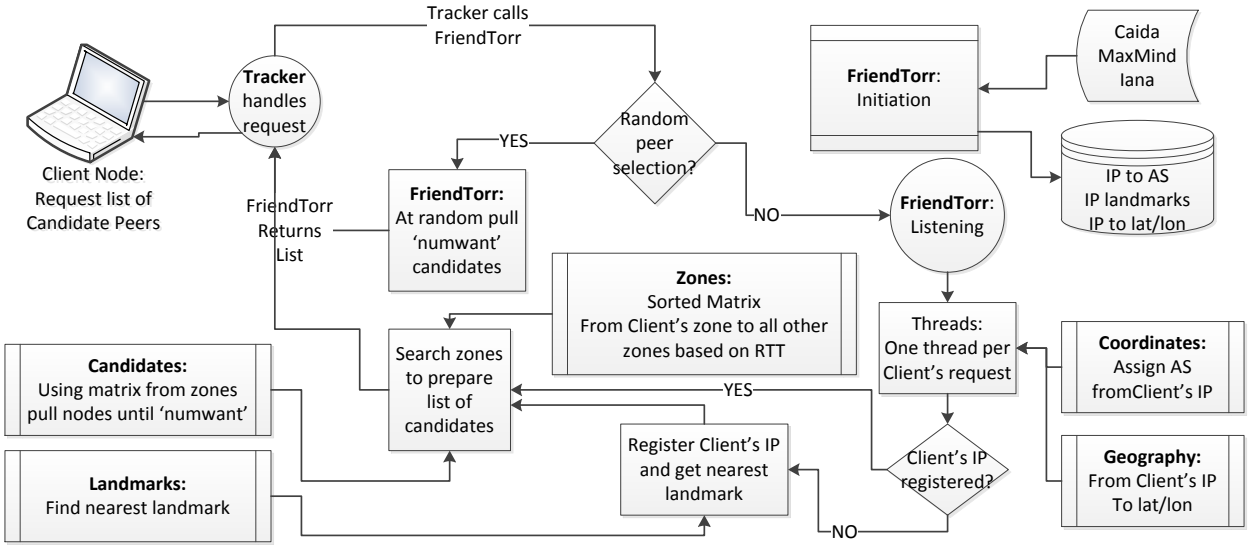

Figure 3. Node Positioning System

This initiation process takes an amount of processing time directly proportional to the number of countries selected for the experiments and also based on the size of the countries under consideration. To this end, the prototype reads files that can be serviced over the Internet. Three such files are the landmark file (Caida, 2011), the geographic file (MaxMind, 2011) and

the AS-IP-blocks file (Iana, 2011). From these files, matrixes are initiated and made available to the algorithms to be discussed shortly. Completion of the matrixes was the signal to end the initiation state.

During the listening state (highlighted in Figure 3), a new thread in FriendTorr is created and associated to an AS by invoking the coordinates algorithm. At this point, the logic in FriendTorr (executing in a thread for each Client request), determines if the requesting Client is new or already registered. If it is a new node, then it must be registered, its longitude and latitude determined (using the geography algorithm), and the node's nearest landmark (IP address) identified. Once a node is registered, then the process continues by creating a link-list of zones based on RTT starting from the zone of the requesting node to all other zones (ascending order in terms of RTT). The zone membership is assigned based on the nearest landmark that was determined during registration. This closest landmark assignment is completed by the zones algorithm in two steps: first the nearby landmarks are determined based on geospatial distance. In the second step, the peer measures its RTT to the nearby landmarks using either ping or traceroute to pick up the closest one. The thread in FriendTorr goes on to pull candidate nodes form the zones in the link-list returned by the zones-algorithm and by the logic in the landmarks-algorithm. The thread is eliminated from the system when a list of candidates is returned to the requesting client.

The coordinates algorithm

This algorithm identifies the AS for a given IP address. When it receives an IP address as input, it returns the AS number for the requesting IP address. A hash data structure is used to map an ASN (number is the key) to its associated range of IP values: fromIP and toIP.

```
•Algorithm: Coordinates
•Input: client's IP address and array matching IP to AS
•Output: AS number
1. c ← client's IP address
2. a ← hash of IP ranges by AS: a(AS-Number) = {fromIP, toIP}
3. r ← empty
4. ForEach row in a
5.         IF c >= fromIP and c <= toIP THEN  r ← AS-Number
6.         IF r <> empty THEN EXIT ForEach  loop
7. END ForEach
8. Return r
```

Algorithm 1. Coordinates

The zones algorithm

The input to the zones algorithm is the IP address of the requesting client, IP of the nearest landmark and the IDS hash containing the RTT delay between each pair of landmarks (the two landmarks are the key to the hash). The sort is done by matching the nearest landmark to the first key in the hash and ordering the hash from the nearest landmark to the farthest.

```
•Algorithm: Zones
•Input: client's IP address, IP nearest landmark, IDS array
•Output: link-list of landmarks from closest to farthest based on RTT
1. c ← client's IP address
2. fromL ← IP of closest landmark
3. toL ← sort IDS using fromL as a key from closest RTT to farthest RTT
4. toL ← push fromL on top of the array (1st element)
5. return toL
```

Algorithm 2. Zones

The candidate algorithm

When a client requests a list of nearby peers using the proximity peer selection strategy, this algorithm is invoked as a final step. The input is the IP address of the client, the IP of the nearest landmark, the length of the list (numwant) requested by the client and the link-list array obtained from the zones algorithm. The output is the list of nearby peers. Peers from the same zone are

selected until the requested number of peers is satisfied. If more peers are needed, then the next zone based on proximity from the landmark as given in the link-list is invoked and nodes belonging to this zone (for the landmark) are retrieved. This is done until the number of selected members matches the numwant parameter (passed by the client when a request was sent to the tracker). This process of searching zones in the link-list is repeated as needed until numwant is met or the maximum number of registered nodes is exhausted.

```
•Algorithm: Candidates
•Input: client's IP address, IP nearest landmark, toL (from zones algorithm), numwant
•Output: list of candidate nodes
1.  c ← client's IP address
2.  toL ← list from zones algorithm
3.  list ← empty
4.  numwant ← numwant
5.  ForEach zone in toL
6.      ForEach peer registered
7.          Next if peer == c
8.          l ← closest landmark for peer
9.          If l == zone THEN list  ← push peer
10.         EXIT both loops if numwant == length of list
11.     END ForEach
12. END ForEach
13. return list
```

Algorithm 3. Candidates

The landmark algorithm

When a node registers for the first time, this algorithm is invoked. The input is the IP address of the client. The output is the IP address of the closest landmark. A hash of possible nearby landmarks is identified using the Haversine distance algorithm (Algorithm 4) and ordered from closest to farthest in terms of the RTT between landmarks. In this Haversine algorithm, the input are the longitude and latitude of two locations. Also, "dlon" and "dlat" are the delta longitude and latitude between the client and the landmark under consideration. The sine (sin), cosine (cos) and tangent (tan) are from geometry. R is the radius of earth and sqrt is the square root. The arctangent of the longitude and latitude in the range between $-\pi$ to π (-PI to PI) is

given by atan2. In other words, Haversine performs a geospatial calculation between two geographical locations on planet earth and returns the distance in miles. Once the list of potential landmarks is established, a latency check could be conducted (but it is not in the algorithm) to determine from the candidate landmarks the closest in terms of RTT using "ping or traceroute" utilities as described earlier.

```
•Algorithm: Haversine
•Input: lon1, lon2, lat1 and lat2
•Output: d
1.  dlon ← lon2 - lon1
2.  dlat ← lat2 - lat1
3.  a ← (sin(dlat/2))^2 + cos(lat1) * cos(lat2) * (sin(dlon/2))^2
4.  c ← 2 * atan2( sqrt(a), sqrt(1-a) )
5.  d ← R * c
6.  return d
```

Algorithm 4. Haversine

```
•Algorithm: Landmarks
•Input: client's IP address, latitude and longitude, AS number, index_landmarks, country, city
•Output: IP of nearest landmark
7.  c ← client's IP address
8.  l ← empty (nearest landmark)
9.  lat1 ← latitude for client's IP address
10. lon1 ← longitude for client's IP address
11. AS ← AS number for client's IP address
12. CC ← country for client's IP address
13. City ← city for client's IP address
14. nearestIP ← empty
15. nearestD ← infinity
16. /* index_landmark{IP,AS-Number,Country,Index-City} = {latitude,longitude} */
17. Index ← index_landmarks
18. newIndex ← search Index for AS-Number==AS and cc==Country match
19. newIndex ← search newIndex for City==Index-City match
20. ForEach i in newIndex
21.         lat2 ← lat from Index for this i
22.         lon2 ← lon from Index for this i
23.         d = harvesine(lat1,lat2,lon1,lon2)
24.         If d < D then DO
25.             nearestIP ← IP from Index for this i
26.             nearestD ← d
27.         END DO
28. END ForEach
```

```
29. return nearestIP
```

Algorithm 5. Landmarks

The geography algorithm

Given an IP address, this algorithm returns the longitude and latitude along with the name of the country, region and city where the given IP is located. This information is helpful to reduce the search space when looking for nearby landmarks. If latencies between all peers were probed it would require O(n**2) probes where "n" is the number of peers. If this probing was to be done by each peer, the problem would become intractable. Network coordinates reduce the number of probes to O(n) but that is still a significant burden, even if the probing is done by the tracker. By working with landmarks spread around the world (Figure 4), the search problem can be significantly simplified to a polynomial complexity.

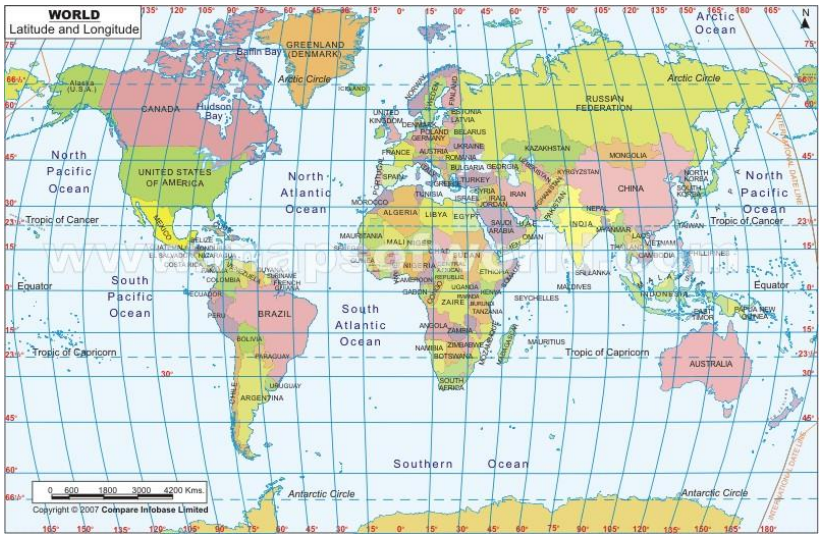

Figure 4. World Latitude and Longitude

MaxMind is a provider of geographical location maps in the form of a file that can be used to map IP addresses to longitudes and latitudes on planet earth (MaxMind, 2011). This map is read during the initiation state. MaxMind shows a breakdown of the accuracy of their GeoLite map on a city by city basis around the world. The GeoLite database is said to be 99.5% accurate.

Their accuracy figures are derived by checking known IP address and location pairs against their database (IP Locations, 2011).

```
•Algorithm: geography
•Input: client's IP address, Map
•Output: latitude, longitude, country, city, region
6.  c ← client's IP address
7.  map ← Map
8.  lat ← map{lat}
9.  lon ← map{lon}
10. country ← map{country}
11. city ← map{city}
12. region ← map{region}
13. return lat,lon,country,city,region
```

Algorithm 6. Geography

Resources

Software licenses were required using a combination of Microsoft and GPL licenses. For the Windows platform, the Microsoft Windows XP and Windows 7 software were needed. For the Linux platform, the CentOS distribution (GPL licenses) was used. For the experiments, WEB hosting services from providers had to be contracted.

Summary

After the prototype was built using the framework described in this chapter and the benchmarks could now be conducted, the goal set for this work was within reach. That is, a biased alternative to the random method for selecting peers implemented in a prototype that could be used for comparing multiple peer selection techniques. The presented framework built upon the work from other researchers reviewed in chapter 2. The algorithms made it possible to implement the concepts into a testing environment. The next chapter discloses the results obtained from the benchmarks.

Chapter 4

Results

Introduction

The literature shows a high variability in P2P download times when distant peers were present. Researchers noticed unnecessary P2P traffic delays due to peers ignoring the topology of the underlying physical network at the time peers were selected for interactions. The random selection technique, the most prevalent selection method, was suspected to be the culprit. In addition, the literature revealed that peering points between ISPs to be congestion points that could be avoided if nearby peers were given preference for interactions. Hardware and bandwidth solutions were not an option because ISPs had no incentive in such investments to improve their peering points since revenues are not linked to P2P traffic; a fast growing source of traffic in the Internet. As a result, some ISPs have resorted to the practice of trimming their P2P traffic at peering points. This practice causes traffic delays in the network due to retransmissions. The thesis was that algorithms could be developed independently from ISP's involvement for the purpose to group peers by neighborhoods after taking into account the topology of the Internet. The stated goal was to reduce the cumulative download time and variance in P2P networks when peers have a wide geographical diversity.

The methodology, presented in chapter 3, was to develop a new framework for grouping peers by neighborhoods and for comparison purposes, benchmarks would be conducted to compare the random versus the new proximity based peer selection techniques. This new framework resulted in a set of new algorithms, also presented in chapter 3, that were implemented in a prototype named FriendTorr. The anticipated tradeoff to this new selection

technique was additional processing overheads to be incurred while running FriendTorr. Success was stated to depend on the ability to design the algorithms in such a way to keep the footprint small and the impact on scalability down to a minimum.

FriendTorr was designed to assist the BT Tracker with the new functionality needed to position nodes in the topology of the Internet and to group nodes that are likely to be in the same neighborhood. This new prototype was a requirement to make it possible to test the random and proximity selection techniques via benchmarks to be conducted in a lab setup and over the public Internet. This closer to real life approach for testing is a departure by the mostly event-driven simulations, a low fidelity method, noticed during the literature review. It is expected that the benchmarks with the actual system should be a more convincing argument to prove the advantages of one technique over the other.

In this chapter, the results from the benchmarks are presented with the objective to verify if the proximity-based peer selection technique does indeed show a significant improvement in the cumulative download time compared to the random technique. In addition, the expected overheads from FriendTorr are shown. The rest of this chapter is organized into subsections to show how the simulations and experiments over the public Internet were designed, conducted, how data was collected, and most important present the results obtained from the data collected during the benchmarks. The subsections are:

1) Simulation (stage 1) was setup to verify that the new algorithms can distinguish between local and remote nodes.
2) Simulation (stage 2) was run to compare download times in both homogenous and heterogeneous networks for the random and proximity based peer selection techniques in an environment where network delays could be controlled.

3) For nodes in the public Internet, benchmarks were conducted to gather empirical evidence of the download times from the random and proximity based peer selection techniques.
4) A closer look at the overheads introduced by FriendTorr by comparing processing, memory, and disk footprints with and without FriendTorr. This overhead is the tradeoff between the random's low overhead method for selecting peers and the more complicated proximity technique.

Simulation

The simulation was designed for the purpose of conducting experiments with the ultimate objective to compare the random and biased peer selection strategies. The simulation was necessary because taking the experiments directly to the public Internet would have not allowed the experimenter to verify the algorithms presented in chapter 3 in a controlled environment where network conditions could be manipulated to test ideas. As indicated in chapter 3, the metric used for evaluation of the different tests is the cumulative distribution function (CDF) of the download time for the participating nodes. The simulation had the following characteristics: nodes that represent users in the process of downloading a file, the BT Tracker (tracker) with the role to present a list of candidate peers to requesting nodes, FriendTorr assisting the tracker to position nodes and group nodes into neighborhoods and a publisher node containing the file being downloaded.

Overview of the simulation

The simulation models a group of peers that are interested in downloading a file. The simulation was set up to run in a Local Area Network (LAN). Given the dual need to verify the algorithms and the objective to compare peer-selection strategies, the simulation was conducted in two stages.

For either of the two stages of the simulation (Figure 5), peers must first contact the tracker in use by the publisher node hosting the file of interest. The simulation consists of the following components: the BT tracker (with some minor changes to interface to FriendTorr), FriendTorr (based on the algorithms presented in chapter 3), the original publisher (seed hosting the published file), and one or more nodes interested in downloading the published file. As indicated in chapter 3, the nodes should be spread across different countries to increase the diversity of locations. If all nodes were located in the same location, there would be no difference between selection strategies.

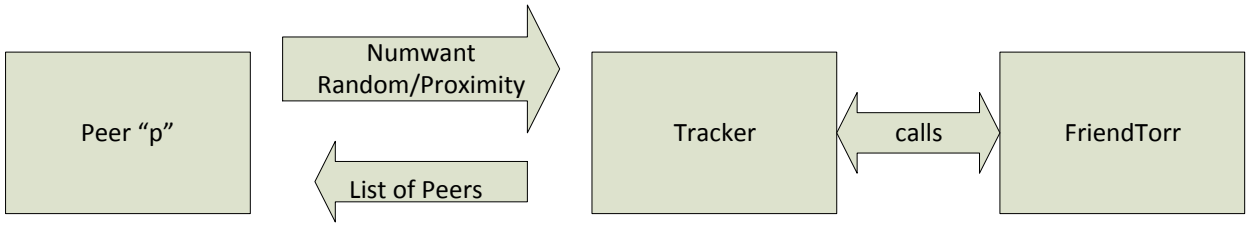

Figure 5. Simulation

As per the BT protocol, each simulated peer "p", upon first joining the network, contacts the tracker using the "numwant" parameter. The "numwant" parameter indicates to the tracker the length of the list to be returned to "p". The tracker obliges by selecting "numwant" nodes, out of all the registered nodes participating in the torrent, and returns the list back to "p".

Stage 1 for the simulation

The objective for the first stage was to verify that FriendTorr is capable of assisting the tracker in positioning nodes in the Internet and to identify neighbors to a requesting client. Instead of using multiple client nodes generating requests, a driver prototype was written to generate requests to the tracker as if the requests were generated from several nodes. To facilitate the running of multiple tests for each of the selections strategies to be compared, a

parameter for the trackers can be set ahead of each experiment to run in either random or proximity mode.

During the simulation “p” would receive a list of candidate peers followed by an attempt to initiate connections with, by default, five of those nodes in the list by opening a TCP/IP connection to each one of them. However, for stage 1 and different from the real client software, the driver prototype does not attempt to open a TCP/IP connection to each of its 5 peers. Instead, the driver prototype is set to gather statistics to be reported as follows: for each peer “p”, the average number-of-hops per packet between “p” and each of its closest 5 peers are added together. The number-of-hops is used as a proxy for the average RTT between nodes. Given that this is a simulation without real number-of-hops measurements as would be the case in the public Internet, a publicly available database with measured number-of-hops as a substitute of Internet delays was obtained, a topic to be discussed in more detail later in this section. Once the number-of-hops for each and every “p” are obtained and sorted, the number of hops percentile could be reported. The reported results are presented later in this section to confirm that the algorithms are indeed working as designed.

Design, formulas and assumptions for stage 1

The design for stage 1 consisted of simulating multiple peers with a driver prototype that generates requests to the tracker as if they were being generated by separate BT clients. At this point, some simplifications to real world were assumed: the network had no congestion and every node had exactly the same network capacity on a symmetrical link (upload/download speeds are identical).

Similar to the real world, all participating peers downloaded the same file simultaneously. For this first stage of the simulation the file was set to a small size so it could be broken into 5

pieces; each piece to fit in a single IP packet. The tracker was set to return a list of 5 peers (numwant=5). It was also assumed that the client would be using its entire bandwidth (BW) to talk to one peer at a time to download the needed piece of the file. This approach of communicating with one peer at a time was needed to simplify the simulation given that in the real world clients can talk to multiple peers at the same time. The assumptions for stage 1 were the following:

- No network congestion, thus constant TCP Window Size for each node.
- Every peer participating has the same network capacity and a symmetrical link.
- All peers in torrent download the same file, i.e. constant number of bits to download, and the file is broken into 5 pieces of equal size each. Assume each piece is 256 kilobits. In other words, each client downloads the same amount of data from every peer.
- A client talks to one peer at a time using its full bandwidth (BW). Thus, each client alternates one peer at a time sequentially to download all the pieces of the file.
- The number-of-hops as a proxy for the RTT. The RTT between two nodes represents the time needed to download one piece of the file.
- Once a client selects 5 peers with the needed 5 pieces, no other peers are selected.

In general, for a TCP connection in an IP networks, bandwidth (BW) refers to the channel capacity as measured by its throughput = (TCP Receive Window) / RTT (round trip time). In addition, the download time of a file = file size / BW. Based on the above assumptions for the first stage, it follows that the download time for an arbitrary peer "i" is equal to a constant times $\left(RTT_{i,1} + \cdots + RTT_{i,5}\right)$ where each $RTT_{i,j}$ is the RTT delay between the arbitrary node "i" and peer "j" for j between 1 and 5. From the assumptions it follows that the constant should always be the same constant regardless of the client. Therefore, the total download time for all

the peers in the torrent = k * [$(RTT_{1,1} + \cdots + RTT_{1,5})$ + … + $(RTT_{n,1} + \cdots + RTT_{n,5})$ where "k" is a constant that belongs to the set of real numbers; and "n" belongs to the set of integer numbers and it is the arbitrary "n" peer.

The Driver Prototype for Stage 1

A prototype simulator program known as the driver was written to simulate one or more active peers participating in a BT network to download a file. The driver prototype generates requests where each request "p" behaves as described in the overview. The tracker interprets these requests as if they were originated from real clients following the BT protocol and responds according to the type of request. When the request is for a list of neighbors, the tracker returns a list selected from all of the registered nodes participating in a file download. For those situations when there are less than "numwant" registered nodes, the tracker returns all the registered nodes.

The driver is organized to run based on parameters set before execution. One of the parameters is to pre-set the simulation to run as either a "random" or "proximity" peer selection strategy. Once the FriendTorr is brought up to listen to the tracker, it runs in one of the two modes (random or proximity) for selecting peers. Another parameter controls the creation of a file that contains valid IP addresses for the nodes to be included. As described in chapter 3, the IP addresses should be known to belong to the different countries under consideration for the simulation so the experimenter can verify if the simulation is indeed working as if it was the real world. This file had to be shuffled to ensure the IP addresses are placed at random inside the file. From the framework described in chapter 3, when an arbitrary node makes its first request to the tracker, FriendTorr registers the node for tracking purposes. For the simulation, yet another parameter controls the number of nodes to be pre-processed to ensure that the BT network has a warm up period and a base of peers to work with before statistics are collected and

reported for additional nodes arriving during a given test. After each simulation run, the drive reports the download time from every client to each of its five peers.

