January 2009

Design and Evaluation of Distributed Algorithms for Placement of Network Services

Todd Sproull
Washington University in St. Louis

WASHINGTON UNIVERSITY

Sever Institute
School of Engineering and Applied Science
Department of Computer Science and Engineering

Dissertation Examination Committee:
Roger Chamberlain, Chair
Ron Cytron, Co-Chair
John W. Lockwood, Co-Chair
Chenyang Lu
Chris Gill
Young H. Cho
Robert E. Morley

DESIGN AND EVALUATION OF DISTRIBUTED ALGORITHMS FOR PLACEMENT OF NETWORK SERVICES

by

Todd S. Sproull

A dissertation presented to the
Graduate School of Arts and Sciences
of Washington University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

August 2009

Saint Louis, Missouri

Acknowledgments

My inexpressible thanks go to my research advisor Dr. Roger Chamberlain. His willingness to take me under his guidance very late in my research demonstrates his kind and giving nature. I am very much thankful for his amazing ability to quickly absorb all of the details of my research and provide great feedback. His probing questions help to dive into the heart of my research and highlight the most important aspects of my work. His ability to work hard and be nice made the last year of my dissertation one of the most enjoyable in my life.

I am also very grateful to my original research advisor Dr. John Lockwood. He instilled upon me an amazing work ethic with a never quit attitude, for which I am truly thankful. Through the many years of working together, he never stopped believing in me or my abilities and always encouraged me to continue with my research. I was fortunate enough to work with him on many projects that made a difference in the world. I also had the luxury of working with him on a few startup companies, these helped light an entrepreneurial flame which will never burn out.

I would next like to thank my co-advisor Dr. Ron Cytron. His willingness to meet with me twice a week helped drive this dissertation to completion. His thoughtful questions and careful analysis led to many insights and discoveries in my research. He is also responsible for my new found love of teaching. I was very reluctant to take on a teaching role in the last year of my research but now it is something I hope to do for the rest of my life.

I am also very thankful for the support of Dr. Young Cho. He consistently asked the tough questions and pushed my research to another level. His frank and honest nature helped me focus on the right problems and work with the right people.

I would also like to thank Dr. Chris Gill, Dr. Chenyang Lu, and Dr. Robert Morley. Their suggestions during my PhD proposal and during update meetings were very useful and helped strengthen my work.

I next would like to thank all of my coauthors whose contributions assisted me in producing great publications. In particular I want to thank Sarang Dharmapurikar, Praveen Krisnamurthy, Haoyu Song, David Taylor, and Jack Meier.

I am grateful to my research sponsors, NSF, Global Velocity, and Boeing for supporting my work.

I would also like to thank Myrna Harbison, Peggy Fuller, Stella Sung, Madeline Hawkins, and Andrea Levy. They helped make my life much easier when dealing with the wide range of administrative issues every graduate student encounters.

Finally, I wish to thank my parents and my two brothers for their unconditional love and support. I cannot express how much they have contributed to enriching my life. Their patience and support will always be remembered.

Todd S. Sproull

Washington University in Saint Louis
August 2009

To my Mother and Father, Phyllis and Stephen Sproull, for always encouraging me to learn, love, and laugh.

Contents

List of Tables

List of Figures

ABSTRACT OF THE DISSERTATION

Design and Evaluation of Distributed Algorithms for Placement of Network Services

by

Todd S. Sproull

Doctor of Philosophy in Computer Engineering

Washington University in St. Louis, 2009

Roger Chamberlain, Chairperson,

Ron Cytron, Co-Chair, John W. Lockwood, Co-Chair

Network services play an important role in the Internet today. They serve as data caches for websites, servers for multiplayer games and relay nodes for Voice over IP (VoIP) conversations. While much research has focused on the design of such services, little attention has been focused on their actual placement. This placement can impact the quality of the service, especially if low latency is a requirement. These services can be located on nodes in the network itself, making these nodes supernodes. Typically supernodes are selected in either a proprietary or ad hoc fashion, where a study of this placement is either unavailable or unnecessary. Previous research dealt with the only pieces of the problem, such as finding the location of caches for a static topology, or selecting better routes for relays in VoIP. However, a comprehensive solution is needed for dynamic applications such as multiplayer games or P2P VoIP services. These applications adapt quickly and need solutions based on the immediate demands of the network.

In this thesis we develop distributed algorithms to assign nodes the role of a supernode. This research first builds off of prior work by modifying an existing assignment algorithm and implementing it in a distributed system called Supernode Placement in Overlay Topologies (SPOT). New algorithms are developed to assign nodes the supernode role. These algorithms are then evaluated in SPOT to demonstrate improved SN assignment and scalability. Through a series of simulation, emulation, and experimentation insight is gained into the critical issues associated with allocating resources to perform the role of supernodes. Our contributions include distributed algorithms to assign nodes as supernodes, an open source fully functional distributed supernode allocation system, an evaluation of the system in diverse networking environments, and a simulator called SPOTsim which demonstrates the scalability of the system to thousands of nodes. An example of an application deploying such a system is also presented along with the empirical results.

Chapter 1

Introduction

The rapid expansion of the Internet has brought about changes in the way computers communicate. One such change is the increase in communication between individual end hosts or nodes. This type of communication is called Peer-to-Peer (P2P). In a P2P network, the nodes themselves exchange content or provide services with each other. This is in contrast to the more common client/server architecture where nodes request content or services from a central server. As these P2P networks grow in size, it is often advantageous to create a tiered architecture for the nodes to communicate. Here, a smaller collection of upper-tiered nodes take on additional responsibilities in the network. These upper-tiered nodes are often referred to as supernodes (SNs). The responsibilities SNs provide range from bootstrapping nodes joining a Voice-over-IP (VoIP) network to hosting a multi-player game for a group of nodes to assisting in the discovery of content in a file-sharing network. The role of the SN in these examples assumes some of the responsibilities of a server, however the benefits of a P2P system rely on the nodes themselves. For example, in a file sharing application, the files are distributed across the nodes in the network as opposed to a centralized server in the client/server example. This dissertation focuses on the placement and evaluation of supernodes in P2P networks.

1.1 Supernode Placement

Proper placement of an SN plays an important role in the overall performance of applications utilizing a tiered architecture. Consider a P2P VoIP application that wishes to offer an audio conversation between two nodes, A and B, as shown in Figure 1.1. Now suppose an additional node, such as node C or D is needed to act as a relay between node A and B due to firewall or other network restrictions. This relay is an example service that an SN provides. The VoIP application is now tasked with locating the best available SN to serve as a relay. For example, node C is the preferred node to serve as the SN to relay traffic between nodes A and B. If node D was selected instead of node C, traffic between A and B would experience more delays due to the additional routers connected to node D. This causes an increase in latency and can result in decreased call quality. In some circumstances the application is unable to find a suitable existing SN due to high latency or lack of resources. In these situations moving a lower tiered node to the upper tier to serve as an SN can be considered. Assigning this SN role to a closer or more capable P2P node can improve the quality of the conversation.

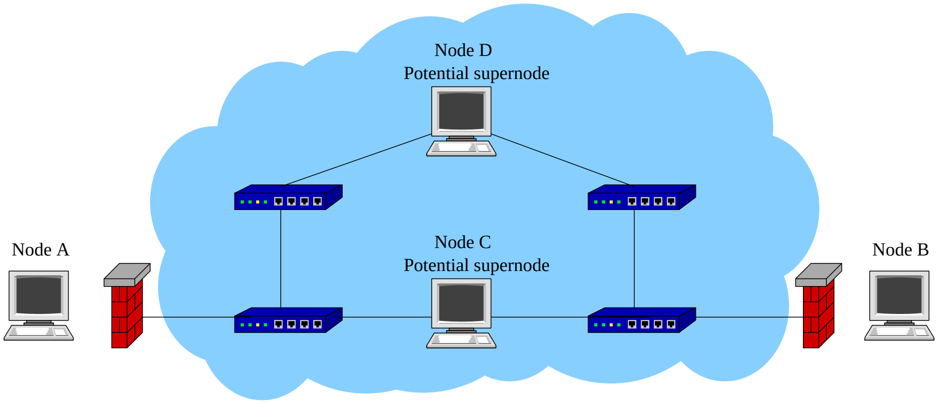


Figure 1.1: Example topology of two nodes A and B communicating with VoIP using either supernode C or D to relay the conversation.

The previous example illustrates the impact a poorly placed SN can have on a P2P network. Several factors influence the task of selecting a set of nodes to act as SNs. For example, nodes with large amounts of storage or high performance computing may prove to be attractive options for content storage or replication services. Nodes with low latency, high bandwidth communication links are desirable for realtime applications such as voice or video conversations. Finally, performing the role of a server in an ad hoc multi-player game may require both high performance computing and high speed network links.

In order to determine the number and assignment of SNs, a cost metric is typically utilized. This cost is composed of metrics of interest for determining placement. Metrics such as location, CPU type, available network bandwidth, distance relative to peers, and available storage are a few examples. This information is then used to evaluate the best subset of nodes subject to the relative importance of each metric.

For small sized (tens of nodes) networks a centralized approach to collect and evaluate metrics of interest is generally applicable. However, as the size of the network increases, this approach becomes prohibitively expensive due to the required communication and computation overheads. Therefore, distributed methods are needed. Distributed placement allows for increased scalability over centralized solutions. Quantifying the quality of SN placement for distributed and centralized approaches provides much needed insight in designing network applications.

P2P networks are rarely static in nature. Over time participants join and leave networks (known as churn). As portions of the network change, it may be advantageous to reevaluate the placement of SNs for all of or a portion of the network. In addition, quantifying the tradeoffs between relocating SNs or adding new SNs provides valuable information for developers in network management.

Current research with the placement of services, or more specifically SNs, fails to address several key issues. First, previous work is typically concerned with placing services near the nodes requiring service at designated service centers [34][59]. Second, once services are placed the problem is considered solved, with little discussion of reassigning SNs as nodes join and leave the network. Third, previous research explicitly dealing with SN placement fails to investigate the problem beyond high level placement strategies and simple simulations [38].

This dissertation furthers the state of the art research by building large deployable SN placement systems using new dynamic distributed SN placement algorithms. A software implementation called Supernode Placement in Overlay Topologies (SPOT) is introduced. SPOT focuses on assigning services to individual nodes. This research first examines previous approaches in the placement of services. The previous state of the art algorithms are extended to the more specific problem of SN placement. Applications utilizing this service are then investigated to determine how to take advantage of informed placement of SNs.

1.2 Evaluation of Supernode Placement Algorithms

With the design of any distributed system, its evaluation is vital for a deeper understanding of the design tradeoffs that are being considered. Systems level research requires careful implementation and rigorous evaluation to gain these insights and discoveries when studying distributed systems. Creating systems that exist only as simulations can be limited in the lessons they provide. Moving beyond simulation to emulation and Internet-testbed deployment develops a deeper understanding of a real system on a larger scale.

This dissertation explores a breadth of evaluation strategies regarding SNs. The strategies include: simulations of SN placement with thousands of nodes, emulation of different topologies with hundreds of nodes, and Internet deployment for a diverse selection of nodes and network communications. In addition, comparisons to traditional client/server communication models are studied. Finally, a hardware accelerated SN service is created and investigated. This range of exploration provides a better understanding of the impact SNs provide on a variety of applications and platforms.

1.3 Contributions

The contributions of this dissertation are in the areas of network algorithms and evaluation of P2P networking services. Specific contributions are:

- Design and implementation of dynamic distributed SN placement algorithms
  - Developed new dynamic algorithms that assign SNs in P2P networks with limited topology information to provide node level deployment (r-mod and r-SPOT)
  - Developed and deployed a software service that positions SNs in P2P networks (SPOT)
- Evaluation of distributed placement with Internet topologies and emulation testbeds

  - Evaluated performance of distributed architecture in emulation using hundreds of nodes
  - Evaluated diverse collection of nodes and network topologies in a global research testbed
  - Created software simulator of SN placement algorithms to investigate topologies with thousands of nodes using realistic Internet topologies

- Evaluation of applications using SN placement service
  - Proposed extensions for third-party applications to support SN placement
  - Evaluated multiplayer first person shooter game utilizing dynamic SN placement algorithms

- Design of satellite P2P communication service and evaluation in emulation testbed compared to client/server approach

- Design and implementation of P2P hardware accelerated SN with a comparison to a software implementation

1.4 Overview of Dissertation

The next chapter provides related work on SN placement and different methods to evaluate large scale network services. Chapter 3 describes the SN placement problem and a formal description of the problem. Chapter 4 describes the Supernode Placement in Overlay Topologies (SPOT) system developed for placing SNs in P2P networks and the new dynamic algorithms that execute inside of SPOT. Chapter 5 explores the evaluation of SPOT in various topologies, introduces the SPOTSim simulator and provides an example of an application utilizing the SN placement service. Chapter 6 investigates a P2P architecture in satellite avionics and provides a comparison to a client/server approach. In Chapter 7 an accelerated SN is explored for avionic networks demonstrating the performance of a hardware accelerated SN. Finally, in Chapter 8 conclusions are provided along with future work.

Chapter 2

Background and Related Work

The following discusses background and related work on placing services in a network. First, several problems from operational research will be discussed. Next, previous work on the placement of SNs is introduced. Then, a discussion about P2P applications is provided. Next, several testbeds to evaluate distributed systems are presented. Finally, different evaluation methods for distributed systems are considered.

2.1 General Placement Problem

The placement of SNs is closely related to the uncapacitated k-median problem or (k-median). The term uncapacitated means that the SN can provide an unlimited amount of service to an unlimited number of nodes. Hakimi was the first to prove the optimality of node locations for the k-median problem [27]. Hakimi describes the problem as the placement of k nodes in a graph where the average distance from every node to these k nodes is minimal. An enumeration technique was provided as an initial solution.

The k-means and k-centers are also similar to the k-median problem. The k-means problem finds k points on a graph, but those points do not have to coincide with actual node locations. The k-centers problem minimizes the longest distance from every node to these k nodes.

The SN placement problem is also similar to the uncapacitated facility location or (facility location) problem which places switches in communication networks or public services such as hospitals or fire and police stations [27][63]. This problem investigates the cost of locating a facility in order to minimize the average distance to its nodes. The solution also considers the additional costs associated with creating a facility. This constraint allows for a more dynamic deployment of facilities as demand and topologies change. In [29], the k-median and facility location problems are shown to be NP-Hard. The terms facilities and service centers are synonymous with SNs. Related work also discusses capacitated versions of the problem, where each SN has a limit on the amount of service it can provide [13]. In this chapter our discussion will stay with the uncapacitated version of the problem where SNs can provide an unbounded amount of service to an unbounded number of nodes.

2.1.1 k-median Example Solution

An example of the k-median problem is now discussed. We present a sample topology with n nodes where $n = 10$ and provide solutions for $k = 1$ and 2. Consider the graph in Figure 2.1 which is based on Example 2.2 in [29]. The nodes are labeled A through J with link costs associated with communication from one node to the next. Let c_{ij} equal the transition

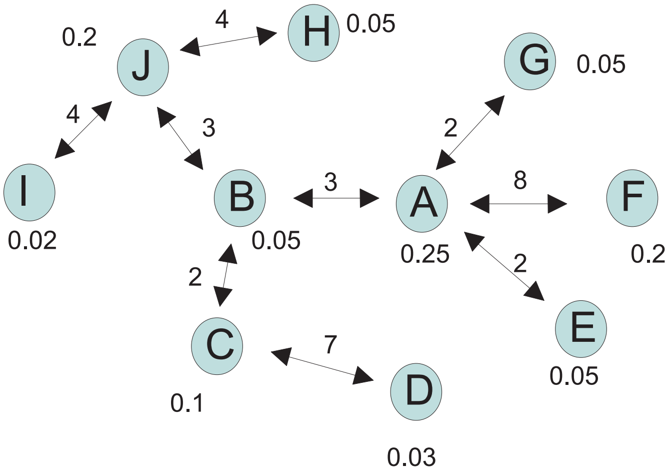


Figure 2.1: Example topology of 10 node network with link costs and demand for service from each node.

	SN A	SN B	SN C	SN D	SN E	SN F	SN G	SN H	SN I	SN J
A	0	0.75	1.25	3	0.5	2	0.5	2.5	2.5	1.5
B	0.15	0	0.1	0.45	0.25	0.55	0.25	0.35	0.35	0.15
C	0.5	0.2	0	0.7	0.7	1.3	0.7	0.9	0.9	0.5
D	0.36	0.27	0.21	0	0.42	0.6	0.42	0.48	0.48	0.36
E	0.1	0.25	0.35	0.7	0	0.5	0.2	0.6	0.6	0.4
F	1.6	2.2	2.6	4	2	0	2	3.6	3.6	2.8
G	0.1	0.25	0.35	0.7	0.2	0.5	0	0.6	0.6	0.4
H	0.5	0.35	0.45	0.8	0.6	0.9	0.6	0	0.4	0.2
I	0.2	0.14	0.18	0.32	0.24	0.36	0.24	0.16	0	0.08
J	1.2	0.6	1	2.4	1.6	2.8	1.6	0.8	0.8	0
Total Costs	4.71	5.01	6.49	13.07	6.51	9.51	6.51	9.99	10.23	6.39

Table 2.1: Total costs for each node serving as the single SN in the 10 node example

from some node i to another node j. Consider the transition from A to B or c_{AB}, it has a distance cost of 3, where c_{AB} is the sum of edge costs along the minimum path from A to C. Each node j also has a demand probability for the service d_j. For example, d_A equals 0.25 or node A will request service 25% percent of the time it is in operation. In this example a single type of service is available to all nodes. The goal is to place k SNs that minimize the total cost of providing the service based on the demand of each node and link costs. More formally, for the $k = 1$ example, a set of n costs are calculated. The cost of some node i to serve as the SN has a cost C_i defined as

$$C_i = \sum_{j=1}^{n} c_{ij} d_j$$

In the case of $k = 1$, a solution is provided by enumerating all possible SN locations and selecting the least expensive C_i. The least expensive (or best) location for the SN is at node A. The total cost of locating the SN at this node is 0.05 * 3 + 0.1 *(2+3) + 0.03 * (7 + 2 + 3) + 0.05 * 2 + 0.2 * 8 + 0.05 * 2 + 0.05 * (4 + 3 + 3) + 0.02 * (4 + 3 + 3) + 0.2 * (3 + 3) = 4.71. When $k = 2$, the best location for two SNs are at A and J. Even though nodes A and B have the two lowest costs for k=1, the best pair of SN nodes, where each node selects the closest SN has the lowest cost for k=2. This cost involves assigning nodes B, C, D, E, F, and G to SN A and nodes H and I to SN J, giving a total cost of the A and

J SN components (from Table 2.1) (0.15 + 0.5 + 0.36 + 0.1 + 1.6 + 0.1) + (0.2 + 0.08) = 3.09.

2.1.2 Previous Solutions and Approximations for the k-median problem

Various solution methods exist for solving the k-median problem. Reese [61] provides a survey of techniques from the initial problem formulation to advanced approximation and genetic algorithms. The best known approximation for k-median to date is from Arya et al. [4] with a $(3 + \frac{2}{p}\ )$ times the optimal solution using a local search heuristic. The running time is $O(n^p)$ where p is the number of facilities the local search is able to simultaneously swap out at a given time. Previous works involved different solution techniques. Shmoys, Tardos and Aardal [67] and Charikar, Guha, Shmoys and Tardos [12] formulated the problem as linear programs and rounded the results for a $6\ \frac{2}{3}$ times the optimal placement cost. Jain and Varizani [32] provided a primal-dual solution which lowered the k-median to 6 times the optimal placement cost. Charikar and Guha [11] improved the k-median to 4 times the optimal placement cost.

Placing SNs in a P2P network is more than just the k-median or facility location problem. Although the inputs to both problems are similar, the distance to each node and the demand for service, the SN placement problem is concerned with the state of each node. Nodes are not obligated to participate in the P2P network and may exit at any time. Developing a mechanism to quickly and efficiently communicate with nodes and monitor their status is necessary. Finally, in order to scale the solution to support hundreds or thousands of nodes with a large number of constraints, a centralized k-medians or facility location algorithm alone may prove too costly in execution time.

2.2 Placement of Services

This section describes the placement of service center or facilities as it relates to networking. The related work is interested in placing services in a network for a range of applications from content distribution networks (CDN) to web server caches. One of the first references to the k-median problem in P2P networks deploying CDNs is by Qui [59]. In this work, various algorithms for the placement of CDNs are investigated. An optimal model is compared to various competing strategies such as greedy, random, and hot spot. The hot spot and greedy methods demonstrated comparable performance to the super-optimal model. These algorithms are centralized, though, and their scalability was demonstrated up to a few hundred nodes in simulation. The evaluation of larger experiments using distributed algorithms was proposed in the future work.

2.2.1 Distributed Algorithms

The use of distributed algorithms can greatly reduce the computational complexity in placing SNs in a P2P network. The following describes several distributed algorithms used to place services.

Neighborhood Based Distributed Topology

Laoutaris et al. [34] describes a distributed algorithm for the k-median and facility location problems, referred to as the r-ball algorithm. The nodes in the network form groups called r-balls. An r-ball is a collection of nodes that are r hops away from a facility. All nodes connect to exactly one facility and are either considered inside or outside of the r-ball. Nodes within a predefined radius r are inside the r-ball and all others are outside of it.

Facilities maintain an exact network topology of the nodes inside the r-ball. The facilities also monitor the demand for service from all nodes connected to it. This includes nodes inside and outside of the r-ball. A node outside of the r-ball communicates its demand for service through its closest *ring node*. The ring nodes are those nodes furthest from the facility yet still within a radius of r from it. This approximate value is placed on the edge of the r-ball ring. The approximate demand of nodes on the ring of the r-ball increases proportionally to account for aggregate flows from outside the r-ball. The information is then used to solve the k-median or facility location problem locally using integer programming. This approach is called a limited horizon view because complete knowledge is available for nodes close to a facility and only an approximation of the demand and topology is available for rest of the nodes.

An example topology is provided in Figure 2.2. Here two r-balls are shown, with radius $r = 1$. Nodes B and I are the facilities in each r-ball. In r-ball 1, nodes A, E, or D may become the new facility based on the solution from the locally performed k-median or facility location problem. Nodes C and F are more than r hops away from the facility and thus are not eligible to become facilities in the next iteration of the algorithm.

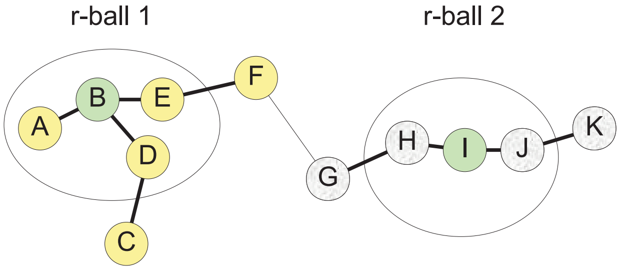


Figure 2.2: Distributed algorithm example topology. Two r-balls are shown with facilities located at nodes B and I. Nodes inside the ring are eligible to become new facilities based on the local solution of k-median or facility location.

Due to the limited knowledge of the r-ball algorithm, the local solution is only able to relocate a facility to another node inside its r-ball. If $r = 1$, then a facility is only able to relocate to a node one hop from its current location. Matlab simulations have shown setting r to small values such as 1 or 2 can provide SN placement solutions close to those obtained with an optimal centralized algorithm [34]. A trade-off exists between the scalability and the overall performance of the algorithm. As the size of r increases, the benefits of a distributed algorithm diminish, ultimately limiting the scalability.

A discussion of the r-ball algorithm used by Laotaris et al. is now provided. Notation and definitions are first discussed. A network $G = (V, E)$ is defined by a node set $V = \{v_1, v_2, ..., v_n\}$ and an undirected edge set E. Let $d(v_i, v_j)$ denote the length of the shortest path between v_i and v_j. Let $s(v_j)$ represent the service demand originating from node v_j. Let $F \subseteq V$ denote a set of facility nodes. Each iteration of the r-ball algorithm is represented with the superscript m. Let $F^{(m)} \subseteq V$ represent the set of facilities at the mth iteration. Let $V_i^{(m)}$ be defined as the r-ball of facility node v_i and all nodes that are within a radius r of v_i. Let $U_i^{(m)}$ denote nodes outside the r-ball V_i that selected v_i as their closest facility. These nodes receive service from the facility v_i but do not provide demand and topology information to it. The domain $W_i^{(m)} = V_i^{(m)} \bigcup U_i^{(m)}$ consists of a facility node of its r-ball and the surrounding ring. From the previous definitions $V = V^{(m)} \bigcup U^{(m)}$ where $V^{(m)} = \bigcup_{v_i \in F^{(m)}} V_i^{(m)}, U^{(m)} = \bigcup_{v_i \in F^{(m)}} U_i^{(m)}$.