The lab for stage 1

The simulation was set to handle an arbitrary number of IP addresses but based on the experiments conducted by Bindal et-al (2006), one thousand nodes constitute a good representation of what has been observed in the real world. Therefore, a parameter under the experimenter's control was introduced to limit the number of nodes participating in the BT network. Based on this parameter, the driver prototype obtains valid IP addresses that were selected at random from a publicly available list of real IPv4 addresses for each of the countries under consideration. These IP addresses were used to contact the tracker as if they were coming from multiple clients. The tracker needed these IP addresses to have the FriendTorr prototype do its job of positioning clients in the topology of the Internet.

As pointed out earlier, one of the challenges for the first stage of the simulation in a lab set up was the lack of meaningful RTT or number-of-hops measurements between nodes. To overcome this challenge, the CityEdges files from DIMES (Shavitt & Shir, 2005) public information that provides number of hops (as a replacement to RTT) measurements between cities around the world was used to provide empirical evidence of actual average delays between nodes for those cases when the nodes are located in those cities for which measurements are available from CityEdges. In the real world the nodes would have taken the measurements in the public Internet during the time of execution but for simulation purposes, it provided an estimate of what the public Internet may have been like had the nodes exchanged traffic at the time when the CityEdges information was collected.

The method for taking advantage of the CityEdges information was to associate each peer's IP address pulled out from the file of random IP addresses with a specific city available in

the CityEdges database. The same method was repeated for each peer by associating each node with a specific city. Once those cities were identified, number-of-hops readings were taken from the CityEdges information. This would work without any further refinements if it was not for the fact that not every city in every country has a value for the number-of-hops entry that corresponds to every other city in every other country. Also, not every client or peer is located in exactly one of the cities contained in the CityEdges. The solution was to modify the driver prototype to use the latitude and longitude associated with each peer's IP address to find its closest city in each country available from CityEdges. There could be cases when the source city does not have a number-of-hops entry to the destination city where the candidate peer resides. In such cases, a nearby city with a valid number-of-hops was used as an approximation.

Verification and repeated searches were necessary to ensure that the selected intermediate city had a number-of-hops from CityEdges to the destination city. This was done for each client "p" and each of its peers resulting anywhere between one and three number-of-hops readings that must be added together to get the total estimated average delay between the client and each identified peer. Since each client "p" has up to a pre-determined maximum (e.g. 5) peers selected by either the proximity or random approach, and since each client "p" interacts with those (e.g. 5) peers, the average number-of-hops to each peer were added together to produce the total number-of-hops from a client "p" to all of its peers.

This approach is thought to be more realistic and much closer to what would happen in the real world had the client been deployed in the public Internet. During the simulation, the driver prototype reports the estimated geographical distances and the number-of-hops as an alternative to RTT delays between peers along with the percentiles from all participating nodes.

The validity of the simulation depends on the accuracy of the GeoLite (MaxMind, 2011) database which contains the estimated longitude and latitude given an IP address. Since the RTT between nodes has no meaning during the lab simulation, the second important ingredient is the validity of the Dimes (Shavitt & Shir, 2005) number-of-hops as a replacement to RTT between cities in the CityEdges information. Each node selected for processing was assigned to a city based on its closest proximity to a city shown in the CityEdges. At that point, the number-of-hops between peers corresponds to the number-of-hops between the cities where the nodes are located. This approach is not as accurate as when experiments are conducted in the public Internet but for our simulation purposes, it can assist in measuring the effectiveness of the proposed algorithms in finding neighbors by proximity after taking into account empirical evidence of the real number-of-hops at the time when the CityEdges information was collected.

Results from stage 1 of the simulation

The experimental results obtained from the stage 1 of the P2P simulation are presented in this subsection. The objective for this stage was to prove the algorithms were indeed differentiating between local and remote nodes; thus grouping nodes into neighborhoods with short proximity between the nodes inside each of the neighborhoods.

The results shown in Figure 6 were obtained after the first stage of the simulation was pre-set to run either the random or the proximity peer selection strategies (one at a time). In both cases, 700 nodes were processed. The first 500 nodes were pre-processed as part of the warm up period as described earlier. Statistics were generated for the remaining 200 nodes. Each of the sample nodes received a list of peers; in this case the length of the list was set to 5. In the case of the proximity selection strategy, the number-of-hops (where each entry corresponds to the sum of the known number-of-hops between a client and each of its 5 peers) had a median value of onr

with and 90-percentile value of three. In the case when the random peer selection strategy was employed, the median value was six and the 90-percentile was fifteen.

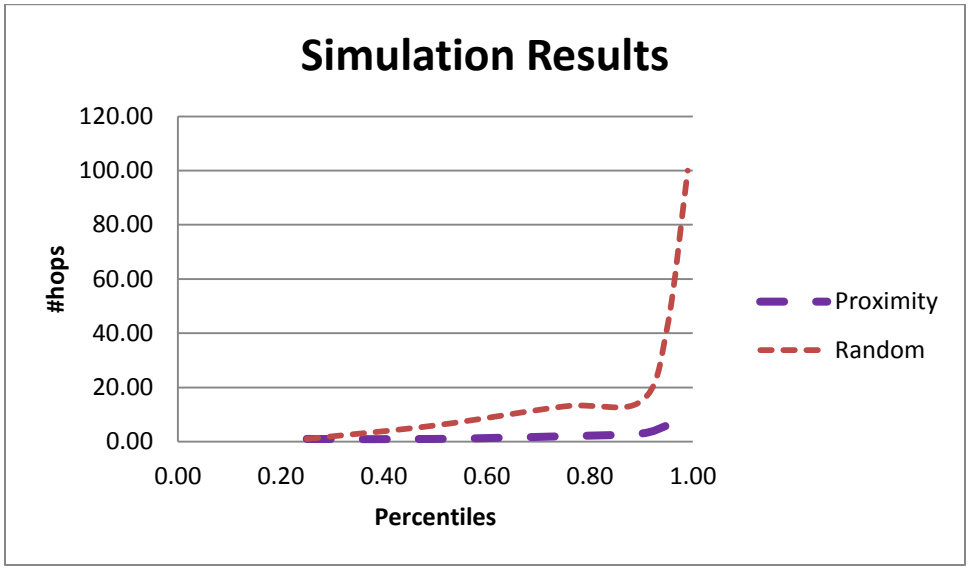

Figure 6. Proximity versus Random Peer Selection

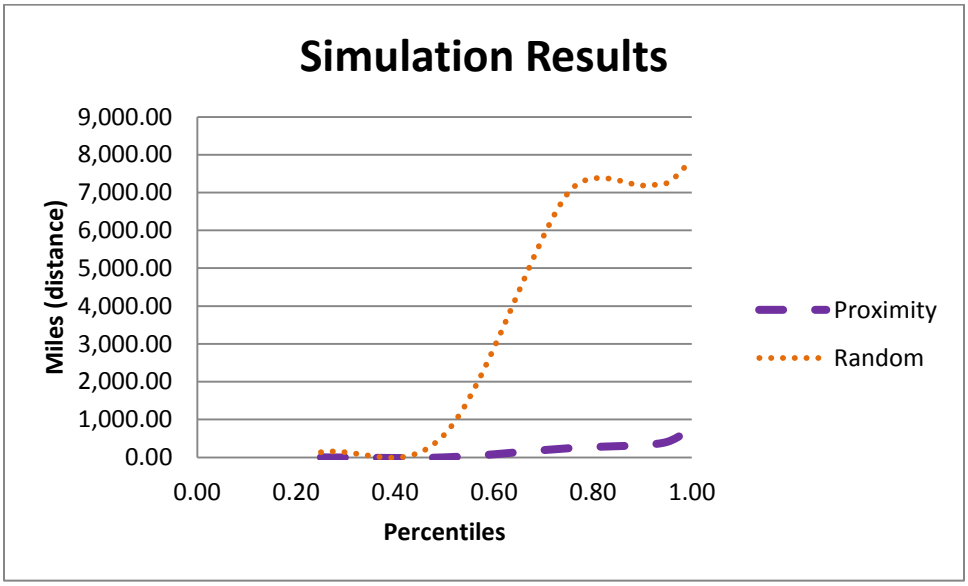

Figure 7. Proximity versus Random Peer Selection (Miles)

As a point of reference, the distance between Rome and Buenos Aires is about 7,000 miles. From the simulation (Figure 7), the proximity solution had a median of one mile between a client and its neighbor and a 90th-percentile of 324 miles. The random strategy had a median of 605 miles between a client and its neighbor and a 90th-percentile of 7,185 miles. The difference

in miles is not surprising when considering that in the random strategy, about half the peers exchanged packets with peers in a different country. This is proof that FriendTorr is working as designed and that it is grouping peers to a client based on either the random or proximity strategies.

From the simulation results it can be observed that proximity plays a significant role in the latency (based on number-of-hops) experienced by the peers in the 50 to 95 percentiles of the peers participating in a P2P overlay. The results show that the proximity peer selection should perform better than the random peer selection strategy. At this point, the FriendTorr prototype proved to be doing its job as designed.

Once this proof was obtained that FriendTorr was working as designed and that it was grouping peers either at random or into neighborhoods based on whether the random or proximity strategies were employed, stage 2 for the simulation was initiated.

Stage 2 for the simulation

In stage 2 of the simulation as in stage 1, the objective was to model a group of peers that were interested in downloading a file. Different from stage 1, there is no driver prototype involved but instead multiple nodes running the BT client software generate traffic over a LAN where WAN emulation permits the experimenter to control bandwidth, delays and packet loss. Through this WAN emulation, the experimenter can control the effects of arbitrary network performance parameters such as bandwidth traffic shaping at peering points between ISPs. The emulator works on real IP packets and creates the impression that the packets are going through a WAN with some of the same characteristics of the public Internet. This environment made it possible to repeat experiments with network conditions under the experiment's control.

The nodes were simulating two types of end-users: university and cable-modem nodes. University node would represent symmetrical up/down link bandwidth. Cable-nodes would

represent home users with asymmetrical bandwidth connections to the Internet. For this purpose, 180 cable-modem nodes and 7 university nodes were set up to run the BT client software for the purposes of downloading a 91.5 megabyte file. During the simulation, the IP traffic goes over a LAN (as opposed to the public Internet). Linux based software was used to perform Wide Area Network (WAN) emulation. Different from stage 1, there is meaningful RTT packet delay information between nodes that was captured at the time of execution so network statistics could be reported as it would be done over the public Internet.

The concept of ISPs, nodes, and bandwidth needs some further elaboration. A computer server running the Linux Operating System with the KVM virtualization software was set up as a host to run multiple guests. Each guest runs the Linux Operating System as well as the BT client software. There is a virtual LAN connecting the multiple guests inside each host. In this model, a host simulates an ISP. The bandwidth (up-link/down-link) for each guest is controlled for the BT client software using the "trickle" software package available for Linux. The communication between hosts (i.e. ISPs) must go via the one and only active physical Network Interface (NIC) card in each machine. Each host machine was set to run a WAN emulator that induces the needed traffic shaping for the NIC it controls as if they were ISP peering points. The publisher (i.e. seed node) was set up in a different computer.

The lab for stage 2

The lab consisted of one private LAN network where several physical computers that were connected to the LAN via a physical NIC (Figure 8). Each physical computer was under the control of the host operating system, in our case Linux CentOS (version 5.5) running the Kernel Virtual Machine (KVM) software. Each host contains up to 50 nodes running as virtual machines (a.k.a. guests) that represent the cable-modem nodes. The university nodes run in a

separate host with the same CentOS and KVM software. The University nodes share the same host with the publisher node.

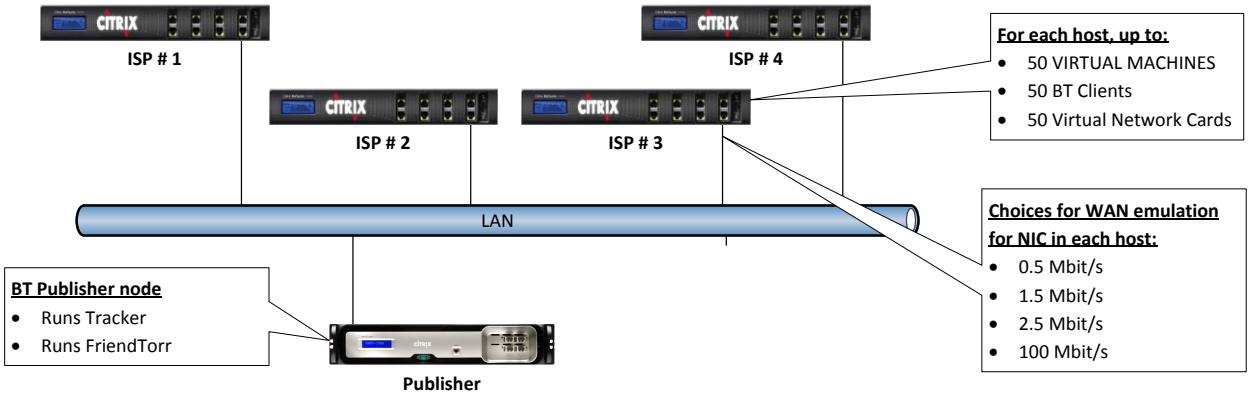

Figure 8. WAN Emulation

The following is a summary of the facilities employed for the simulation.

NODES – virtual guests running under the hosts

- OS: Linux (CentOS 6.2)
- Virtual CPUs: 1
- Memory: 512 Mbytes

HOST # 1 (50 nodes)

- CPU model: AMD Quad-Core Opteron 2356 x86_64
- CPU cores: 8
- Cores per socket: 4
- Threads per core: 1
- CPU frequency: 2,300 MHz with 512 KB cache
- Memory: 48 GBytes
- OS: Linux (CentOS 6.2) with the KVM software

HOST # 2 (50 nodes)

- CPU model: AMD Quad-Core Opteron 2356 x86_64
- CPU cores: 8
- Cores per socket: 4
- Threads per core: 1
- CPU frequency: 2,300 MHz with 512 KB cache
- Memory: 48 GBytes
- OS: Linux (CentOS 6.2) with the KVM software

HOST # 3 (50 nodes)

- CPU model: AMD Quad-Core Opteron 2356 x86_64
- CPU cores: 8
- Cores per socket: 4
- Threads per core: 1
- CPU frequency: 1,150 MHz with 512 KB cache

- Memory: 48 GBytes
- OS: Linux (CentOS 6.2) with the KVM software

HOST # 4 (30 nodes)

- CPU model: Intel Xeon x5450 x86_64
- CPU cores: 8
- Cores per socket: 4
- Threads per core: 1
- CPU frequency: 3,000 MHz with 6,144 KB cache
- Memory: 16 GBytes
- OS: Linux (CentOS 6.2) with the KVM software

PUBLISHER HOST (8 nodes)

- CPU model: Intel Core Quad CPU Q9400 x86_64
- CPU cores: 4
- Cores per socket: 4
- Threads per core: 1
- CPU frequency: 1,998 MHz with 3,072 KB cache
- Memory: 8 GBytes
- OS: Linux (CentOS 6.2)

Compared to the public Internet, the emulator permits for experiments to be repeated under similar network conditions but with different peer selection strategies (random or proximity). Therefore, in stage 2 of the simulation, there was a real network with real clients running the BT client software, with a real publisher presenting a file for peers to download. What is not real is that ISPs were represented by a physical computer, nodes were guests under the KVM software, and the bandwidth for each node and for each NIC was controlled by software.

Design details for stage 2

The objective was to simulate 180 cable-modem nodes running the BT client software and 7 university nodes downloading a 91.5 megabyte file. The cable modem nodes were set to have an upload bandwidth capacity of 100 Kbit/s and a download capacity of 1024 Kbit/s. The University nodes were set at 400 Kbit/s uplink/downlink capacities. This approach made it possible to simulate each ISP with a Linux virtualized machine where multiple guests (virtual machines) were running; one guest machine per peer. The virtual machines were connected to

the host machine via a (virtual) bridge; the bridge was set up and managed by the host machine under the experimenter's control. The host was set up to manage the physical Ethernet (eth0) connection (NIC card) using the traffic controller (tc) tool. This tool is part of the "iproute" package under Linux. The traffic controller was set up to perform traffic shaping and had to be preset before each simulation to run at 0.5 Mbit/s, 1.5 Mbit/s or 2.5 Mbit/s to simulate ISPs doing the same throttling activities in the real world.

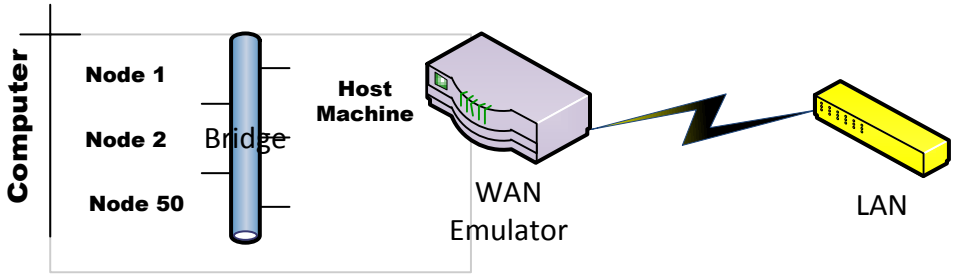

Figure 9. Simulation stage 2 design

Figure 9 is a graphical representation of the simulation. In this figure, an x86 type machine was set up using the Linux operating system. This machine was virtualized using the KVM software under the host machine and each node was an independent virtual machine running the BT client software. The client machines communicate to the host machine via a bridge (a virtual LAN). The IP traffic on the bridge is relayed to the Ethernet port via the host machine. The WAN conditions between the nodes (in this case virtual machines) as members of an ISP and other nodes in a different host were emulated using the traffic controller tool working as a WAN emulator.

The simulation runs were conducted for networks that are either homogeneous (i.e. only cable-mode nodes) or heterogeneous (i.e. both cable-mode and university nodes). For each of these networks, the simulations enable the random and locality based peer selection to be compared. The publisher node was preset to have an upload and download link capacity of 400 Kbit/s.

Results from stage 2 of the simulation

The experimental results obtained from stage 2 of the P2P simulation are presented in this section. This stage brings a more real-life like environment compared to stage 1 where the network can be under the control of the experimenter (s). The objective was to have an environment where experiments could be repeated with very similar conditions without the uncertainties expected from the public Internet. The objective was to gain confidence in the results obtained from both selection techniques in preparation for the experiments to be conducted over the public Internet.

Results for the case of homogeneous networks

Download times were found to be affected by traffic throttling when ISPs engage in traffic shaping activities for both the random and the proximity (a.k.a. biased) selection strategies in a homogeneous network. Figure 10 shows the results obtained for the random selection strategy with no ISP bottlenecks (100 Mbit/s) versus the cases when peering-point bottlenecks between ISPs were set to 2.5, 1.5 and 0.5 Mbit/s.

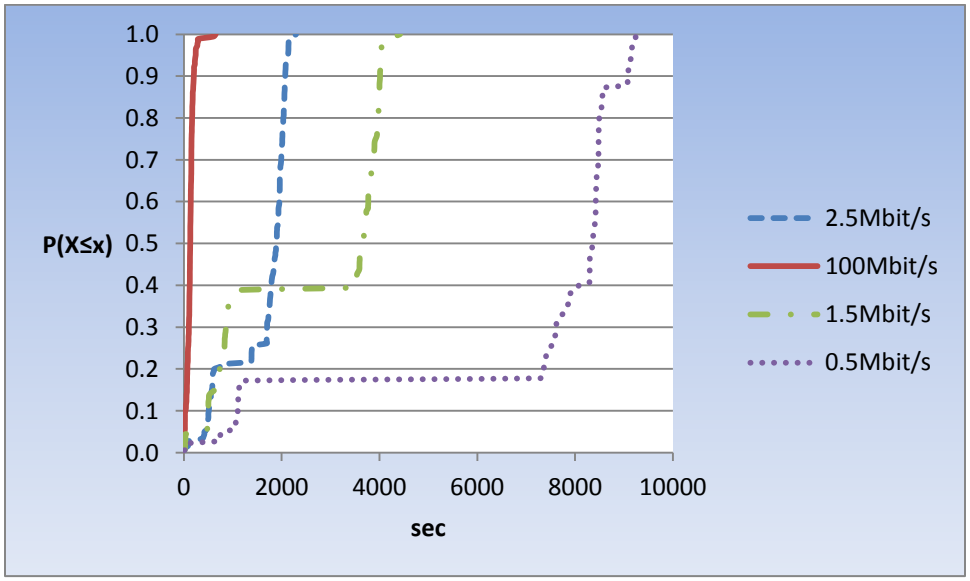

Figure 10. ISP's BW throttling for random peer selection

A similar behavior to the random strategy can be observed when the proximity (a.k.a. biased) peer selection strategy was employed (Figure 11). In both these figures, the more limited the bandwidth, the worse the download time became.

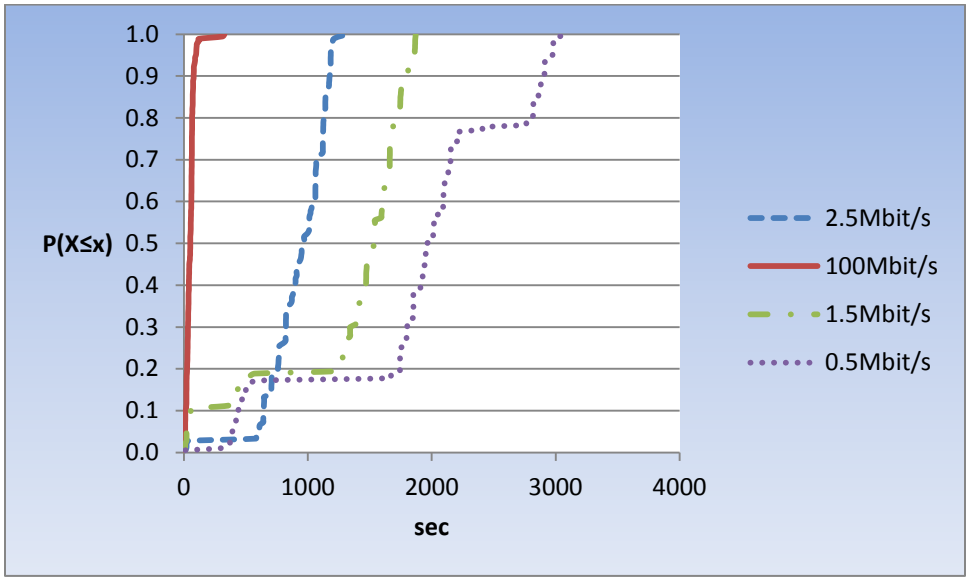

Figure 11. ISP's BW throttling for proximity peer selection

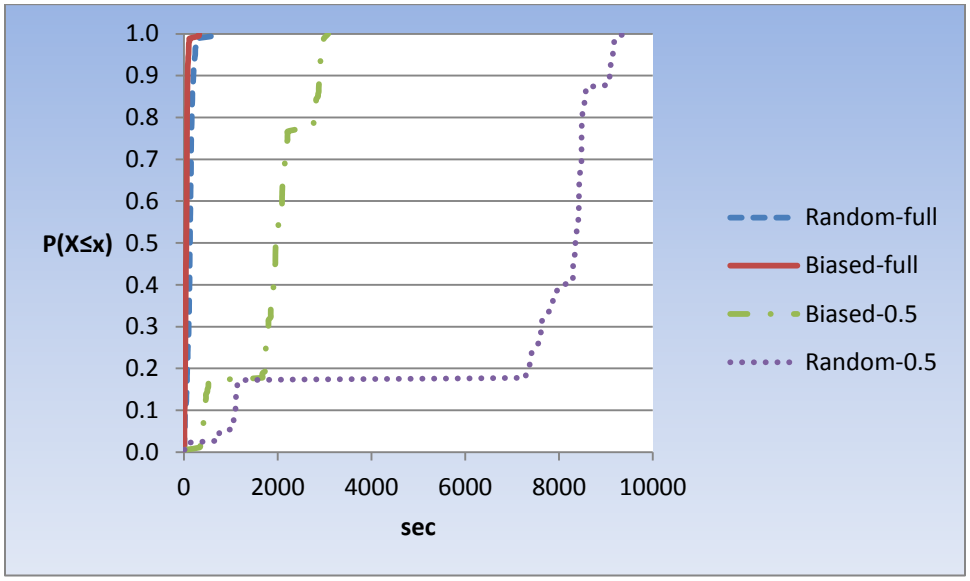

Figure 12. Random versus Proximity peer selection

When the two peer selection strategies were compared for the case of no bandwidth throttling (100 Mbit/s) and the worst case of bandwidth throttling (0.5 Mbit/s), it was observed

that the proximity outperforms the random selection strategy (Figure 12). In a similar manner, for the 2.5 and 1.5 cases of BW throttling, again, the proximity outperforms the random selection strategy (Figure 13).

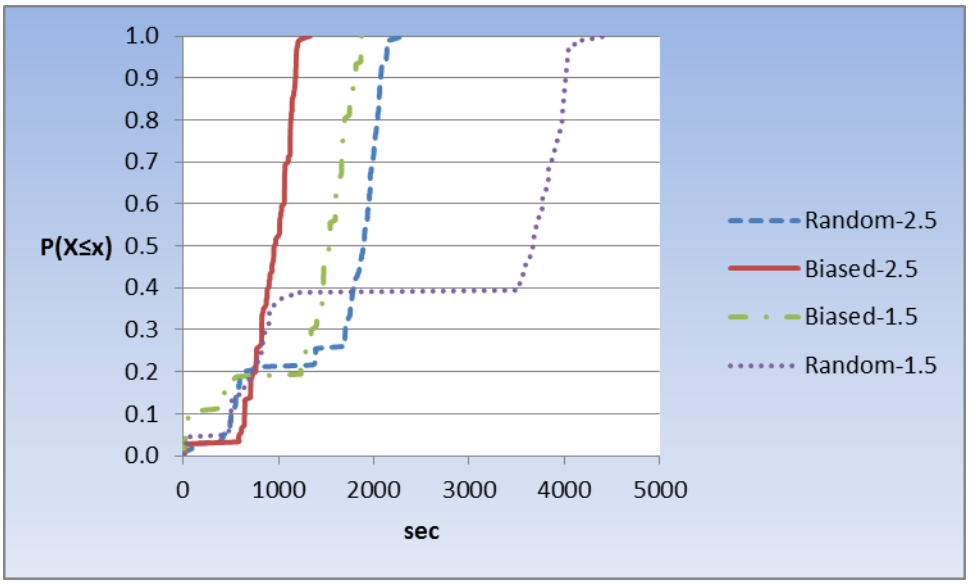

Figure 13. BW-restricted Random versus Proximity peer selection

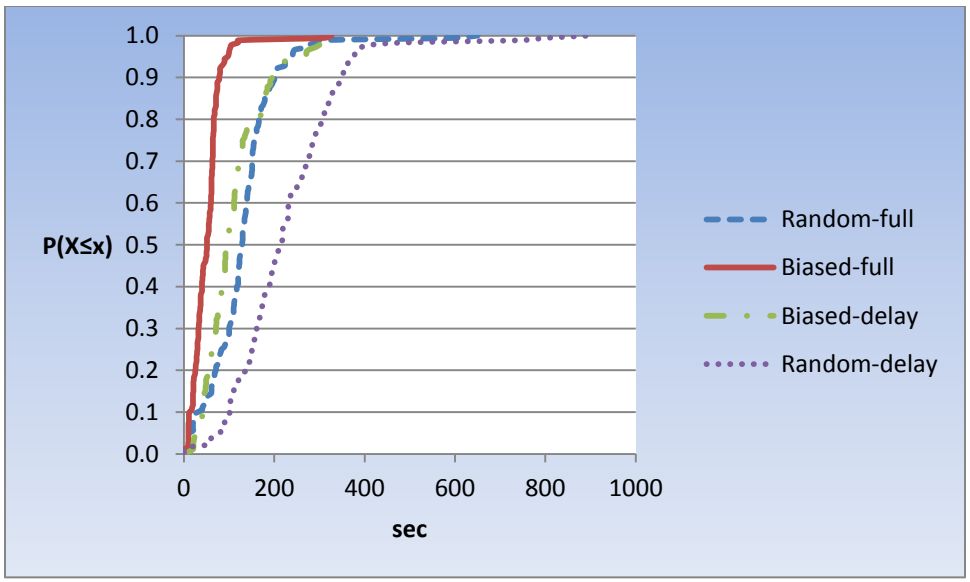

Figure 14. Delay=50ms at peering points

In a different experiment with homogeneous networks, the random and proximity peer selection were compared when a 50 milliseconds delay was introduced at ISP's peering points.

As shown in Figure 14, the proximity peer selection strategy performs better in the presence of delays at peering points.

The next analysis is to show the variance of the observed download time obtained during the simulation of the homogeneous network. Table 3 and Table 4 contain the variance (VAR) for the 50% and 95% of the nodes. It can be observed from both tables that in all cases including with and without traffic throttling, the variance is consistently larger for the case of the random technique. The times are in seconds (sec).

Table 3: Variance from Homogeneous Simulation

	Biased-full	Random-full	Biased-0.5	Random-0.5
Mean (sec)	51.46	129.52	1,907.80	7,042.74
VAR 0-50%	165	1,559	467,384	11,085,294
VAR 0-95%	481	2,909	585,856	8,280,998

Table 4: Variance from Homogeneous Simulation with Traffic Shaping

	Biased-2.5	Random-2.5	Biased-1.5	Random-1.5
Mean (sec)	926.02	1,605.71	1,325.12	2,606.64
VAR 0-50%	44,176	412,684	348,092	1,865,669
VAR 0-95%	55,917	378,678	318,980	2,485,673

Results for the case of heterogeneous networks

When university nodes are brought into the BT network, the proximity selection does a better job at protecting download times for the majority of the participating peers in a heterogeneous network. In such situation (Figure 15 and Figure 16), the proximity peer selection performs better than the random peer selection regardless if the network is flawless (no bandwidth limitations) or experiencing packet loss (0.5 or 1.5 Mbit/s traffic shaping).

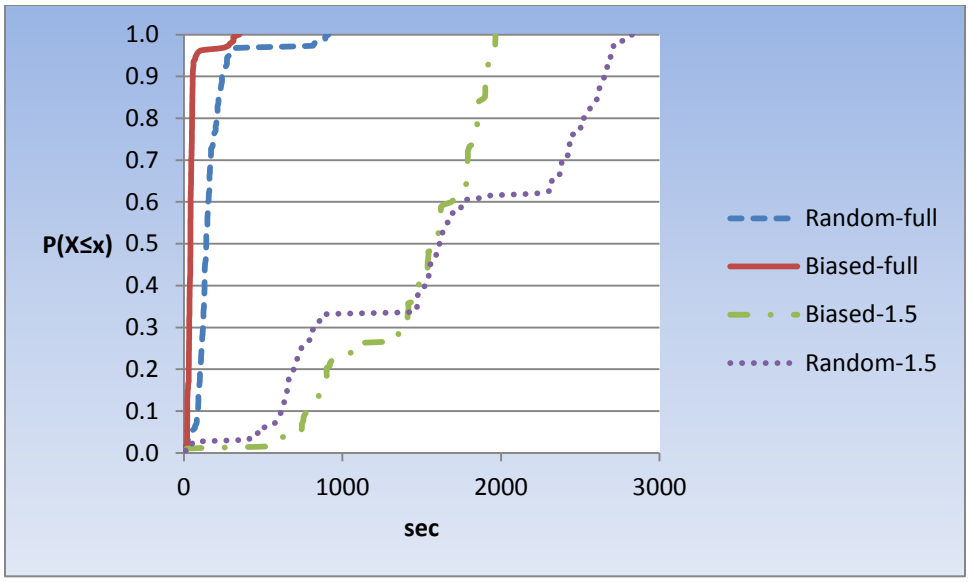

Figure 15. Heterogeneous BT Network

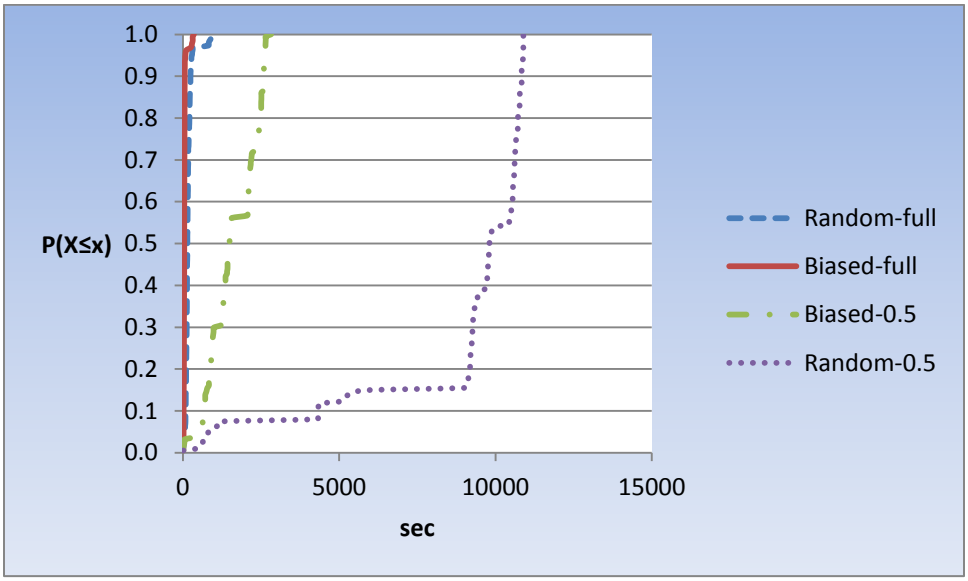

Figure 16. Heterogeneous 0.5 BT Network

Following a similar approach as it was done for the homogeneous network, Table 7 and Table 6 contain the variance of the observed download time obtained during the simulation of the heterogeneous network. It can be observed that in all cases including with and without traffic throttling, the variance is larger for the case of the random technique. The times are in seconds (sec).

Table 5: Variance from Heterogeneous Simulation

	Biased-full	Random-full	Biased-0.5	Random-0.5
Mean (sec)	49.56	170.31	1,626.68	9,025.66
VAR 0-50%	62	1,068	153,750	10,688,798
VAR 0-95%	148	3,157	541,097	8,168,676

Table 6: Variance from Heterogeneous Simulation with Traffic Shaping

	Biased-full	Random-full	Biased-1.5	Random-1.5
Mean (sec)	49.56	170.31	1,463.58	1,637.31
VAR 0-50%	62	1,068	136,018	218,268
VAR 0-95%	148	3,157	188,249	649,190

Findings from results of stage 2 of the simulations

It is evident that the random selection strategy does very well when the network is not being exposed to ISP's traffic throttling or packet-loss in general. The proximity strategy brings some of the same benefits as the random strategy along with more protection to peers in terms of download-time for all different kind of network conditions. In other words, the proximity peer selection strategy outperforms the random peer selection strategy under most circumstances. The results obtained from stage 2 of the simulation are congruent with the observations made by Bindal et-al (2006). What remains to be seen is how the algorithms would perform once nodes were deployed in the public Internet.

Experiments using the public Internet

The results from the simulation proved that the algorithms were working as designed and that indeed the proximity strategy performs better in the presence of adverse network conditions. The next step was to run experiments using the public Internet to gather more empirical evidence to compare the two selection strategies. Different from the simulation, there was no need for a WAN emulator or any other kind of artificial network traffic controls to be introduced by the

experimenter. Once running over the public Internet, there was no reliable way to tell if ISPs were engaging in traffic shaping or if congestions were present at the time the experiments were conducted.

Tests were conducted to observe and record the cumulative time it takes all participating peers to download a file when the nodes scattered across three continents (Figure 17). The nodes were VPS (virtual private servers) set up across multiple datacenters located in North America, South America and Europe. The publisher node was setup separately in a different datacenter located in Canada. In each datacenter, one or more host servers had up to 10 different VPS. Each VPS was set up to execute the BT client software upon request every time a test was conducted. Communication between nodes was handled by the ISP networks that were servicing the datacenters at the time the experiments were conducted. The multiple ISPs are members of the public Internet. Since the BT system favors peers with high download speed, the nodes were selected to have a similar network connection and processing speed.

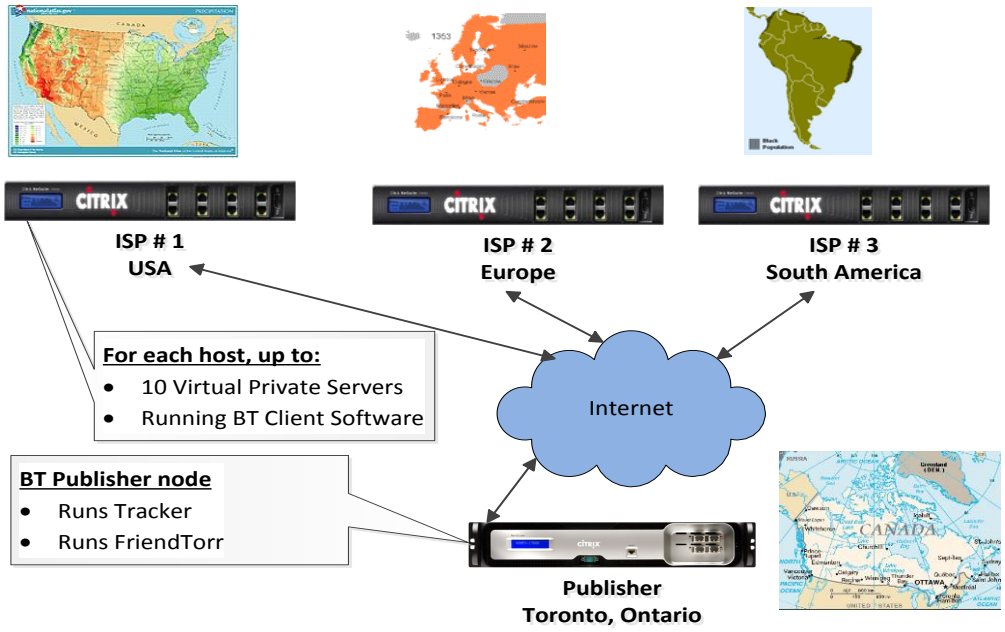

Figure 17. Experiments over the Internet

Overview of the experiments using the public Internet

The P2P network functions as described in the simulation section: each node "p" contacts the tracker to get the list of other nodes participating in the file download using either the random or proximity peer selection strategy. The length of the list of candidate nodes returned from the tracker (i.e. numwant) any time it got a request from a node was preset to 10 to reflect the fact that up to 10 nodes were setup per country. Once "p" gathered such a list, it initiated TCP/IP sessions with up to 10 peers to request downloading pieces of the file. It also accepts requests from up to 4 peers to upload pieces it had collected (Table **7)**. Once "p" completed downloading a file, it kept a log showing time stamps for the start and end times. The starting time for each experiment, similar to the simulation, was based on the "flash crowd" scenario. That is, all clients start to download soon after a new file debuts (Guo et-al, 2007 and Bindal et-al, 2006).

Table 7: BT parameters

Parameter	Publisher	Peers
Maximum bits/sec (between peers)	10,000	5,000
Maximum TCP connections	20	10
Maximum upload connections	10	4

Similar to the simulation, the intention was to compare the random peer selection strategy to the proximity based strategy. Different from the simulation, the host machine for the multiple VPS nodes were managed by the hosting company. The experimenter had full access and control of each VPS but no control over the host.

The experiments were launched by pre-setting a start time for a script executing in the publishing node using cron (cron is a time-based job scheduler in Linux). The cron script (Table 8) was set up to run inside the Publisher node to control the scheduling of the experiments. All the scripts required for the testing can be found in Appendix B. Cron would schedule the submission of the proximity.sh and random.sh shell scripts. These two shell scripts would in

turn toggle the parameter that control FriendTorr between random and proximity. In addition, the cron script would also schedule the execution of FriendTorr (select_peers.pl), the module assisting the tracker for either proximity or random selection techniques. FriendTorr would listen to requests from the tracker and return the list of peers grouped either at random or by neighborhoods. It should be pointed out that the BT client implementation in use for the experiments is known as Snark. Finally, the cron script would schedule the scheduleSimStart.pl script to initiate the experiments by contacting each remote BT nodes participating in the experiments (i.e. all of the contracted VPS machines with a fixed IP address).

In turn, after being contacted, each remote node executes the scheduleTest.sh script to start the BT client via the testClient.sh script and the killTest.sh script. The testClient.sh uses the "AT" UNIX command to stop execution of the BT client after a pre-set period of time (2 hours). The 2-hours pre-set time was long enough to give all peers a chance to complete with sufficient spare time. There was always the possibility that a node would not be available to run experiments due to factors outside the control of the experimenter; the hosts are controlled by the datacenters. In those cases, the results were ignored.

Table 8. Script from cron to schedule experiments

```
02 2,4,6,8,11,14,17,19 * * * root cd /root/BT-Publisher/FriendTorr/;./select_peers.pl 2>&1 | /usr/bin/logger -t
FriendTorr
10 2,4,6,8,11,14,17,19 * * * root cd /root/BT-Publisher/snark-0.5/;./scheduleSimStart.pl 2>&1 | /usr/bin/logger -t
snark
01 2,6,11,17 * * * root cd /root/BT-Publisher/snark-0.5/;./proximity.sh 2>&1 | /usr/bin/logger -t proximity
01 4,8,14,19 * * * root cd /root/BT-Publisher/snark-0.5/;./random.sh 2>&1 | /usr/bin/logger -t random
```

As part of the BT client software, each peer would log the start and end times of the file download. The end time occurred when the file was fully downloaded. At the end of each test, the scheduleSim.pl script running on the Publisher would collect the statistics from the log file of each participating peer.

During the simulation, the file to be downloaded by all the participating peers was a fixed file of 95 megabytes. For the Internet experiments, the file size was increased to 190 megabytes. This was done to make each participating node to take longer to download a file since fewer nodes were participating in the experiments. Once set, the file size was never changed between experiments.

Multiple "ping" (Unix command to measure RTT between hosts) were issued to measure latency between nodes participating in the download. The results are shown in Table 9.

Table 9: Observed RTT between locations

From	To	Average RTT
Canadian Node	European Nodes	117 milliseconds
Canadian Node	American Nodes	78 milliseconds
Canadian Node	South American Nodes	175 milliseconds
American Nodes	European Nodes	176 milliseconds
American Nodes	South American Nodes	233 milliseconds
European Nodes	South American Nodes	233 milliseconds

Same as the simulation, the cumulative download time was the main evaluation criteria to compare the two selection strategies. Since the conditions of the Internet are likely to change over time, the date and time for each experiment were recorded. The experiments were run several times a day. Before each experiment, the tracker was re-started to remove the list of registered peers from previous experiments. FriendTorr was pre-set to use either the random or proximity strategy, but not both simultaneously, for each experiment. The starting time was considered time=0. As time goes on, peers record their activities in a log. The log is the basis for calculating the start and end time of each file download. Based on the collected information, statistics such as download times were recorded to make it possible to compare the selection strategies.

As pointed out in chapter 3, the cumulative distribution function (CDF) for the download time was selected as a metric for the experiments to be consistent with the results presented by

Bindal et-al (2006) and following the advice from Lehrfeld and Simco (2010) who found that download time is the best performance indicator when dealing with a P2P overlay.

Results from experiments using the public Internet

The experimental results obtained from the public Internet are presented in this section. As indicated earlier, the testing is no longer a simulation but the real world, thus the network is no longer controlled by the experimenter. The two selection strategies were run for several months, mostly at night to minimize busy hosts. Busy hosts would skew results by causing delays that cannot be controlled and have nothing to do with comparing the two strategies.

Comparisons were made by pairing each proximity test to a random experiment. The experiments were conducted around the same time from one day to the next. This was done to keep the comparison as fair as possible on the assumption that the conditions of the Internet between nodes would not change that much during the time the two experiments were being run and from one day to the next.

Figure 18 shows the results obtained when running the biased and the random strategies twice each. It can be noticed that the biased (proximity) selection technique dominated the results with a faster download time for most of the nodes involved. The percentiles from these two tests are shown in Table 10 where Q1 and Q3 represent the 25^{th} and 75^{th} percentile respectively. It can be observed that the proximity (a.k.a. biased) selection technique showed faster download times than the random technique in the 25^{th} to 95^{th} (P95) range. In the case of Q1, the difference (delta) between biased-1 and random-1 is minus 103 (-103). Therefore a negative delta would indicate that the proximity technique produced a faster download time for the given percentile of the peers during the experiment.

Table 10: Percentiles from tests

Test	Q1	Median	Q3	P95
Biased-1	326	579	818	1,196
Random-1	429	680	1,074	1,226
Biased-2	397	549	874	1,464
Random-2	550	812	1,054	1,557

A more extensive set of results from multiple experiments is shown in Table 11. In this table, each biased-random pair is shown one row at a time. The 25, 50, 75 and 95 percentiles are shown to compare the two selection strategies. The delta is the difference between the random and the proximity techniques for each of the percentiles. As indicated earlier, a negative number indicates an improvement in download time introduced by the proximity technique.