Some initialization is required before executing the algorithm. First create a set $F^{(0)} \subseteq V$ of $k_0 = |F^{(0)}|$ nodes to act as the initial set of facilities. Also, let $\mathcal{F} = F^{(0)}$ denote the unprocessed facilities. Finally, let $\mathcal{F}^- = F^{(0)}$ denote a variable containing the initial group of facilities.

Algorithm 1 r-ball Distributed Algorithm

```
 1: while $\mathcal{F} \neq \emptyset$ do
 2:   $v_i \in \mathcal{F}$
 3:   $DiscoverLocalTopology(v_i)$
 4:   $G_J \leftarrow TestRballNeighborMerge(v_i)$
 5:   $F^{(m)} \leftarrow OptimizeRballGroup(G_J)$
 6:   $\mathcal{F} \leftarrow \mathcal{F} \setminus (J \bigcap \mathcal{F}^-)$ {Remove Processed Facilities}
 7:   if $\mathcal{F} = \emptyset$ then
 8:     if $F^{(m)} \neq \mathcal{F}^-$ then
 9:       $\mathcal{F} \leftarrow F^{(m)}, \mathcal{F}^- \leftarrow F^{(m)}$
10:     end if
11:   end if
12: end while
```

From Algorithm 1, the first step is to select some facility v_i from the batch of facilities. Next, discover the topology of nodes close to it using a neighborhood discovery protocol (lines 2-3). No details are provided for the discovery protocol, however a reference to related work is offered [44]. Next, test if the r-ball can merge with neighbor r-balls. This test checks for intersections between r-balls (line 4). An intersect exists if at least one node is within a distance r from all facilities under consideration. Let $J \subseteq F^{(m)}$ denote a composition of v_i with the facilities that it can merge with. J produces an r-shape $G_J = (V_J, E_J)$ which is a subgraph of G composed of the facilities of J, their neighbors up to a distance of r, and the edges between them. Restrictions can also be placed on the maximal size of the r-shape so it is much smaller than $O(n)$.

The next step is to optimize the single r-ball or if joined with neighboring r-balls the r-shape G_J. This is accomplished through integer linear programming to solve the

k-median problem using the facilities and nodes inside the r-shape as inputs. The solution of the k-median problem returns the best locations for the $|J|$ facilities (line 5).

After determining the locally optimal solutions, remove the processed facilities, both v_i and the merged facilities, from $\mathcal{F}$. Also update the set of current facilities $F^{(m)}$ with the new locations from the local optimization (line 6).

Finally, test if all facilities have converged to locally optimal facilities (lines 7 - 11). If $\mathcal{F} = \emptyset$, then all facilities of the latest batch have been processed, if not iterate through the algorithm again. After processing all facilities, check if the locations of the current set of $F^{(m)}$ are different than the initial set $\mathcal{F}^{-}$. If they are different, re-iterate with a new batch of facilities where $\mathcal{F} = F^{(m)}$ and $\mathcal{F}^{-} = F^{(m)}$. Otherwise terminate by returning the locally optimal solutions.

Using the algorithm presented, Laoutaris et al. performed Matlab simulations comparing the placement of the r-ball algorithm to the optimal placement of facilities. The impressive simulation results helped motivate a starting point for this dissertation. My research takes advantage of the limited horizon view similar to r-balls and extends it. My work places emphasis on a slightly different problem where the placement of nodes requires more precision than selecting a suitable autonomous systems (AS). Also, assumptions such as inferring node topology and communication with multiple SNs in a single r-ball are left unexplored in the previous work. This dissertation embraces the limited horizon view of service placement and focuses on placement down to the node level, making it more applicable for a different set of applications.

2.3 Peer to Peer Applications and Supernodes

This section provides background on a range of topics dealing with P2P and supernodes. The rise of P2P and early academic research in P2P networking is first discussed. Then some background about supernodes and applications deploying them is provided. Finally, a discussion of APIs available to deploy and maintain P2P networks is presented.

The advances of P2P are arguably due to the popularity of Napster [51]. Napster provided a mechanism to download music from a centralized search through nodes connected to the Napster P2P network. The research community responded to the interest generated from Napster and its clones by creating services which provided distributed file indexing and sharing. These research efforts demonstrated a decentralized, scalable approach to the Napster phenomenon. One popular technique, utilizing distributed hash tables (DHT), was developed to organizes node placement and file indexing. Examples of a few academic P2P networks are now provided.

Initial research in developing *structured* P2P networks originated from Plaxton [57]. This work established the idea of creating structured P2P networks based on routing data to a unique identifier (ID) for each node.

One of the first widely deployed academic projects is Pastry [64]. Pastry's structured network came from the P2P ring formed with uniquely identifiable nodes which maintained pointers to their neighbors and the nodes spread throughout the network. Similar to other first generation P2P networks, Pastry maintained an $O(\log N)$ query lookup time where N was the number of hops needed to resolve a file query. Other examples of first generation P2P networks include Chord [72] and Content-Addressable Networks (CAN) [60].

Gnutella [25] competed with Napster in the commercial P2P market. The Gnutella network provided decentralized search capabilities and consisted of an *unstructured* P2P network. In order for a query to propagate through the network, a Gnutella node flooded its P2P neighbors with a request to locate the file. Later versions of the software introduced the concept of supernodes (SNs). That created a two-tier P2P network where a subset of the regular nodes connected directly to a single SN. Each SN maintained indices of a file subset in the network and communicated with additional SNs to locate files. In addition to providing file indices, SNs also served as a bootstrapping mechanism for nodes joining the P2P network.

Another widely popular P2P application is Skype, a VoIP service with over 246 million registered users [69] and up to 10 million users online at a given time [68]. Here the SNs provide two useful services. First, they serve as a bootstrap device necessary to provide initial connectivity. Second, in the event that direct communicate between the parties is unavailable, due to Network Address Translation (NAT) or firewall constraints, the SN can serve as a relay node to connect both parties.

2.3.1 SNs

SNs provide a range of capabilities such as distributing data throughout the network, balancing load, and providing network services. SNs manage other nodes joining the P2P network, route search queries, and act as data rendezvous points. They generally have more bandwidth and processing power than other nodes in the P2P network. In current systems, the process of selecting a SN is typically arbitrary. The selection of SNs may be based on the order in which nodes join or random selection.

Recent work demonstrates the importance of SN placement [38]. Here, three different placement solutions, based on the type of P2P network evaluated studying, were introduced. The paper considered structured, unstructured and coordinate based networks. Their research also relates the SN location problem to the k-median or absolute centers problem. Unfortunately, their research lacks evaluation of the proposed protocols and cites a need for more algorithmic approaches to solve this problem.

In addition to placing SNs, P2P networks also need to determine the number of SNs to deploy. This decision depends on several properties of the network. The first is the amount of communication between SNs and regular nodes in the network. If the communication is fairly small, maintaining a SN ratio of 100:1 or 1000:1 may prove sufficient. However, if large amounts of communication are necessary, additional SNs need to be deployed. Another property to consider is the amount of computation SNs perform. In addition to providing bootstrapping capabilities, SNs can perform additional computational services such as maintaining the state of nodes in the network.

2.3.2 Application Specific Placement Solutions

One application receiving some attention in regards to SN placement is VoIP. Ren et al. [62] introduce a concept of deploying an P2P network to shortcut the small percentage (10%) of VoIP traffic where the default routing provides RTT times greater than than 300 ms.

Another area of research for improving VoIP traffic is through the use of fast path switching [74]. Here the communication path for VoIP traffic is based on measurements of path quality and changes dynamically. Both of these approaches are effective at improving VoIP applications, particulary Skype. Our research approaches the problem differently by reducing the maximum cost of SN placement for the entire network.

2.3.3 Application Programming Interfaces (API)

P2P development tools and APIs provide useful abstractions for the creation and use of P2P networks. JXTA [78] is an open source API that simplifies this process. This API provides an open infrastructure allowing developers to create loosely-consistent distributed hash table (DHT) networks that operate across a variety of platforms. The original implementation was developed in Java, but versions for the C language and low powered mobile devices also exist. The default message distribution communicates with nodes via broadcast messages. JXTA also supports a two-tiered hierarchy for P2P networks using SNs similar to SNs in Gnutella. JXTA uses the term rendezvous node when discussing SNs.

As structured P2P networks became more popular, a common API was also developed, called OpenDHT [33]. This allowed developers to use commands such as *put* and *get* to communicate with a DHT. Providing such a simple API allows for application developers to easily extend OpenDHT for useful networking services. An example service built on top of the OpenDHT architecture is $i3$ [73]. The $i3$ service attempts to generalize the point-to-point communication in the Internet with services like multicast and anycast.

The growth of P2P applications has demonstrated a need for SNs to provide more efficient communication. Tools and APIs that allow for easier deployment of P2P applications are invaluable. The power and ease of use of the JXTA API is explored in later chapters as an efficient tool to develop networking applications.

2.4 Tools to Evaluate Network Services

This section describes different approaches for evaluating distributed network services. These approaches vary from software simulation to distributed emulation on a large number of hosts.

2.4.1 Network Simulators

One common method to evaluate network protocols and applications is through software simulation. Different software suites exist to evaluate P2P protocols and range from ad hoc simulators written specifically for one application to plug-ins for commonly used network simulators.

One of the most successful networking simulators is ns-2 [53]; a discrete event simulator that models network protocols. Experiments are created with the Tcl language. The experimenter describes node topologies, traffic patterns, link delays and bandwidth constraints on the network. The simulator generates traffic based on input models for applications and provides transport layer communication between nodes.

The ns-2 simulator has evolved considerably since its initial deployment. It now provides models for wireless links, quality of service, multicast, and other services. The software simulator is very popular for simulating arbitrary network topologies; however, it lacks the ability to accurately model complex interactions between higher level applications. Support does exist to create addon modules for specific application layer protocols, but, the simulation complexity greatly increases. This prevents ns-2 from executing a P2P simulation with more than a few nodes in a reasonable amount of time, especially compared to emulation testbeds or P2P network simulators.

Another method of software simulation is through ad hoc simulators written for a specific application. Consider a P2P network service such as Pastry [10] or Chord [72]. Typically, these software distributions provide a simulation model of nodes joining and leaving the network. Unfortunately, they also lack the realism of an actual implementation in terms of capturing network link characteristics and underlying protocols.

More generic P2P simulators were developed in an attempt to model the behavior of various P2P protocols. However, these P2P simulators ignore link level details due to the complexity and processing involved. PeerSim [55] is one of the most widely used examples of this type of simulator. It implements the Chord and Pastry protocols and provides an API to augment the simulator, allowing more complex applications to run on top of it. P2PSim [54] is another example of a P2P Simulator. Similar to PeerSim, P2PSim also provides an implementation of the Chord protocol.

2.4.2 Emulation Testbeds

A different model for evaluating P2P networks is through emulation testbeds. Network emulation testbeds vary in size, administrative control, accessibility, and reprogrammability. This section discusses the most popular testbeds and future directions in this field.

The first large scale emulation testbed developed for researchers was Emulab [80]. Emulab is a network testbed that provides researchers a range of environments to evaluate systems. Emulab currently has 365 PCs available for a single experimenter to connect in an arbitrary topology. Users configure each PC with a different operating system from various distributions of Linux to Windows XP. It allows users to create experiments, maintain complete control over the type of experiment, and specify the latency and bandwidth between nodes. Each node supports multiple interfaces and allows the creation of arbitrary network topologies. Emulab also supports virtualization using FreeBSD jails [30]. Jails allow a researcher to increase the size of the experiment beyond the physical limitations of the number of nodes available.

The Emulab framework has been replicated for specialized testbeds such as DETER [6] and WAIL [79]. The framework provides users with a Web interface that allows the creation of topologies, management of experiments, and remote access to testbed nodes.

Of the testbeds deploying the Emulab infrastructure, the cyber-DEfense Technology Experimental Research Testbed (DETER) [6] is the most notable. DETER performs large scale security experiments from virus propagation to DNS root server failures. It contains 260 nodes and is collocated at UCSD and ICSI. Specialized hardware performs additional security processing, such as the NetFPGA board [40] and commercial routers from Juniper like the M7i [31].

Another replication of the Emulab efforts exists in the Wide Area Internet Lab (WAIL) [79]. WAIL emphasizes experimentation with network equipment through the configuration of various Cisco switches and routers. It offers 120 PCs, 50 IP routers and switches, and hardware traffic generators. Researchers also developed a scalable network path emulation tool [1] that provides reliable link delays between nodes and models latencies.

The Open Network Lab (ONL) [15] is a testbed that provides reconfigurable hardware in the network. ONL uses gigabit switches with per port processing so users can deploy transport level protocols and network services. Through the use of the Smart Port Card (SPC) [16] and the Field Programmable Port Extender (FPX) [39] researchers can create environments utilizing network processing previously unavailable with commodity PCs. A Java Graphical User Interface (GUI) allows remote access to the testbed to configure links between ports and monitor traffic between nodes.

The largest distributed testbed available for researchers is PlanetLab [56]. PlanetLab currently contains 917 nodes connected over the Internet. Each node uses virtualization to provide administrative control over a slice of a node. This presents each user with their own share of CPU, memory, and I/O. Machines are typically located within universities and accounts are issued to universities that contribute machines to the testbed. PlanetLab also provides priority scheduling of slices that allow 25% of the CPU and 2 Mbps of network bandwidth.

One of the next generation testbeds proposed is the Virtual Network Infrastructure (VINI) [5]. This architecture would provide the accessibility of Planet Lab and the reproducibility and isolation of Emulab to create a virtual network of nodes capable of behaving as routers using open-source software. VINI will provide multiple paths between nodes and allows Internet traffic to pass through the testbed. Researchers can then better transition experiments from emulation to deployment. The preliminary results show a substantial performance penalty from modern CPUs performing routing in software.

A slightly different direction for future testbeds is Flexlab [19]. It combines Emulab with Internet traffic models that will allow researchers to experience Internet-like behavior from an application's point of view. The over utilization of resources in PlanetLab helps motivate a testbed such as Flexlab. Flexlab aims to provide the best of both worlds in terms of controllable experiments and Internet realism.

In this dissertation experiments are conducted using Emulab and Planetlab. The isolation of Emulab is invaluable for developing and debugging distributed systems. The virtualization of Emulab also allows for larger isolated experiments to better understand scalability issues. Planetlab is also utilized because of the diversity obtained through shared networking and compute infrastructure. Deploying the experiment in such a diverse environment leads to interesting and unexpected system behavior.

2.5 Evaluation of Distributed Network Services

Large scale systems research has grown in recent years due to the increase in computing power and falling prices of commodity hardware. These advances allow researches to create bigger experiments in less time. A discussion is now provided describing the several related works studying large systems.

The Julia Content Distribution Network (CDN) [8] is a P2P distribution tool that demonstrates a fairly common experimentation methodology. First a custom discrete event simulator is created to verify the algorithm. Then, experiments are deployed with a few hundred nodes on PlanetLab. The simulation consists of approximately 1200 nodes and 250 nodes are used in PlanetLab.

As with most large systems, some characteristics of the Julia CDN simulations agree with results observed experimentally. However, at least one metric, the absolute time associated with distributing a piece of data, fails to demonstrate simulation and emulation results that agree. In fact, data was not even reported on 50 of the 250 nodes due to non-ideal operating situations. The authors state that the simulation did not capture the properties of the network well. Although not surprising, it motivates the use of emulation and development of better simulation models to introduce as much realism as possible.

The evaluation of SNs is another area of much interest, even though much of the previous work is limited to simulation models. Mastroianni et al. [45] present a super peer model for resource discovery. They provide an interesting design and analysis of the costs to deploy super peers. The simulation model is an event-based object simulator that reports metrics such as the message load on a super peer and the average response time of a super peer query depending on the size of the network. The overall system performance is left unexplored.

Lin et al. [37] present a technique to simulate very large (over 1 million nodes) systems. They introduce Slow Message Relaxation (SMR) that trades simulation accuracy for performance. It allows the simulator to execute events further ahead in time than traditional simulators. SMR avoids the waiting endured by previously used minimum network delay models. The authors claim SMR is a reasonable approach because of its distributed protocol resiliency. With the use of a 250 machine cluster, these simulations are able to exceed 1.5 million nodes. The technique provides an order of magnitude increase in speed while maintaining statistically accurate simulations. What remains to be seen in this work is a smaller simulation would agree with an implementation.

With these different experimentation approaches, it is clear that one size does not fit all for all types of experiments. Depending on the size and complexity of the algorithms, experiments may be better suited for one type of evaluation over another. In this dissertation a range of simulation and emulation results are gathered. Relationships between simulated models and experimental data are also investigated thoroughly.

Chapter 3

Supernode Placement Problem

3.1 Preliminaries

Definitions are now provided on the types of network elements used to model the supernode (SN) placement problem. The network is made up of components described in Figure 3.1. The network's routers provide communication between nodes. Connections can exist between routers and routers and between routers and a node. At any given time, nodes are classified into four states. The nodes themselves operate in the active or inactive state. Nodes in the active state may also operate in the willing to become SN state or SN state. Only nodes in the willing to become SN state may be assigned to the SN state. A hierarchical relationship between the message sending nodes is depicted in Figure 3.2.

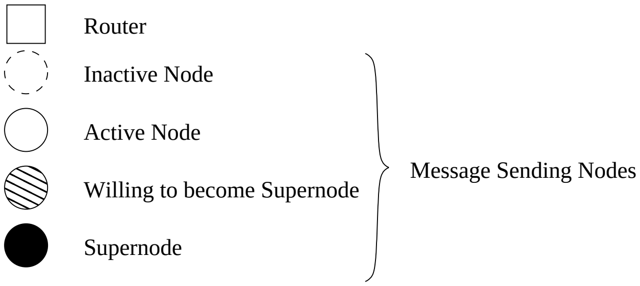


Figure 3.1: Components in the network graph topology.

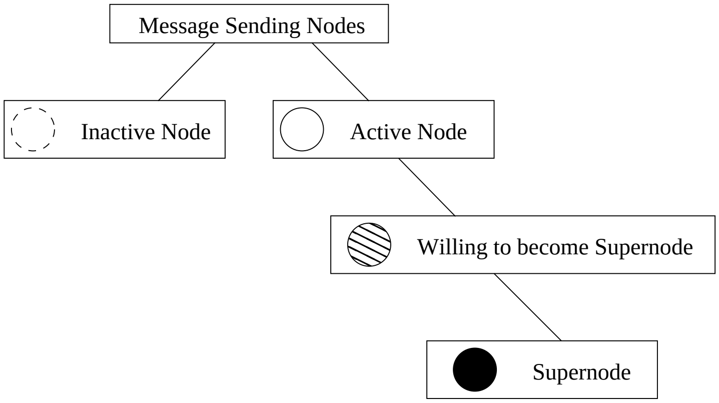


Figure 3.2: Hierarchy of Message Sending Nodes.

In order to evaluate the assignment of SNs at particular nodes, a method for computing a global cost is provided. This cost metric is based on the node state and topology. Consider the network topology T_1 shown in Figure 3.3. This topology contains three nodes A, B, and C and one router providing communication between the nodes. From Figure 3.3 we can see that Node A is one unit of distance from the router and two units from Node B.

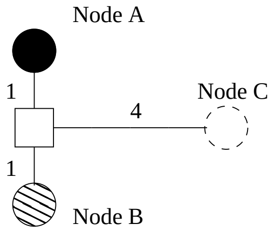


Figure 3.3: Example topology with three nodes A, B, and C connected through a router. Nodes A and B are active and A is the SN.

When describing the state of each node we define a labeling partition P_i of the topology T as $P_i(T)$ or just P_i when referring to a single topology. Let a particular partition P_i of topology T indicate the state of each node as inactive, active, willing to become SN, or SN. The partition P_1 in Figure 3.3 depicts node A as an SN, node B as willing to become an SN and node C as inactive.

We determine the assignment of SNs based on their distance to other active nodes and the demand of each node to communicate with an SN. Initially, the demand to use SNs is uniform. The number of nodes assigned as SNs is k, where k is determined a priori. The distance between nodes is measured in nonnegative units. In general, we define a cost for a given partition P_i as Cost(P_i). This cost refers to the service cost for k nodes operating as SNs in a particular partition. This does not include transition costs dealing with nodes moving from one state to another. A simple example is now provided. Consider the cost of the previously described partition P_1 with node A as the only assigned SN, denoted as Cost(P_1). The cost is now computed below.

$$Cost(P_1) = d(A, A) + d(B, A) = 0 + (1 + 1) = 2$$

With uniform demand, the cost is simply the distance from node B to node A. Note, the cost with node B as the SN is the same as Cost(P_1). Therefore, the selection of node A instead of node B as SN is arbitrary.

Now consider Figure 3.4 where node C becomes active and willing to become an SN, called partition P_2. If node A remains as the SN, the cost becomes simply the distance from node C to the SN plus the cost Cost(P_1) as displayed below.

$$Cost(P_3) = Cost(P_1) + d(C, A) = 2 + (4 + 1) = 7$$

Suppose two SNs were required and assigned to topology T_1 as shown in Figure 3.5. Here, assigning SNs at nodes A and C (called partition P_4) results in the lowest cost. When

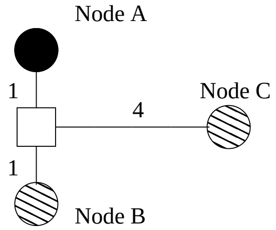


Figure 3.4: Example topology with all three nodes active and A is the SN.

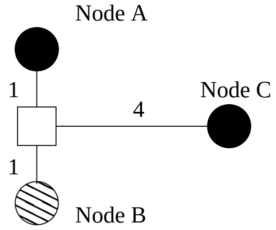


Figure 3.5: Example topology with three active nodes, where nodes A and C are SNs.

multiple SNs are available, the active node selects the SN closest to it in terms of distance. The cost of this partition is computed below.

$$Cost(P_4) = d(A, A) + d(B, A) + d(C, C) = 0 + 2 + 0 = 2$$

Clearly adding more SNs will result in a lower cost partition if SNs are chosen carefully. However, if every node is assigned as an SN, the benefits of this two-tiered architecture disappear. Next we provide a formal statement of the problem.

3.2 Problem Statement

Let $G = (V, E)$ be a weighted, undirected graph, where V represents the routers and message sending nodes. We define a partition in V where V_N represents the message sending nodes and V_R represents the routers. Therefore $V_N \cup V_R = V$ and $V_N \cap V_R = \emptyset$. All V_N nodes have a degree of 1 and all V_R nodes have a degree > 1. Also, we define a partition in V_N where V_I represent the inactive nodes and V_A represent the active nodes. Therefore, $V_I \cup V_A = V_N$ and $V_I \cap V_A = \emptyset$. Let V_W represent the active nodes that are willing to become SNs. Let V_k represent the active nodes that are assigned as SNs. Therefore $V_k \subseteq V_W \subseteq V_A$. E represents the direct physical connections between vertices in V. Associated with each edge $e \in E$ is a positive weight $w(e)$, the communication metric of interest.

Let $d(u, v)$ represent the shortest distance between nodes u and v according to weights w. Note: $d(u, v)$ is defined on $u, v \in V$; however our usage will be constrained to the circumstance $u, v \in V_N$, hence the use of the term node for u and v. Each node u is assigned a demand $t(u)$. Let $d(u, J)$ be the shortest distance from the node u to its nearest element in the set $J \subseteq V_N$. Our problem is to find a set of exactly k nodes in V_W, called V_k, that will be assigned as SNs. Determine V_k such that

$$\forall S \in 2^{V_W}, |S| = k \rightarrow (\sum_{u \in V_A} t(u)d(u, V_k) \leq \sum_{u \in V_A} t(u)d(u, S)). \tag{3.1}$$

3.3 Cost Properties of SN Topologies

Next we consider different properties of SN placement. The costs of moving a node from inactive to active state, relocating an SN, and transition an additional node from willing to become SN to the SN state, will be discussed.