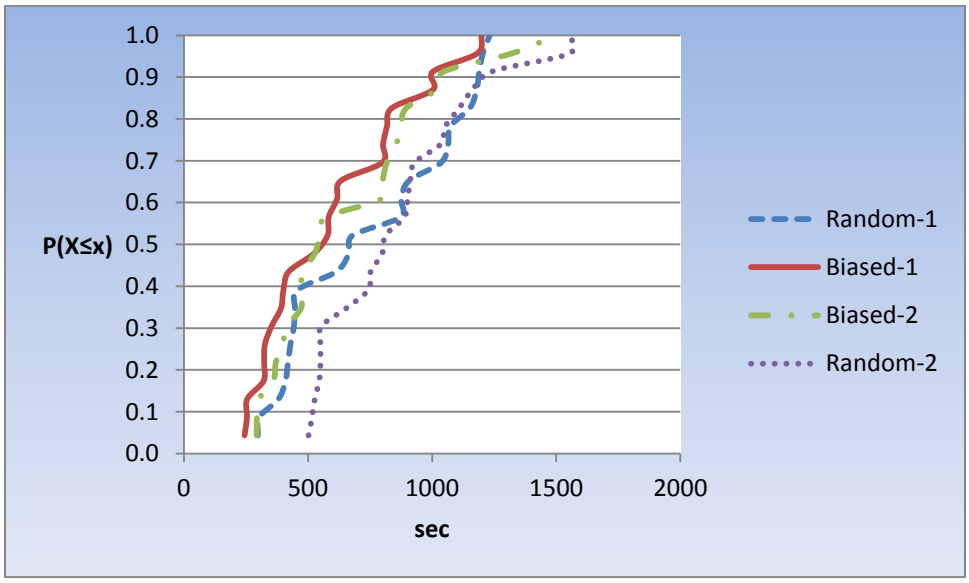

Figure 18. Results from the public Internet

Table 11. Statistics from public Internet

Biased	Q1	Median	Q3	P95		Random	Q1	Median	Q3	P95		Delta	Q1	Median	Q3	P95
Biased-1	232	435	890	1,049		Random-1	528	779	1,003	1,997		Delta-1	-296	-344	-113	-948
Biased-2	256	578	924	1,412		Random-2	284	651	863	1,128		Delta-2	-28	-73	61	284
Biased-3	352	513	917	1,967		Random-3	527	919	1,071	2,820		Delta-3	-175	-406	-154	-854
Biased-4	258	442	824	1,564		Random-4	550	689	940	1,503		Delta-4	-292	-247	-116	61
Biased-5	142	672	989	1,456		Random-5	405	717	813	1,486		Delta-5	-263	-45	176	-31
Biased-6	309	706	902	1,167		Random-6	613	747	890	5,720		Delta-6	-304	-41	12	-4,553
Biased-7	374	661	1,019	1,285		Random-7	422	875	1,094	1,729		Delta-7	-48	-214	-75	-444
Biased-8	325	473	906	1,211		Random-8	393	797	955	1,623		Delta-8	-68	-324	-49	-412
Biased-9	325	462	1,023	1,336		Random-9	629	839	1,041	1,250		Delta-9	-304	-377	-18	85
Biased-10	332	493	930	1,173		Random-10	498	635	944	1,080		Delta-10	-166	-142	-14	93

It can be observed that the proximity (a.k.a. biased) approach dominated the cumulative download time results in most of the experiments; not only for the median and higher percentiles

as other research has shown. The results are consistent with the findings from the experiments run by Bindal, et-al (2006) and from the results obtained during stage 2 of the simulation as documented earlier in this chapter. The one wrinkle that could be observed is that the proximity strategy can perform worse than the random strategy for a small number of participants (P95 percentile). The one possible explanation is that those peers are being starved of the pieces needed to complete the download. This could be as a direct result of the small number of peers used for the real Internet compared to the experiments in the lab where each neighborhood had up to 50 nodes.

Findings from experiments using the public Internet

The results from the tests indicate that the download time experienced by the majority of participating nodes were mostly improved when the proximity strategy was employed. Since adverse conditions are difficult to anticipate in the public Internet, the results from the simulations are probably more indicative on how adverse network conditions would favor the proximity strategy.

FriendTorr Resource Usage

Snark, the BT client software used for this work, does not have either a random or proximity strategy. Instead, it returns a list of nodes based on first-in-first-out basis. FriendTorr was designed to assist the Tracker to either return a random list of peers or in the process of grouping peers into neighborhoods. Both techniques have processing implications for the node running FriendTorr with the proximity more heavily taxing resources. The disk space, memory and processor requirements for both techniques are discussed in this section.

The GeoLite file, which maps IP addresses to longitudes and latitudes, takes 27 megabytes of disk space. This file is referenced via a binary executable provided by the vendor

so it does not require RAM memory once FriendTorr is executing. This information is obtained by subscription and changes to the file are received either automatically or by request from the vendor over the Internet. The cost for obtaining this file is some bandwidth to receive the updates and processing time to download the file. For the testing conducted, the file was downloaded once to avoid changing the environment in the middle of the testing that took place over several months.

The disk space to store the DNS longitudes and latitudes plus the RTT for DNS to DNS amounts to 106 megabytes for 3 continents (as used during testing) and 501 megabytes for the entire world. For the testing, all of the available DNS for 3 continents were pulled out for a total of 1,493 records. It might be possible to reduce this number by trial-and-error but considering that the literature indicated that 3,000 DNS might be sufficient to represent all locations around the world, 1,493 may be reasonable. This DNS information can be purchased from vendors and downloaded into the node running FriendTorr with some regularity. DNS servers do not change that often so this file tends to remain very stable over time. The RTT delays between DNS servers do change and for non-academic purposes this file should probably be replaced daily.

The DNS data is read from disk into three hash tables in memory. One hash table contains by IP the latitude, longitude, country-code, region-code and city-code. The second hash is the same size but indexed by country-code, region-code and city-code. The last hash contains two IP address (from/to) plus the RTT between them in either direction. The observed total memory utilization for FriendTorr using the proximity technique was 251,658 kilobytes of memory for the 3 continents as tested. It is estimated that twice this amount would be needed if the DNS information for the entire world was to be loaded. FriendTorr required 16,777 kilobytes of memory for the random technique. The memory requirements discussed so far are

fixed but both selections techniques have variable memory requirements. The variable memory requirements are dictated by the number of registered peers for either selection technique but with different requirements.

The random technique requires memory to store the IP address of each registered peer. In contrast, the proximity technique requires a hash with a key of each peer that registers along with the closest DNS for a total of two IP addresses. The algorithms do not need to keep any other details per peer for the proximity technique since the membership to a zone is based on a common DNS/beacon. Different from the DNS hash tables that can be controlled in size and fixed regardless of the number of peers; the IP addresses for registered peers would grow linearly with the number of peers that register for either selection technique.

During testing using the Internet, FriendTorr ran in a node with one gigabyte of real memory, 350 gigabytes of disk and four (virtual) processors. Also, the Tracker, and therefore FriendTorr, was contacted every 5 minutes by the nodes instead of the 15 minutes default, therefore creating more processor consumption by FriendTorr. It was observed that CPU for FriendTorr drops to less than 2% for the entire time when either technique was used.

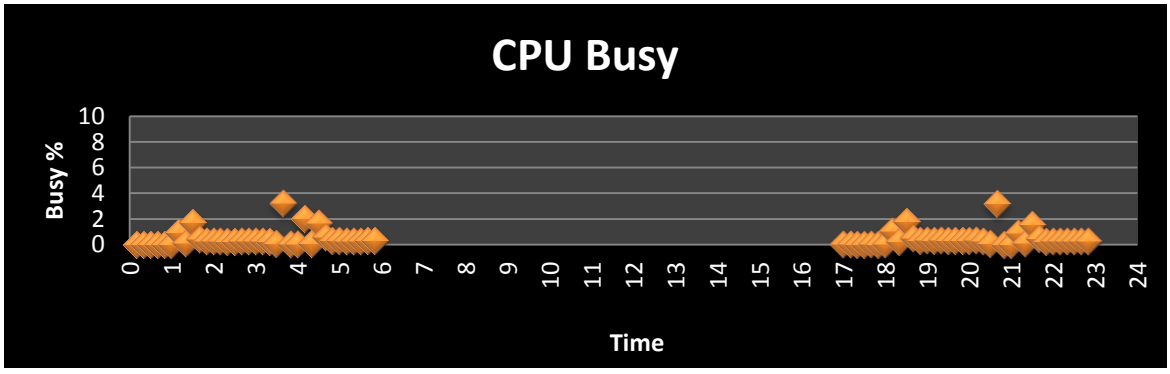

Figure 19: CPU usage by publisher

FriendTorr CPU utilization for the entire node that was collected using “sar” (UNIX utility) is shown in Figure 19. The statistics reported by “sar” are averaged in 10 minute intervals. The proximity technique was ran at 1:10 AM and again at 9:10 PM (21); the random

at 4:10 AM and 6:10 PM (18). It can be observed that the averaged CPU utilization was low and about the same for both strategies. It was concluded that using the data from the testing over the Internet was not providing sufficient information to compare the possible scalability potentials for either of the selection techniques. Therefore, a different approach was followed.

The issue of scalability could not be explored using the public Internet. Therefore, it was necessary to use the same techniques used during stage 1 of the simulation. The driver was employed for the purpose of registering up to 20,910 nodes. Since the Tracker and FriendTorr do the same work regardless of where the requests came from, the results should be very close to what is to be expected in real life. The question was one of knowing the distribution and frequency of arrivals to register with the Tracker/FriendTorr. Normally the Tracker is queried every 15 minutes by each participating peer. When FriendTorr is introduced to assist the Tracker, the expectation is that the scalability for the proximity technique would be reduced because it takes more processing per request compared to the random approach. This expectation was observed when testing with 20,910 peers. As a reference, Guo et-al (2007) found that most BT P2P networks are less than 1,000 peers with an average of 102 peers. But what if a P2P network had to be scaled past the 10,000 maximum found by Guo et-al to say, 20,910 peers?

It was observed that when 20,910 peers attempt to register as fast as possible, that CPU requirements by FriendTorr (not the entire system) for the random strategy taxed the four processors up to 35% and the proximity strategy up to 55% (the numbers oscillated between the multiple tests conducted). In terms of memory, 1,800 megabytes was needed for the random technique and 24,700 megabytes for the proximity. Another difference was the throughput of both techniques. The random strategy could register about 97 peers per second (obtained

anywhere from 95 to 100 peers per second during multiple tests); the proximity was limited to 27 peers per second (observed anywhere from 25 to 30 peers per second). These throughputs remain stable after the first few thousand nodes were registered so they represent steady-state averages. This is the real cost of the proximity strategy, the processing requirements would lower the throughput capabilities during the registration process. Once registered, the proximity strategy would improve its throughput but capacity should be dictated based on the registration throughput. However, based on the findings from Guo et-al (2007) showing that for the most popular trackers, about 550 of them, the peer born rate, which must register, amounts to 0.33 per second and the number of tracker requests per hour amounts to 2.23 per second, the numbers achieved from FriendTorr should surpass most expectations.

Considering the throughput limitations of both techniques and when a very high level of scalability is needed, beyond 21,000 peers, it would be advisable to run multiple copies of FriendTorr to increase performance and reliability. BT is not limited to one Tracker so when multiple Trackers are employed, then each Tracker would have either a dedicated FriendTorr or multiple copies of FriendTorr running in one or multiple nodes. For instance, if the Tracker is at www.file-to-download.mytracker.com, then the case for multiple Trackers could be done by working with the DNS to have the authoritative servers re-direct to a non-authoritative DNS where the World could be split into continents, for example: Europe.file-to-download.mytracker.com. Then each Tracker would work with its assigned FriendTorr where only the European countries are loaded. The entire set of algorithms in FriendTorr could be put in the DNS system but DNS delays are problematic since name server resolvers are supposed to be fast. The case for one Tracker and multiple FriendTorr nodes is less involved. For example, the Tracker could be modified to check the country membership for the IP (same as done with

FriendTorr) and call different FriendTorr nodes (since Tracker uses HTTP to make calls to FriendTorr) based on country or continent. Each FriendTorr would then be expected to load only the countries it would handle. This hierarchical approach is likely to have limitations but at extremely high levels of scalability. Given that in this research testing was done for up to 20,910 peers in a node with 4 logical CPU processors and one gigabyte of memory, it is expected that the majority of P2P networks as reference by the literature can be handled with a low grade computer.

Therefore, in theory it might be possible that a machine with more memory and logical processors could handle more than 20,910 peers with FriendTorr. But testing such configuration would carry very little value since such high number of peers would be better serviced by configuring multiple trackers to improve reliability. The trackers could be grouped into geographical locations based on nameservers and DNS services as done today by large content distribution networks. In such a design, FriendTorr could be set up in a hierarchical design to handle a subset of countries or regions around the world.

Summary

The conjecture was that the biased peer selection technique would outperform the random technique by eliminating unnecessary network delays. This conjecture had already been proven valid using event-driven simulation but lacking was an actual implementation that could be taken to the public Internet for a high fidelity validation. This was accomplished by implementing the algorithms presented in chapter 3 into a prototype named FriendTorr. The prototype was tested in three stages. In the first stage, a simplified simulation was designed to test the ability of the new prototype to separate nodes by neighborhoods as designed.

In the second stage, a high-fidelity simulation was conducted in a controlled environment where all the components were as if deployed in the public Internet except for the fact that the wide area network was being manipulated with software to create the illusion of events that would normally take place in the public Internet. The final step was to take the testing to the public Internet. The results presented in this chapter proved that the proximity technique matches the random technique in a perfect, but dubious in the real world, network and outperforms the random technique in all situations during the simulations and in most cases when using the public Internet. The simulation also showed that the variance from the average download time can be consistently reduced with the proximity based peer selection technique. However, this performance improvement when using the proximity technique comes at a cost.

The node running the Tracker and FriendTorr set to use the proximity strategy will always consume more memory than the random proximity strategy due to the need to store hash tables for the selected landmarks per Autonomous System. This fixed memory requirement, which remains the same regardless of the number of registered nodes, for the proximity strategy was a quarter of a gigabyte when loading information from three continents. Information about the world is estimated to take half a gigabyte. As the number of registered peers grow, the

random uses less variable memory requirements but the difference is not significant. CPU is a cost as well.

The other cost is in terms of CPU cycles for the node running the Tracker and FriendTorr. It is not significant for the majority of P2P networks observed in the real world where 102 nodes is the average. For those more extreme cases when a large number of nodes is expected, the simulation with 20,910 peers and when running the publisher, Tracker and FriendTorr in the same node, showed that the throughput for the proximity strategy was no worse than 25% of the throughput for the random strategy with the proximity consuming in the vicinity of 50% more processing cycles. These important issues of scalability can be further explored using the public Internet in future work by employing the framework and prototype presented in this work.

Chapter 5

Conclusion

This investigation ratifies the claim that proximity based peer selection outperforms the random technique when distant peers are present in the P2P overlays. The random technique is the most prevalent method in use today. This technique is based on the assumption that selecting a subset of peers for interactions at random from a large pool of candidate peers was a pre-requisite for its success. In spite of its success, researchers started to notice wide discrepancies in download times and suspected that distant peers could be the culprit. As it turns out, when distant peers are selected at random without taking into account their location, both cumulative download times and variance from the average were affected. However, testing other peer selection options was not possible due to the lack of non-random peer selection client implementations. This absence of non-random client implementations has been forcing researchers to rely on event-driven simulations to support their claims. This research closed this testing gap by implementing a fully functioning prototype that could be used to compare multiple peer selection techniques. The arguments to support the assertions to be presented in this chapter were developed and documented in the previous four chapters.

In chapter 1, the research problem was identified and an alternative to the random method for selecting peers for interactions in P2P networks was proposed. In chapter 3, a framework to implement such alternative was presented. The goal was to reduce both the cumulative download time and what recent research was showing to be a wide variance experienced by nodes when downloading files in the Internet. The prototype made it possible to compare the time it takes all of the peers to download a file when using either the proximity or random

strategies in both a lab environment and the public Internet. In addition, in chapter 3 it was indicated that the concept of proximity in the Internet is complicated by the fact that the topology of the Internet is hidden and constantly changing due to the volatile business arrangements between ISPs. Therefore, a node positioning system was needed that would enable each independently running peer an opportunity to select other peers for interactions after taking into account their locality.

The concept of locality had to go beyond distances based on latitudes and longitudes because two nodes that belong to different ISPs may be geographically close to each other but the two ISPs could be exchanging their inter-traffic at a remote location. Geographical distances within an ISP are a good indication of proximity but this is not the case across different ISPs. To deal with this issue and to develop the needed Internet positioning system, a new framework that breaks up the Internet address space was presented. In this new framework, the Internet-IP-address-space is separated by IP blocks that belong to the different Autonomous Systems (AS). The aggregate of all the active AS is what constitutes the Internet. Each block of IP addresses in each AS is mutually exclusive of the IP addresses that belong to a different AS. Then, within each AS beacons are identified ahead of time. The beacons/landmarks are known Domain Name Servers (DNS) that have a known geographical location. Finally, nodes that join a P2P network can be placed in relation to the beacons, and those nodes that share the same beacon are said to be in the same zone. The distance between beacons are measured in round-trip-delays (RTTs) and the distance between nodes and beacons are calculated based on latitudes and longitudes.

The results presented in chapter 4 confirmed that the proximity strategy can significantly improve the cumulative download time for nodes in a P2P network along with the variance from

the average download time. This cause-effect relationship between a selection technique and both the cumulative download times and variance from the average are undisputable.

In all of the tested network contingencies during the simulations, the proximity strategy matched the performance of the random strategy in a perfect network where delays and packet loss conditions were not present; most likely an unrealistic situation in the public Internet. When the testing was taken to the real world where the network is exposed to congestions and failures, the proximity strategy continued to consistently outperform the random strategy in a noticeable manner. But there was a cost in terms of throughput. The proximity strategy slows down the speed at which nodes could register with the Tracker. This slowdown was deemed to be insignificant once the size of most P2P networks is taken into account. Therefore, given the observed throughput results from the testing in chapter 4 and the size of most P2P networks available from the literature, FriendTorr is more than sufficient to handle real world demands.

Henceforth, the framework presented to position nodes in the topology of the Internet should be beneficial not only to researchers but also to developers of P2P client software in the Internet application domain.

Implications and contributions to knowledge

Considering the large P2P share of Internet traffic, the implications from this research are numerous and significant. Today, when traffic travels across long distances between nodes, packets end up touching more points (e.g. routers) along the way, particularly peering points when going across multiple Autonomous Systems. If P2P clients were to select local peers for interactions, then a reduction in bandwidth requirements needed on the aggregate from all Autonomous Systems would be expected. This would be highly beneficial to the ISP community in terms of cost. It would also have the side benefit of removing unnecessary P2P traffic from

their peering points; the source of congestions and delays. This reduction in traffic will open new possibilities for file distribution for new media such as video on-demand, Voice over IP, and software distribution using P2P clients. It was noticed during this work that P2P has a high share of the entire capacity of the Internet. Therefore, freeing up capacity by switching from the random to the proximity peer selection technique will make it possible to continue the exponential growth experienced by the Internet without ISPs having to resort to traffic shaping to protect their non-P2P traffic. This can be accomplished without having to put aside additional capital expenditures to handle the unnecessary traffic generated by the random technique. What is important is that this can be done independently of the ISPs and without relying on changes to the standards currently in place governing the implementation of software for the Internet.

This work also made contributions to the body of knowledge by pointing out that the proximity strategy outperforms the random technique. In addition, early in this research it became obvious that researchers had limited tools to support their claims when comparing multiple peer selections strategies. In fact, most researchers depended on event-driven simulations to make their points, not high fidelity testing based on real clients and real networks such as the public Internet. Therefore, this research's most significant contribution to knowledge is the algorithms that solved the search in the Internet Delay Space problem. These algorithms made it possible to implement a prototype to test multiple peer selection techniques. Searching the Internet Delay Space is an NP-hard problem. This problem needed a solution to ensure the prototype could scale. This work presented a method for overcoming this obstacle that made the prototype scale as the number of nodes joining the P2P network increased. This was accomplished by introducing the concept of zones based on beacons. The number of beacons would remain constant regardless of the number of nodes joining the P2P network, thus keeping

the search space constant in size. The prototype software implementation was accomplished with a sufficiently small footprint so it could fit into a standard computer.

Concisely, the framework and algorithms presented in this work can help other researchers to conduct high-fidelity testing using FriendTorr, the software prototype contributed by this work, when comparing multiple peer selection techniques.

Recommendations and future work

The framework and algorithms presented in this study provide an alternative to the random peer selection strategy in P2P networks. This made it possible to test multiple peer selection techniques under realistic lab and public Internet conditions without having to exclusively rely on event-driven simulations. This allowed questioning the validity of the assertion that the random peer selection was a requirement for the success of P2P networks. Nonetheless, this framework and the presented prototype are the starting point for future work that still needs further investigation in regards to P2P peer selection strategies. In addition, there is room for improving the efficiency and effectiveness of the algorithms presented in this work. Some ideas:

1) For instance, currently the number of landmarks per Autonomous System is a crude determination based on the number and diversity of DNS available from public sources. There is the possibility that many of these landmarks can be clustered together into one single point that does not necessarily have to be a DNS, it can also be a router.
2) Multiple zones, even across Autonomous Systems, can be collapsed into one zone given their similarities in terms of delays in the Internet Delay Space. When ISPs and Corporations obtain or exchange ASN (Autonomous System Numbers) based on commercial transactions, they would have a strong incentive to consolidate their

operations by removing unnecessary DNS servers and peering points. Therefore, it might be possible to improve the algorithms by working on the most up to date information regarding the ownership of the ASNs to reduce the search space.

3) There are places such as Internet Exchange Hotels (or NAPs) that reduce the value of using landmarks at those locations. The list of landmarks can be further reduced by removing such landmarks from the list, thus reducing memory requirements for the prototype.
4) In hindsight, the city and country kept by Autonomous System may be further reduced by picking up a subset of latitudes/longitude locations within each country of interest that fully represents the coverage of locations within a country. Then, when a new client arrives to register with the Tracker, the geographical distance based on the Haversine distance could be used to assign nodes to zones. This would reduce the amount of memory requirements further improving the footprint and probably the throughput of the algorithms during the registration process.
5) The translation from IP addresses to geographical location can be further improved by purchasing commercially available databases with such information to be kept in the same machine where the Tracker software is running. Such databases are not much larger in size than the one used during this research. What is interesting about some of these databases is that they also contain information about link failures and packet loss. It might be possible to group peers into zones based not only on distance and delays but also based on connectivity quality to other nodes within the same zone.

6) Budget restrictions made it difficult to go beyond 30 nodes when testing in the public Internet. Re-running the testing for 102 nodes, the observed averaged from most popular Trackers (Guo et-al, 2007), would make the results more accurate.

Summary

This research compared the performance of the random and biased peer selection techniques. Since there was no client implementation based on biased peer selection technique, an implementation framework and algorithms became a necessity. Based on the framework and algorithms presented, a prototype was implemented with the option to switch back and forth between both techniques for testing purposes. The comparison between selection techniques was conducted in both a lab and in the public Internet. The lab was set up in two stages. In stage 1, the algorithms were tested to verify that peers were being grouped by proximity. In stage 2, the two selection techniques were compared under different levels of bottlenecks (none, 0.5, 1.5 and 2.5 Mbit/s). Once testing was completed in a lab environment, the experiments were taken to the public Internet. Nodes were set up in Europe, North American and South America. The idea was to keep a very similar configuration to what had been done in the lab.

For the experiments in the lab, the prototype was set up to run with realistic conditions as nodes would experience in P2P networks operating in the real world. The experiments were setup to introduce the effects of traffic shaping and peering points. This setup resulted in a network that experienced packet loss, latency, (delay) distance between nodes, and peering points that could be manipulated and pre-determined for each experiment.

During the experiments, the two selection techniques were subject to adversarial situations such as traffic shaping activities in the lab and packet losses in the public Internet. As indicated, the lab was setup with a highly realistic local network but with a simulated WAN.

The experiments in the public Internet were setup to be a close approximation to the way P2P networks operate.

It was observed that the biased peer selection technique performs as well as the random selection technique in a perfect network without packet loss, link failures, or delays. For those networks where the traffic is exposed to real world conditions, the biased technique consistently outperformed the random technique.

The results from stage 2 of the simulation would have been sufficient to prove the advantages of the biased technique but questions regarding its performance in the public Internet may have persisted. Therefore, to gather empirical evidence from the real world, the experiments were also conducted over the public Internet. This new evidence brings a strong sense of reality to support the findings obtained during the simulation. Unfortunately, taking the testing to the Internet could only be done on a limited basis.

The ideal situation when taking the testing to the public Internet would have been to keep the same number of nodes as when the simulation was conducted. However, setting up nodes for testing required signing contracts with multiple datacenter hosting facilities and given the budget limitations of this project, their number and number of locations had to be minimized as much as possible without compromising the results. As it turned out, three continents and 24 nodes were set up and reliable test results were compiled to complete the analysis.

Based on the test results, the assertion made is that the biased technique not only helps individual users to download files but also helps network service providers by reducing unnecessary inter-ISP P2P traffic that can be avoided when peers interact with nearby peers. But it came at a cost. The random strategy could handle a higher rate of peer arrival during the registration with the tracker process. Succinctly, the biased strategy has its advantages but

comes at a cost in terms of reduced speed during the registration process and more memory/processing requirements compared to the random strategy.

The goal set for this research was to reduce both the cumulative download time and what recent research was showing to be a wide variance experienced by nodes when downloading files in the Internet. This goal was achieved as proven by the test results.

One of the contributions from this work is proof that the biased technique outperforms the random technique. This is trivial compared to the framework and algorithms made available to other researchers that may want to pursue multiple selections techniques without relying exclusively on analytical/simulation models. Future work may consider adding more nodes to the testing in the public Internet and by improving the algorithms contributed by this work.

References

Abrahao, B. and Kleinberg, R. (2008). On the Internet Delay Space Dimensionality. ACM/SIGCOMM Internet Measurement Conference, Vouliagmeni, Greece, 2008.

Androutsellis-Theotokis, S., and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. ACM Computing Surveys, 36(4). ACM.

Azureus (2010), standard BT client at http://www.azureus.org

Bindal, R., Cao, P., Chan, W., Medved, J., Suwala, G., Bates, T. and Zhang, A. (2006). Improving traffic locality in bittorrent via biased neighbor selection. In Proceedings of International Conference on Distributed Computing Systems, Lisbon, Portugal, July 2006.

Biskupski, B., Cunningham, R. and Meier, R. (2007). Improving Throughput and Node Proximity of P2P Live Video Streaming through Overlay Adaptation. In International Symposium on Multimedia (ISM'07), IEEE.

BitTorrent (2010), the site is at http://www.bittorrent.com

Cache Logic (2011), the site is at www.dcia.info/activities/p2pmsla2006/CacheLogic.ppt

CAIDA. (2011). The Cooperative Association for Internet Data Analysis: www.caida.org

Cheng, L., Hutchinson, N.C., and Ito, M.R. (2008). RealNet: A Topology Generator Based on Real Internet Topology. 22nd International Conference on Advanced Information Networking and Applications , AINAW 2008, 2008, pp. 526–532.

Chiu, Y. M., and Eun, D.Y. (2008). Minimizing file download time in stochastic peer-to-peer networks. IEEE/ACM Transaction on Networking (TON), 1692), pp. 253-266.

Chiu, Y.M. and Eun, D.Y. (2006). Minimizing File Download Time over Stochastic Channels in Peer-to-Peer Networks. 40th Annual Conference on Information Science and Systems, pp. 159-164. Print ISBN: 1-4244-0349-9.

Choffness, D. and Bustamante, F.E. (2008). Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems, In Proc. of ACM SIGCOMM 2008.

Cidr. (2011). A list of ASes advertising routes: www.cidr-report.org/as2.0/

Dick, M., Pacitti, E., and Kemme, B. (2009). Flower-CDN: a hybrid P2P overlay for efficient query processing in CDN. Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM.

Efthymiopoulos, N., Christakidis, A., Denazis, S. and Koufopavlou, O. (2008). L-CAN: Locality Aware Structured Overlay for P2P Live Streaming. Proceedings of the 11th IFIP/IEEE international conference on Management of Multimedia and Mobile

Networks and Services: Management of Converged Multimedia Networks and Services. ISBN: 978-3-540-87358-7.

Flegg, H. G. (2001). From Geometry to Topology. Dover Publications. ISBN: 0-486-41961-4

Gnutella (2011). A P2P implementation: http://www.gnu.org/philosophy/gnutella.html.

Guo, L., Chen, S., Xiao, Z., Tan, e., Ding, X. and Zhang, X. (2007). A Performance Study of BitTorrent-like Peer-to-Peer Systems. IEEE Journal, 25(1).

Grolimund, D., Meisser, L. Schmid, S. and Wattenhofer, R. (2006). Havelaar: A Robust and Efficient Reputation System for Active Peer-to-Peer Systems. Proceedings 1st Workshop on the Economics of Networked Systems (NetEcon).

Han, S. C., Yu, I. K., Xia, Y., and Newman, R. (2007). A Node Selection Algorithm for Many-to-many Mapping in Peer-to-peer Networks. Second International conference on Internet and Web Applications and Services.

Hilbert, D. (1891) Über die stetige Abbildung einer Linie auf ein Flächenstück. Mathematische Annalen 38 (1891), 459–460.

IANA. (2011). The Internet Assigned Numbers Authority (IANA) is responsible for the global coordination of the DNS Root, IP addressing, and other Internet protocol resources: www.iana.org

IP Locations (2011), IP lists from known geographical locations. The site can be found at www.ipaddresslocation.org/ip_ranges/get_ranges.php

Lafon, S. and Lee, A. B. (2006). Diffusion Maps and Coarse-Graining: A Unified Framework for Dimensionality Reduction, Graph Partitioning, and Data Set Parameterization. IEEE Trans. Pattern Anal. Mach. Intell., vol. 28 , pp. 1393-1403.

Laoutaris, N., Carra, D., and Michiardi, P. (2008). Uplink allocation beyond choke/unchoke: or how to divide and conquer best. Proceedings of the 2008 ACM CoNEXT Conference. ACM ISBN: 978-1-60558-210-8.

Le, T.M., and But, J. (2009). Bittorrent traffic classification. CAIA Technical report 091022A. Also at http://caia.swin.edu.au/reports/091022A/CAIA-TR-091022A.pdf.

Ledlie, J., Gardner, P., and Seltzer, M. (2007). Network Coordinates in the Wild. Proceedings of the Fourth USENIX Symposium on Network Systems Design and Implementation (NSDI).

Lehrfeld, M., and Simco, G. (2010). Choke-Based Switching Algorithm in Stochastic P2P Networks to Reduce File Download Duration. Proceedings of IEEE SoutheastCon 2010, 127-130.

Lipschutz, R.P. and Clyman, J. (2012). P2P Programs: Popular and Perilous. http://www.pcpitstop.com/spycheck/p2p.asp

Liu, Z., Dhungel, P., Wu, D., Zhang, C. and Ross, K.W. (2010). Understanding and Improving Ratio Incentives in Private BitTorrent Systems. IEEE International Conference on Distributed Computing Systems (ICDCS'10), to appear.

Locher, T., Moor, P., Schmid, S., and Wattenhofer, R. (2006). Free Riding in BitTorrent is Cheap. 5th Workshop on Hot Topics in Networks (HotNets), Irvine, California, USA, November 2006. Also at http://www.dcg.ethz.ch/alumni/thomasl.html

Madhyastha, H.V., Isdal, T., Piatek, M., Dixon, C., Anderson, T. Krishnamurthy, A. and Venkataramani, A. (2006). iPlane: An informationplane for distributed services. Operating Systems Design and Implementation (OSDI). USENIX.

Madhyastha, H.V., Katz-Bassett, E. and Anderson, T. (2009). iPlane nano: Path Prediction for Peer-toPeer Applications. In Proceedings of the 6yh USENIX Symposium on Netowkred Sytesm Design and Impllementation (pp 137-152). Boston, MA, USA.

MaxMind. (2011). GeoLite database to map IP addresses to longitudes and latitudes on planet earth. http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz.

Salmon, R., Tran, J. and Abhari, A. (2008). Simulating a file sharing system based on BitTorrent. Proceeding of the 2008 Spring simulation muticonference. Society for Computer Simulation International San Diego, CA, USA ©2008. ISBN:1-56555-319-5.

Shavitt, Y. and Shir, E. (2005). DIMES: Let the Internet Measure Itself. http://arxiv.org/abs/cs.NI/0506099

Sherman, A., Nieh, J., and Sten, C. (2009). FairTorrent: bringing fairness to peer-to-peer systems. International Conference On Emerging Networking Experiments And Technologies. Proceedings of the 5th international conference on Emerging networking experiments and technologies, pp. 133-144. ACM. ISBN: 978-1-60558-636-6.

Sirivianos, M., Park, J.H., Chen, R., Yang, X. (2007). Free-riding in BitTorrent Networks with the Large View Exploit. International Workshop on Peer-to-peer Systems (IPTPS). Also at http://www.cs.duke.edu/~msirivia

Snark (2010), built by Mark J. Wielard. It is a BitTorrent client written in Java in 2003. It has not been updated ever since. Site at http://www.klomp.org/Snark

Wang, Hui. (2007). ISP's Traffic Engineering and Peering Strategy. Publisher Chinese University of Hong Kong. ISBN 0549773738, 9780549773733

Wong, B. and Sirer, E.G. (2004). A Lightweight Approach to Network Positioning. Cornell University, Computing and Information Science Technical Report TR2004-1949, August 2004.

Wong, B., Slivkins, A., and Sirer, E.G. (2005). Meridian: A Lightweight Network Location Service without Virtual Coordinates. Cornell University, Computing and Information Science Technical Report TR2005-1982, February 2005.

Wong, B., Slivkins, A., Sirer, E. G. (2008). Approximate Matching for Peer-to-Peer Overlays with Cubit. Computing and Information Science Technical Reports. Issued 16-Dec-2008. https://dspace.library.cornell.edu/handle/1813/11651

Xie, H., Yang, Y.R., Krishnamurthy, A., Liu, Y.G. and Silberschatz, A. (2008). P4p: provider portal for applications. Proceeding SIGCOMM '08 Proceedings of the ACM SIGCOMM 2008 conference on Data.

Yin, H., Liu, X., Zhan, T., Sekar, V., Qiu, F., Lin, C., Zhang, H. and Li, B. (2010). LiveSky: enhancing CDN with P2P. Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) , 6(3), ACM.

Zhang, C., Dhungel, P., Wu, D., Ross, K.W. (2010). Unraveling the BitTorrent Ecosystem. IEEE Transactions on Parallel and Distributed Systems, 99(1), pp. 1, ISSN=1045-9219.

Zhang, M., Zhang, C., Pai, V.S., Peterson, L., and Wang, R. (2004). PlanetSeer: Internet path failure monitoring and characterization in wide-area services. Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, VOL 6 (pp. 12-21). San Francisco, CA: USENIX Association.

Appendices

Appendix A: BT Algorithms

Queueing

Peers are expected to keep a few unfulfilled requests on each connection. This prevents a full round trip from the download of one block to the beginning of the download of a new block. In the case of high BDP (bandwidth-delay-product, high latency or high bandwidth), this can result in a substantial performance loss.

This is the most crucial performance item. A static queue of 10 requests is reasonable for 16KB blocks on a 5 Mbit/s link with 50ms latency. Links with greater bandwidth are becoming very common so UI designers are urged to make this readily available for changing as a parameter. Cable modems which are known to be exposed to traffic policing might benefit by increasing the number of parallel connections.

Piece downloading strategy

Clients have the choice to download pieces in random order. A better strategy is to download pieces in rarest first order. The client can determine which strategy to follow by keeping the initial bit-field from each peer, and updating it with every message received. This way, the client can download the pieces that appear least frequently in the set of peer bit-fields. Note that any Rarest First strategy should include randomization among at least several of the least common pieces, as having many clients all attempting to jump on the same "least common" piece would not be advisable.

End Game

When a download is almost complete, there's a tendency for the last few blocks to trickle in slowly. To compensate, the client sends requests for all of its missing blocks to all of its active

peers. For the sake of efficiency, the client sends a cancel to everyone else every time a block is received.

Choking and Optimistic Unchoking

Choking prevents the client from setting up too many simultaneous uploads to other peers. This is done to achieve good TCP performance. The TCP congestion control algorithm degrades in performance when too many connections are active. In addition, choking enable each peer to take advantage of the tit-for-tat algorithm in an attempt to achieve a consistent download rate.

Optimistic Unchoking enables the client to try new peer nodes. The idea is to find better nodes than the current active peers.

Reciprocation and number of uploads capping is managed by unchoking the four peers which have the best upload rate and are interested. This maximizes the client's download rate. These four peers are referred to as downloaders, because they are interested in downloading from the client.

Peers who have a better upload rate (as compared to the downloaders) but aren't interested get unchoked. If they become interested, the downloader with the worst upload rate gets choked. If a client has a complete file, it uses its upload rate rather than its download rate to decide which peers to unchoke.

For optimistic unchoking, at any one time there is a single peer which is unchoked regardless of its upload rate (if interested, it counts as one of the four allowed downloaders). Which peer is optimistically unchoked rotates every 30 seconds. Newly connected peers are three times as likely to start as the current optimistic unchoke as anywhere else in the rotation. This gives them a decent chance of getting a complete piece to upload.

Anti-snubbing

Occasionally a peer will be choked by all peers whom it was formerly downloading from. In such cases it will usually continue to get poor download rates until the optimistic unchoke finds better peers. To mitigate this problem, when over a minute goes by without getting any piece data while downloading from a peer, it is considered to be "snubbed" by that peer and doesn't upload to it except as an optimistic unchoke. This frequently results in more than one concurrent optimistic unchoke, (an exception to the exactly one optimistic unchoke rule mentioned above), which causes download rates to recover much more quickly when they falter.

Appendix B: Scripts used during testing

FriendTorr: code to listen to requests from Tracker

```
# wait for an outside request using http protocol:
while (my $c = $d->accept)
  {
     my $head =  "<html><head></head><body><p>";
     my $tail =  "</p></body></html>";
     while (my $r = $c->get_request)
       {
          # OK, someone requested services using an URL
          print "url=".$r->url->path."\n" if ($debug);
          # We now take the GET requests for http://localhost:9999/peers
          #-------------------------------------------------------
          #     GET request for PEERS
          #     using the proximity based selection strategy
          #-------------------------------------------------------
          if ($r->method eq 'GET' and $r->url->path eq "/peers")
            {
               # we must have the IP address of the requestor
               # supposed to be a peer requesting neighbor peers
               my $date = `date`;
               # print "$date\n";

               my $query = $r->url->query;
               print "$date $PROCESS has request from: $query\n";
               my @tmp = split(/\=/,$query); # separate ip=x.x.x.x
               my $client_ip = 0;  # in case we were sent a bad IP address
               my $msg  =  "WARNING: no peers found"; # default message
               #  we must have received at least ip=x.x.x.x in the query
               if ( defined($tmp[0]) && defined($tmp[1]))
                 {
                    # ok, we got something like ip=x (but is it an ip@?)
                    # if (index($tmp[1],'.')>0)
                    if (is_ipv4($tmp[1]))
                      {
                         # it seems to be a valid IP address
                         $client_ip = $tmp[1];
                         # go get the closest peers from this client
                         $msg = closest_peers($client_ip);
                      }
                 }
               print "HTTP response for client=$client_ip\n" if ($debug);
               print "HTTP BODY: $msg\n" if ($debug);
               #
               # send response :-)
               # $c->send_file_response("/etc/hosts");
               #
               my $response = HTTP::Response->new(200);
               $response->header("Content-Type" => "text/html");
               $response->content("$head $msg $tail");
               $c->send_response($response);
               print "response msg=$msg\n" if ($debug);
            }
          # We now take the GET requests for http://localhost:9999/random
```

```
#----------------------------------------
#     GET request for RANDOM
#     using Random Peer selection strategy
#----------------------------------------
elsif ($r->method eq 'GET' and $r->url->path eq "/random")
  {
     # we must have the IP address of the requestor
     # supposed to be a peer requesting neighbor peers
     my $query = $r->url->query;
     print "request from: $query\n";
     my @tmp = split(/\=/,$query); # separate ip=x.x.x.x
     my $client_ip = 0;  # in case we were sent a bad IP address
     my $msg  =  "WARNING: no peers found"; # default message
     #  we must have received at least ip=x.x.x.x in the query
     if ( defined($tmp[0]) && defined($tmp[1]))
       {
          # ok, we got something like ip=x (but is it an ip@?)
          # if (index($tmp[1],'.')>0)
          if (is_ipv4($tmp[1]))
            {
               # it seems to be a valid IP address
               $client_ip = $tmp[1];
               # go get the closest peers from this client
               $msg = random_peers($client_ip);
            }
       }
     print "HTTP response for client=$client_ip\n" if ($debug);
     print "HTTP BODY: $msg\n" if ($debug);
     #
     # send response :-)
     # $c->send_file_response("/etc/hosts");
     #
     my $response = HTTP::Response->new(200);
     $response->header("Content-Type" => "text/html");
     $response->content("$head $msg $tail");
     $c->send_response($response);
     print "response msg=$msg\n" if ($debug);
  }
#
# We now take other query type requests such as
# the list of countries, cities or regions to list
# per DNS.  Or optionally, the number of peers per
# country, region or city that have been registered.
#
#----------------------------------------------------
#     GET request for DNS information
#----------------------------------------------------
elsif ($r->method eq 'GET' and $r->url->path eq "/dns")
  {
     # supposed to be requesting information about DNS
     # servers (country/region/city)
     my $query = $r->url->query;
     print "query: $query\n" if ($debug);
     my @tmp = split(/\=/,$query); # type of request?
     my $msg  =  "ERROR: invalid GET request"; # default message
     #  we must have received at least xx=yy in the query
```

```
            if ( defined($tmp[0]))
              {
                # ok, we got something like xx=yy (but what is it?)
                #
                if ( defined($tmp[0]) && defined($tmp[1]))
                  {
                    if ($tmp[1] =~ 'countries')
                      {
                        $msg = dns_countries($tmp[0]);
                      }
                    if ($tmp[1] =~ 'cities')
                      {
                        $msg = dns_cities($tmp[0]);
                      }
                    if ($tmp[1] =~ 'regions')
                      {
                        $msg = dns_regions($tmp[0]);
                      }
                    if ($tmp[1] =~ 'peers')
                      {
                        $msg = peers_counter($tmp[0]);
                      }
                  }
              }
            print "HTTP DNS response\n" if ($debug);
            #
            # send response :-)
            # $c->send_file_response("/etc/hosts");
            #
            my $response = HTTP::Response->new(200);
            $response->header("Content-Type" => "text/html");
            $response->content("$head $msg $tail");
            $c->send_response($response);
          }
        #-----------------------------------------------------
        #     A request we cannot handle
        #-----------------------------------------------------
        else
          {
            # the requester did not use http://localhost:9999/peers
            # in their request
            print "$PROCESS ERROR: request cannot be handled\n";
            print "request=". $r->url->path ."\n";
            print "query=  ".  $r->url->query ."\n";
            $c->send_error(RC_FORBIDDEN)
          }
      }
    print "close\n" if ($debug);
    $c->close;
    undef($c);
  }
```

FriendTorr: code to build the hash tables

```
#
#############################################################################
###     BUILD %dns2dns, %dnsInfo                                          ###
###     where %dns2dns will contain the RTT latency between DNS servers#
```

```
###     and %dnsInfo will contain the coordinates of the DNS servers ###
###     note: the coordinates are in an array [longitude,latitude]   ###
##############################################################################
#
# open the facility to convert from IP address to coordinates
my $gi   = Geo::IP->open(GEO_LITE_FILE, GEOIP_STANDARD);
#
# open and read file that contains the DNS to DNS RTT latencies
my $file = INPUT_DNS_FILE;
open (DATA,"<$file") or die "file=$file not found: $!\n";
print "open file=$file\n" if $debug;
#
while (my $line = <DATA>)
  {
     $count++;                    # count number of records read
     chomp($line);
     $line           =~ s/^\s+//;
     $line           =~ s/\s+$//;
     next if $line    =~ /^$/;
     next if $line    =~ /^#/;
     # line format: dns=xxx country=xxx region=xxxx city=xxx lon=xxx lat=xxx
     my @values       = split(/\=/,$line);
     next unless defined $values[1];
     next unless defined $values[2];
     next unless defined $values[3];
     next unless defined $values[4];
     next unless defined $values[5];
     my ($dnsip,$x1)   = split(/\s+/,$values[1]);
     my ($country,$x2) = split(/\s+/,$values[2]);
     my ($region,$x3)  = split(/city/,$values[3]);
     my ($city,$x4)    = split(/lon/,$values[4]);
     my ($lon,$x5)     = split(/\s+/,$values[5]);
     $values[6] = MISSING unless defined($values[6]);
     my $lat          = $values[6];
     next unless defined $lon;
     next unless defined $lat;
     #
     next unless (is_ipv4($dnsip));          # valid IP address?
     #    for each DNS server, save the coordinates
     $dnsInfo{$dnsip} = (     [$lat,
                               $lon,
                               $country,
                               $region,
                               $city]);
  }
close DATA;
#
# open and read file that contains the DNS to DNS RTT latencies
$file = INPUT_DNS2DNS;
open (DATA,"<$file") or die "file=$file not found: $!\n";
print "open 2nd file=$file\n" if $debug;
#
$num=0;
while (my $line = <DATA>)
  {
     $count++;                    # count number of records read
```

```
    chomp($line);
    $line          =~ s/^\s+//;
    $line          =~ s/\s+$//;
    next if $line    =~ /^$/;
    next if $line    =~ /^#/;
    my @values      = split(/\=/,$line); # dns1=xxx dns2=xxx RTT=xxx
    next unless defined $values[1];
    next unless defined $values[2];
    next unless defined $values[3];
    my ($dns1ip,$x1)   = split(/\s+/,$values[1]);
    my ($dns2ip,$x2)   = split(/\s+/,$values[2]);
    my $RTT           = $values[3];
    #
    next unless (is_ipv4($dns1ip));          # valid IP address?
    next unless (is_ipv4($dns2ip));          # valid IP address?
    #    for each DNS pair, save the RTT
    $dns1ip         =~ s/^\s+//;
    $dns1ip         =~ s/\s+$//;
    $dns2ip         =~ s/^\s+//;
    $dns2ip         =~ s/\s+$//;
    $RTT            =~ s/^\s+//;
    $RTT            =~ s/\s+$//;
    $num++;
    print "$num RTT=$RTT\n" if $debug;
    $dns2dns{$dns1ip}{$dns2ip} = $RTT;
  }
close DATA;
#
###############################################################################
###                BUILD @index_dns by country,region,and city    ###
### to quickly identify the loction of the DNSs under consideration. ###
###############################################################################
#
# DNS
#
my %index_dns = ();
print "build index\n" if $debug;
foreach my $item (keys %dnsInfo)
  {
    my $cc     = MISSING;
    my $region = MISSING;
    my $city   = MISSING;
        #
    $cc     = $dnsInfo{$item}[2] if (defined($dnsInfo{$item}[2]));
    $region = $dnsInfo{$item}[3] if (defined($dnsInfo{$item}[3]));
    $city   = $dnsInfo{$item}[4] if (defined($dnsInfo{$item}[4]));
    $cc     = MISSING unless (defined($cc));
    $region = MISSING unless (defined($region));
    $city   = MISSING unless (defined($city));
    #
    $cc     = MISSING if (length($cc)     < 2);
    $region = MISSING if (length($region) < 2);
    $city   = MISSING if (length($city)   < 2);
    # keep an array of all those DNS by country, region and city
    $cc             =~ s/^\s+//;
    $cc             =~ s/\s+$//;
```

```
    $region         =~ s/^\s+//;
    $region         =~ s/\s+$//;
    $city           =~ s/^\s+//;
    $city           =~ s/\s+$//;
    $item           =~ s/^\s+//;
    $item           =~ s/\s+$//;
    push @{ $index_dns{$cc}{$region}{$city} },$item;
  }
```