3.3.1 Cost for a single node to become active

Consider a network that assigns SNs based on the problem statement previously described. Let us denote the optimal cost associated with the assignment of a set V_k of k nodes as SNs in a partition P_j as $\mathrm{Cost}^*(P_j)$. We next describe the cost of a new partition P_m that results when a single node u transitions from the inactive to active state and a new assignment of SNs is not computed (i.e., the set V_k is unaltered). The cost of this new partition P_m, with the old assignments of SNs, is described in Equation 3.2, where $d(u, V_k)$ is the shortest distance from node u to its nearest node in the set V_k.

$$Cost(P_m) = Cost^*(P_j) + t(u)d(u, V_k)) \tag{3.2}$$

This new cost is just the distance of the recently active node u to the closest SN in S and the previous optimal cost P_j before u became active.

3.3.2 Reducing cost with better SN placement

Consider a topology T_2 with five nodes, a partition P_5 in which four nodes are initially active and one is inactive and all demands are unity ($\forall u, t(u) = 1$), as shown in Figure 3.6. Each active node in the figure is a distance of 6 from each other. This symmetry between nodes makes the SN assignment arbitrary with node A being assigned as the SN in this example. The optimal cost of this partition with an SN assignment of node A is displayed below.

$$Cost^*(P_5) = d(A, A) + d(B, A) + d(C, A) + d(D, A) = 18$$

Now let node E become an active node in the willing to become SN state and call this

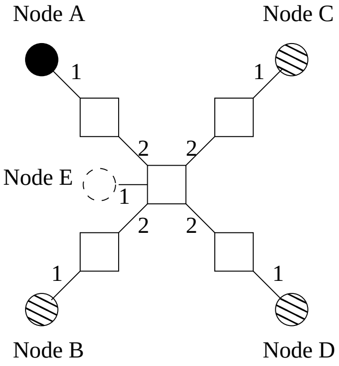


Figure 3.6: Example topology with four active nodes with node A assigned as the SN.

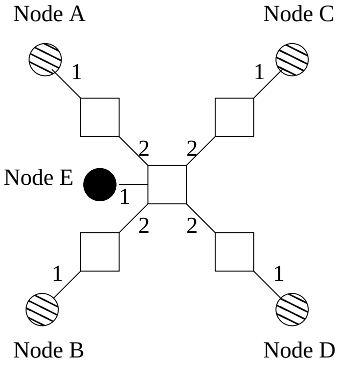


Figure 3.7: Example topology with five active nodes with node F assigned as the SN.

partition P_6. The cost for node E to join without recalculating the SN assignment is shown below, which is the same as Equation 3.2.

$$Cost(P_6) = Cost^*(P_5) + d(E, A) = 18 + 4 = 22$$

If the optimal cost of the new partition is determined, node E is assigned as the SN as shown in Figure 3.7 and called P_7. The cost for this assignment is shown below.

$$Cost^*(P_7) = d(A, E) + d(B, E) + d(C, E) + d(D, E) + d(E, E) = 12$$

From this example we can see that it is possible to lower the cost of an SN assignment for a given topology with an additional node becoming active. Intuitively, one might expect the cost of an SN assignment to increase as more nodes become active. However, opportunities for reducing cost can exist as additional nodes become active.

3.3.3 Bounds for a single node becoming active with SN reassignment

Next we seek to find a lower bound of cost when a single node in a partition moves from the inactive to active willing to become SN state. Suppose for some topology T, and some partition P_j the SN is assigned at some node u, and the optimal cost is $\text{Cost}^*(P_j)$. Assume one node, v moves from the inactive to active state to create the partition P_m. Now perform a reevaluation of the SN assignment for all nodes in the topology. Let y represent the new

SN with optimal cost. Now we can discuss the upper and lower bound cost of this new partition P_m with respect to P_j.

First we will show the upper bound of the new cost, Cost$^*(P_m)$ in Equation 3.3.

$$Cost^*(P_m) \leq Cost^*(P_j) + d(y, v) \tag{3.3}$$

Suppose Equation 3.3 is not true. Thus, Cost(P_2) is more expensive than Cost$(P_1) + d(y, v)$. However, by definition of the original problem, Cost(P_2) can not be more expensive, because a valid SN assignment of Cost$(P_1) + d(y, v)$ exists and would be selected instead of assigning the SN at node y, thus a contradiction. Therefore, Equation 3.3 is an upper bound for the cost of one additional node becoming active with a single SN reassignment.

Now we provide a lower bound for the cost of the partition P_m after a single new node transitions from inactive to willing to become SN state. Assume v is located as close as possible to every other node in the topology. The best assignment of any node as the SN, by definition of the original problem, is an assignment that provides the shortest distance from it to all other nodes. Therefore, the best assignment for a single SN assignment is one that cuts the distance in half to all other nodes in the topology. Thus reducing the costs to no more than half the cost of Cost(P_j) plus the cost of the old SN connecting to the new SN, as shown in Equation 3.4.

$$Cost^*(P_m) \geq \frac{1}{2}Cost^*(P_j) + d(u, v) \tag{3.4}$$

Therefore, the lower bound on a node transition from the inactive to active state is at least $\frac{1}{2}$ the cost of the partition before the node moved from the inactive to active willing to become SN state.

This section described a few properties of the SN placement problem. What follows are the initial algorithms used to place SNs in a P2P network.

3.4 Distributed Supernode Placement Algorithms

This section presents the distributed algorithm developed to assign nodes as SNs in an overlay network. Intuitively, the algorithm divides the entire overlay network into smaller groups called *neighborhoods*. A neighborhood consists of a SN and nodes that are within a distance r from the SN and that are in active willing to become SN state. A network topology is then obtained for all the nodes inside the neighborhood. The demand to use the SN service is also collected from the nodes inside the neighborhood as well as an approximation of demand from the nodes outside the neighborhood. This information is then used to compute a locally optimal assignment for a SN inside each neighborhood.

As the assignment of SN changes, so does the neighborhood surrounding the SN. The movement of the neighborhood can create an overlap between two neighborhoods. When this occurs, the neighborhoods are able to merge together to form larger neighborhoods. These larger neighborhoods may eventually split up due to future SN assignments. This SN placement completes once each SN reaches its locally optimal assignment, where any additional move does not lower the cost of the SN placement.

3.4.1 General Behavior

More detail is now provided about the general behavior of the placement algorithm along with an example. Nodes initially joining the network connect to a well known bootstrap node for authentication and initialization. Once authenticated, a node is either promoted to SN status or provided the address of an SN to connect to. Each is then notified by that SN if the node is close enough to join the neighborhood. Those nodes not joining the neighborhood locate the closest node in the neighborhood to act as a *neighborhood representative*. A neighborhood representative provides mechanisms for nodes outside the neighborhood to influence the future assignments of SNs. This is accomplished by the neighborhood representatives aggregating demand from nodes outside the neighborhood and representing it as their own. Where demand represents the desire for a node to use the service and we are assuming a demand of unity for each node. An example is now provided to explain how nodes, neighborhoods and neighborhood representatives interact.

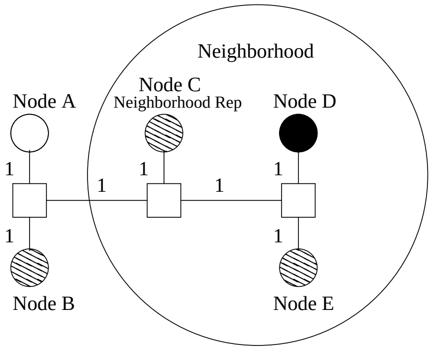


Figure 3.8: Example neighborhood with SN at node D and neighborhood representative at node C with an r value of 3.

Figure 3.8 depicts a network with three nodes inside and two nodes outside of a neighborhood. In this figure, the radius metric r=3, therefore nodes nodes C and E are inside the neighborhood (with distances of 3 and 2 to the SN, node D, respectively). Nodes A and B are too far from the SN therefore placed outside the neighborhood, node A is also not willing to become an SN (it is in the active state) and would not join the neighborhood even if it was close enough. This is because any node not in the willing to become an SN state can not join the neighbor because all nodes in the neighborhood must be eligible to become an SN. Nodes A and B must find their closest neighborhood representative and will select node C. With this topology, nodes A and B use node C as their neighborhood representative, and node C reports a demand of 3 to the SN, with node E reporting a demand of 1. With the node demands and topology information for nodes C, D, and E, the SN, node D, is ready to determine if it will reassign the SN to a new location. To do this, the SN solves the local k-median problem for three nodes with the specified demand and topology information using integer linear programming (ILP). In this example, the output of the k-median problem assigns node C as the SN and the neighborhood in Figure 3.9 is created.

This SN placement strategy strikes a balance between complete global knowledge and a very limited local view. With complete local knowledge of the distance to each node and associated demand, the SN is able to solve the k-median or problem while not

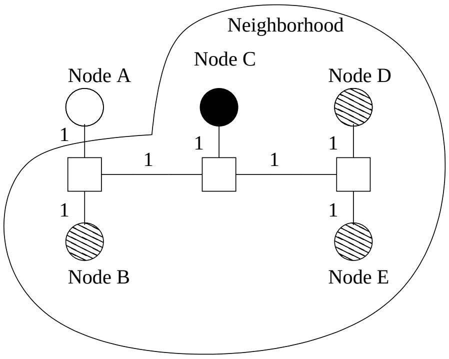


Figure 3.9: Example neighborhood with SN at node C.

completely ignoring nodes outside of the neighborhood by considering aggregate demand information.

3.4.2 Definitions

A description of the distributed algorithm will now be discussed using notation presented earlier. The initial algorithm is a slightly modified version of the r-ball algorithm from Chapter 2 and will be referred to as r-mod (Algorithm 2). We will discuss the r-mod algorithm using the notations discussed in this chapter and throughout the rest of the dissertation.

We describe each iteration of r-mod with the superscript m. Let $V_k^{(m)} \subseteq V_w$ represent the set of SNs at the mth iteration. We define $N_i^{(m)}$ as the neighborhood of nodes of SN v_i. Thus $N_i^{(m)}$ contains all nodes willing to be an SN within radius r of SN v_i. Let $U_i^{(m)}$ denote the set of nodes outside the radius r or unwilling to become SNs, but still connected to the SN node v_i.

The domain $W_i^{(m)} = N_i^{(m)} \bigcup U_i^{(m)}$ of a SN is itself and all nodes connecting to it. We can also describe $V_N = N^{(m)} \bigcup U^{(m)}$ where $N^{(m)} = \bigcup_{v_i \in V_k^{(m)}} N_i^{(m)}, U^{(m)} = \bigcup_{v_i \in V_k^{(m)}} U_i^{(m)}$.

3.4.3 Neighborhood Costs

In order to determine if an SN placement is better than another SN placement a cost metric is used to evaluate each topology. Earlier in this chapter, a cost metric was introduced in the problem statement. Now we discuss cost as it relates to distributed placement and neighborhoods. Therefore, we are in interested in the costs associated with a particular neighborhood given the current SN placement and the new SN placement. We will formally define two costs one known as the *neighborhoodCost* and the other as the *trueNeighborhoodCost*. The neighborhoodCost is the cost of placing SNs with respect to the costs and demands of nodes inside the neighborhood. This includes the aggregate demands from the neighborhood representatives. A few definitions are now introduced. Let $Q_u^{(m)}$ be the set of nodes connected to the neighborhood representative u at iteration m, where $Q_u^{(m)} \subseteq U_i^{(m)}$. Also we introduce $T(u)$ as the aggregate demand of some neighborhood node u as defined below.

$$T(u) = t(u) + \sum_{w \in Q_u^{(m)}} t(w) \tag{3.5}$$

With these definitions we can now present the neighborhoodCost which computes the cost for locating an SN in a neighborhood based on the aggregate demands of all nodes in the neighborhood and their distances to the SN.

$$neighborhoodCost = \sum_{u \in N_i^{(m)}} T(u)d(u, v_i) \tag{3.6}$$

The trueNeighborhoodCost is very similar to neighborhoodCost. However, the trueNeighborhoodCost includes the individual demands of all nodes connected to the SN in a particular neighborhood. Also, the aggregate demands are no longer applicable as distance and demand information is received from every node connecting to the SN in this neighborhood. The trueNeighborhoodCost is now defined below.

$$trueNeighborhoodCost = \sum_{u \in W_i^{(m)}} t(u)d(u, v_i) \tag{3.7}$$

As mentioned before, this cost provides complete information regarding the distance and demand of all nodes associated with a neighborhood. This provides a means to determine SN placement improvement in absolute terms from from a previous SN placement, assuming the set of nodes connected to the neighborhood has not changed.

3.4.4 Distributed Supernode Placement Algorithm

Some initialization is necessary before executing r-mod. First, randomly pick an initial set $V_k^{(0)} \subseteq V_w$ of $k_0 = |V_k^{(0)}|$ nodes to act as SNs. Also, let $V_k = V_k^{(0)}$ denote the unprocessed SNs, or SNs that have not performed an iteration of the placement algorithm. Finally, let $V_k^- = V_k^{(0)}$ denote a variable containing the initial group of SNs.

Algorithm 2 Placing SNs in a P2P Network (r-mod)

```
$ConnectToBootstrapNode()$
**while** $V_k \neq \emptyset$ **do**
    $v_i \in V_k$
    $DiscoverLocalTopology(v_i)$
    $G_J \leftarrow TestSupernodeNeighborMerge(v_i)$
    $V_k^{(m)} \leftarrow OptimizeSupernodeGroup(G_J)$
    $V_k \leftarrow V_k \setminus (J \bigcap V_k^-)$ {Remove Processed Supernodes}
    $UpdateBootstrapNode(V_k)$
    **if** $V_k = \emptyset$ **then**
        **if** $V_k^{(m)} \neq V_k^-$ **then**
            $V_k \leftarrow V_k^{(m)}, V_k^- \leftarrow V_k^{(m)}$
        **end if**
    **end if**
**end while**
```

The similarities and differences between the two algorithms, r-ball and r-mod, will now be discussed. The initial changes made to the r-ball algorithm arose from implementation issues and subtle differences in the problems each aims to solve. The first change made to the r-ball algorithm provides for a dynamic topology. The previous work assumed a fixed size for the topology and ran simulations based on this size. In r-mod, nodes are able to join the topology at any time. The second notable change is in the use of a bootstrap node. As mentioned earlier the bootstrap node is used to assign nodes as temporary SNs or provide an address of an existing SN (line 1). The bootstrap node also maintains some state for each SN to inform other SNs if it is able to merge at a particular time.

In lines 2 - 7 both algorithms perform nearly identical functionality. However, the DiscoverLocalTopology method (line 4) was not explicitly described in the r-ball literature, and simply referred to previous work [50]. We provide our description of the DiscoverLocalTopology method below. In line 8 another call is made to communicate with the bootstrap node. Here, any changes to the SNs are reflected back to the bootstrap node to maintain the state of the SNs. The remainder of r-mod is nearly identical to the r-ball algorithm lines 9 - 14, where each SN determines whether to continue trying to improve its position.

The DiscoverLocalTopology method (Algorithm 3) is now described. This method determines whether a node is inside or outside of the SN neighborhood (also known as an interior or exterior node). First, each node connects to the SN and sends an initialization string which reports its distance to the SN. The SN then decides if that nodes is an interior or exterior node. After the initialization period is over, the interior nodes receive a list of all other interior nodes from the SN. These nodes then calculate with all other interior nodes. These distance vectors are then sent to the SN. The SN uses this information to create a complete routing table of the neighborhood.

Now that we have presented an initial placement solution, an evaluation of this algorithm will be explored. The next chapter investigates an implementation and evaluation of this initial placement algorithm along with enhancements to provide improved SN placement.

Algorithm 3 Discover Local Topology

```
for all $v_i \in W_i^{(m)}$ do
    $DistanceToSN \leftarrow CalculateDistanceToSN(v_i)$
    if $DistanceToSN \leq r$ then
        if $WillingToBecomeSN(v_i$ then
            $N_i^{(m)} \leftarrow AddNodeToNeighborhood(v_i)$
        else
            $U_i^{(m)} \leftarrow AddNodeToOutsideNeighborhood(v_i)$
        end if
    else
        $U_i^{(m)} \leftarrow AddNodeToOutsideNeighborhood(v_i)$
    end if
end for
for all $v_i \in N_i^{(m)}$ do
    $SendNeighborInformation()$
end for
for all $v_i \in N_i^{(m)}$ do
    $ReceiveRouteTableInformation()$
    $ReceiveDemandInformation()$
end for
$CreateRouteTable()$
$CreateDemandArray()$
```

Chapter 4

Evaluation of Supernode Placement in Overlay Topologies

To better understand the behavior of the algorithms discussed in Chapter 3, a wide range of empirical data is presented. Careful analysis of this data yields insights into key characteristics of the current state of the art in SN placement and motivates additional improvements. This chapter focusses on understanding the initial behavior of the r-mod algorithm and a new algorithm called r-SPOT. Both of these algorithms operate inside the the Supernode Placement in Overlay Topologies (SPOT) system.

4.1 SPOT

SPOT is a distributed system written in Java that uses node and topology information to find a subset of the nodes to serve as SNs. This decision is based upon the cost metric described in the previous chapter. SPOT is designed as a service for other applications to utilize. A brief description of SPOT is now provided followed by an evaluation of the different aspects of the system.

4.2 General Behavior

Nodes participating in the SPOT service first connect to a well known bootstrap server for authentication. Once connected, these nodes are either promoted to SN status or provided a SN to communicate with. If communicating with an SN, that SN decides whether the node is close enough to become a member of its neighborhood, these nodes are also referred to as interior nodes. Those nodes outside of the neighborhood are referred to as exterior nodes. The eligibility is based on whether or not that node is a distance of r or less from the SN. In order for the outside nodes to influence SN placement they must communicate with an interior node. These interior nodes are called neighborhood representatives and communicate on the exterior node's behalf. These neighborhood representatives are the interior nodes that are closest to the exterior node.

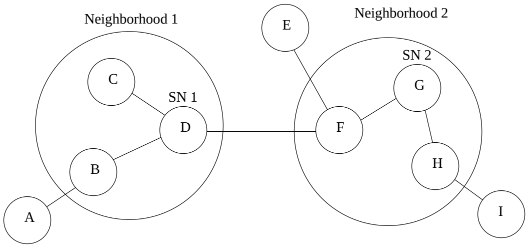


Figure 4.1: Figure displays two communities with SNs located at positions C and H.

Figure 4.1 depicts two neighborhoods of nodes with SNs placed at locations D and G. The exterior nodes are A, E, and I and the neighborhood representatives are B, C, F, and G. Once all nodes are established as interior or exterior and all exterior nodes have found their neighborhood representatives, SPOT tries to improve each SN's location. First the exterior nodes send service usage information (called demand) to their neighborhood representatives. The neighborhood representatives aggregate that demand with their own and send it to the current SN. All other interior nodes also send their demand to the SN. The SN then sends all interior nodes a list of all other interior nodes. Each interior node discovers the distance to all other interior nodes. This information is also sent to the current SN. The SN creates a distance matrix of all interior nodes and a demand vector for service requirements. The SN combines all of the route data along with the cost data to determine if the current location of the SN is in the optimal position in the local neighborhood. This is accomplished by computing the k-median problem using Integer Linear Programming (ILP). The ILP solver finds the best location in the local community using Equation 3.1 in Chapter 3. If the cost of the new SN placement is better than the current cost, the SN is relocated. This process continues until no further improvement is possible.

4.2.1 Nodes dynamically joining the network

Due to the dynamic nature of the SN placement problem, SPOT supports an arbitrarily large network of nodes with an initial ratio of nodes to SNs. Initially, when the first node joins the network it becomes the SN. All nodes after that are assigned to this SN. As the ratio of node to SN is reached a new SN is created and subsequent nodes are assigned to it. The ratios can vary depending of the type of service the SNs provide with values from 10-1 to 1000-1 nodes to SNs. Due to this difference, the original design of the r-ball problem does not translate exactly to r-mod and r-SPOT. In the r-ball problem, the total number of nodes is known in advance and a fixed partition is set. Also, since all SNs are known in advance, it is easy for a node to evaluate the distance to each one and connect to the closest. In SPOT this is not the case and the initial SN connection may not be the ideal choice. Improvements to this are mentioned later in the chapter.

4.2.2 Merging Neighborhoods

In addition to individual neighborhoods relocating SNs, two neighborhoods can merge when close enough to each other (within a distance of r). Merge neighborhoods allows SNs to migrate outside of initially selected set of nodes assigned to it when more than one SN is specified. In order for this process to occur, one SN is designated as the master SN and all other SNs become slave SNs. The master SN collects the route and cost information from all of its connected nodes as well as the costs and route information from the slave SNs. The master SN then executes the k-median ILP again to place the k master and slave SNs in the expanded neighborhood. If the solution set of new SNs is within the r distance of each other, the large neighborhood stays intact, otherwise it separates. If more than one SN exists in the newly created neighborhood, a master SN is elected as before.

4.2.3 SPOT Metrics

4.2.4 Distance

Several different constraints influence the ILP solver in deciding a good SN location. As mentioned before, the first constraint considered is distance. A few distance metrics are available for the SN to use in determining placement. The first two metrics are very simple and use ICMP ping messages, this distance can be measured through round trip time (RTT) or a packet hop count (using the time to live (TTL) field). This provides a simple although sometimes inaccurate metric for determining distance. In order to deploy more advanced techniques, we interface with the Harvard Pyxida coordinate system [58]. Because SPOT takes supports metrics such as RTT or the Pyxida coordinate system it is important to note the units of measure involved. Unlike the hop count, where values of 1, 2, or 3 seem appropriate, the RTT reports values in terms of milliseconds which range from tenths to thousandths. This depends on the distance traveled, number of hops along the path, and other characteristics of the network. In order to simplify the process, SPOT treats the RTT in milliseconds the same as a hop count. Therefore, nodes deal with neighbors that are hundreds of *units* away from other nodes. This does not become a large problem though due to SPOT's mechanism to dynamically scale the neighborhood when needed. This technique also works in a similar fashion with the Pyxida coordinate system. Despite the dynamic behavior though, larger default neighborhoods are still evaluated and shown an ability to reduce SPOT's SN placement costs.

4.2.5 Demand

Another important constraint in the k-median problem is node demand. Assuming homogeneous SN utilization from each node reduces the effectiveness of the placement solution. A simple demand strategy is considered; however more complex strategies could be implemented. The strategy deployed is a simple static demand for all nodes. More advanced demand strategies could involve the SN maintaining statistics for all connected nodes in its local neighborhood or local neighbor nodes predicting their utilization and sending that information to the SN.

4.2.6 CPU and Bandwidth Utilization

To consider real world systems with multiple users and congested links, SPOT allows for additional ILP constraints. In addition to distance and demand, CPU utilization and available network utilization are examples of additional constraints that can serve as inputs to the ILP configuration. For example, when nodes send demand information, they might also include the current load on their system. The bandwidth utilization or bottleneck of their link can also be determined with currently available Planetlab tools [35]. Integrating these tools with SPOT is left as future work.

4.2.7 Software Implementation

SPOT is written in 12000 lines of multithreaded Java code. Each node initializes by listening on a well-known TCP socket and forks a thread for each incoming command. The ILP software uses for the local k-medians problem is the GNU Linear Programming Kit (GLPK) [24]. This open source solver performs provides reasonable performance compared to other solvers [75].

4.3 Evaluation of r-mod

At the core of SPOT is the placement algorithm to determine the location of SNs. SPOT currently supports two different algorithms, r-mod and r-SPOT. The r-mod algorithm was developed first and is an extension of the related work [34]. An evaluation of it is now provided, along with improvements made to it to create the r-SPOT algorithm.

The initial target testbed for performing experiments is Emulab. This testbed provides a reasonable balance between simulations and full scale Internet deployment. Previous works demonstrated similar placement algorithms in simulation environments, therefore in order to move the body of research forward an emphasis on increasing the realism of the experiments is appropriate. The initial goal was to first create a distributed system that would place the SN functionality on nodes in a network. Using the r-mod algorithm such a system was realized.

We are now interested in the how well the r-mod algorithm performs in finding low cost SN placement solutions. The Emulab testbed [80] was utilized to perform these emulation experiments. The first metric of interest is the total cost metric from Equation 3.1 in the previous chapter. This calculates the sum of the products of each node's distance and demand to its chosen SN.

4.3.1 Experimentation Details for r-mod

The order in which nodes connect to the bootstrap server can influence the overall placement score due to SNs being located in local minimums. Therefore 100 experiments were run for each topology with random nodes starting as the initial SNs. The initial set of SNs is determined by the order in which the nodes connect to the bootstrap server. To randomize this initialization a sleep function is placed before the nodes initialize. This sleep function returns a random *sleepTimer* value between 1 and n, where n is the size of the network. The sleepTimer value is the number of seconds to wait before connecting to the bootstrap server. In order to vary the sleepTimer values for each experiment, a different initial seed to the sleep function is provided.

The network topologies consisted of hierarchical networks of size 100, 200, 300, and 400 as shown in Figure 4.2. The Emulab physical nodes were of the type pc3000. The pc3000 are 3 GHz Pentium 4 CPUs with 2 GBytes of RAM. There are 160 nodes on Emulab of this type. In order to create larger sized networks, virtualization is used with Emulab. The virtual machine consisted of FreeBSD jails with an assignment of 10 virtual machines per physical node. This allowed for larger experiments (greater than 160 nodes) while not over-utilizing a single physical machine. The radius value (r) was set to 2 in all of the experiments, consistent with related work [34]. The experiments were evaluated with a k value (number of SNs) of 1 and 3. The SPOT software was loaded on each node and a

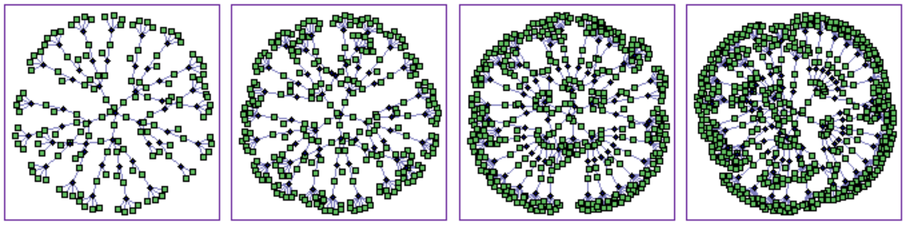


Figure 4.2: Figure displays the various topologies created to deploy the SPOT architecture. Shown are the 100, 200, 300, and 400 node topologies.

special bootstrap node was also created running the bootstrap software. Each node ran a script which would execute the SPOT Java application and connect to the bootstrap node.

The results of these experiments are shown in Figures 4.3 and 4.4, with the number of SNs set to k=1 and k=3, respectively. The results are the average scores out of the 100 executions using different nodes as the initial SNs. Also included in the graphs are the optimal and worst case results for each the topology. These optimal and worst case results were obtained using global knowledge of the topology and solved with the GLPK software using a minimization or maximization constraint.

From Figure 4.3 it is clear the r-mod performs very close to the optimal placement score with a single SN. These results are impressive with the r-mod obtaining placement costs that are no more than 10% greater than optimal placement. In the k=3 case, however, Figure 4.4 shows that the r-mod is placing SNs with an average cost of 60% more than optimal.

4.4 Improving the $r-$mod algorithm

From these preliminary experiments it is evident that some opportunities for improvement exist. This section describes those improvements and presents the new algorithm r-SPOT.

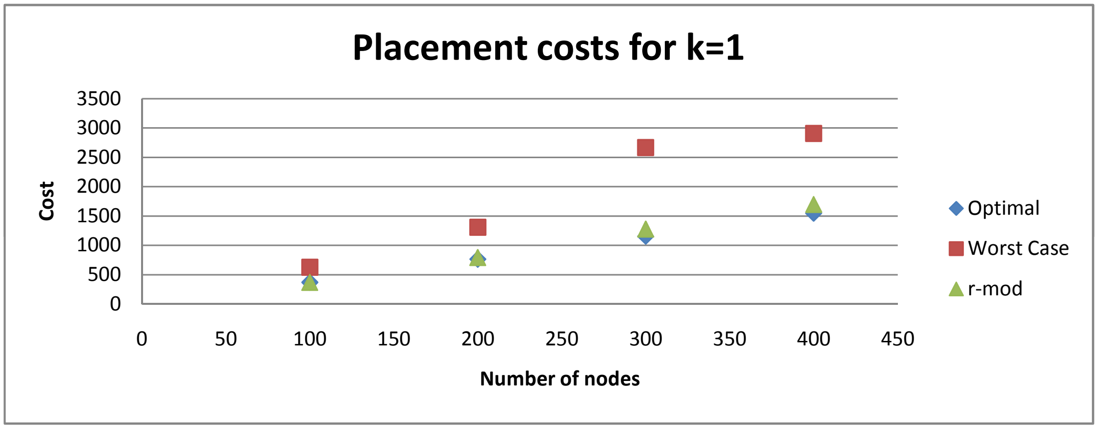


Figure 4.3: Results of placement for one SN with various sized network topologies.

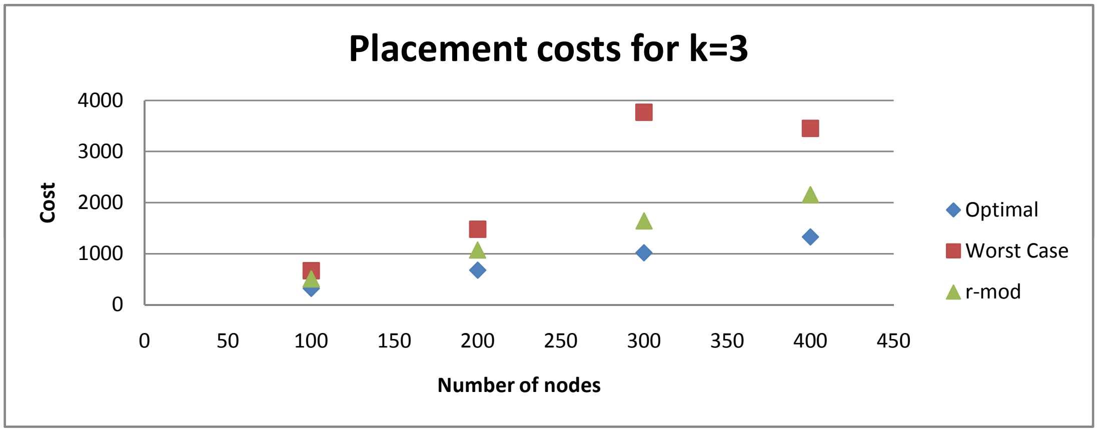


Figure 4.4: Results of placement for three SNs with various sized network topologies.

4.4.1 Multiple Iterations with Informed Placement

The first improvement made is to insert an outer for loop to the r-mod algorithm. This allows for multiple iterations of r-mod with the output of the previous iteration used as the initial set of SNs for the next iteration. This helps reduces the overall importance of poorly chosen initial SNs. In the first iteration of SPOT, nodes may not initially connect to the best SN, due to the order of nodes joining and assigning SNs. By reusing the placement information and not randomly selecting the SNs an improvement in placement costs can be obtained. In the next iteration, each node connects to the closest SN available. Therefore, we are restoring some of the properties of the original r-ball placement algorithm in which nodes connected to the closest SN. The choice though, comes after one iteration of the algorithm and provides an improved placement of SNs for the second iteration of the algorithm.

4.4.2 Dynamically Expanding Neighborhood

In r-mod it is possible for an initial SN to be selected that is more than distance r from any other node that is willing to become an SN. This is not ideal as it does not permit the SN to relocate if necessary. The previous work assumed a fixed value r for the radius of nodes eligible to join the neighborhood. We propose a dynamically sized neighborhood in order to deal with poorly selected initial SNs and to accommodate topologies where nodes are more than a distance of r away from each other. This increases the size of r as needed until at least one interior node is found. This strategy improves the mobility of SNs within a neighborhood and promotes better placement scores by increasing the information available within a neighborhood and reducing the chances of a stranded initial SN.

4.4.3 Preventing Loops in SN Placement

Currently, r-mod compares the cost of placing an SN at a new location against the costs of the current SN location. This cost reflects distances and aggregated demands of those nodes in the neighborhood and does not consider the distance costs of nodes outside the neighborhood. An SN is relocated if the costs of the new SN is less than the current cost. A

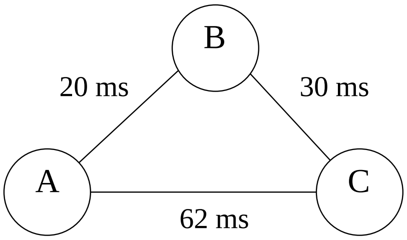


Figure 4.5: An example of a triangle inequality, where the distance from node A to node C (62 ms) is greater than the distance from node A to node B (20 ms) plus the distance from node B to node C (30 ms).

problem can arise if the metric to compute distance creates a Triangle Inequality Violation (TIV). Consider Figure 4.5 from [42], here nodes A, B, and C are reporting ping times in milliseconds to all other nodes. If the distance from node A to node C is longer than the distance from node A to node B plus node B to node C, then that is a violation of the Triangle Inequality. Savage et al. [65] have demonstrated experiments on the Internet where approximately 20% of the nodes exhibit the Triangle Inequality Violation. A similar distance metric problem can occur on Emulab when computing TTLs from nodes on complex topologies. When either of these occurs, r-mod may end up in a state of flip flopping the SN location between two SNs. In order to correct this a *true neighborhood cost*, one for all nodes using that SN, is calculated. The true neighborhood cost is defined in Section 3.7. This technique eliminates the problem from trying to compare costs of networks with different configurations of interior and exterior nodes. Here, an SN may improve its location only if the true neighborhood cost is lower. This incurs some additional communication overhead but prevents cycles of SN placement.

4.4.4 Description of r-SPOT

A formal algorithm is now provided for r-SPOT. This builds on the previous algorithms from Chapter 3 and is presented below. Both the r-SPOT placement algorithm and a new DiscoverLocalTopologies method are presented in Algorithms 4 and 5 respectively.

The changes necessary to both 4 and 5 are now described.

- Added outer for loop to run multiple iterations of the placement algorithm.

Algorithm 4 Improved placement of Supernodes in a P2P Network (r-SPOT)

```
1:  for $iter = 1$ to $PlacementIter$ do
2:      $ConnectToBootstrapNode()$
3:      while $V_k \neq \emptyset$ do
4:          $v_i \in V_k$
5:          $DiscoverLocalTopology(v_i)$
6:          $G_J \leftarrow TestSupernodeNeighborMerge(v_i)$
7:          $V_k^{(m)} \leftarrow OptimizeSupernodeGroup(G_J)$
8:          $V_k^{(m)} \leftarrow EvaluateSupernodeGroup(V_k^{(m)})$
9:          $V_k \leftarrow V_k \setminus (J \bigcap V_k^{-})$ {Remove Processed Supernodes}
10:         $UpdateBootstrapNode(V_k)$
11:         if $V_k = \emptyset$ then
12:             if $V_k^{(m)} \neq V_k^{-}$ then
13:                 $V_k \leftarrow V_k^{(m)}, V_k^{-} \leftarrow V_k^{(m)}$
14:             end if
15:         end if
16:     end while
17: end for
```

  - Allows for improved placement by using the output of the first iteration as the input of the next iteration.
  - Described in Section 4.4.1.
  - Found in lines 1 and 17 of Algorithm 4.

- Introduced dynamically expanding neighborhoods.
  - Enables neighborhoods to grow when an interior node is needed as a neighborhood representative.
  - Described in Section 4.4.2.

- Added topology loop detection.
  - Introduced mechanisms to detect loops from Triangle Inequality Violations (TIV).
  - Described in Section 4.4.3.
  - Found in line 8 of Algorithm 4 and Algorithm 6.

Algorithm 5 Discover Local Topology

```
for all $v_i \in W_i^{(m)}$ do
    $DistanceToSN \leftarrow CalculateDistanceToSN(v_i)$
    if $DistanceToSN \leq r$ then
        if $WillingToBecomeSN(v_i)$ then
            $N_i^{(m)} \leftarrow AddNodeToNeighborhood(v_i)$
        else
            $U_i^{(m)} \leftarrow AddNodeToOutsideNeighborhood(v_i)$
        end if
    else
        $U_i^{(m)} \leftarrow AddNodeToOutsideNeighborhood(v_i)$
    end if
end for
if $NeighborhoodSize < 1$ then
    $N_i^{(m)} \leftarrow findClosestNodeToSN(U_i^{(m)})$
end if
for all $v_i \in N_i^{(m)}$ do
    $SendNeighborInformation()$
end for
for all $v_i \in N_i^{(m)}$ do
    $ReceiveRouteTableInformation()$
    $ReceiveDemandInformation()$
end for
$CreateRouteTable()$
$CreateDemandArray()$
```

Algorithm 6 EvaluateSupernodeGroup

```
if $currentCosts \geq oldCosts$ then
    if $totalCost \geq oldTotalCosts$ then
        $V_k^{(m)} \leftarrow V_k^{(-)}$
    end if
end if
```

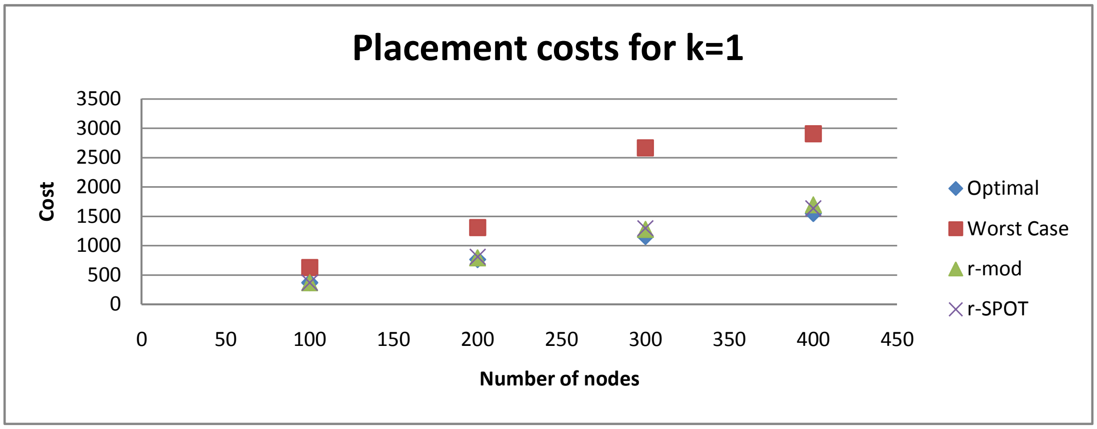


Figure 4.6: Results of the r-mod and r-SPOT placement for one SN with various sized network topologies.

4.5 Evaluating r-SPOT

Experiments with r-SPOT are now presented. The same initial placement experiment, from Section 4.3 is run for r-SPOT. The results for k=1 and k=3 are presented in Figures 4.6 and 4.7. The results for k=1 show r-SPOT and r-mod performing at comparable levels. Both algorithms average placement costs range from 3% to 12% over the optimal. In Figure 4.7 the results for k=3 show improved placement for r-SPOT compared to r-mod. For example, in the 300 node experiment r-SPOT was only 13% over the optimal compared to r-mod with an average placement cost that is 61% higher than optimal.

Next, we evaluate the effects of each individual modification to the r-mod algorithm. Here we are considering the same placement experiment in Emulab focusing only on the 100 node topology with k=3 SNs. In Figure 4.8 a whisker box plot is presented. The whisker-box plot presents the lower quartile (0.25), median, and upper quartile (0.75) values along with the sample minimum and maximum. An interquartile range (IQR) is also calculated which is the upper quartile minus the lower quartile values. Any value which is 1.5 $*$ IQR less than the lower quartile or 1.5 $*$ IQR above the upper quartile is considered an outlier and denoted as a circle. Again the experiments were run 100 times with SPOT implementing variations of the placement algorithms. The options were r-mod, r-mod with the ability to remove loops (r-mod + NoLoop), r-mod plus the expanding neighborhood (r-mod +

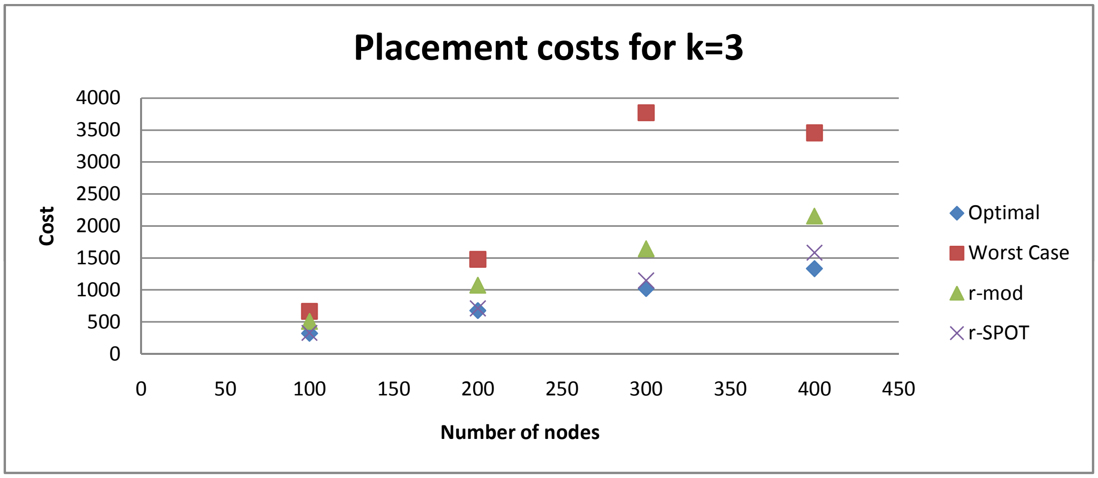


Figure 4.7: Results of placement for three SNs with various sized network topologies.

Expand), r-mod with two executions of the algorithm (r-mod + Iters). Finally, r-SPOT, which is the combination of all three improvements.

From Figure 4.8 the r-mod plus the removal of infinite looping had very little effect on the distribution. This was expected, as it does not improve performance but simply allows SPOT to handle unexpected networking conditions, the average score (not shown) was 511 compared to 510 for the original r-mod. When r-mod allows for expanding neighborhood sizes, a considerable improvement is observed. Here the median placement score improves from 533 to 363, and the average drops to 338. This expanding neighborhood also improves the score of a poorly placed initial SN. Next, more improvements are made when the r-mod algorithm executes twice and uses the first set of SNs as an input for the initial set of SNs. Here the median value is 336 and the average is 347. Finally, when combining all of these together into r-SPOT the average placement cost drops to 334 and the median value is 328. The combination of all three of these improvements allows for reduced costs relative to r-mod.

In addition to the cost of the resulting network topology upon algorithm completion, other metrics of interest are also evaluated for r-SPOT. The total number of iterations to reach a finishing state and total system time necessary before the experiments finished are discussed next.

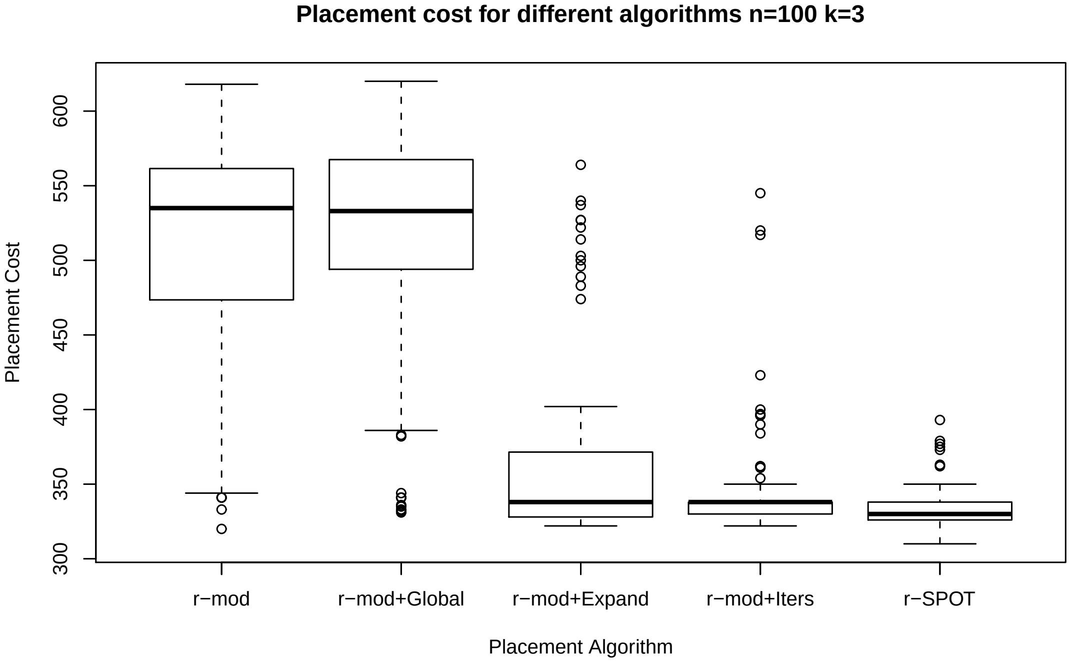


Figure 4.8: Performance of the r-mod algorithm with enabling different improvements along with the r-SPOT algorithm topologies.

The first experiment (Figure 4.9) uses a whisker-box plot again to display the total number of iterations, (lines 3 - 16 in 4) needed for a stable placement. The number of iterations needed increases as the number of SNs to place increase. The minimum number of iterations is the value PlacementIter from line 1 of 4. This graph displays the total number iterations for k values of 1 - 3 in the 100 node graph. Here as the number of SNs increases so too does the number of iterations needed to find locally optimal solutions.

Using the same experimental setup from the last graph, the total system time necessary to locate SNs in a network topology is shown in Figure 4.10. From this graph we can see total system time increase as the number of SNs increase. This is to be expected as increasing the number of SNs to place increases the total amount of work and the time to complete it.

Additional experiments are run to understand some of the tradeoffs between r-SPOT and an optimal placement strategy. In Figure 4.11 the total time to locate three SNs were

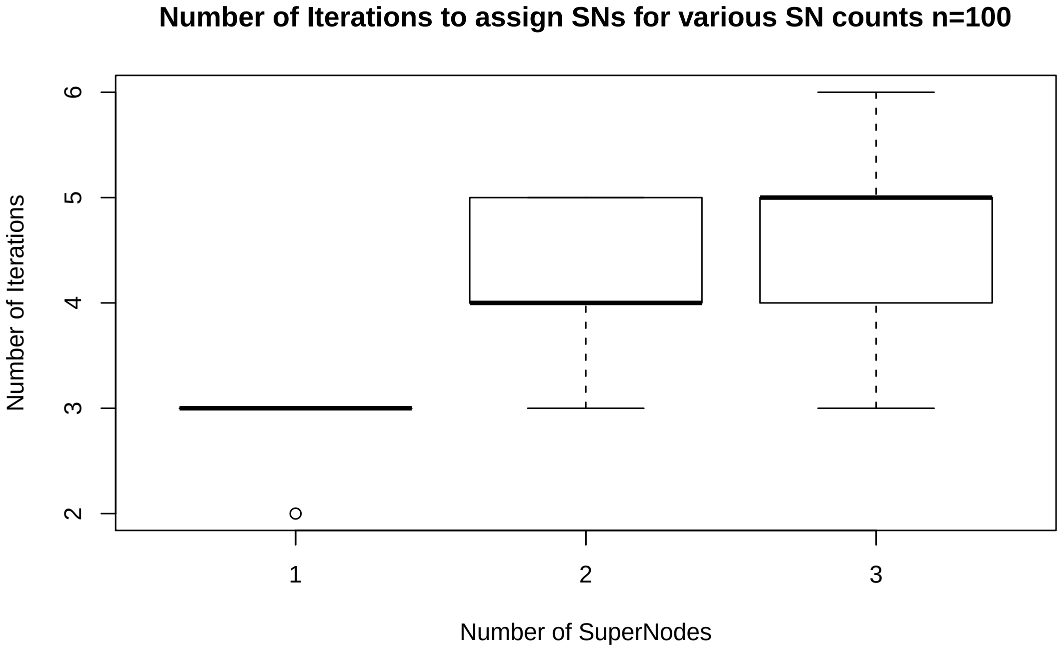


Figure 4.9: Number of iterations for the outer for loop in r-SPOT algorithm for various SN counts.

computed for various topology sizes in comparison to a global solution. The global solution required fully topological information, which is equivalent to increasing the neighborhood size to include all nodes in the network. At the 250 node size it starts to become increasingly more expensive to place nodes as solving the centralized optimal solution is NP-Hard.