FriendTorr: code to implement the algorithms from Chapter 3

```
#
# ----------------------------------------------------------------------
#   SUBROUTINES
# ----------------------------------------------------------------------
#
##############################################################################
### random_peers:
###         return neighbor list based on a random selection.         ###
##############################################################################
sub random_peers
  {
    my $client = shift;
    # note: the array @random_peers has list of registered peers
    my @peer_list = ();  # place to put list of peers to return
    #
    # The first node to register:
    #
    if ($#random_peers < 0)
      {
        push (@random_peers,$client);
        return MISSING;
      }
    #
    # let's make sure we don't register the same node twice
    #
    my $item = '';
    my $flag = FALSE;
    foreach $item (@random_peers)
      {
        $flag = TRUE if ($item =~ $client);
      }
    #
    # Case when the requested list of peers is more than
    # the number of peers already registerd
    #
    if ($#random_peers < DEFAULT_NUM_PEERS)
      {
        foreach $item (@random_peers)
          {
            push (@peer_list,$item) unless ($item =~ $client);
          }
        push (@random_peers,$client) unless ($flag);
        return (join('', ',@peer_list));
      }
    #
    # Now we can pick nodes at random from @random_peers
```

```
    # to include in the @peer_list
    for (my $i=1; $i <= DEFAULT_NUM_PEERS; $i++)
     {
        my $random = rand($#random_peers);
        $random    = int($random);
        push (@peer_list,$random_peers[$random]);
     }
    push (@random_peers,$client) unless ($flag);
    return (join(", ",@peer_list));
  }
#############################################################################
### closest_peers:                                        ###
###         return neighbor list of location biased peers.         ###
#############################################################################
sub closest_peers
  {
    my $client = shift;
    # do we already have the closest DNS to this client?
    unless (exists($reg_peers{$client}))
     {
        print "Go get the nearest DNS for $client\n" if ($debug);
        $reg_peers{$client} = assign_nearest_dns($client);
        print "$reg_peers{$client} is nearest DNS to $client\n" if ($debug);
     }
    #
    ###################################################################
    ###   find and return peer list:                          ###
    ###             Go through the list of all DNS servers, one at ###
    ###             a time, for each of the selected DNS servers   ###
    ###             go through registered peer list and pick up    ###
    ###             those neighbor peers up to DEFAULT_NUM_PEERS.  ###
    ###################################################################
    #
    my @peer_list = ();  # place to put list of peers to return
    $count        =  0;
    #
    # We now switch from Geographical distances to RTT latencies
    # This means that triangular violations can take place due to
    # Internet risks causing further delays along one path and not another.
    #
    my $fromDns = $reg_peers{$client}; # get the closest DNS to this client
    #
    #  @ToDns array sorted from closest RTT to farthers RTT latency
    #
    my @ToDns = sort { $dns2dns{$fromDns}{$a} <=>
                       $dns2dns{$fromDns}{$b} }
      keys %{$dns2dns{$fromDns}};
    if ($verbose)
     {
        print "RTT from/to DNS\n";
        foreach my $close (@ToDns)
         {
            print "$fromDns to $close: RTT=$dns2dns{$fromDns}{$close}\n";
         }
     }
        #
```

```
  #   We also need to look for peers that have $fromDns as their
  #   closest DNS.  The idea is that peers that share the same $fromDns
  #   are likely to be located nearby the client under consideration.
  #
  unshift(@ToDns,$fromDns);  # put $fromDns at the top of the array
  #
  # Given $fromDns, the nearest DNS to $client, we have @ToDns
  # that contains the list of destination DNS sorted from closest to
  # farthest based on the RTT between source and destination DNS.
  #
CLOSE:
  #
  # starting from $fromDNS to other DNSs in order of RTT proximity,
  # find registered nodes until quota for list of peers to return is met.
  #
  foreach my $close ( @ToDns )
   {
      # $close has the IP of the "TO" destination DNS
      # in ascending order from closest to farthest in terms of RTT
      # (note: I verified this claim)
      # Now, go get the peers that are registered to this "TO" DNS
      #
      if ($verbose)
        {
           print "source DNS=$fromDns to destination DNS=$close\n";
        }
     FIND:
      #
      # %reg_peers has the closest DNS for each registered node
      # For the DNS of each registered node, find a match to $close DNS.
      #
      foreach my $peer (keys %reg_peers)
        {
           # For the destination DNS=$close, find registered nodes
           # that may have the same $close DNS as its nearest DNS.
           if ($verbose)
             {
                print "check if peer=$peer has the same ";
                print "nearby DNS=$close as $client\n";
             }
           #    go through list of registered nodes and
           #    check to see if DNS is the same as nearby DNS = $close
           if ($reg_peers{$peer} =~ $close)
             {
                #
                # OK, this peer is registered to $close as nearest DNS
                #
                if ($verbose)
                  {
                     print "peer=$peer share DNS=$close ";
                     print "with $client\n";
                  }
                # avoid a peer that is the same as the client
                # i.e. cannot be a peer to itself.
                if ($peer =~ $client)
                  {
```

```
                    # ignore since it is the same as requestor
                    print "peer same as requestor\n" if ($debug);
                  }
                 else
                  {
                    # add to our potential list of peers
                    print "$client has neighbor=$peer\n" if ($debug);
                    push(@peer_list, $peer) ;
                    # stop looking when maximum is found
                    last FIND if ($#peer_list+1 >= DEFAULT_NUM_PEERS);
                  }
               }
           }
        # stop looking at more nearby DNS's when maximum is found
        last CLOSE if ($#peer_list+1 >= DEFAULT_NUM_PEERS);
      }
    #
    print "Done with finding neighbors for $client\n" if ($debug);
    @ToDns = ();
    $#peer_list = DEFAULT_NUM_PEERS -1 if ($#peer_list+1>=DEFAULT_NUM_PEERS);
    if ($#peer_list >= 0)
      {
        print "List of peers is being returned\n" if ($debug);
        my @shuffle =shuffle(@peer_list);
        return (join(", ",@shuffle));     # 01/07/2012
        # return (join(", ",@peer_list)); # 01/07/2012
      }
    else
      {
        print "Empty list of peers is being returned\n" if ($debug);
        return MISSING;
      }
  }
##############################################################################
### assing_nearest_dns:                                    ###
###         return nearest DNS based on geographical coordinates.   ###
##############################################################################
sub assign_nearest_dns
  {
    # find nearest DNS to client
    my $client = shift;
    print "looking for closest DNS to $client\n" if ($debug);
    my $rec    = $gi->record_by_addr($client);
    warn   "WARNING: ip $client without coordinates\n" unless defined ($rec);
    #
    my $lon    = MISSING;
    my $lat    = MISSING;
    my $cc     = MISSING;
    my $region = MISSING;
    my $city   = MISSING;
    #
    # 3/14/2012 - for testing inside lab:
    #
    my @data     =   split(/\./,$client);
    #
    #
```

```
$lon      = MISSING unless (defined($lon));
$lat      = MISSING unless (defined($lat));
$cc       = MISSING unless (defined($cc));
$region   = MISSING unless (defined($region));
$city     = MISSING unless (defined($city));
#
$lon      = MISSING if (length($lon)    < 1);
$lat      = MISSING if (length($lat)    < 1);
$cc       = MISSING if (length($cc)     < 2);
$region   = MISSING if (length($region) < 2);
$city     = MISSING if (length($city)   < 2);

if ($debug)
 {
    print "Client: $client\n";
    print "Location: $cc $region $city ";
    print "longitude($lon) latitude($lat)\n";
 }
#
#########################################################################
###   STEP 1:                                         ###
###            Find the country, region, or city to match.    ###
###            Get DNS_RANGE candidates for the nearest.      ###
#########################################################################
#
my @candidates = ();   # add to this array those nearby DNS servers
my @tmp        = ();
my $flag       =  1;   # to keep looking for peers
#      in case there is no DNS for this client in his country:
unless (exists($index_dns{$cc}))
 {
    print  "WARNING: No DNS in country $cc for client=$client\n" ;
    return "WARNING: No DNS in country $cc for client=$client" ;
 }
#
#     we have at least one nearby DNS in this country
#
if ($debug)
 {
    print "At least one nearby DNS in country=$cc for client=$client\n";
 }
@tmp = ();  # reset
#      check for peers for this client in his city:
my $tmp =  exists( $index_dns{$cc}{$region}{$city} );
if ($tmp)
 {
    #  we found some nearby DNS in this client's city
    print "found all nearby DNS in city=$city\n" if ($debug);
    @tmp = @{ $index_dns{$cc}{$region}{$city} }; # take entire list
 }
#
#   There may have been none or some DNS in this client's city but
#   have we met our quota for number of near-by DNS servers?
#
if ($#tmp+1 >= DNS_RANGE)
 {
```

```
    $#tmp = DNS_RANGE -1;  # cut off extras
    @candidates = @tmp;
    $flag = 0;  # we are done, stop looking
    warn "stop looking for more nearby DNS, we got them\n" if ($debug);
  }
#
#  If we have to continue looking for near-by DNS servers,
#  check in the region (city by city) until we meet our quota.
#  note: do not check for the client's city since it is already
#        searched.
#
if ($flag)
  {
    # look for more DNS candidates in the region
    print "We need to look for more DNS candidates\n" if ($debug);
    if ( exists( $index_dns{$cc}{$region} ) )
      {
        # let's see what DNS we can find in each city (i.e. item)
        print "Look for nearby DNS in region=$region\n" if ($debug);
        REGION: foreach my $item (keys %{ $index_dns{$cc}{$region} } )
          {
            # we already look in the same city so avoid
            next if ($city =~ $item); # avoid duplicates
            # item is the DNS in a given city
            print "City=$item in same region=$region\n" if ($debug);
            push @tmp, @{ $index_dns{$cc}{$region}{$item} };
            # have we met our quota?
            last REGION if ($#tmp + $#candidates +2 >= DNS_RANGE);
          }
      }
  }
#
#   There may have been none or some DNS in this client's region but
#   have we met our quota for number of near-by DNS servers?
#
if ($#tmp + $#candidates + 2 >= DNS_RANGE)
  {
    $#tmp = DNS_RANGE -1 - $#candidates;  # cut off extras
    push @candidates, @tmp;
    $flag = 0;  # we are done, stop looking
    warn "stop looking for more nearby DNS, we got them\n" if ($debug);
  }
else
  {
    # just push whatever near-by DNS we have found so far, if any
    push @candidates, @tmp;
    print "List of nearby DNS candidates: @tmp\n" if ($verbose);
  }
#
#  if we have to continue looking for near-by DNS servers
#  check in the country (region by regions and city by city)
#  until we meet our quota.
#  Note: skip region and city for client since we already searched them.
#
@tmp = ();  # reset
if ($flag)
```

```
 {
    if ($debug)
     {
        print "Now search for nearby DNS to the entire country\n";
     }
    # look for more DNS candidates in the country
    if ( exists( $index_dns{$cc} ) )
     {
        print "Looking in country=$cc for nearby DNS\n" if ($debug);
        # let's see what DNS we can find in each region
       REGION2:
        foreach my $reg  (keys %{ $index_dns{$cc} } )
         {
            next if ($region =~ $reg); # avoid duplicates
            print "Look in region=$reg\n" if ($debug);
           CITY2:
            foreach my $item2 (keys %{ $index_dns{$cc}{$reg} } )
             {
                next if ($city =~ $item2); # avoid duplicates
                # item is the DNS in a given city
                if ($#{ $index_dns{$cc}{$reg}{$item2} } >= 0)
                 {
                    if ($debug)
                     {
                        print "found nearby DNS ";
                        print "in city=$item2\n";
                     }
                    push @tmp, @{$index_dns{$cc}{$reg}{$item2}};
                 }
                last CITY2 if ($#tmp + $#candidates+2>=DNS_RANGE);
             }
            last REGION2 if ($#tmp + $#candidates+2 >= DNS_RANGE);
         }
     }
 }
#
#   There may have been none or some DNS in this client's country but
#   have we met our quota for number of near-by DNS servers?
#
if ($#tmp + $#candidates +2 >= DNS_RANGE)
 {
    push @candidates, @tmp;
    $#candidates = DNS_RANGE -1;  # no need to return longer list
    $flag = 0;  # we are done, stop looking
     warn "stop looking for more nearby DNS, we got them\n" if ($debug);
 }
else
 {
    # just push whatever near-by DNS we found, if any
    push @candidates, @tmp;
    print "More nearby DNS candidates: @tmp\n" if ($verbose);
 }
#   We checked earlier but may be no near DNS server after all
if ($#candidates < 0)
 {
    warn "WARNING: After all, no near DNS in country $cc\n";
```

```
    return "After all, no near DNS in country $cc";
  }
#
if ($debug)
  {
    if ($#candidates +1 < DNS_RANGE)
      {
        my $tmp2 = $#candidates + 1;
        print "client=$client has $tmp2 candidate nearby ";
        print "DNSs \n ";
      }
  }
#
#########################################################################
###   STEP 2:                                          ###
###         Keep only unique DNS candidates (no duplicates)    ###
#########################################################################
#
# remove duplicates from candidate list
my %seen   = ();
my @unique = ();
foreach my $item3 (@candidates)
  {
    # print "item=$item3\n";
    unless (exists($seen{$item3}))
      {
        $seen{$item3} = 1;
        push (@unique, $item3);
      }
  }
@candidates = (); # reset
%seen       = (); # reset
print "only unique nearby candidates to be considered\n" if ($debug);
#
#########################################################################
###   STEP 3:                                          ###
###             For those candidate (unique) DNS servers,      ###
###             calculate the distance (Haversine) from        ###
###             the requestor $client to the candidate         ###
###             DNS servers                                    ###
###             using haversine on the coordinate system of    ###
###             latitudes and longitudes.                      ###
#########################################################################
#
print "get geo distance to nearby DNS\n" if ($debug);
my $gis = GIS::Distance->new();
$gis->formula( 'Haversine' );
my %close = ();
my $closest_dns = '';  # the one with minimu distance in miles
my $min_miles   = 999999999;
foreach my $item3 (@unique)
  {
                                # for the candidate DNS:
    my $dns_lat  = $dnsInfo{$item3}[1];   # pull out the latitude
    my $dns_lon  = $dnsInfo{$item3}[0];   # pull out the longitude
    print "(lat,long): ($lat,$lon) ($dns_lat,$dns_lon)\n" if ($debug);
```

```
        #      calculate the distance from client to candidate DNS
        my $distance = $gis->distance ( $lat,$lon => $dns_lat,$dns_lon );
        my $miles    = $distance->miles();
        $miles       = sprintf("%0.2f",$miles);
        print "miles=$miles\n" if ($debug);
        if ($miles < $min_miles)
          {
            $min_miles = $miles;
            $closest_dns = $item3;
          }
        #     save the distance from requestor to DNS in miles
        ##    $close{$item3} = $miles;
      }
    @unique = ();
    #          sort distance from requestor to all DNS candidates
    ##  my @dns_near = sort { $close{$a} <=> $close{$b} } keys %close;
    %close = ();
    #          pick closest DNS server
    if ($debug)
      {
        print "Closest DNS: $closest_dns\n" ;
        print "Location: $dnsInfo{$closest_dns}[2] ";
        print "$dnsInfo{$closest_dns}[3] ";
        print "$dnsInfo{$closest_dns}[4] ";
        print "longitude($dnsInfo{$closest_dns}[0]) ";
        print "latitude($dnsInfo{$closest_dns}[1])\n";

      }
    #
    # is the $client and $closest_dns in the same country?
    #
    unless ($cc =~ $dnsInfo{$closest_dns}[2])
      {
        print "WARNING: client $client and closest DNS ";
        print "$closest_dns are not in the same country ";
        print "$cc and $dnsInfo{$closest_dns}[2] respectively\n";
      }
    #
    # take the first one, i.e. the closest DNS to $client
    #
    print "For $client found nearest DNS=$closest_dns\n" if ($debug);
    ### my $closest_dns = $dns_near[0]; # eureka
    ### @dns_near = ();
    #if ($verbose)
    #  {
    #     print "Registered Peers:\n";
    #     foreach my $peer (keys %reg_peers)
    #       {
    #          print "$peer -> DNS: $reg_peers{$peer}\n";
    #       }
    #  }
    return ($closest_dns);
  }
##############################################################################
### dns_countries:                                    ###
###         return list of countries.                   ###
```

```
#############################################################################
sub dns_countries
  {
     my $request = shift;
     my @tmp     = ();
     foreach my $country (keys %index_dns )
       {
          push @tmp, $country;
       }
     if ($#tmp >= 0)
       {
          return (join('', ',@tmp));
       }
     else
       {
          return MISSING;
       }
  }
#############################################################################
### dns_regions:                                          ###
###         return list of regions.                         ###
#############################################################################
sub dns_regions
  {
     my $request = shift;
     my @tmp     = ();
     foreach my $cc (keys %index_dns )
       {
          foreach my $region (keys %{$index_dns{$cc}} )
            {
               push @tmp, $region unless ($region =~ /\./);
            }
       }
     if ($#tmp >= 0)
       {
          return (join('', ',@tmp));
       }
     else
       {
          return MISSING;
       }
  }
#############################################################################
### dns_cities:                                          ###
###         return list of cities.                         ###
#############################################################################
sub dns_cities
  {
     my $request = shift;
     my @tmp     = ();
     foreach my $cc (keys %index_dns )
       {
          foreach my $region (keys %{$index_dns{$cc}} )
            {
               foreach my $city (keys %{$index_dns{$cc}{$region}} )
                 {
```

```
                push @tmp, $city unless ($city =~ /\./);
              }
           }
       }
     if ($#tmp >= 0)
       {
          return (join('', ',@tmp));
       }
     else
       {
          return MISSING;
       }
  }
################################################################################
### peers_counter:                                          ###
###         Number of peers per country, region and city.          ###
################################################################################
sub peers_counter
  {
     my $request = shift;
     my $count = 0;
     my %stats   = ();
     foreach my $client (keys %reg_peers )
       {
          my $rec    = $gi->record_by_addr($client);
          #
          # 3/14/2012 - for testing inside lab:
          #
          my $lon     = MISSING;
          my $lat     = MISSING;
          my $country = MISSING;
          my $region  = MISSING;
          my $city    = MISSING;
          #
          # 3/14/2012 - for testing inside lab:
          #
          my @data     =   split(/\./,$client);
          #
          #
          $lon       = MISSING unless (defined($lon));
          $lat       = MISSING unless (defined($lat));
          $country   = MISSING unless (defined($country));
          $region    = MISSING unless (defined($region));
          $city      = MISSING unless (defined($city));
          #
          $lon       = MISSING if (length($lon)    < 1);
          $lat       = MISSING if (length($lat)    < 1);
          $country   = MISSING if (length($country)     < 2);
          $region    = MISSING if (length($region) < 2);
          $city      = MISSING if (length($city)   < 2);
          #
          if ( exists($stats{$country}{$region}{$city}) )
            {
               ($stats{$country}{$region}{$city})++; # counter + 1
               $count++;
            }
```

```
        else
          {
            $stats{$country}{$region}{$city} = 1; # initialize
            $count++;
          }
      }

    my $msg = '';
    foreach my $country (keys %stats)
     {
        foreach my $region (keys %{$stats{$country}})
          {
            foreach my $city (keys %{$stats{$country}{$region}})
              {
                $msg .= "<p>";
                $msg .= "$country $region $city: ";
                $msg .= "$stats{$country}{$region}{$city}";
                $msg .= "\n";
                $msg .= "</p>";
              }
          }
      }
    $msg .= "<p>Number of Peers: $count</p>";
    return "$msg";
  }
```