The next experiment compares the amount of network traffic generated for a single node (on average) with the r-SPOT solution and an optimal solution which requires global topology information. The network traffic comparisons are shown in Figure 4.12. From the figure, the total amount of network traffic generated per node for the centralized optimal solution is 20 times greater than r-SPOT. This would continue to grow for larger network sizes. This graph demonstrates the scalability of r-SPOT compared to a centralized approach.

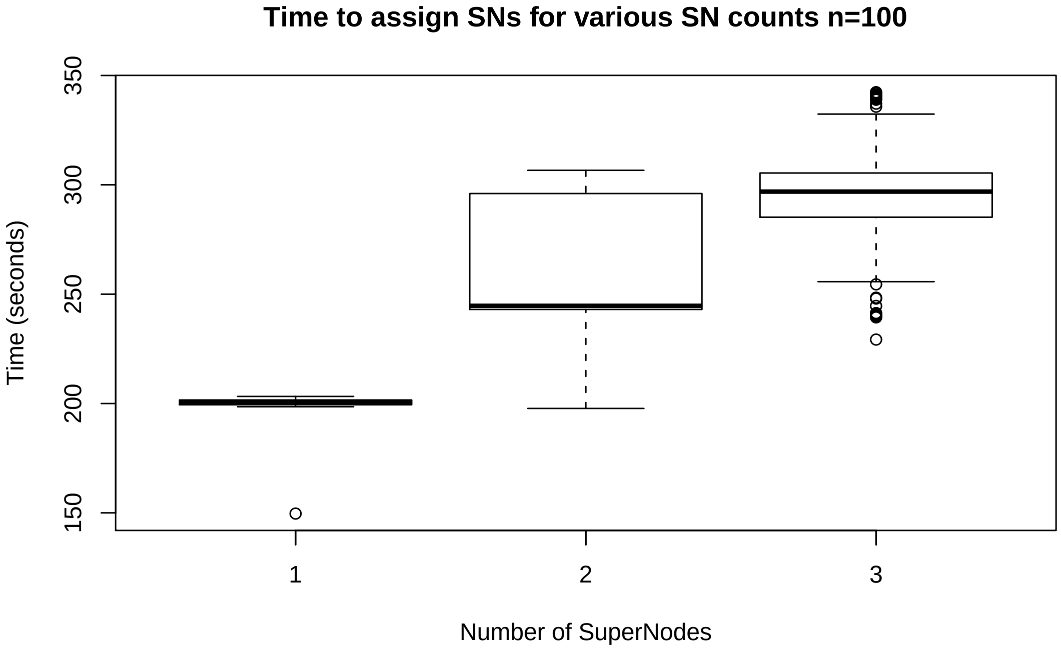


Figure 4.10: Total time to place SNs in r-SPOT algorithm.

4.6 Summary

This chapter has presented the SPOT system, along with the two placement algorithms r-mod and r-SPOT. The evaluation demonstrates the improved placement performance of the r-SPOT algorithm and how both approaches compare to the optimal results. The next chapter explores other environments to evaluate SPOT and assesses the use of an application taking advantage of this informed service placement.

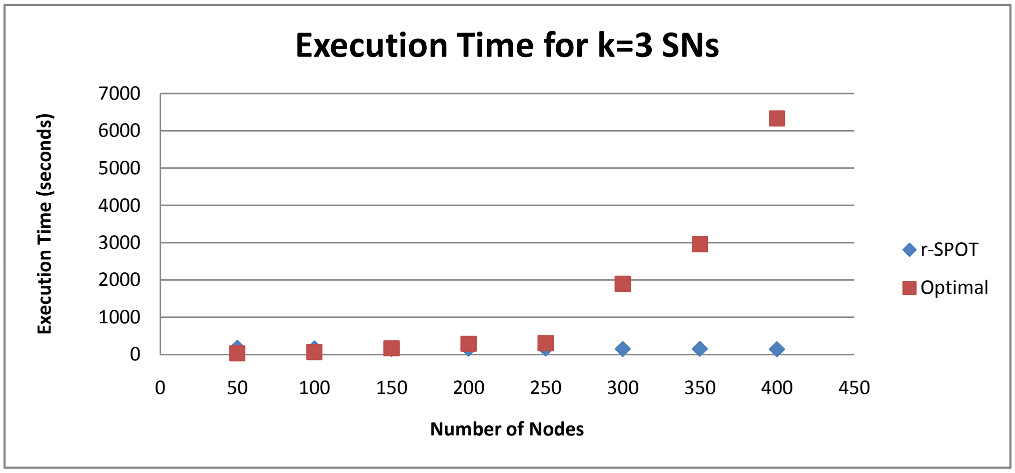


Figure 4.11: Total time to place three SNs in r-SPOT algorithm compared to the optimal placement.

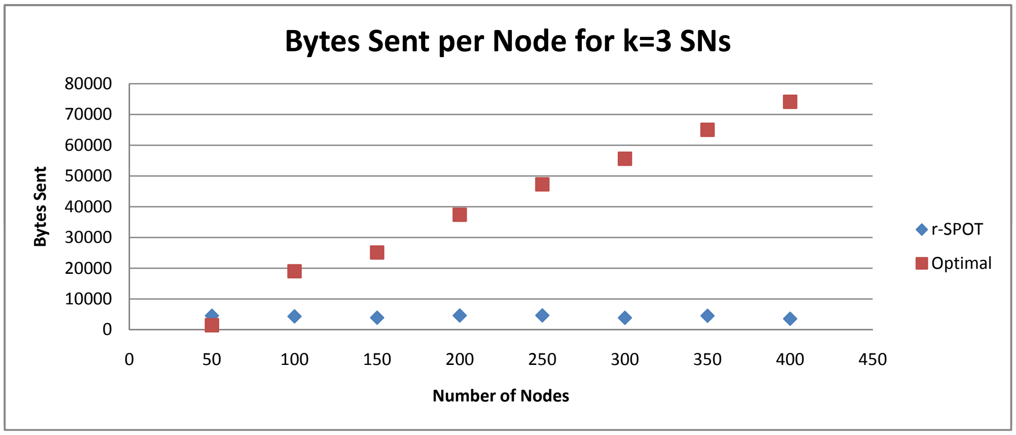


Figure 4.12: Total amount of network traffic sent per node in placing three SNs compared to an optimal solution requiring global knowledge.

Chapter 5

Evaluation of SPOT in Diverse Environments and Applications

This chapter investigates the performance of r-SPOT using the SPOT infrastructure under a variety of environments. They include simulation, emulation, and experimentation. This breadth of study provides a better understanding of the entire system and how it will perform in diverse scenarios.

5.1 Simulation

5.1.1 Introduction

Simulation provides an excellent opportunity to extend the previous results to large and more realistic network topologies. With research testbeds, a node size limit is reached somewhere in the hundreds of nodes. In order to evaluate systems larger than that, simulation is very useful. Also with simulation, the experiments can run on different topologies very easily.

In order to provide simulation with SPOT, a discrete event simulator called SPOTSim was developed. SPOTSim was written in Java and models all of the communication between nodes running SPOT. It also interfaces with the same ILP solver (GLPK) as SPOT.

The simulator was developed after creating SPOT, therefore the true functionality of the working system was captured in the simulation environment. Typically the simulator is developed first and the final implementation ends up behaving somewhat differently due to real world constraints. This is not the case with SPOTSim, which provides a fairly realistic model of SPOT's behavior. A few differences exist between a real execution of SPOT and a SPOTSim simulation. First, SPOTSim does not model the runtime of SPOT. Some very simple estimations are made regarding the amount of time it takes to send messages between nodes. SPOTSim does not account for varying latency between nodes or model the capacity of links to send messages. SPOTSim also does not model when SNs are unable to merge with other SNs due to node timeouts or random networking failures. These events occur in real systems and SPOT has timeout mechanisms built into it to exit gracefully after a period of time of not communicating with any other node. SPOTSim never misses an opportunity to merge with an available SN and the consistency of the placement scores reflect that behavior.

5.1.2 SPOTSim Evaluation

In order to test the validity of the model, experiments were run on Emulab with SPOT and on SPOTSim with the topology deployed on Emulab. The first experiment illustrates the placement scores of both SPOT and SPOTSim for k=1 and 3 in Figures 5.1 and 5.2 respectively.

From these simulation and emulation experiments the simulation results closely match the best emulation results. Differences however are found in some of the more costly emulation placement results. These results are not captured by the simulation model. The higher scores in emulation are due to the missed opportunities for SNs to join with other SNs and create larger neighborhoods. In SPOT the SNs operate within a specific timeout and if other nodes are unable to join or communicate with that SN it eventually reports a stable position and exits. In the simulator, the merge operations occur with perfect knowledge of the other SNs available, therefore all potential neighborhood merges will successfully occur.

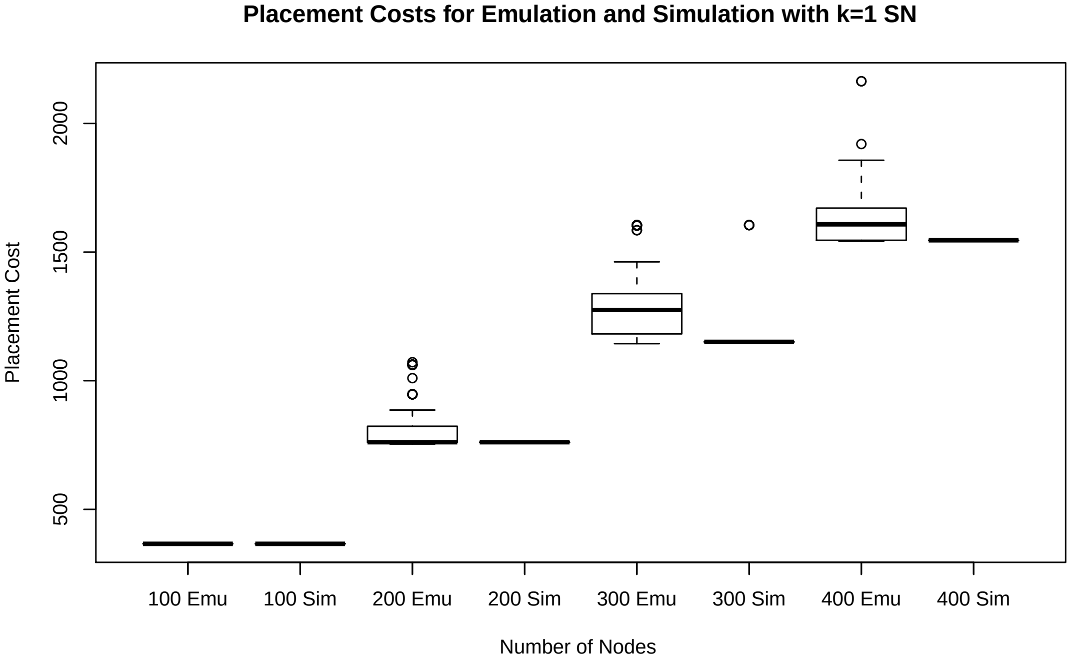


Figure 5.1: Comparison of Emulation and Simulation placement results for various network topologies locating one SN.

Evaluating SPOTSim on an Emulab topology is useful to determine the validity of the model. However, the real strength of the simulator is to experiment on a larger number of nodes and more interesting topologies. To accomplish this a topology generator was used to aid in the design and creation of larger more realistic topologies. The topology generator used is BRITE [46]. BRITE provides an intuitive GUI interface which allows the researcher to specify the number of nodes, average degree, and general structure of the topologies. A large number of more advanced configuration options are available as well. The types of networks include Waxman, BA, BA-2, and GLP and the model supports AS and router level models. Based on previous related work [34] the BA-2 router level topology was selected. A range of sizes were created (500, 1000, and 1500 nodes) using the BRITE's default growth rate parameters. An example topology is displayed in Figure 5.3.

The simulation experiments investigated the placement costs of SPOTSim scaling for larger sized topologies. Using the three topologies created with BRITE, simulations

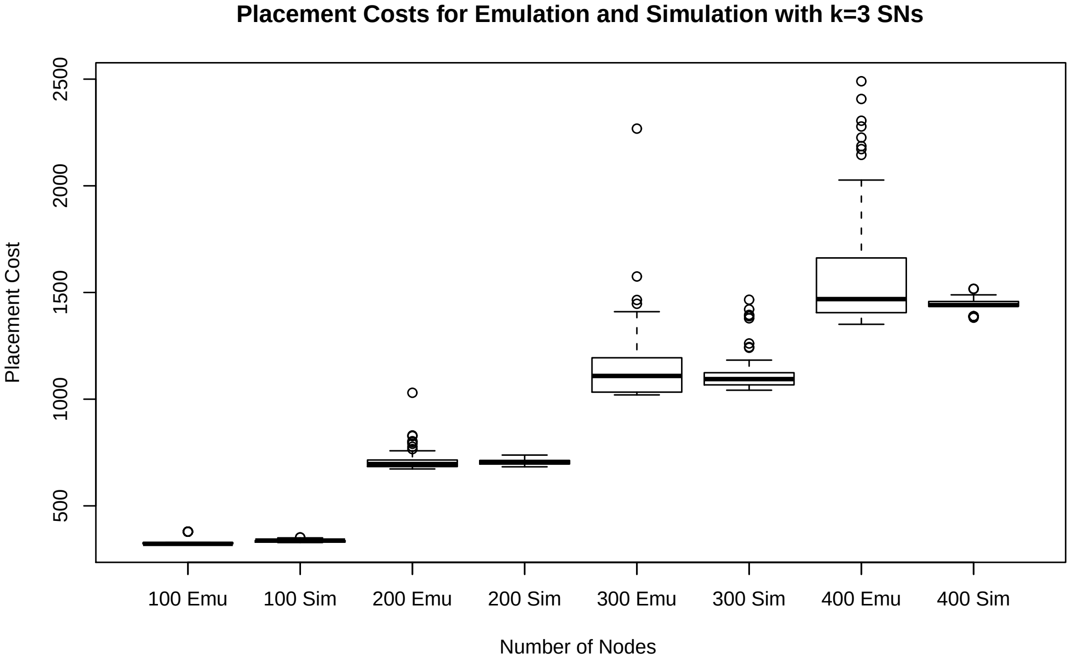


Figure 5.2: Comparison of Emulation and Simulation placement results for various network topologies locating three SNs.

were performed and the average placement cost is now presented. In Figure 5.4 SPOTSim and the optimal results are presented. From the figure, SPOTSim is able to place SNs with a cost of less than twice that of the optimal.

Simulations were also run with various default neighborhood node sizes. Thus far, all simulation and emulation results were executed with a r or neighborhood size of two units, where units are some metric such as network hops. In Figure 5.5 three different values for the default neighborhood size are experimented with placing k=3 SNs in the 500 node router topology. From the figure, increasing the default neighborhood size r reduces the cost of placing SNs. Here the average placement costs are 1045, 954, and 965 for r values of 1, 2, and 3. Therefore setting the default neighborhood size (r) to two provides a 7% reduction in cost and setting r to three reduces the costs by an additional 1%.

Finally, experiments were run to determine the effect of the outer loop added to r-SPOT. In Figure 5.6 one, two, and three iterations of SPOTSim were run before accepting

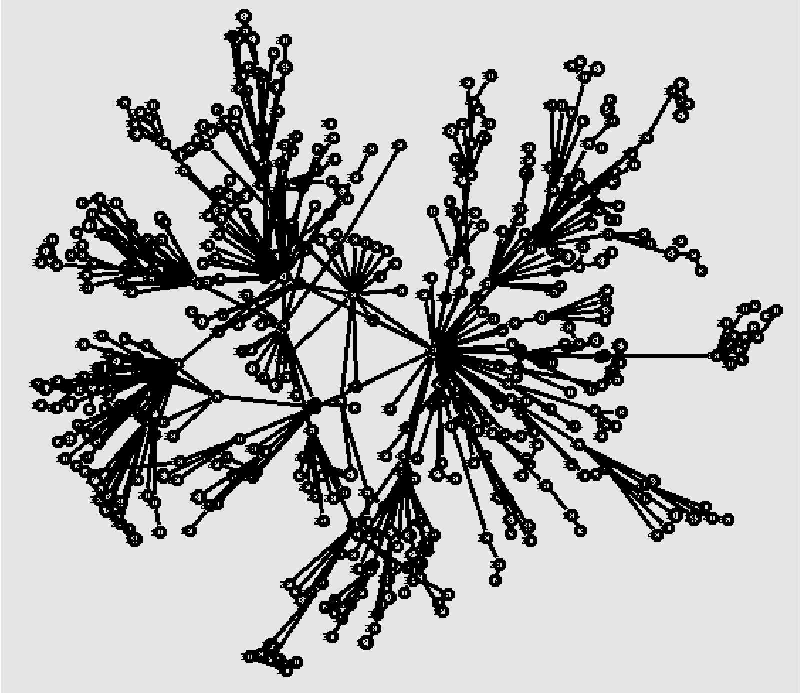


Figure 5.3: 500 node router topology generated using BRITE.

the placement of k=3 SNs in the 500 node router topology. As with increasing the default neighborhood size, increasing the number of iterations of the outer loop (PlacementIter) leads to better SN placement and reduced cost. The average placement costs for PlacementIter values of 1, 2, and 3 where 1046, 977, and 951 respectively. Therefore, adding one additional iteration of the outer loop reduces average placements cost by approximately 7%, with two iterations the costs are reduced by 9%.

5.2 Planetlab

5.2.1 Introduction

The next type of evaluation is via *experimentation*. Here, we are deploying the SPOT system on the Planetlab testbed [56]. This testbed consists of over 1000 nodes distributed around the world. The nodes are hosted at universities and corporations, with various hardware and networking configurations. Each researcher is allowed access to a *slice* of

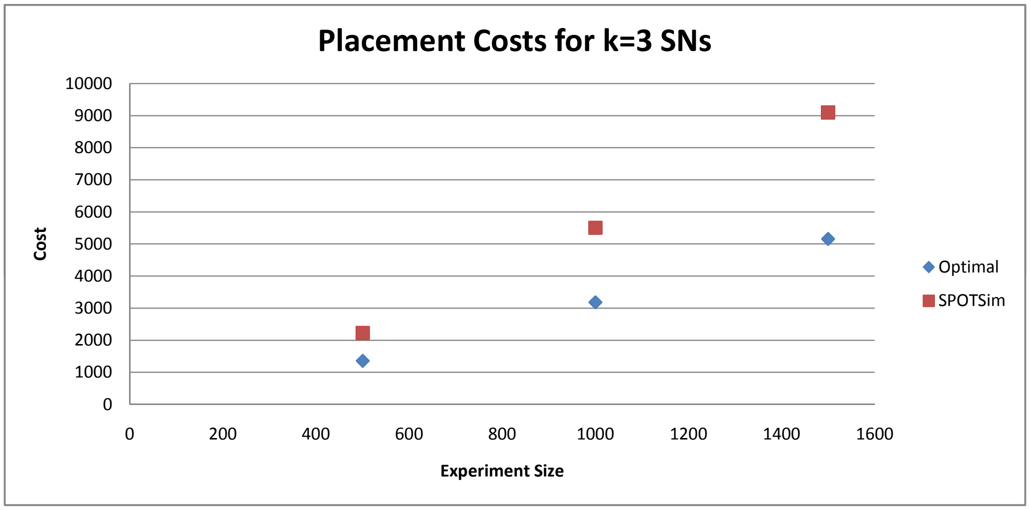


Figure 5.4: Placement costs for SPOTSim simulations compared against optimal placement costs.

every single node in the network. This is very useful with regards to the diversity of systems and networking environments. It is common to encounter heavily loaded CPUs and over-utilized low bandwidth network links. This type of environment increases the realism and quality of experimentation greatly.

The test setup involved deploying SPOT on 50 nodes in the Planetlab environment. The size of the experiment may appear small, however due to the unpredictable nature of Planetlab, hundreds of nodes are unavailable at any given time. Also, when running many different experiments using the same set of nodes is crucial given the large number of variables present. The idea of network cost is also unreliable if the set of nodes ever changes during the experimentation. For these reasons, a size of 50 nodes was selected. This number is also common with other researches working with distributed systems [70].

5.2.2 Planetlab Evaluation

The initial experiments on Planetlab involved collecting statistics about the nodes. In Figures 5.7 and 5.8 the cumulative distribution functions are provided for the number of hops necessary to reach all of the nodes along with the round trip time (RTT) for each node.

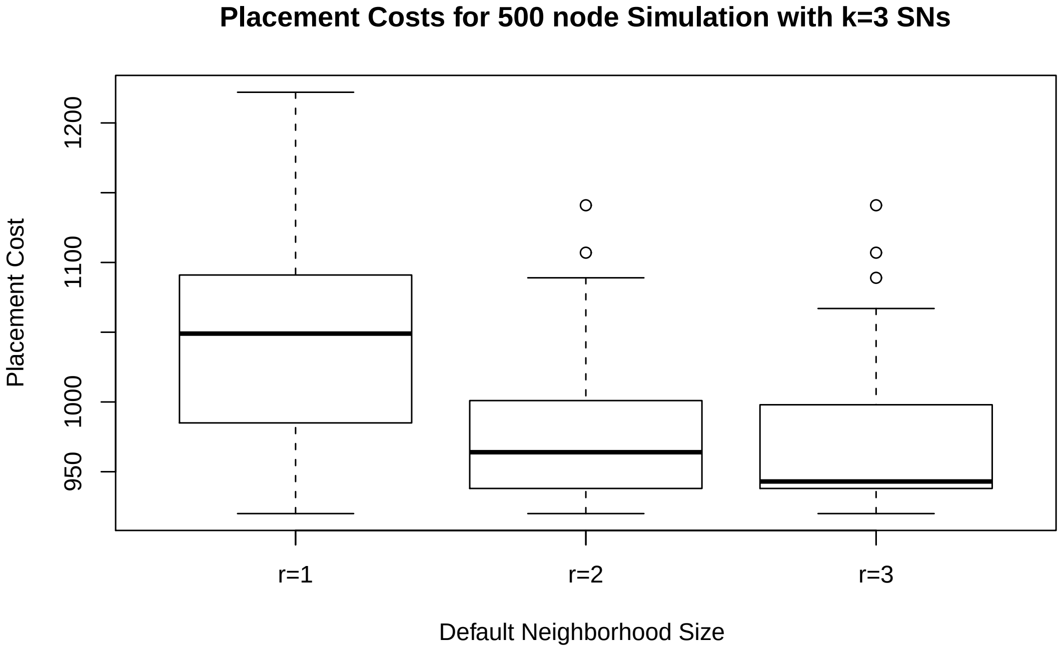


Figure 5.5: Whisker-box plot of placement costs for SPOTSim on a 500 node topology with varying initial neighborhood sizes.

Next we deployed the SPOT system on all 50 nodes and selected an arbitrary node (not one of the 50) to operate as the bootstrap server. Once the software was deployed, 50 experiments were run with k values of 1, 2, and 3 SNs. The results of the experiments are shown in Figure 5.9. These results use the same distance metric as Emulab, the hop count metric. From the results, the total cost decreases as the number of SNs increase. This is to be expected as adding more SNs should decrease the total network cost.

In these experiments we measured the total time to locate SNs, the number of iterations to find SNs, and finally the total amount of network traffic generated from all of the nodes in the experiment. This are illustrated in Figures 5.10, 5.11, 5.12 respectively.

In general, all three of the whisker-box plots demonstrate similar trends. As the number of SNs increase, so too does the number of iterations, total time, and network traffic necessary to assign SNs.

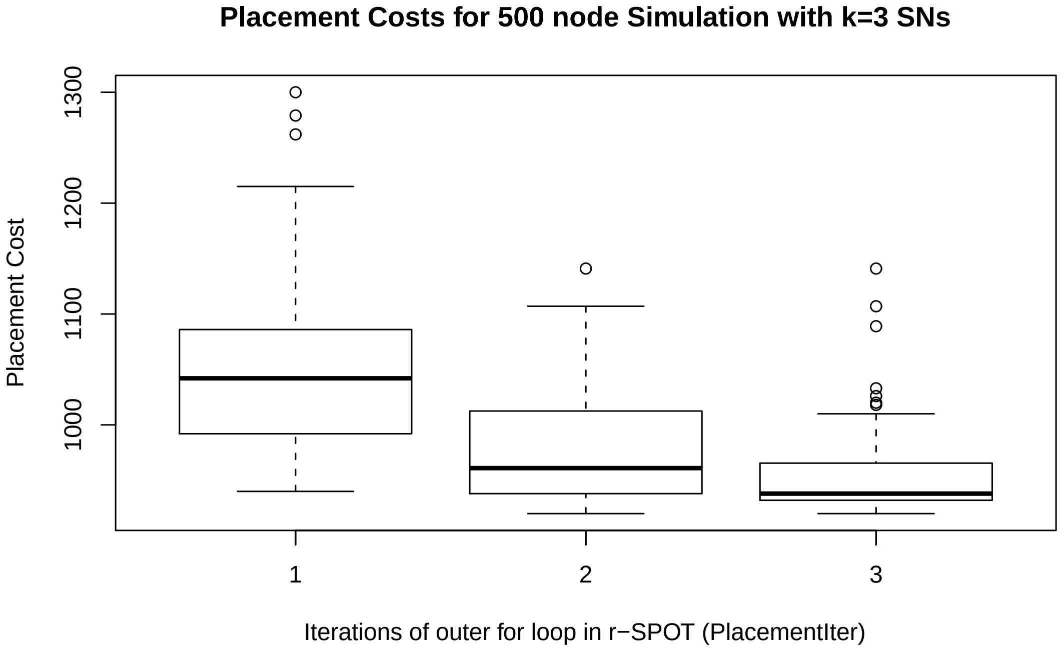


Figure 5.6: Whisker-box plot of placement costs for SPOTSim for various iterations of the outer-loop (PlacementIter).

5.3 Game Servers

5.3.1 Introduction

All of the previous results have dealt with SPOT, its ability to place SNs throughout a network and measurements associated with it. Ultimately, the improvements that SPOT provides for applications utilizing its service are important. In order to evaluate that, we turn our focus to online video games, namely multi-player first-person shooters. Multiplayer first person shooters (such as Quake III Arena and Half-Life [3]) are very sensitive to the latency from the client to the game server. Typically anywhere from 16 - 64 people connect to a single server or host. This host sends game updates to all players connected to the server. If the client has a RTT to the server greater than 180 - 200 ms, it can greatly reduce the quality of the experience as well as fairness in the game itself [2]. Therefore when creating a multiplayer game, it is important to choose the game server carefully.

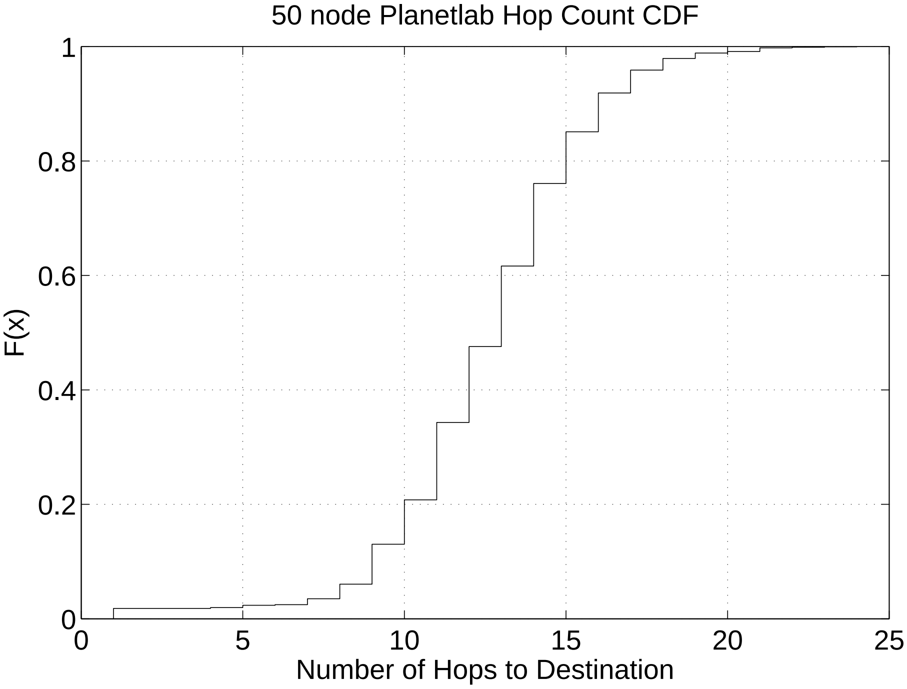


Figure 5.7: CDF of the number of hops necessary for nodes to reach each other in the 50 node Planetlab experiments.

5.3.2 Planetlab Evaluation

In order to evaluate the effects of server selection, the 50 node setup in Planetlab was studied. A map indicated the location of the 50 nodes is illustrated in Figure 5.13. Using these 50 nodes, the ping data collected earlier was used to evaluate the RTTs letting each node become the *server* in a online game. Therefore, we are interested in the RTT from each client to that server. From this data 7 of the 50 servers or 14% of the nodes would be unable to satisfy the requirement that every node maintain a RTT under 180 ms. This demonstrates the importance of carefully selecting a SN. The RTTs are shown in Figure 5.14, after removing the four largest outliers a closer view is shown in Figure 5.15.

Next SPOT was run across all 50 nodes with k=1 and it selected node 8 as the SN, the average RTT is 60 ms to the SN and the maximum RTT is 139 ms as shown in Figure 5.16. The optimal value is selecting node 6 as the SN with an average RTT of 40 ms and

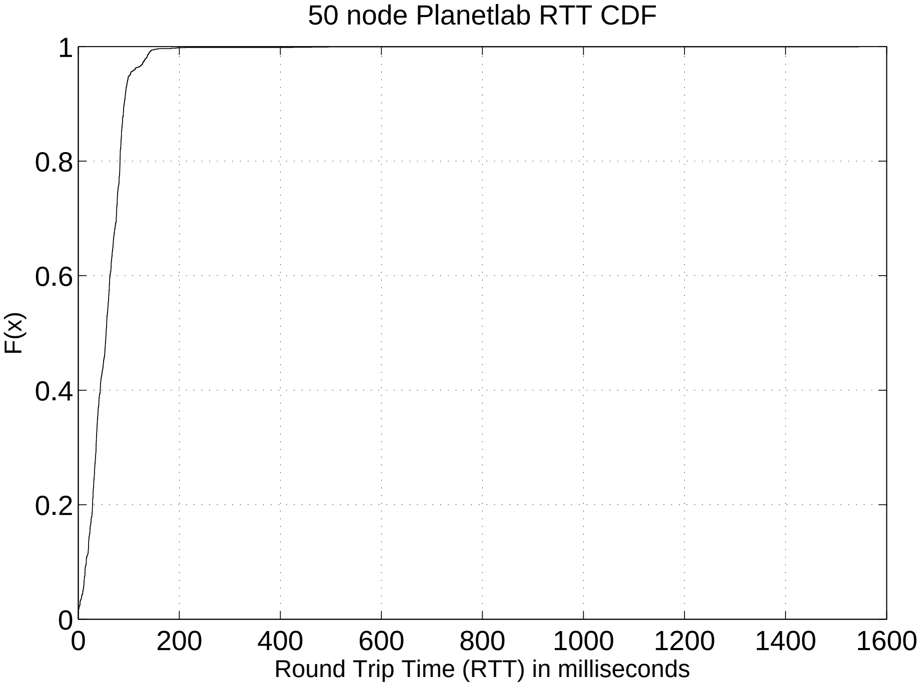


Figure 5.8: CDF of round trip time (RTT) in milliseconds for nodes to reach each other in the 50 node Planetlab experiments.

a maximum RTT of 118 ms. The worst case selection is node 42, with an average RTT of 127 ms, a worst case RTT of 1545 ms and three nodes over the 180 ms threshold.

A larger topology of 263 Planetlab nodes was also evaluated. Figure 5.17 shows a map of the node locations around the world. The RTTs were collected to and from all nodes in the evaluation. The average RTT time with each node serving as the candidate SN is illustrated in a whisker-box plot in Figure 5.18. From the figure, the nodes planetlab1 and planetlab2 at citadel.edu experience very high average RTT delays around 1.4 seconds. Next we evaluated the individual RTT from each node to the candidate SN. Figure 5.19 depicts the maximum number of players that can join the candidate SN server. A player can join the server if the RTT to that server is less than 180 ms. From the figure, 107 potential SNs can support 200 or more players in a single game (the largest is 236), also 116 potential SNs support 100 or more players. The least number of players came from the pair of nodes located in Uruguay, supporting 4 and 5 players each. Finally, a comparison is provided showing the relation between solving the k-medians problem and finding an SN

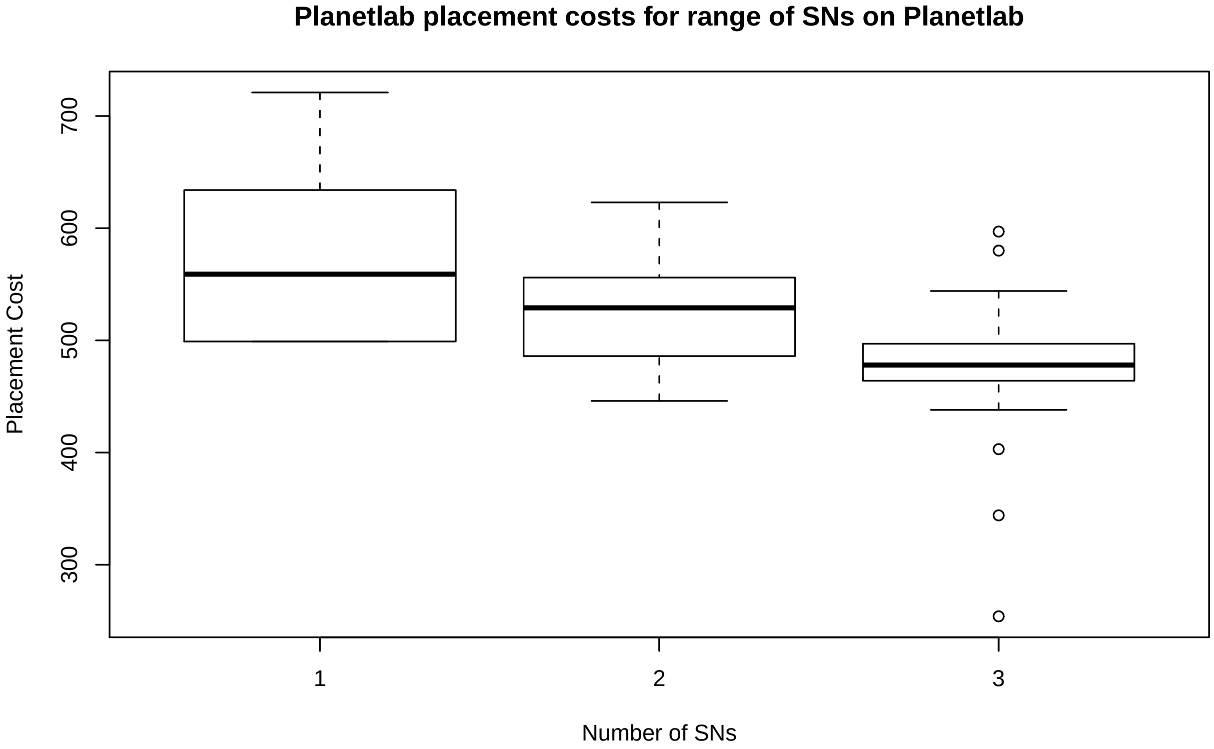


Figure 5.9: 50 node Planetlab r-SPOT experiment using the TTL distance metric illustrating the placement costs for k=1,2, and 3 SNs.

that supports the most number of players. In Figure 5.20, a bar graph represents the total number of players that could connect to a single SN in the best and worst case. Also shown are the results of the k-medians optimal solution and the SPOTSim solution with respect to the number of players each SNs supports. From the results, the maximum number of players in a single game with the best SN placement is 236 players, while the ILP solver and SPOTSim, selected nodes supporting 227 and 226 players respectively. This helps to demonstrate the ability of the k-median algorithm to determine SN locations for a first person shooter.

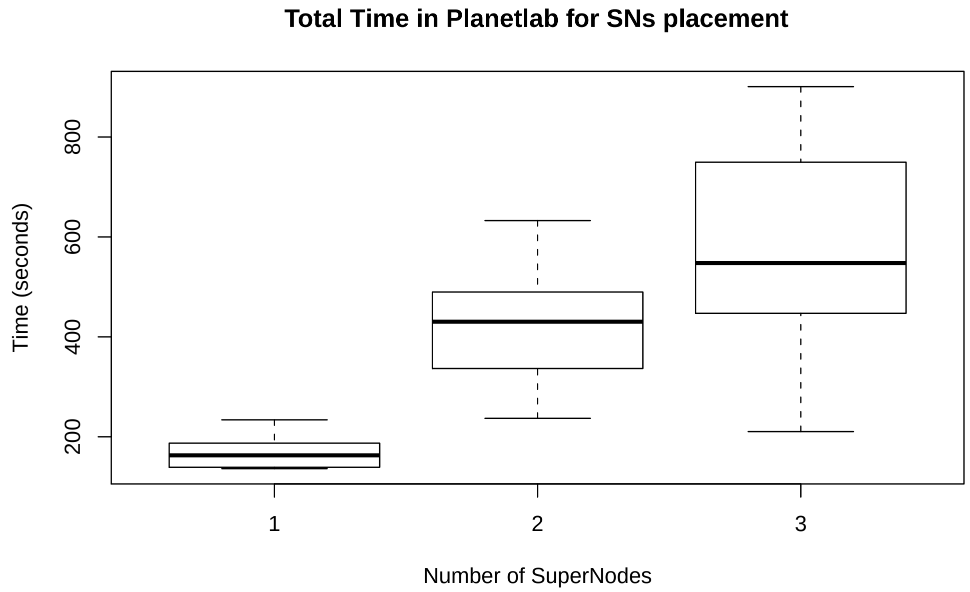


Figure 5.10: 50 node Planetlab r-SPOT experiment illustrating the total time to place k=1, 2, and 3 SNs.

5.3.3 Updating the SN as the topology changes

We are now interested in the effects of a dynamic game state where players join the game after some period of time. Here, we are interested in whether it is necessary to recalculate the location of the SN after a number of new players join. Consider a 40 player node topology taken from the original set of 50 nodes in the previous experiment. We use r-SPOT to determine a location of the SN (node 7) and measure the RTT from all the players to that SN. Now suppose 10 more players join the game and the SN is not re-evaluated with all 50 players. With node 7 still serving as the SN, one of the new nodes joining is unable to play the game due to a large RTT (1545 ms to node 7). However, if the SN is re-evaluated and moved to node 8, all players are able to participate. The results of this experiment are shown in Figure 5.21 with a whisker-box plot of all RTTs. From the figure, when node A is the SN in the 40 node experiment the average RTT is 44 ms. Once the 10 additional nodes join, the average RTT jumps to 81 ms with the outlier node experiencing a large delay to SN 7. When the topology is re-evaluated the SN moves to node B and the average RTT

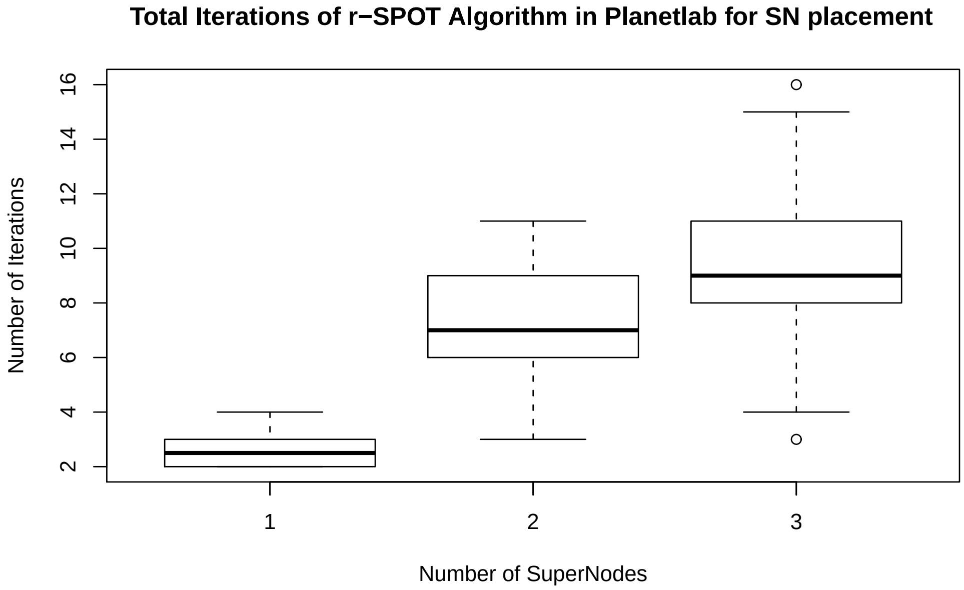


Figure 5.11: 50 node Planetlab r-SPOT experiment illustrating the number of iterations place k=1, 2, and 3 SNs.

drops to 60 ms. This illustrates the importance of re-evaluating the SN assignment in order to maximize the number of players in the game.

5.4 Summary

This chapter evaluated SPOT in a range of environments. The SPOTSim simulator was introduced and compared against SPOT. This proved to be very helpful in creating large (thousands of nodes) experiments to evaluate the placement of network services. The Planetlab experimentation environment was also utilized to explore SPOT's performance on a diverse set of nodes. Finally, an application of SPOT for selecting game servers for first person shooters was motivated with data from the Planetlab environment.

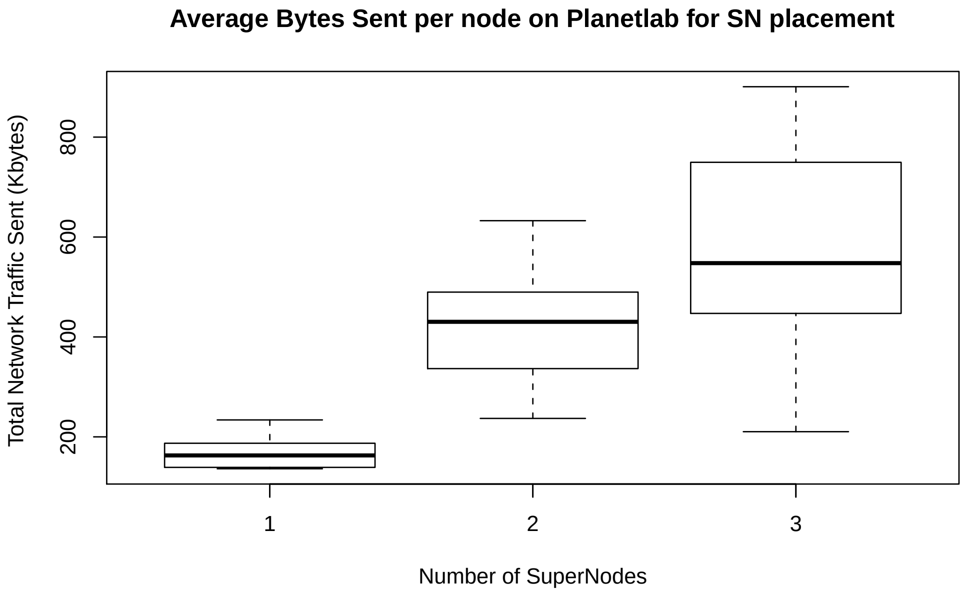


Figure 5.12: 50 node Planetlab r-SPOT experiment illustrating the total traffic sent from all nodes in order to place k=1, 2, and 3 SNs.

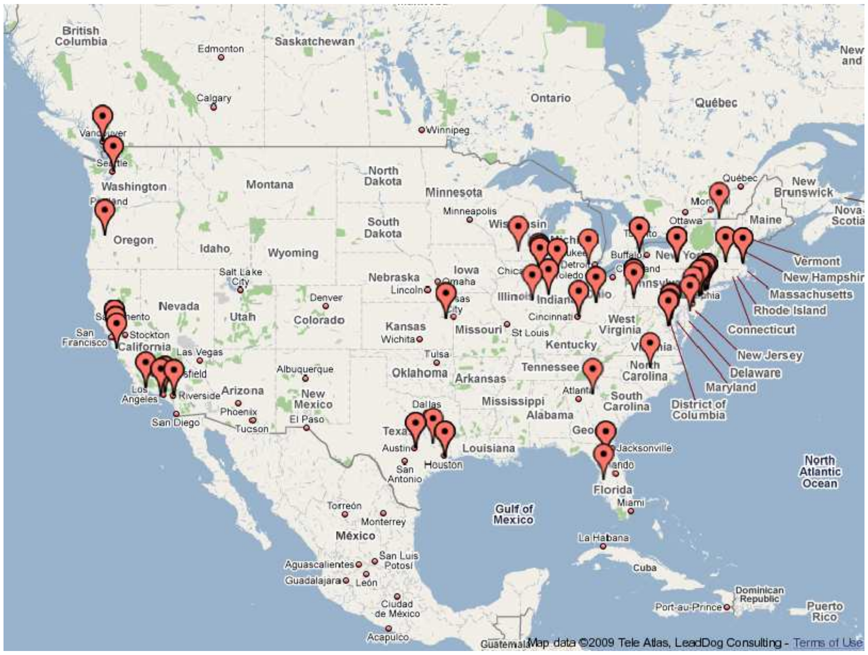


Figure 5.13: Map of the 50 nodes used in the Planetlab experiments.

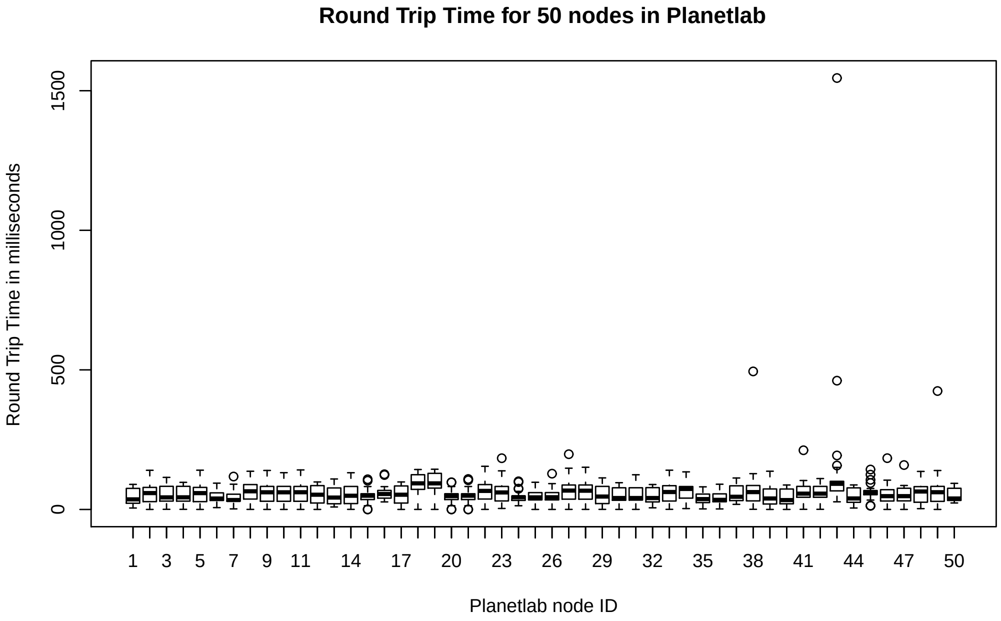


Figure 5.14: Round Trip Times from each node to all 50 nodes in Planetlab.