Driver: code for stage 1 of the simulation

```
#
##############################################################################
###     Generate Calls to Tracker/FriendTorr     ###
##############################################################################
#
my $t0 = Benchmark->new;
my $start = `date`;
chomp($start);
print "******************************************************************************\n";
print "$PROCESS: processing started: $start\n";
if (RANDOM_STRATEGY)
  {
    print "PARAMETER: RANDOM STRATEGY FOR PEER SELECTION\n";
  }
else
  {
    print "PARAMETER: PROXIMITY STRATEGY FOR PEER SELECTION\n";
  }
#
#    get object that contains the GeoLite database (via C-API interface)
#
my $gi = Geo::IP->open("./GeoLite/GeoLiteCity.dat", GEOIP_STANDARD);
#
#    initialize some variables
#
my %country = ();
my @ip      = ();
my $count   =  0;
my $record  =  0;
my $used    =  0;
```

```
#
#
my $code1 = COUNTRY_1;  # 1st country to select for IP ranges
my $code2 = COUNTRY_2;  # 2nd country
$country{'AR'} = AR_IP_FILE if ($code1 =~ 'AR' || $code2 =~ 'AR');
$country{'AT'} = AT_IP_FILE if ($code1 =~ 'AT' || $code2 =~ 'AT');
$country{'CA'} = CA_IP_FILE if ($code1 =~ 'CA' || $code2 =~ 'CA');
$country{'FR'} = FR_IP_FILE if ($code1 =~ 'FR' || $code2 =~ 'FR');
$country{'HK'} = HK_IP_FILE if ($code1 =~ 'HK' || $code2 =~ 'HK');
$country{'IT'} = IT_IP_FILE if ($code1 =~ 'IT' || $code2 =~ 'IT');
$country{'NZ'} = NZ_IP_FILE if ($code1 =~ 'NZ' || $code2 =~ 'NZ');
$country{'US'} = US_IP_FILE if ($code1 =~ 'US' || $code2 =~ 'US');
#
#----------------------------
# Create sample and urn files
# Comments:
# --------
# this is done to be able to
# run multiple times without
# having to recreate the
# SAMPLES and URN arrays
# each time.
#----------------------------
#
my $sFile  = SAMPLE_FILE;
my $uFile  = URN_FILE;
if (CREATE_SAMPLES)
  {
     print "PARAMETER: CREATE SAMPLES = TRUE\n";
     my $rc     = create_files($sFile,$uFile) ;
     print "URN and SAMPLES files were created\n";
     print "Forced to EXIT, run $PROCESS with parameter CREATE_SAMPLES set to 0
\n";
     print "Also in $PROCESS, set parameter SKIP_REGISTER to 0 to pre-register
nodes\n";
     print "in addition, bring up daemon named sim_select_peers.pl\n";
     exit OK;
  }
#
# -------------------------------
# load the urn and sample arrays
# from the save files from
# previous create_file subroutine.
# -------------------------------
#
my @urn    = ();  # place for the client IPs to be registered with daemon
my @sample = ();  # place for the peers that will need to find neighbors
open FILE, "$sFile" ||
  die "cannot open file $sFile:$!\n";
chomp( @sample = <FILE> );
close FILE;
open FILE, "$uFile" ||
  die "cannot open file $uFile:$!\n";
chomp( @urn = <FILE> );
close FILE;
#
```

```
#############################################################################
#   Register all the IP addresses from the URN                  ###
#   This is equivalent to a warm up period before               ###
#   we start to collect statistics from SAMPLES.                ###
#############################################################################
#
if ($debug)
  {
     print "-------------------------------\n";
     print "Register IP addresses from URN \n";
     print "-------------------------------\n";
  }
print "Make sure SIM_SELECT_PEERS daemon is running with READY message\n";
$count = 0;
unless (SKIP_REGISTER)
  {
     my $time = `date`;
     print "PARAMETER: register peers = TRUE\n";
     print "Process to register peers started: $time\n";
     #
     # process each IP address representing each peer from inside @urn
     #
     foreach my $client (@urn)
       {
          $count++;
          #
          # call sim_select_peers.pl (daemon must be running in same host)
          # according to strategy: random or proximity
          #
          if (RANDOM_STRATEGY)
            {
               # when random peer selection strategy is being used:
               $query = "http://localhost:9999/random?ip=$client";
            }
          else
            {
               # when the proximity based peer selection strategy is used:
               $query = "http://localhost:9999/peers?ip=$client";
            }
          sleep 1;  # give it some time so we don't burden the daemon
          #
          #  check to see what we got in return from daemon
          #
          my $rc = `/usr/bin/wget -O wget.out "$query" 2>&1`;
          #
          #  should be a list of neighbors unless something wrong
          #
          if ($rc =~ 'failed')
            {
               warn  "BAD WEB PAGE FROM FriendTorr daemon\n";
               print "$PROCESS ERROR: ip=$client count=$count\n";
               print "sim_select_peers daemon might be down\n";
               print "bring daemon up and run $PROCESS again\n";
               exit -1;
            }
          #
```

```
        #  tell user about progress in the registration process
        #
        my $xmod = $count % 50;
        unless ($xmod)
          {
            print "Sub-total number of registered peer nodes = $count\n";
          }
      }
    #
    #  tell user the number of peers registered with FriendTorr daemon
    #
    print "Total number of pre-registered peers = $count\n";
    $time = `date`;
    print "Peer registration process completed: $time\n";
  }



unless (SKIP_REGISTER)
  {
    print "This run was set to perform registration (SKIP_REGISTER=0)\n";
    print "Forced to EXIT after all the peers from the URN were used\n";
    print "Modify $PROCESS to set SKIP_REGISTER=1 and re-run $PROCESS\n";
    exit OK;
  }



#
#  Now that we have pre-register some nodes (peers) with
#  FriendTorr, it is time
#  to start collecting statistics by continuing
#  to register peers from the SAMPLES array.
#
if ($debug)
  {
    print "---------------------------------\n";
    print "Process IP addresses from SAMPLES\n";
    print "---------------------------------\n";
  }
my $file = '';   # output file with the needed statistics report
#
#  based ont he type of peer selection strategy, open OUTPUT file
#
if (RANDOM_STRATEGY)
  {
    # when random peer selection strategy is being used:
    $file = './DATA/random_statistics.txt';
  }
else
  {
    $file = './DATA/proximity_statistics.txt';
  }
open (FILE,">$file") or
          die "file=$file cannot open: $!\n";
#
```

```
# print header record for CSV output file
#
print FILE "client_IP, country, longitude, latitude, ";
print FILE "neighbor_IP, country, longitude, latitude, miles, RTT (msec)\n";
#
my $time = `date`;
print "Process to pull peers from SAMPLES started: $time\n";
$count = 0;
#
# process each IP address representing each peer from inside @samples
#
foreach my $client (@sample)
  {
     my ($cn1, $cn2, $region, $city, $city2, $lat1, $lon1, $lon2, $lat2, $gis);
     $record      = $gi->record_by_addr($client);
     $cn1         = $record->country_name;
     $region      = $record->region;
     $city        = $record->city;
     $lat1        = $record->latitude;
     $lon1        = $record->longitude;
     $cn1         = MISSING unless (defined($cn1));
     $region      = MISSING unless (defined($region));
     $city        = MISSING unless (defined($city));
     $lat1        = MISSING unless (defined($lat1));
     $lon1        = MISSING unless (defined($lon1));
     #
     # call sim_select_peers.pl (daemon must be running in the same host)
     #
     if (RANDOM_STRATEGY)
       {
          # when random peer selection strategy is being used:
          $query = "http://localhost:9999/random?ip=$client";
       }
     else
       {
          # when the proximity based peer selection strategy is used:
          $query = "http://localhost:9999/peers?ip=$client";
       }
     sleep 1;  # give daemon sometime to catch up
     #
     # check to see what we got back from daemon
     #
     my $rc = `/usr/bin/wget -O wget.out "$query" 2>&1`;
     # my $rc = `/usr/bin/lynx -dump "$query" 2>&1`;
     if ($rc =~ 'failed')
       {
          warn  "BAD WEB PAGE received from FrienTorr daemon\n";
          print "$PROCESS ERROR: ip=$client count=$count\n";
          print "sim_select_peers daemon might be down\n";
          print "bring up sim_select_peers and run $PROCESS again\n";
          exit -1;
       }
     $rc = `more wget.out 2>&1`;
     $rc    =~ s/^\s+//;
     $rc    =~ s/\s+$//;
     if ($display)
```

```
 {
    print "----------------- CLIENT --------------------------------\n";
    print "Reporting neighbors for client IP=$client\n";
    print "Client is in country($cn1), region($region),";
    print "city($city), lon=$lon1, and lat=$lat1\n";
    print "List of neighbors: $rc\n" if ($verbose);
 }
my ($header,$x1) = split(/<p>/,$rc);
my ($body,$x2)   = split(/<\/p>/,$x1);
my @tmp = split(/\,/,$body);
#
#  Let's process each neighbor to our client
#
foreach my $item (@tmp)
 {
    $item    =~ s/^\s+//;
    $item    =~ s/\s+$//;
    # print "client=$client item=$item (from:to IP addresses)\n";
    next unless (is_ipv4($item));
    $record       = $gi->record_by_addr($item);
    next unless defined $record;
    $cn2          = $record->country_name;
    $region       = $record->region;
    $city2        = $record->city;
    $lat2         = $record->latitude;
    $lon2         = $record->longitude;
    #
    $cn2          = MISSING unless (defined($cn2));
    $region       = MISSING unless (defined($region));
    $city2        = MISSING unless (defined($city2));
    $lat2         = MISSING unless (defined($lat2));
    $lon2         = MISSING unless (defined($lon2));
    $gis          = GIS::Distance->new();
    $gis->formula( 'Haversine' );
    #
    # both countries must be from pre-set list of countries to process
    #
    # print "from country ($cn1) to country ($cn2)\n";
    my $cn1_2alpha = country2code("$cn1",LOCALE_CODE_ALPHA_2);
    my $cn2_2alpha = country2code("$cn2",LOCALE_CODE_ALPHA_2);
    $cn1_2alpha    = uc($cn1_2alpha);
    $cn2_2alpha    = uc($cn2_2alpha);
    # print "2-alpha format: from ($cn1_2alpha) to ($cn2_2alpha)\n";
    if (($cn1_2alpha =~ COUNTRY_1) ||
       ($cn1_2alpha =~ COUNTRY_2))
     {
       # print "from country ($cn1_2alpha) is valid\n";
     }
    else
     {
       print "WARNING: invalid FROM country ($cn1_2alpha)\n";
       next;
     }
    if (($cn2_2alpha =~ COUNTRY_1) ||
       ($cn2_2alpha =~ COUNTRY_2))
     {
```

```
    # print "to country ($cn2_2alpha) is valid\n";
  }
else
  {
    print "WARNING: invalid TO country ($cn2_2alpha)\n";
    next;
  }
#next unless ($cn1 =~ m/Argentina/gi || $cn1 =~ m/Italy/gi);
#next unless ($cn2 =~ m/Argentina/gi || $cn2 =~ m/Italy/gi);
# print "$client FROM ($city)($cn1) TO $item in ($city2)($cn2)\n";
#
# we are good:
#    continue with the valid countries
#
my $distance = $gis->distance( $lat1,$lon1 => $lat2,$lon2 );
my $miles = $distance->miles();
$miles  = sprintf("%0.2f",$miles);
#
# print "city($city) and city2($city2)\n" if ($debug);
#
my $RTT = 0.01; # assume this default RTT between peers
if (($lon1 =~ $lon2) &&
   ($lat1 =~ $lat2))
  {
    # Take default RTT when both nodes are in the same place
  }
else
  {
    # found client and neighbor are in different locations.
    # But is it the same or different countries?
    # print "get_rtt is being requested for $item\n" if ($debug);
    if ($cn1 =~ /$cn2/)
      {
        # print "same country $cn1 and $cn2\n";
        $RTT =  &get_rtt_same_country(FNLO=>$lon1,
                    FNLA=>$lat1,
                    FNCC=>$cn1,
                    TNLO=>$lon2,
                    TNLA=>$lat2,
                    TNCC=>$cn2
                  );
      }
    else
      {
        # print "different countries $cn1 and $cn2\n";
        $RTT =  &get_rtt_different_countries(FNLO=>$lon1,
                    FNLA=>$lat1,
                    FNCC=>$cn1,
                    TNLO=>$lon2,
                    TNLA=>$lat2,
                    TNCC=>$cn2
                  );
      }
  }
push (@numbers,$RTT);  # keep track to calculate percentiles
#
```

```
        # keep track of the total RTT per client node
        #
        $CLIENT_RTT{$client} = 0 unless (exists($CLIENT_RTT{$client}));
        $CLIENT_RTT{$client} += $RTT if (defined($RTT) && $RTT > 0);
                #
        # keep track of the total MILES per client node
        #
        $CLIENT_MILES{$client} = 0 unless (exists($CLIENT_MILES{$client}));
        $CLIENT_MILES{$client} += $miles if (defined($miles) && $RTT >= 0);
        #
        ##  print "distance=$miles miles and RTT=$RTT\n";
        if ($display)
          {
            print "Found neighbor for client IP=$client:\n";
            print "The neighbor IP=$item at $miles miles away from ";
            print "client=$client\n";
            print "Neighbor is located in country($cn2), region($region)";
            print " in city($city2), at lon=$lon2, lat=$lat2\n";
            print "The RTT between client and neighbor is $RTT\n";
          }
        #   write resulting statistics
        print FILE "$client, $cn1, $lon1, $lat1, ";
        print FILE "$item, $cn2, $lon2, $lat2, $miles, $RTT\n";
      }
    $count++;
    my $xcount = $count % 20;
    print "Number of peers from SAMPLES processed: $count\n" unless ($xcount);
    #
    # last if ($debug);
  }
#
#  let's put the RTT information into an array so we can do percentiles
#
my @rtts = ();  # initialize
foreach my $item (%CLIENT_RTT)
  {
    if (exists($CLIENT_RTT{$item})  &&
        defined($CLIENT_RTT{$item}) &&
        ($CLIENT_RTT{$item} > 0))
      {
        push @rtts,$CLIENT_RTT{$item};
      }
  }
#
#  let's put the RTT information into an array so we can do percentiles
#
my @MILES = ();  # initialize
foreach my $item (%CLIENT_MILES)
  {
    if (exists($CLIENT_MILES{$item})  &&
        defined($CLIENT_MILES{$item}) &&
        ($CLIENT_MILES{$item} >= 0))
      {
        push @MILES,$CLIENT_MILES{$item};
      }
  }
```

```
#
#  Save results in our statistics file:
#
if (RANDOM_STRATEGY)
  {
     # when random peer selection strategy is being used:
     print FILE "RANDOM PEER SELECTION STRATEGY RESULTS\n";
  }
else
  {
     # when the proximity based peer selection strategy is used:
     print FILE "PROXIMITY BASED PEER SELECTION STRATEGY RESULTS\n";
  }
print "Total Number of Client Nodes Processed: $count\n";
print FILE "Total Number of Client Nodes Processed: $count\n";
$count = $#numbers + 1;
print "Number of RTT values collected from Client Nodes to each Neighbor Node: $
count\n";
print FILE "Number of RTT values collected from Client Nodes to each Neighbor No
de: $count\n";
print "Percentiles for all RTT values from each client to each neighbor\n";
print FILE "Percentiles for all RTT values from each client to each neighbor\n";
printf "Percentile %d%% RTT= %f\n", $_, percentile($_,\@numbers) for qw/25 50 75
 90 95 99/;
printf FILE "Percentile %d%% RTT= %f\n", $_, percentile($_,\@numbers) for qw/25
50 75 90 95 99/;
#
$count = $#rtts + 1;
print "Percentiles after adding RTT values from each Client to its Neighbor Node
s, count=$count\n";
print FILE "Percentiles after adding RTT values from each Client to its Neighbor
 Nodes, count=$count\n";
printf "Percentile %d%% RTT= %f\n", $_, percentile($_,\@rtts) for qw/25 50 75 90
 95 99/;
printf FILE "Percentile %d%% RTT= %f\n", $_, percentile($_,\@rtts) for qw/25 50
75 90 95 99/;
print "Percentile in Miles\n";
print FILE "Percentile in Miles\n";
printf "Percentile %d%% MILES= %f\n", $_, percentile($_,\@MILES) for qw/25 50 75
 90 95 99/;
printf FILE "Percentile %d%% RTT= %f\n", $_, percentile($_,\@MILES) for qw/25 50
 75 90 95 99/;
#
$time = `date`;
chomp($time);
print "Results in file=$file\n";
print "$PROCESS: processing completed: $time\n";
my $t1 = Benchmark->new;
my $td = timediff($t1,$t0);
print "Processing time=",timestr($td,'all'),"\n";
#
print FILE "$PROCESS: processing started: $start\n";
print FILE "$PROCESS: processing completed: $time\n";
print FILE "Processing time=",timestr($td,'all'),"\n";
close FILE;
print "**********************************************************************\n";
```

```
exit OK;
#
# ------------------------------------------------------------------------
#   SUBROUTINES
# ------------------------------------------------------------------------
#
#
sub create_files
  {
     #   Read each file by country and save the IP of either FROM or TO
     #   that is present in each row.
     #   I.e. ignore the values in between From and To
     #   note: the IP addresses are saved in @IP (an array)
     #
     my $time = `date`;
     print "Create_files for URN and SAMPLES started: $time\n";
     my $sample_file = shift;  # file to write
     my $urn_file    = shift;  # file to write
     my $urn_size    = URN_SIZE;
     my @urn    = ();  # place for the client IPs to be registered with daemon
     my @sample = ();  # place for the peers that will need to find neighbors
     foreach my $pais (keys %country)
       {
          @ip = ();  # reset the set of IP addresses for each country
          my $num_urns = 0;
          my $file = $country{$pais}; # file with the range of IP addresses
          #
          # IDEA: Divide 10,000 by number of records in $file
          # Each record has a range of IP addresses, select at random
          # the results from the division from the range.
          #
          open (DATA,"<$file") or
            die "file=$file not found: $!\n";
          #
          print "file=$file\n" if ($debug);
          $count = 0;
          $used  = 0;
          #
          while (my $line = <DATA>)
            {
               # Get all the IP addresses for this country
               $count++;                    # count number of records read
               chomp($line);
               $line           =~ s/^\s+//;
               $line           =~ s/\s+$//;
               next if $line    =~ /^$/;
               next if $line    =~ /^#/;
               my @tmp = split(/\-/,$line);
               next unless (defined($tmp[0]));
               $tmp[0]         =~ s/^\s+//;
               $tmp[0]         =~ s/\s+$//;
               next unless (is_ipv4($tmp[0]));
               $record         = $gi->record_by_addr($tmp[0]);
               next unless defined $record;
               next unless defined $record->city;  # 10/04/2011
```

```
    my $cn  = $record->country_name;
    my $cn_2alpha = country2code("$cn",LOCALE_CODE_ALPHA_2);
    $cn_2alpha    = uc($cn_2alpha);
    # print "2-alpha format: $cn_2alpha)\n";
    if (($cn_2alpha =~ COUNTRY_1) ||
       ($cn_2alpha =~ COUNTRY_2))
     {
        # print "valid country: $cn_2alpha)\n";
     }
    else
     {
        print "WARNING: invalid country ($cn_2alpha)\n";
        next;
}
    # push(@ip,$tmp[0]);
    next unless (defined($tmp[1]));
    $tmp[1]         =~ s/^\s+//;
    $tmp[1]         =~ s/\s+$//;
    next unless (is_ipv4($tmp[1]));
    $record         = $gi->record_by_addr($tmp[1]);
    next unless defined $record;
    next unless defined $record->city;  # 10/04/2011
    print "Range for $pais: $tmp[0] to $tmp[1]\n" if ($debug);
    my $next = new Net::IP("$tmp[0] - $tmp[1]") ||
      die ("cannot allocte next IP from $tmp[0]: $!\n");
    # note: $next is an object with a range of IP addresses
    # visit CPAN for more information
    my $limit = 0;  # limit per IP range and to avoid infinity
    do
     {
        $limit++;
        #
        # get IP addresses for each range given in each
        # record read from input file.
        # limit the number of IP to consider per given range
        #
        my $newIP         = $next->ip();
        # print "new IP=$newIP limit=$limit\n";
        my $record      = $gi->record_by_addr($newIP);
        push(@ip,$newIP)  if (defined($record));
        #
        my $xmod = $limit % 500;  # maximum per record read
        my $see  = $limit % 10;
        print "#" unless ($see);
        unless ($xmod)
          {
            print "\n country($pais) records=$count ";
            print "ip=$newIP limit=$limit and ";
            print "IP collected=$#ip\n";
            $next = new Net::IP("$tmp[1]"); # end this WHILE
          }
      } while (++$next);
    #
    # limit the total number of IP addresses per country to 100k
    #
    $used++;
```

```
        #    to keep the number of IP addresses to consider:
        last if ($#ip > 100000);  # maximum IP addresses per country
      }
    if ($debug)
      {
        print "country=$pais: records count($count)used($used) ";
        print "IP collected=$#ip\n";
      }
    close DATA;

    die ("number of IPs is less than $urn_size") if ($#ip < $urn_size);
    #
    # pick up $urn_size IP addresses from each country for the URN
    #
    for (my $i = 0; $i < $urn_size; $i++)
      {
        my $random = rand($#ip);   # random number
        $random    = int($random);
        $random++  unless defined($ip[$random]); # one more chance
        next unless defined($ip[$random]);
        my $client = $ip[$random];
        delete $ip[$random];  # to make them unique
        push(@urn,$client);
        $num_urns++;
        $i = $urn_size if ($#ip < 1);  # in case we took them all out
      }
    die ("number of IPs is less than 100") if ($#ip < 100);
    #
    # Pick up 100 IP address from each country for the SAMPLE
    #
    for (my $i = 0; $i < 100; $i++)
      {
        my $random = rand($#ip);   # random number
        $random    = int($random);
        $random++  unless defined($ip[$random]); # one more chance
        next unless defined($ip[$random]);
        my $client = $ip[$random];
        delete $ip[$random];  # to make them unique
        push(@sample,$client);
        $i = 100 if ($#ip < 1);  # in case we took them all out
      }
  }
#
# at this point we have collected IP addresses for both
# the URN and SAMPLE arrays.  Let's go ahead and save the
# arrays into files for later use in this program.
#
open OUT, "+>$sample_file" ||
  die ("cannot open $sample_file: $!\n");
my @shuffled = shuffle(@sample);
while (@shuffled)
  {
    my $tmp = shift(@shuffled);
    next unless (length($tmp)>2);
    print OUT "$tmp\n";
  }
```

```
    close OUT;
    open OUT, "+>$urn_file" ||
      die ("cannot open $urn_file: $!\n");
    #
    @shuffled = shuffle(@urn);
    while (@shuffled)
      {
         my $tmp = shift(@shuffled);
         next unless (length($tmp)>2);
         print OUT "$tmp\n";
      }
    close OUT;
    $time = `date`;
    print "create_files ended: $time\n";
    return OK;
  }