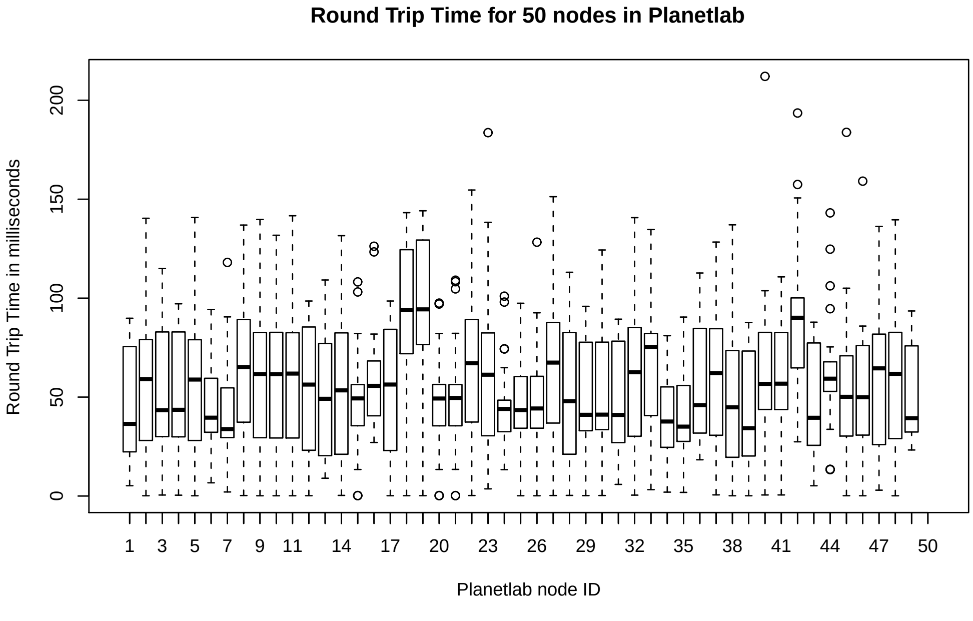


Figure 5.15: Round Trip Times from each node to all 50 nodes after removing four outlier RTT times.

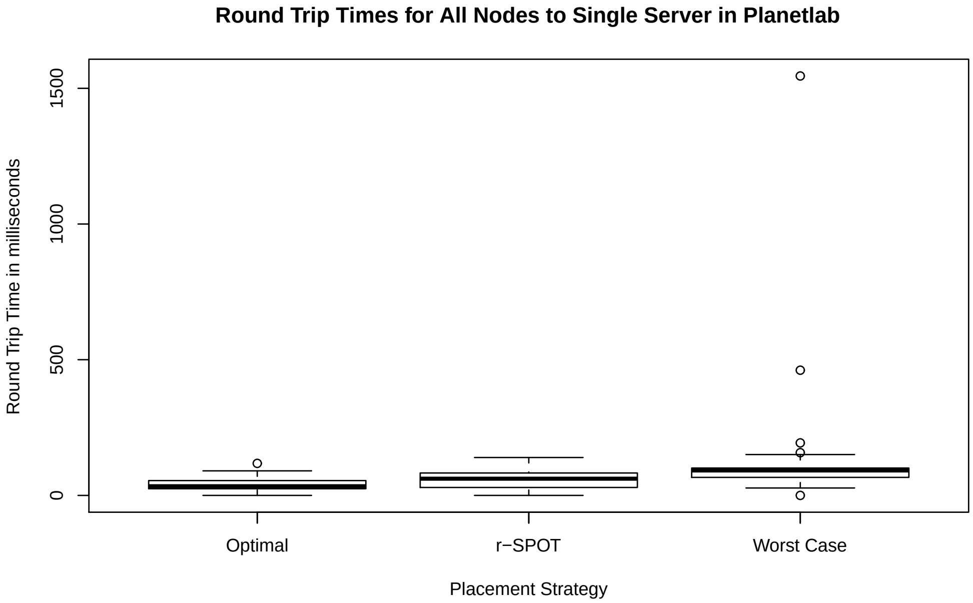


Figure 5.16: Round Trip Times from each node to a single server selected based on the particular placement algorithm times.

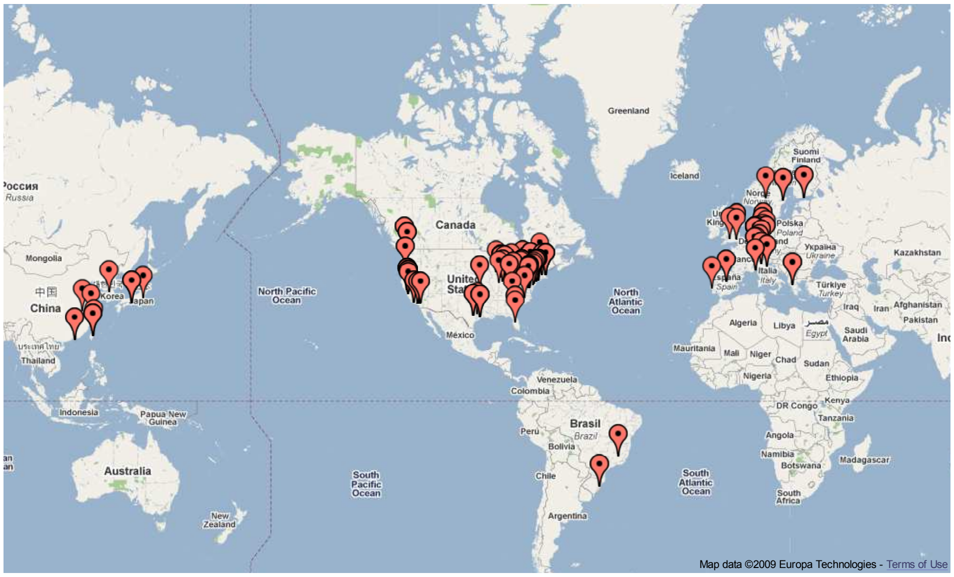


Figure 5.17: Map of the 263 nodes used in the larger Planetlab experiments.

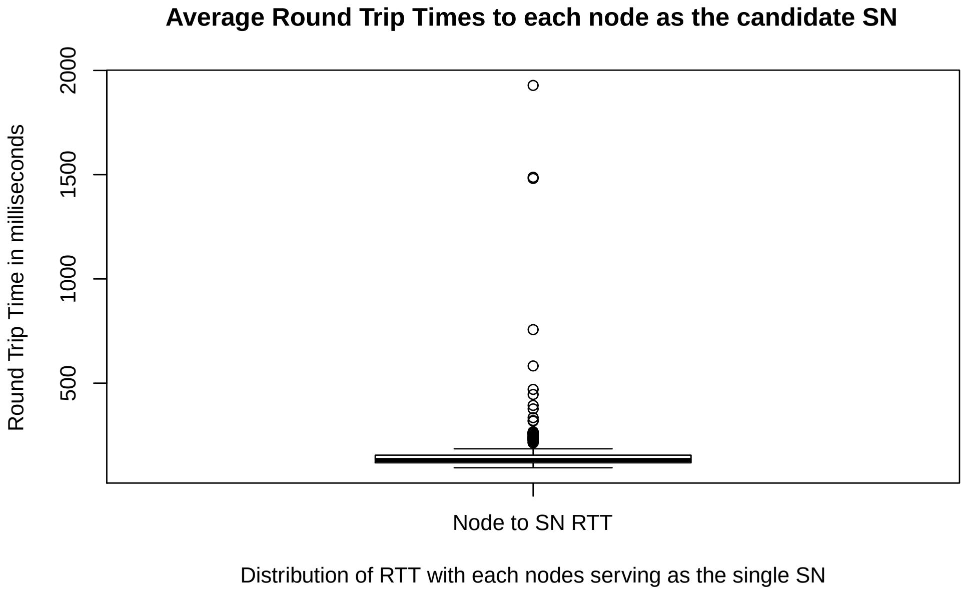


Figure 5.18: Average Round Trip Times for each node serving as the candidate SN node to all other nodes.

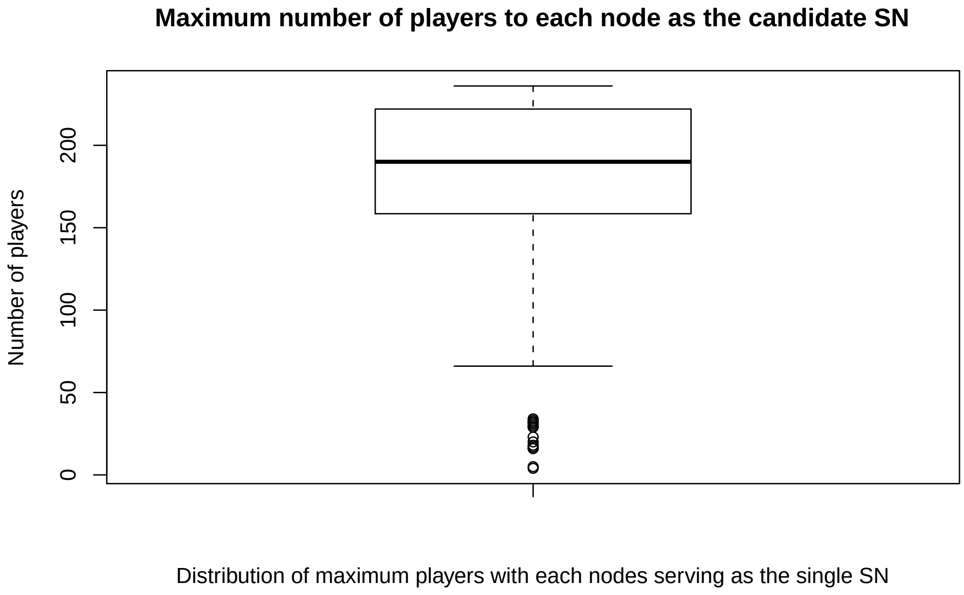


Figure 5.19: Maximum number of players each node supports if serving as the SN. This is based on a 180 ms RTT required to connect to the SN.

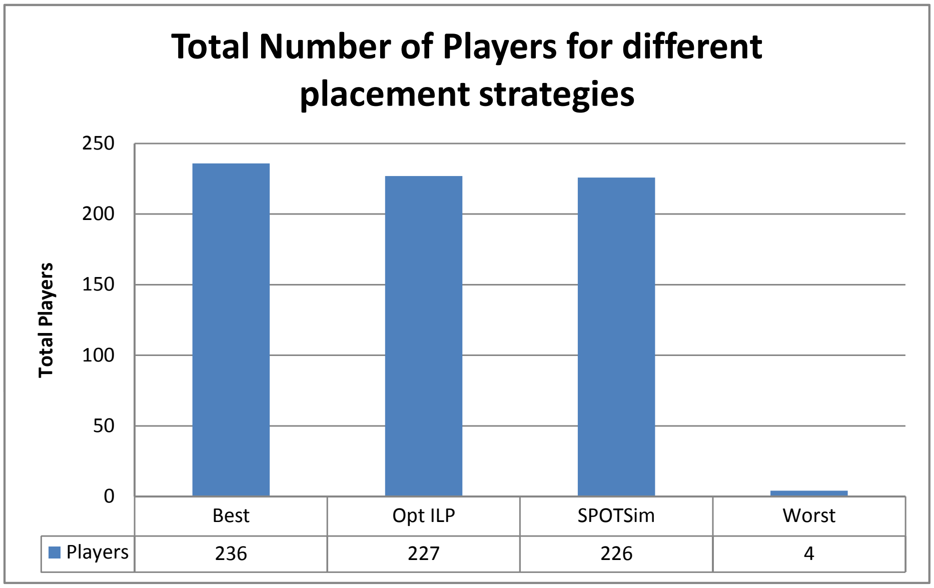


Figure 5.20: Bar graph of the number of players the best case, optimal k-median, SPOTSim, and worst case SN selection.

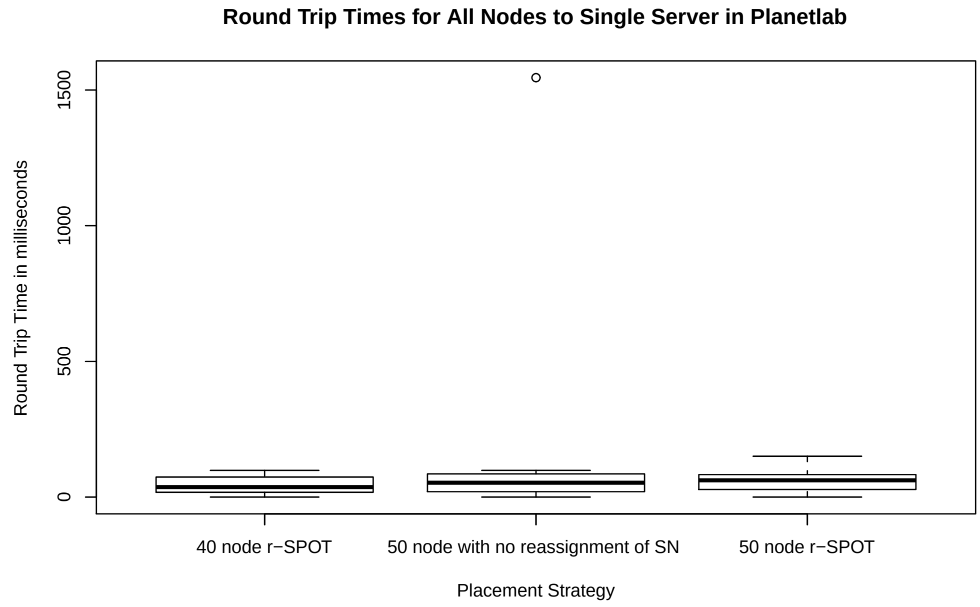


Figure 5.21: Round Trip Times from each node to a single server for a 40 node topology with r-SPOT SN placement, a 50 node topology using the 40 node SN placement, and a 50 node topology with a new r-SPOT SN placement.

Chapter 6

Evaluation of Supernodes in Satellite Networks

This chapter explores the value of deploying supernodes in a peer-to-peer (P2P) satellite network. This network architecture is compared to a traditional client/server architecture. The P2P architecture is evaluated in the Emulab testbed environment using satellite link characteristics. The material in this chapter was originally published in IEEE Aerospace 2007 [71]. Portions of this chapter were written with my co-authors John Meier and John Lockwood.

6.1 Introduction

Communication links transfer data between satellites, unmanned airborne vehicles (UAVs), and devices on the ground. UAVs are used to analyze pollution, relay communications and host a variety of sensors. By providing data processing services within the nodes of the hierarchical networks, raw data can be locally and efficiently transformed into useful information. Overlay networks with *supernodes* placed strategically in the hierarchical network enables traffic to be effectively shaped and filtered.

Many centralized client/server architectures are used to process the compute intensive applications. They channel information between a centralized set of data processing and storage nodes. Often, networked platforms (UAVs, and earth orbiting satellites) are deployed thousands of miles away from a central processing center. Even with caching, client/server architectures are not well suited for network services in media intensive real time networks in dynamic mobile environments, due to changing topologies.

The latency and bandwidth constraints of long-distance communication networks are a challenge for real time media and data fusion applications. As more network devices are deployed, it becomes increasingly difficult to use a centralized processing center. Centralized architectures do not scale well to handle large volumes of information, provide robustness from failure, nor do they provide fast reaction times.

One service benefiting from distributed networks is the deployment of robots to provide medical assistance for injured soldiers. Research efforts such as the Trauma Pod [77] have investigated ways to deploy remote medical services. With *telesurgery*, surgeons perform operations on wounded soldiers using robots. Bandwidth intensive network applications such as streaming video allow surgeons to perform many life-saving operations from a remote location. The need for low latency communication is crucial to increase responsiveness to the remote surgeon during an operation.

Using distributed (rather than centralized) services to interconnect a diverse set of platforms increases scalability and real time performance. This chapter investigates the tradeoffs in deploying a Peer to Peer (P2P) overlay network technology as compared to a centralized client/server approach. The architecture enables mobile devices to efficiently exchange large volumes of information using new P2P services rather than channeling all information through a central server.

In heterogeneous networks, messages can flow between mobile devices on the ground, vehicles in the air, and satellites in earth orbit. Overlay networks with content-based routing services on P2P networks enhance real time decision making for multi-tiered communications. Overlay network services, such as content-based routing, to improve the Quality of Information (QoI) that flows between devices. Improved QoI enables transmission of useful information using minimal bandwidth. By using P2P services, mobile devices can communicate with less latency and bandwidth than they would using a centralized system.

Distributed services running on P2P networks enhance information rendezvous rates while reducing latency of information exchange. P2P establishes efficient overlays to enable content based routing. Overlays in P2P networks decrease routing time for advanced multi-tiered communications. The three tiers of communication (space, air, and ground) must seamlessly integrate low latency services on multiple platforms using overlays to provide scalable real time network services. Long-distance links between geosynchronous satellites and moving vehicles such as UAVs require a highly dynamic network environment.

UAVs like the *Shadow* fly at around 5000 feet while low-flyers such as the *Dragon Eye* and *Honeywell MAV* operate at 100's of feet. The bandwidth between these heterogeneous nodes scales logarithmically as the altitude increases. P2P technology is used by the highly dynamic ad hoc networks to improve reaction times in a service-oriented architecture (SOA).

Adhoc networks use overlays to improve neighbor node discovery, user authentication, and tunneling of sensitive data. Once set up, the overlay network facilitates discovery of additional nodes with minimum reaction time. The P2P API JXTA facilitates discovery of distributed services [26].

This work utilizes distributed P2P networks with an overlay to reduce latency and maximize use of available network bandwidth. In this work, we measure the network metrics of the network latency and bandwidth as a function of the configuration of the network. We also measure an additional Measurement of Performance (MOP) to characterize the number of successful requests for P2P services. Specifically, we measure the MOPs for four services: transfer of streaming target tracking data (40 Kbps), still image transfer (100 Kbytes), streaming video (700 Kbps), and sensor query data (10 Kbyte).

We perform experiments using the emulation testbed laboratory, EMULAB [80]. Our experiments use up to 147 PCs to study link costs and we compare the overlay network against traditional client/server models of tasking resources.

The nodes distribute resource request messages using multicast communication and rendezvous nodes (also referred to as *supernodes*). There were significant challenges to interconnecting and managing a diverse set of mobile platforms, network nodes, and end systems with multicast.

A supernode is a server, router, switch or other network device that has more memory, bandwidth, processing power, or better locality than other nodes in the P2P network. Super nodes reduce the need for multicast traffic and P2P chatter by serving as a rendezvous point for nodes deployed in the network.

Super nodes implemented with reconfigurable hardware, such as the Field Programmable Port Extender (FPX) [39], improve data processing services for applications in the network, enforce Quality of Service for Voice, transport Voice over IP (VoIP), and transcoded video.

6.2 Related Work

Today's adhoc networks support a diverse set of services that require different priorities and different allocations of bandwidth for traffic delivery. P2P topologies are scalable to meet the needs of hundreds, thousands, and even tens of thousands of users [17].

Overlay trees help P2P networks optimize the use of bandwidth by minimizing the overhead required to find peer servers [52]. In the related work of file sharing, it was found that the choice of which peer to use in the overlay had a large impact on performance. Picking the correct peer doubles the media file sharing capability in certain cases.

Simulations, such as *p-sim*, have shown how adaptive P2P topologies reduce latency in overlay links [48]. Past work focused on how the application benefited from a P2P deployment rather than measure the peer dynamics, performance of file sharing and searching, or work load of search queries.

Several network simulators provide some support for large scale P2P network experiments. P2PSim [54] is a discrete event simulator that models overlay networks such as Chord [72] and Tapestry [82]. These P2P implementations are created by P2PSim and do not model all of the features of these protocols [76]. PeerSim [55] is another example of a P2P network simulator. It provides support for supernode topologies similar to our deployment strategy. Unfortunately, it does not model the network transport but, it does scale to large (1000's of nodes) networks. PeerSim utilizes it's own P2P protocols to simulate node behavior. This makes it difficult to compare to the well studied and academic

P2P protocols such as [72]. As with other network simulators, both of these software tools provide some of the functionality necessary to create realistic network experiments but, lack the flexibility and realism gained through an emulation testbed.

The use of redundant *super-peers* (also know as *supernodes*) improves the performance of P2P networks. Guidelines have been developed that suggest how to make best use of redundant nodes. Careful use of super-peer redundancy is needed to handle large aggregate processing loads at bottleneck nodes [7].

Client/Server with Caching

One optimization for the client/server architecture is the use of caching node advertisements in the network itself. This is similar, though with reduced functionality, to the use of supernodes. A caching node would provide some benefit to the client/server architecture when nodes are static. However, we envision nodes continuously moving from location in the grid to the next thus invalidating the cached values. The P2P approaches work well is this scenario by publishing the service advertisements to a super when the node relocates. If the client/server architecture sent update information to a caching node and queried it directly we would argue that it is in effect a P2P architecture and not a client/server with caching.

6.2.1 Node Architecture

Services Offered

We model a node that uses four types of services. The first type of service is for a high bandwidth, constant bit-rate, User Datagram Protocol (UDP) video stream that has a bandwidth of approximately 700 kbps. The second service is a low-bit rate service that uses UDP to send coordinate and sensor information with a bandwidth of approximately 40 kbps. The third service models an aerial camera which transmits 100 kbyte images using the Transmission Control Protocol (TCP). The final service transmits 10 kbyte sensor queries using the TCP protocol. Nodes randomly select a service based on the distribution listed in Table 6.1.

Video	Track	Image	Sensor	Idle
10%	45%	10%	25%	10%

Table 6.1: Distribution of services for each node.

6.2.2 Implementation of Nodes in the Overlay Network

In our experiments, nodes both request for and provide services. All nodes request and offer services for a fixed amount of time. A new service is requested once the previous service completes or times out. After completing a service, nodes either request another service or remain idle for one second before issuing another request. When the time for an experiment expires, the node completes all currently active services before exiting the overlay.

Nodes are assigned an initial physical location in a grid with specific coordinates on an (x,y) grid. Services are requested from and to specific locations. For example, a node at overlay position (32,53) might request a video stream from location (27,92).

Overlay Software

The software that establishes the overlay network was written in Java using approximately 2000 lines of code. The client/server portion of the code utilized Java Sockets for all communication. The P2P portion was implemented with JXTA 2.3.5 and unidirectional JXTA pipes to send and receive messages. In both implementations, each node created multiple Java threads to concurrently request and respond to services. Service locations were randomly distributed and the type of service requested is based on the distribution listed in the Table 6.1.

6.3 Experimentation

6.3.1 Experiment Setup

In order to emulate a multi-tiered communication network, a large testbed was needed. Emulab was chosen since it is the one of largest academic testbed available. Emulab allows machines to be allocated, a network to be created, and experiments to be conducted in a way that is reproducible. The current testbed consists of 365 PCs, of which a subset of nodes can be allocated to perform experiments. Emulab provides a web interface to configure experiments and allows for administrative control of each node. After an experiment is run, a script is executed to collect statistics about the operation of the experiment and to report the number of successful service transactions, average latency per service and bandwidth utilization. Each experiment lasted approximately 15 minutes.

Two different types of topologies were deployed using Emulab. The first was a star topology with each node connecting to a central switch. The star topology was used to investigate how well the applications perform in an idealized environment. All of the experiments with this topology utilized the pc3000s Emulab nodes which are 3 GHz, 64-bit Xeon processors equipped with 2 GBytes of RAM. This equipment minimizes the effects of the computing hardware relative to the network under test.

The second topology is hierarchical configuration with varying link delays and bandwidth constraints. The hierarchical network provides more a more realistic deployment scenario with models for different types of nodes requesting services at various rates. Due to the size these experiments a mix of Emulab hosts were deployed ranging from Pentium 3 850 MHz PCs to the 3 GHz Xeon nodes. In general, the fastest nodes available were deployed, giving a higher priority to assigning the server and supernodes with the most capable machines.

6.3.2 Effects of Latency and Bandwidth

Several experiments were conducted to measure how latency and bandwidth constraints affected the performance of the client/server architecture. The experiments measured performance in terms of the number of successfully completed service operations. In order for a node to complete a successful service, it must locate the service, request use of the service, and finally transfer the data associated with the service.

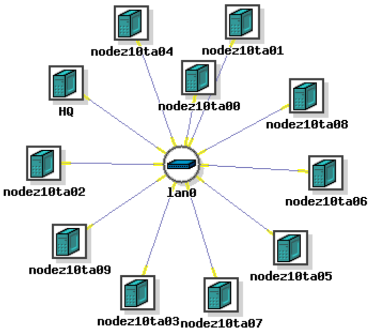


Figure 6.1: View of the 11 node star topology.

Bottleneck Link to the Server

The first experiment measured the total number of services completed as a function of increasing latency between the performance-critical connection to the server. This experiment used a star topology of 11 nodes (10 overlay nodes plus 1 server node) configured with a fixed, 100 ms latency between nodes and variable latency to the server, as displayed in Figure 6.1. Figure 6.2 shows how the number of successful service requests decreased as the latency increased. The P2P architecture, which does not utilize the bottleneck link, performs better than the client/server architecture when the delay constrained server link becomes large. We found that once the delay to the server exceeded approximately 200 ms, the P2P architecture delivers more services than the client/server architecture. In the client/server architecture, all communication is routed from a client to the server then to another client. The performance of the client-server architecture depends on the proximity of the clients to the server as well as the bandwidth of the links. This limits the scalability of the client/server architecture.