#
#  SUBROUTINE: get_rtt_different_countries
#
#    When the client and neighbors are in different countries:
#           Let's define some terminology.  The client node is the
#           IP address obtained from the range of IP addresses that
#           belong to one of the two countries.  We define this country
#           for the client to be the FROM country.
#           By similar reasoning, when the sim_select_peers.pl
#           returns a list of candidate neighbors, and we process one
#           neighbor at a time, we said that the neigbor is in the
#           TO country.
#           In this subroutine, we are dealing with the situation
#           when the FROM and TO countries are not the same.
#           When the proximity peer selection strategy
#           is employed, chances are that
#           the TO and FROM countries will be the same.  For the
#           random strategy, these is not the case (~ 50% of the time).
#           Since we don't have RTT readings for all cases between
#           any arbitrary nodes (clients and neighbors), we need to
#           rely on information available from the CityEdges table.
#           The CityEdges table has the RTT between many of the most
#           important cities in any give country.
#           It is possible that the city for the client (or neighbor)
#           may be in the CityEdges table but when not,
#           then the nearest city must be found.
#           The nearest city for the client might not have an entry for the
#           RTT (i.e. no connection) to the city where the neighbor is
#           located.  More to the point, the neighbor city might not be
#           in the CityEdges table.  When it is not, then the nearest city
#           in the CityEdges table must be found for the neighbor as well.
#           We need to keep in mind that when a client and neighbors
#           are in different countries, chances are that a small subset of
#           cities in the CityEdges table for the FROM country
#           will be connected to cities in the TO country (in other
#           words, the CityEdges has many cities in a country
#           that don't have a reported RTT to all the cities in the
#           TO country where the neighbor resides).
#           Therefore, the approach is to find all the cities in the
#           FROM country from the CityEdges tables with the following
```

```
#           condition: the toCity in the FROM country must be one
#           of those cities connected to the TO country.  This way,
#           we will have many more cities per country to work with that
#           should result in a better selection when looking for the
#           nearest city in the CityEdges table for either a client or
#           a neighbor.
#           In additon, when a city is assigned to the client node,
#           we want to make sure that this city is either connected
#           to the neighbor city in the CityEdges table, or it is
#           at least connected to another city in the same country
#           that in turns is connected to one or more of the cities
#           in the country where the neighbor resides.
#           This will ensure that we will be able to find an RTT
#           delay between the nearest city in the CityEdges table
#           to the client node in the FROM country to a city that
#           has a list of RTT (one or more cities) from the
#           FROM country to the TO country.
#           In summary,
#           Calculate the response time delay between client
#           and neighbor by looking at the Dimes database
#           that has the RTT between cities.
#           Since the client and neighbor are based on IP and
#           estimated latitude and longitude, this routine will
#           find the nearest city based on the latitude and longitude
#           of the client or neighbor to the nearest city.
#  STEPS:
#     1) Given the city in the FROM country of the client node's
#        find the nearest city available from the CityEdges table
#        with the following conditions.
#     2) Make sure that the nearest city for the client in the FROM
#        country is connected to a city in the To country.
#     3) If the nearest city for the client node in the FROM country
#        is connected to at least one city in the TO country,
#        then, there is not need to keep track of an intermediate
#        second city in the same country that is connected to at least
#        one city in the TO country.  If it is not connected, then
#        we must keep track of the RTT between the client's nearest city
#        in the FROM country to each city in the same FROM country that
#        has an RTT entry to one or more cities in the TO country.
#     4) We repeat the previous step but from each of the neighbor's
#        perspective.  Again, we keep track of the RTT in the TO country.
#     5) Next we find the RTT between the two cities that have an RTT
#        in both countries.  These two cities must be the closest to
#        both the client's nearest city and the neighbor's nearest city.
#
sub get_rtt_different_countries
  {
     my ($name,$args,$cursor);
     my ($sql, $table,$db);
     my %args = @_;
     my $fromNodeLon = $args{FNLO};
     my $fromNodeLat = $args{FNLA};
     my $fromNodeCC  = $args{FNCC};
     my $toNodeLon   = $args{TNLO};
     my $toNodeLat   = $args{TNLA};
     my $toNodeCC    = $args{TNCC};
```

```
#
#  Convert Country from full name to an alpha of 2 characters:
#
$fromNodeCC = country2code("$fromNodeCC",LOCALE_CODE_ALPHA_2);
$toNodeCC   = country2code("$toNodeCC",LOCALE_CODE_ALPHA_2);
$fromNodeCC = lc($fromNodeCC);
$toNodeCC   = lc($toNodeCC);
#
# print "get_rtt FROM ($fromNodeLon,$fromNodeLat) in $fromNodeCC\n";
# print "get_rtt TO ($toNodeLon,$toNodeLat) in $toNodeCC\n";
#
#
#
$name = DATABASE;  # name of the database (at ~ home directory)
$table = 'CityEdges';
$args = {AutoCommit => 0,  # I will issue the commit command
        RaiseError => 1,  # complain if something fails
        PrintError => 1};
#
# Connect to DB, create TABLE and PREPARE
#
$db = DBI->connect("dbi:SQLite:dbname=$name","","",$args);
#
#
#  Find all the cities in the fromCountry by forcing the
#  toCountry to be the same as the fromCountry with the
#  following condition.
#  The toCity in the fromCountry must be in the list
#  of cities obtained after selecting those cities
#  that exist (i.e. have an RTT entry) connecting
#  the fromCountry and the toCountry.
#
$sql   = "SELECT ";
$sql  .= "fromCountry,fromCity,fromLon,fromLat,";
$sql  .= "toCountry,toCity,toLon,toLat,RTT ";
$sql  .= "FROM $table ";
$sql  .= "WHERE fromCountry='";
$sql  .= $fromNodeCC;
$sql  .= "' AND toCountry='";
$sql  .= $fromNodeCC;
$sql  .= "' AND toCity in ";
#
#  select only those cities that connect to the other country
#
$sql  .= "(SELECT fromCity FROM $table ";
$sql  .= " WHERE fromCountry='";
$sql  .= $fromNodeCC;
$sql  .= "' AND toCountry='";
$sql  .= $toNodeCC;
$sql  .= "')";
$cursor = $db->prepare($sql);
$cursor ->execute();
#print "sql=$sql\n";
#
#print "Using all cities in the same country as the ";
#print "client country=$fromNodeCC\n";
```

```
    #
    %location    = ();  # reset
    %RTT         = ();
    %destination = ();
    #
    while(my @rec = $cursor->fetchrow_array)
     {
        #print "\t";
        #print join("\t",@rec);
        #print "\n";
        #
        # go city by city in the set and save the Longitude/Latitude
        # for each fromCity in the %location hash.
        #
        my $fromCountry = $rec[0];
        my $fromCity    = $rec[1];
        my $fromLon     = $rec[2];
        my $fromLat     = $rec[3];
        my $toCountry   = $rec[4];
        my $toCity      = $rec[5];
        my $toLon       = $rec[6];
        my $toLat       = $rec[7];
        my $rtt         = $rec[8];
        #
        # print "FROM: city($fromCity) lon($fromLon) lat($fromLat)\n";
        # print "TO: city($toCity) lon($toLon) lat($toLat)\n";
        #
        unless ($fromCity =~ /Unknown/gi)
         {
            $location{$fromLon}{$fromLat}    = $fromCity;
            $RTT{$fromLon}{$fromLat}         = $rtt;
            $destination{$fromLon}{$fromLat} = $toCity;
            # print "($fromCity)->($toCity): $rtt lon($fromLon) lat($fromL
at)\n";
         }
     }
    $cursor->finish;
    #
    # Using the %location hash, go find the closest city
    # available from the CityEdges table to the
    # the client node passed to this routine based on
    # the Latitude and Longitude of the client Node in the fromCountry.
    #
    my $rc;
    $rc              = &find_city($fromNodeLat,$fromNodeLon);
    #print "get_rtt RC for FROM: rc=$rc\n";
    my $fromCity_cc1   = '';
    my @tmp            = ();
    ($fromCity_cc1,@tmp) = split(/\,/,$rc); # nearest city in the CityEdges
    my $rtt_1    = $RTT{$tmp[1]}{$tmp[0]}; # RTT for leg inside same country
    my $toCity_cc1 = $destination{$tmp[1]}{$tmp[0]}; # closest toCity
    unless (defined($fromCity_cc1))
     {
        $fromCity_cc1  = MISSING;
        print "WARNING: missing fromCity_cc1 value\n";
        print "DEBUG: from city($fromCity_cc1) to city($toCity_cc1)\n";
```

```
    print "DEBUG: nearest-city RC=$rc\n";
    print "DEBUG: from country($fromNodeCC) to country($toNodeCC)\n";
  }
unless (defined($toCity_cc1))
  {
    $toCity_cc1    = MISSING;
    print "WARNING: missing toCity_cc1 value\n";
    print "DEBUG: from city($fromCity_cc1) to city($toCity_cc1)\n";
    print "DEBUG: nearest-city RC=$rc\n";
    print "DEBUG: from country($fromNodeCC) to country($toNodeCC)\n";
  }
unless (defined($rtt_1))
  {
    $rtt_1    = 0.01;
    print "WARNING: missing rtt_1 value, rtt_1 reset to 0.01\n";
    print "DEBUG: from city($fromCity_cc1) to city($toCity_cc1)\n";
    print "DEBUG: nearest-city RC=$rc\n";
    print "DEBUG: from country($fromNodeCC) to country($toNodeCC)\n";
  }
# print "fromCity_cc1=$fromCity_cc1 is $tmp[2] miles from client node\n";
#
# NOTE: fromCity_cc1 will represent the city where the client node
#       in the fromCountry is located.
#
# Save the RTT between closest city in CityEdges to our client node
# and the city in the same country that is connected to our
# neighbor node in the toCountry.
#
# print "client in $fromNodeCC: ";
# print "fromCity in CC1=$fromCity_cc1, rtt=$rtt_1, toCity=$toCity_cc1\n";
%location    = ();  # reset
%RTT         = ();
%destination = ();
#
#  At this point we have a city that belongs to the CityEdges table
#  that is the closest to the client node.
#  We also know that this city is connected to at least one other
#  city (could be itself) that in turns is connected to a city
#  in the country where the neighbor node is located.
#  Now we look for the same type of information but in the
#  other country.
#  Remember, we are working with the 2nd country so the
#  pespective as to which is from and to countries are reversed.
#
$sql   = "SELECT ";
$sql  .= "fromCountry,fromCity,fromLon,fromLat,";
$sql  .= "toCountry,toCity,toLon,toLat,RTT ";
$sql  .= "FROM $table ";
$sql  .= "WHERE fromCountry='";
$sql  .= $toNodeCC;
$sql  .= "' AND toCountry='";
$sql  .= $toNodeCC;
$sql  .= "' AND toCity in ";
#
#  select only those cities that connect to the other country
#  but keep in mind that we are going from CC1 to CC2
```

```
#
$sql  .= "(SELECT toCity FROM $table ";
$sql  .= " WHERE fromCountry='";
$sql  .= $fromNodeCC;
$sql  .= "' AND toCountry='";
$sql  .= $toNodeCC;
$sql  .= "')";
$cursor = $db->prepare($sql);
$cursor ->execute();
#print "sql=$sql\n";
#print "Using all cities in the same country as the ";
#print "neighbor country=$toNodeCC\n";
#
while(my @rec = $cursor->fetchrow_array)
  {
     #print "\t";
     #print join("\t",@rec);
     #print "\n";
     #
     # go city by city in the set and save the Longitude/Latitude
     # for each fromCity in the %location hash.
     #
     my $fromCountry = $rec[0];
     my $fromCity    = $rec[1];
     my $fromLon     = $rec[2];
     my $fromLat     = $rec[3];
     my $toCountry   = $rec[4];
     my $toCity      = $rec[5];
     my $toLon       = $rec[6];
     my $toLat       = $rec[7];
     my $rtt         = $rec[8];
     #
     # print "FROM: city($fromCity) lon($fromLon) lat($fromLat)\n";
     # print "TO: city($toCity) lon($toLon) lat($toLat)\n";
     #
     unless ($fromCity =~ /Unknown/gi)
       {
          $location{$fromLon}{$fromLat}    = $fromCity;
          $RTT{$fromLon}{$fromLat}         = $rtt;
          $destination{$fromLon}{$fromLat} = $toCity;
       }
     #
  }
$cursor->finish;
#
# Using the %location hash, go find the closest city
# available from the CityEdges table to the
# the city passed to this routine based on
# the Latitude and Longitude of the neighbor Node.
#
$rc           = ''; # reset
$rc           = &find_city($toNodeLat,$toNodeLon);
#print "get_rtt RC for FROM: rc=$rc\n";
my $fromCity_cc2   = '';
@tmp              = (); # reset
($fromCity_cc2,@tmp) = split(/\,/,$rc); # nearest city in the CityEdges
```

```
my $rtt_3    = $RTT{$tmp[1]}{$tmp[0]}; # RTT for leg inside same country
my $toCity_cc2 = $destination{$tmp[1]}{$tmp[0]}; # closest toCity
# print "fromCity_cc2=$fromCity_cc2 is $tmp[2] miles from client node\n";
unless (defined($fromCity_cc2))
 {
    $fromCity_cc2  = MISSING;
    print "WARNING: missing fromCity_cc2 value\n";
    print "DEBUG: from city($fromCity_cc2) to city($toCity_cc2)\n";
    print "DEBUG: nearest-city RC=$rc\n";
    print "DEBUG: from country($fromNodeCC) to country($toNodeCC)\n";
 }
unless (defined($toCity_cc2))
 {
    $toCity_cc2    = MISSING;
    print "WARNING: missing toCity_cc2 value\n";
    print "DEBUG: from city($fromCity_cc2) to city($toCity_cc2)\n";
    print "DEBUG: nearest-city RC=$rc\n";
    print "DEBUG: from country($fromNodeCC) to country($toNodeCC)\n";
 }
unless (defined($rtt_3))
 {
    $rtt_3    = 0.01;
    print "WARNING: missing rtt_3 value, rtt_3 reset to 0.01\n";
    print "DEBUG: from city($fromCity_cc2) to city($toCity_cc2)\n";
    print "DEBUG: nearest-city RC=$rc\n";
    print "DEBUG: from country($fromNodeCC) to country($toNodeCC)\n";
 }
#
# Save the RTT between closest city in CityEdges to our client node
# and the city in the same country that is connected to our
# neighbor node
#
# print "Neighbor in $toNodeCC: fromCity in CC2=$fromCity_cc2, ";
# print "rtt=$rtt_3, toCity=$toCity_cc2\n";
%location    = ();  # reset
%RTT         = ();
%destination = ();
#
# Now we are ready to see what the RTT (rtt_2) is between
# the two cities that connect both countries.
#
$sql   = "SELECT ";
$sql  .= "fromCountry,fromCity,fromLon,fromLat,";
$sql  .= "toCountry,toCity,toLon,toLat,RTT ";
$sql  .= "FROM $table ";
$sql  .= "WHERE fromCountry='";
$sql  .= $fromNodeCC;
$sql  .= "' AND toCountry='";
$sql  .= $toNodeCC;
$sql  .= "'";
$sql  .= "AND fromCity='";
$sql  .= $toCity_cc1;
$sql  .= "' AND toCity='";
$sql  .= $toCity_cc2;
$sql  .= "'";
#
```

```
    $cursor = $db->prepare($sql);
    $cursor ->execute();
    #
    my $rtt_2;
    # print "List of bridge cities to connect both countries:\n";
    while(my @rec = $cursor->fetchrow_array)
     {
        #print "\t";
        #print join("\t",@rec);
        #print "\n";
        my $fromCountry = $rec[0];
        my $fromCity    = $rec[1];
        my $fromLon     = $rec[2];
        my $fromLat     = $rec[3];
        my $toCountry   = $rec[4];
        my $toCity      = $rec[5];
        my $toLon       = $rec[6];
        my $toLat       = $rec[7];
        $rtt_2          = $rec[8];  # define this value
        #
        # print "FROM: city($fromCity) lon($fromLon) lat($fromLat)\n";
        # print "TO: city($toCity) lon($toLon) lat($toLat)\n";
     }
    $cursor->finish;
    unless (defined($rtt_2))
     {
        $rtt_2    = 0.01;
        print "WARNING: missing rtt_2 value, rtt_2 reset to 0.01\n";
        print "DEBUG: from city($fromCity_cc2) to city($toCity_cc2)\n";
        print "DEBUG: nearest-city RC=$rc\n";
        print "DEBUG: from country($fromNodeCC) to country($toNodeCC)\n";
     }
    #
    # Let's get the total delay
    #
    my $delay = $rtt_1 + $rtt_2 + $rtt_3;
    # print "rtt1=$rtt_1 rtt2=$rtt_2 rtt3=$rtt_3\n";
    # print "get_rtt: total RTT between client and neighbor=$delay (msec)\n";
    #
    #
    #
    $db ->disconnect();
    # exit; # <<<<<<<<<<<<<<<<<<<<<<<<----------------------<<<<<<<<<<<<<<<<<
    #
    return "$delay";
  }
#
#
#
sub find_city
  {
    # print "nearest_city: get geo distance\n";
    my $lat = shift;
    my $lon = shift;
    my @rc = ();
    my $gis = GIS::Distance->new();
```

```
    $gis->formula( 'Haversine' );
    my %close = ();
    my $closest_city = '';  # the one with minimu distance in miles
    my $closest_lat = 0;
    my $closest_lon = 0;
    my $min_miles   = 999999999;
    my @item = ();
    my $value = '';
    my $city = '';
    while (($item[0], $value) = each (%location))
     {
        while (($item[1], $city) = each(%{$value}))
         {
            # for the candidate city (dns) that we found has a bridge
            # to the 2nd country under consideration:
            # 01/27/2012 - CityEdges table fixed, thus reverse lat&lon
            my $dns_lon  = $item[0];   # pull out the latitude
            my $dns_lat  = $item[1];   # pull out the longitude
            #print "(lat,lon): ($lat,$lon) ($dns_lat,$dns_lon)\n";
            #      calculate the distance from client to candidate DNS
            my $distance = $gis->distance ( $lat,$lon => $dns_lat,$dns_lon
);
            my $miles    = $distance->miles();
            $miles       = sprintf("%0.2f",$miles);
            #print "find_city: miles=$miles\n";
            if ($miles < $min_miles)
              {
                $min_miles = $miles;
                $closest_city = $city;
                $closest_lat  = $dns_lat;
                $closest_lon  = $dns_lon;
              }
            #     save the distance from requestor to DNS in miles
            ##    $close{$item} = $miles;
         }
     }
    #          sort distance from requestor to all DNS candidates
    ##  my @dns_near = sort { $close{$a} <=> $close{$b} } keys %close;
    %close = ();
    return "$closest_city,$closest_lat,$closest_lon,$min_miles";
  }
#
#   subroutine to calculate the pecentiles of numbers in an array.
#
# my @numbers = (10.22,20.33,22.3,11.3,12.4,8.3,10.4);
# printf "Percentile %d%% at %f\n", $_, percentile($_,\@numbers)
#       for qw/25 50 75 90 95 99/;
#
sub percentile
  {
    my ($p,$aref) = @_;
    my $percentile = int($p * $#{$aref}/100);
    my @sorted = sort {$a <=> $b} @$aref;
    return (@sorted)[$percentile];
  }
#
```

```
#    When client and neighbor are in the same country:
#           calculate the response time delay between client
#           and neighbor by looking at the Dimes database
#           that has the RTT between cities.
#           Since the client and neighbor are based on IP and
#           estimated latitude and longitude, this routine will
#           find the nearest city based on the latitude and longitude
#           of the client or neighbor to the nearest city.
#
sub get_rtt_same_country
 {
    my ($name,$args,$cursor);
    my ($sql, $table,$db);
    my %args = @_;
    my $fromNodeLon = $args{FNLO};
    my $fromNodeLat = $args{FNLA};
    my $fromNodeCC  = $args{FNCC};
    my $toNodeLon   = $args{TNLO};
    my $toNodeLat   = $args{TNLA};
    my $toNodeCC    = $args{TNCC};
    #
    #  Convert Country from full name to an alpha of 2 characters:
    #
    $fromNodeCC = country2code("$fromNodeCC",LOCALE_CODE_ALPHA_2);
    $toNodeCC   = country2code("$toNodeCC",LOCALE_CODE_ALPHA_2);
    $fromNodeCC = lc($fromNodeCC);
    $toNodeCC   = lc($toNodeCC);
    #
    #print "get_rtt FROM ($fromNodeLon,$fromNodeLat) in $fromNodeCC\n";
    #print "get_rtt TO ($toNodeLon,$toNodeLat) in $toNodeCC\n";
    #
    #
    #
    $name = DATABASE;  # name of the database (at ~ home directory)
    $table = 'CityEdges';
    $args = {AutoCommit => 0,  # I will issue the commit command
          RaiseError => 1,  # complain if something fails
          PrintError => 1};
    #
    # Connect to DB, create TABLE and PREPARE
    #
    $db = DBI->connect("dbi:SQLite:dbname=$name","","",$args);
    #
    #
    #  Find all the cities that belong to the two countries
    #  needed for the FROM client and the TO neighbor nodes.
    #
    $sql   = "SELECT ";
    $sql  .= "fromCountry,fromCity,fromLon,fromLat,";
    $sql  .= "toCountry,toCity,toLon,toLat,RTT ";
    $sql  .= "FROM $table ";
    $sql  .= "WHERE fromCountry='";
    $sql  .= $fromNodeCC;
    $sql  .= "' AND toCountry='";
    $sql  .= $toNodeCC;
    $sql  .= "'";
```

```
$cursor = $db->prepare($sql);
$cursor ->execute();
#print "sql=$sql\n";
#
while(my @rec = $cursor->fetchrow_array)
 {
    #print "\t";
    #print join("\t",@rec);
    #print "\n";
    my $fromCountry = $rec[0];
    my $fromCity    = $rec[1];
    my $fromLon     = $rec[2];
    my $fromLat     = $rec[3];
    my $toCountry   = $rec[4];
    my $toCity      = $rec[5];
    my $toLon       = $rec[6];
    my $toLat       = $rec[7];
    my $rtt         = $rec[8];
    #print "list of all cities that belong to both countries:\n";
    #print "FROM: city($fromCity) lon($fromLon) lat($fromLat)\n";
    #print "TO: city($toCity) lon($toLon) lat($toLat)\n";
    $location{$fromLon}{$fromLat} = $fromCity;
    # $location{$toLon}{$toLat}     = $toCity;
 }
$cursor->finish;
#
# Find the nearby city closest given the Latitude and Longitude
# of the client Node.
#
my $rc;
$rc           = &find_city($fromNodeLat,$fromNodeLon);
#print "get_rtt RC for FROM: rc=$rc\n";
my $fromCity   = '';
my @tmp        = ();
($fromCity,@tmp) = split(/\,/,$rc);
#print "fromCity=$fromCity is $tmp[2] miles from client node\n";
#
%location = ();
#
#  Now look for those cities that match in the FROM client city.
#
$sql   = "SELECT ";
$sql  .= "fromCountry,fromCity,fromLon,fromLat,";
$sql  .= "toCountry,toCity,toLon,toLat,RTT ";
$sql  .= "FROM $table ";
$sql  .= "WHERE fromCountry='";
$sql  .= $fromNodeCC;
$sql  .= "' AND toCountry='";
$sql  .= $toNodeCC;
$sql  .= "'";
$sql  .= "AND fromCity='";
$sql  .= $fromCity;
$sql  .= "'";
#
$cursor = $db->prepare($sql);
$cursor ->execute();
```

```
#
while(my @rec = $cursor->fetchrow_array)
 {
    #print "\t";
    #print join("\t",@rec);
    #print "\n";
    my $fromCountry = $rec[0];
    my $fromCity    = $rec[1];
    my $fromLon     = $rec[2];
    my $fromLat     = $rec[3];
    my $toCountry   = $rec[4];
    my $toCity      = $rec[5];
    my $toLon       = $rec[6];
    my $toLat       = $rec[7];
    my $rtt         = $rec[8];
    #print "List of cities that match $fromCity:\n";
    #print "FROM: city($fromCity) lon($fromLon) lat($fromLat)\n";
    #print "TO: city($toCity) lon($toLon) lat($toLat)\n";
    # $location{$fromLon}{$fromLat} = $fromCity;
    $location{$toLon}{$toLat}     = $toCity;
 }
$cursor->finish;
#
#    Given the latitude and longitude of the neighbor,
#    look for the nearest City to this neighbor node.
#
$rc           = &find_city($toNodeLat,$toNodeLon);
#print "get_rtt RC for TO: rc=$rc\n";
my $toCity    = '';
@tmp          = ();
($toCity,@tmp) = split(/\,/,$rc);
#print "toCity=$toCity is $tmp[2] miles from neighbor node\n";
#
# Now look for not only matching countries but also
# matching FROM client city and TO neighbor city.
#
$sql   = "SELECT ";
$sql  .= "fromCountry,fromCity,fromLon,fromLat,";
$sql  .= "toCountry,toCity,toLon,toLat,RTT ";
$sql  .= "FROM $table ";
$sql  .= "WHERE fromCountry='";
$sql  .= $fromNodeCC;
$sql  .= "' AND toCountry='";
$sql  .= $toNodeCC;
$sql  .= "'";
$sql  .= "AND fromCity='";
$sql  .= $fromCity;
$sql  .= "'";
$sql  .= "AND toCity='";
$sql  .= $toCity;
$sql  .= "'";
#
#
$cursor = $db->prepare($sql);
$cursor ->execute();
#
```

```
    my $delay = 0;
    while(my @rec = $cursor->fetchrow_array)
      {
         #print "\t";
         #print join("\t",@rec);
         #print "\n";
         $delay = $rec[$#rec];
      }
    $cursor->finish;
    # print "get_rtt: RTT between client and neighbor = $delay (msec)\n";
    #
    #
    #
    $db ->disconnect();
    #
    return "$delay";
}
```