In the next experiment, we deployed the same 11 node star topology as before. However, we set the propagation delay of the bottleneck link to a constant then varied the bandwidth. This experiment allowed us to parameterize link bandwidth for a variety of client/server architectures with a P2P approach using a fixed latency (50 ms) on every link. From Figure 6.3, we observe that the P2P architecture completed more services in a fixed

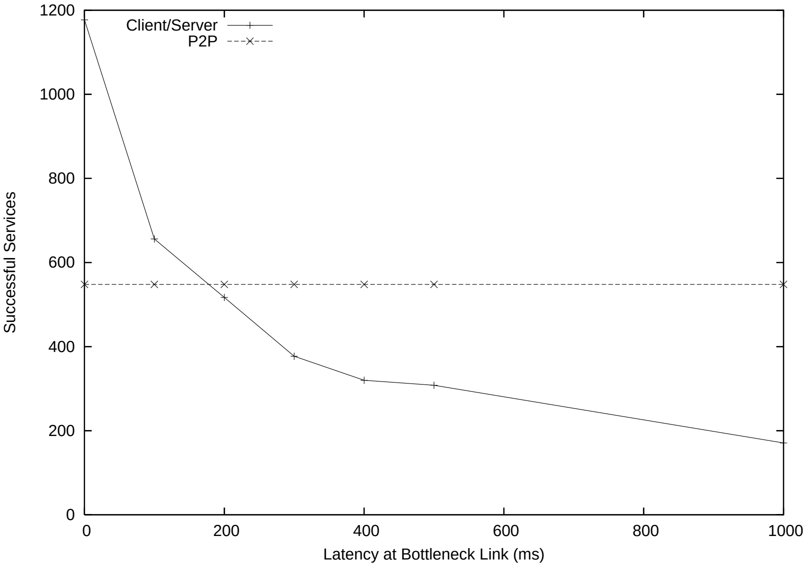


Figure 6.2: Number of successful services for the client/server architecture as bottleneck link increases. The 1 supernode P2P architecture is also shown as a reference.

period of time then the client/server once the bandwidth to the server dropped below 50 Mbits/sec.

6.3.3 Overhead associated with P2P API

A P2P solution adds additional overhead when compared to a client/server architecture for discovery and communication of services. This section describes the amount of overhead that is inherent to the P2P architecture. The P2P overhead is calculated per service by the use of a separate port used for all P2P communication. The first experiment utilized three nodes to calculate the per service overhead from a sender to a receiver communicating through a Super Node. All communication between the sender and receiver was captured using *tcpdump*. Filters were applied to the output of tcpdump to analyze traffic by IP address and port number. The per service overhead includes the overhead introduced by JXTA using XML messages to establish a handshake between two nodes and push out the service. Table 6.2 lists the percentage of traffic that consists of service, and the overhead, with the rest consisting of background traffic on the LAN. In this table, the 700 Kbit/sec service

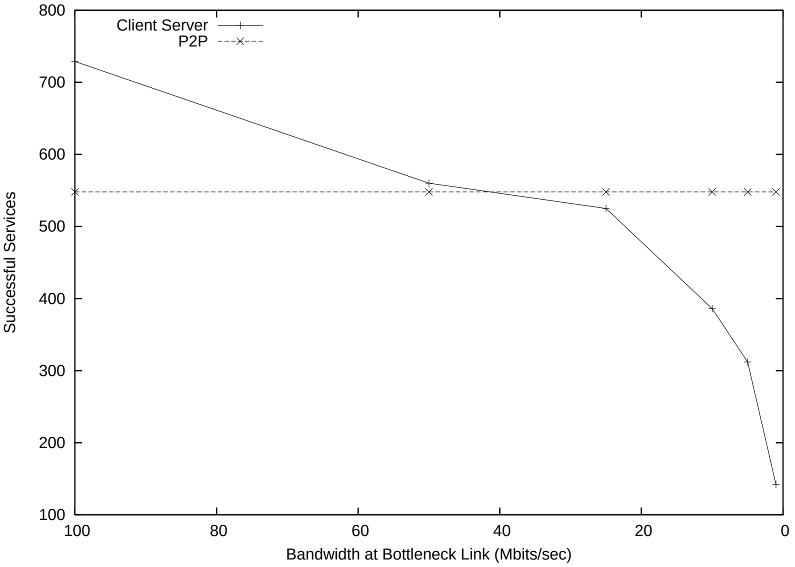


Figure 6.3: Number of successful services as the bandwidth is reduced for the client/server architecture on the bandwidth constrained link. The 1 supernode architecture is provided as a reference.

creates more total traffic than the 40 Kbit/sec service which accounts for the difference in percentage of overhead traffic.

Service Type	Service %	Overhead %
700 Kbps UDP Video Stream	94.5 %	5.0 %
40 Kbps UDP Track Stream	51.7 %	48.3 %
100 Kbyte TCP Image Transfer	96.3 %	3.7 %
10 Kbyte TCP Sensor Reading	71.7 %	28.3 %

Table 6.2: Percentages of traffic associated with the service and the P2P overhead in terms of total bandwidth.

Table 6.3 presents the amount of overhead traffic per successful service. Here the TCP and UDP services require roughly the same amount of overhead traffic for the P2P service requests and discovery, which is what we would expect. From the table, in order to request a 100 Kbyte TCP Image, an additional 41 Kbytes is necessary to discover the service in the overlay network and setup communication between the sender and receiver.

Service	Overhead (Kbytes)
700 Kbps UDP Video Stream	48
40 Kbps UDP Track Stream	48
100 Kbyte TCP Image Transfer	41
10 Kbyte TCP Sensor Reading	40

Table 6.3: Average overhead in bytes per successful service.

6.3.4 Network Scalability

Larger experiments were performed to evaluate how well the P2P and client/server architectures scale. These experiments evaluated the many successful services completed within a fixed period of time as a function of network topologies which had differing sizes. The latency per service and bandwidth utilized per node were measured. In these experiments, a star topology was utilized as a reference that had a fixed latency and bandwidth between each node.

We conducted experiments using 11, 26, and 51-node configurations. The topologies required 17, 39, and 77 Emulab nodes, respectively. Traffic is routed through additional PCs to emulate the desired link characteristics for latency and bandwidth. Each network link incurred a latency of 50 ms between the node and central switch. The link to the remote server was assigned a delay of 125ms. This latency modeled the penalty for accessing a distantly remote server in the client/server architecture. The bandwidth for each link was set to 100 Mbit/sec.

Several different types of architectures were explored in this scenario. The first architecture used a client/server approach. The next four architectures used a P2P overlay. The first P2P overlay used multicast, the next two used one and two supernodes, and finally we deployed a configuration with one supernode in addition to multicast.

Figure 6.4 reports the number of successful services completed as the topologies increase in size. The number of successful services was computed as the sum of total successful services completed on each node. The approach with only the supernode provides the best performance for the larger node experiments. Using multicast with one supernode performs fairly well for the small to medium sized experiments. A multicast only approach is useful with the smaller sized nodes, but as the network increases in size, the performance starts to decline because of the large amounts of traffic created on the network. The client/server architecture performs worse than the P2P approaches.

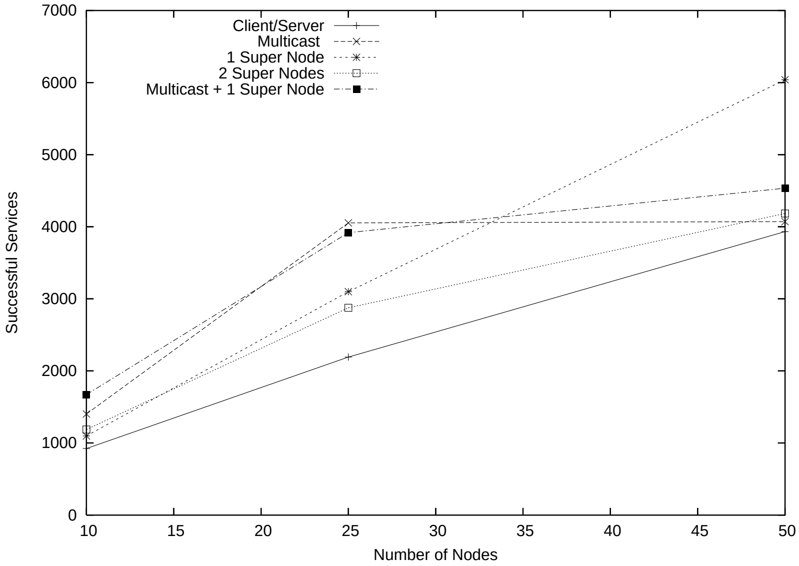


Figure 6.4: Successful number of services as the star topology increased in size. The multicast architecture outperforms the other approaches, with the combined supernode and multicast configuration leading the remaining options.

An additional metric to evaluate the different architectures is measuring the amount of traffic created on the network for each experiment. This is measured in terms of total traffic (Mbytes) and traffic per successful service (Mbytes/service). Figure 6.5 illustrates the total traffic generated by each experiment for a given topology. The results were obtained from reading switch counters before and after each experiment. From the figure, the multicast P2P approach creates the largest amount of traffic with increasing node sizes. The supernode and multicast combination generates the second highest amount of traffic. This is no surprise due to the simple star topology and multicast sending service queries to each node, essentially broadcasting in this configuration.

The traffic per successful service is shown in Figure 6.6. Again, the multicast only approach is the most expensive in terms of bandwidth, requiring over 17 Mbytes per service

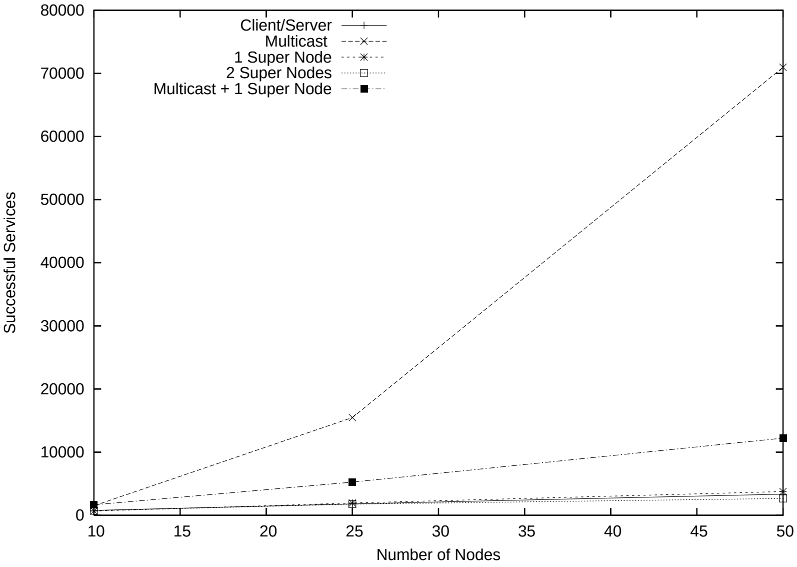


Figure 6.5: Total network traffic as the star topology increased in size with the multicast configuration demonstrating how poorly it scales in a star topology with larger number of nodes.

completed. The supernode P2P and client/server experiments require around 1 Mbyte per service.

The next statistics reported are the latencies associated with each service. Figures 6.7, 6.8, 6.9, and 6.10 measured in milliseconds the latency associated with each service. The latency for the UDP streams was measured from the time that the client requested the stream to the time when the client received the first byte of data. The TCP latency was measured as the time when the data transfer was complete.

The latency experiments can be divided into two groups, the UDP services and the TCP services. The multicast architecture was actually slower than the client/server in the 40 Kbps UDP stream. This is due to the amount of traffic generated from each node searching every node in the overlay for a particular service. Deploying supernodes eliminates that problem by caching service advertisements for nodes utilizing that supernode. In the TCP services the latencies for the P2P approaches were around 2-3 times faster than the client/server model. With the UDP stream services offering around a 30% decrease in latency to discover the service. Excluding the multicast case, as networks grow larger than

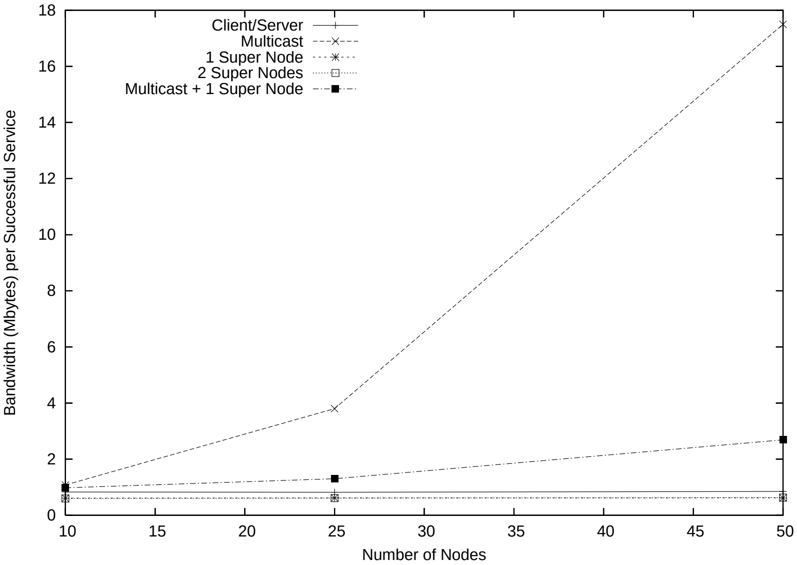


Figure 6.6: Network traffic per successful service as the star topology increased in size, with the multicast configuration providing the most expensive service per megabyte solution.

90 nodes both types of services will benefit even more from the P2P architecture, especially applications transferring large amounts of data.

6.3.5 Hierarchical Network of Super Nodes

This section explores experiments deployed using a hierarchical network topology. In this tree topology, supernodes were placed at various locations near the root of the tree. This network includes a range of bandwidths and link delays with 1 Mbps links on the low flying nodes, 10 Mbps at the Tactical UAVs and 100 Mbps between high flying (X-45) nodes [49]. Delays between links are fixed at 40 ms, 40 ms, and 20 ms for the Low Flying UAV, Tactical UAV and X-45 respectively. The delay link to the server in the client/server architecture was set to 600 ms. The 600 ms delay is a result of the propagation time and queuing that takes place over a multihop satellite link or a wireless to ground infrastructure similar to [81]. The two common methods currently used to route video to a centralized remote set of processors are directly through a satellite (Figure 1) or relayed to a ground station completing the path through the internet. The latency experienced by the satellite

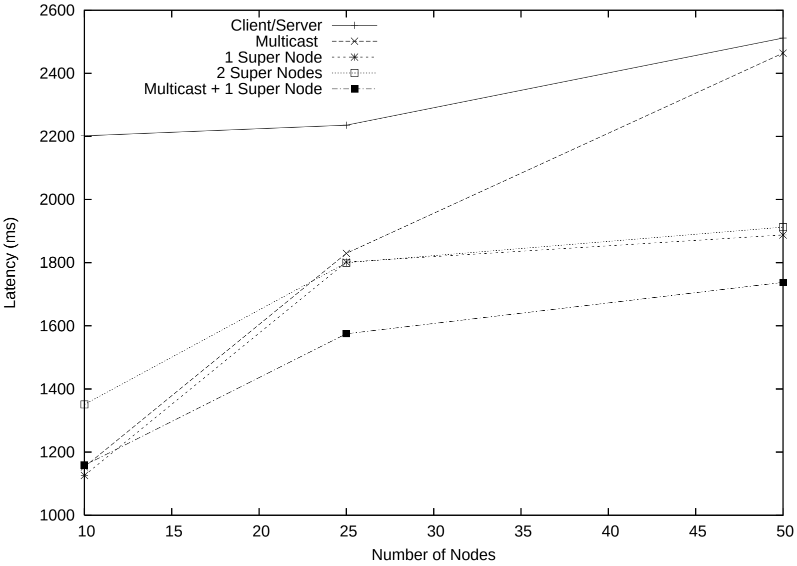


Figure 6.7: 700 Kbps UDP Stream Latency as the star topology increased in size. The client/server and multicast configurations do not scale well with larger sized topologies.

is usually greater than 600 ms because the propagation delay to a geostationary satellite (250 ms), the relay and switching delay to a secondary satellite (250 ms) plus the jitter (100 ms) from the multiplexing and encoding comprise the 600 ms latency. The jitter is due to the multiplexer, modulator, coder, switch, decoder, demodulator and demultiplexer. The use of technology such as Turbo Code provide substantial improvement in error correction however increases jitter due to the large block size required during encoding and decoding. The hierarchical topologies consisted of 11, 31, 54, 75, and 92 nodes in the overlay. The total number of Emulab hosts required to support these experiments ranged from 19 PCs in the 11 node example up to 147 in the 92 node example. Again, this large increase is due to the additional nodes responsible for bandwidth and delay constraints placed between links.

Three different distribution types are simulated at various levels of the hierarchy. Table 6.2 lists the assumed distribution of services.

In order to better exploit the locality of the services, an assumption is made regarding the types of requests issued by the Low Flying UAV nodes. This assumption is that requests are only issued to nodes one hop away, or in the same subnet. This assumption is fairly reasonable given the fact that services are most valuable to the nodes closest to them. No

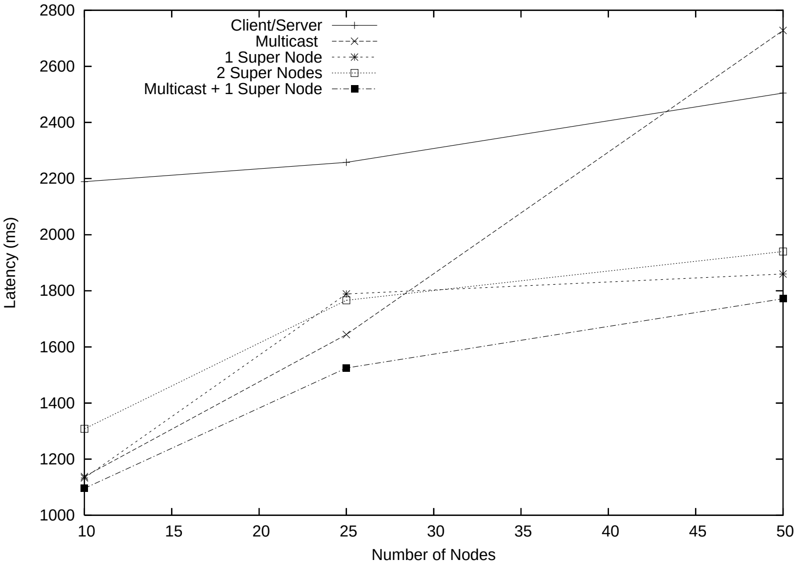


Figure 6.8: 40 Kbps UDP Stream Latency as the star topology increased in size, again the multicast configuration demonstrates a sharply raising latency for the larger topologies.

restriction is placed on the Tactical and X-45 nodes, permitting requests for any node in the topology. These assumptions allow the low flying nodes near regions of interest access to those important services and a global service request scheme for the high flying nodes (Tactical and X-45).

In these experiments multicast was not deployed. With multicast enabled, nodes between routers are unable to communicate with each other in Emulab. Instead, three supernodes are deployed to investigate the benefits of increasing the number of supernodes.

	video	track	image	sensor	idle
Low Flying UAV	10%	45%	10%	25%	10%
Tactical UAV	25%	25%	25%	5%	20%
X-45	40%	5%	20%	5%	30%

Table 6.4: Distribution of different nodes in hierarchical topology

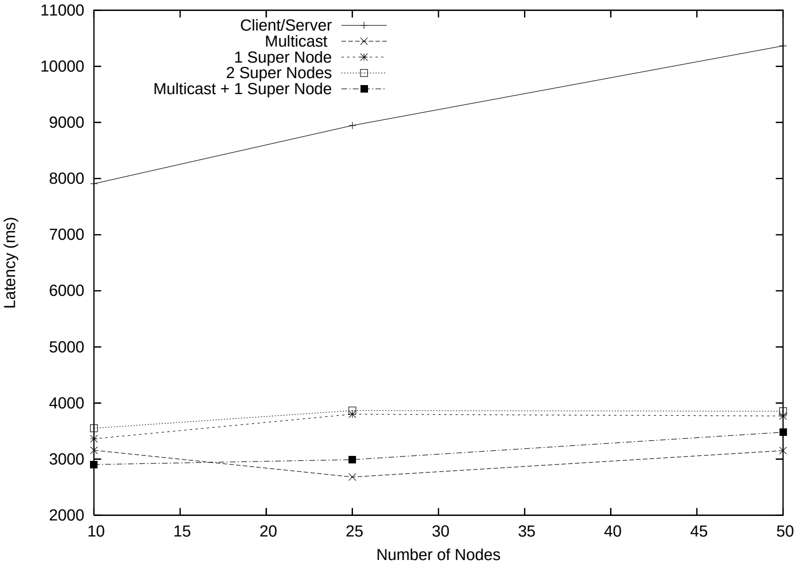


Figure 6.9: 100 Kbyte TCP Transfer Latency as the star topology increased in size, with the client/server configuration unable to scale as well as the P2P architectures.

Results

This section presents the results of the hierarchical topologies. Figures 6.13, 6.14, 6.15 report the successes, bandwidth, and bandwidth per service respectively. Figures 6.16, 6.17, 6.18, and 6.19 report the individual service latencies.

The number of successful services scales well in P2P architectures. The P2P nodes perform almost twice as many services as the client/server in the largest experiment. The client/server experiences an initial decrease in bandwidth per service due to the difference in topologies between the 11, 32, and 54 node experiments. The 54, 75, and 92 node experiments share a similar structure with an increase in total nodes at the edges.

The P2P architectures generate more total traffic as the topologies increase in size, however the number of services completed is also greater. It is important to note the rising costs of services per MByte in the client/server model in Figure 6.15 with larger experiments. The costs for the one supernode example remains fairly stable even for larger experiments which is very encouraging for building larger systems. The bandwidths fluctuate for the two

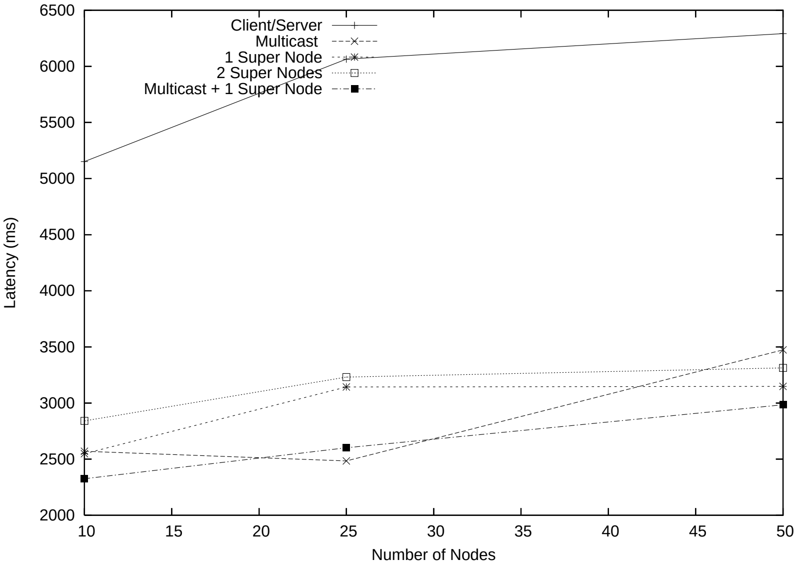


Figure 6.10: 10 Kbyte TCP Transfer Latency as the star topology increased in size. The client/server performs better in this smaller file transfer size, however not at the smaller latencies of the P2P architectures.

and three supernode examples in the larger node topologies, however the costs are always considerably less than the client/server case.

The client/server model performs at 2-5 times the latency of the P2P architectures, depending on topology size and the service. In the 10 KByte TCP transfer, Figure 6.19 the client/server approach increases by 17% moving 75 to 92 nodes, compared to the 1% increase experienced by the single supernode P2P example.

6.3.6 Use of Super Nodes

From the results, the addition of more than one supernode does not necessarily improve performance on all topologies. Strategic location of the supernode will impact the effectiveness of the P2P technology. For the larger experiments though, performance improvements were demonstrated when slower nodes were deployed in Emulab. For example, in the 92 node hierarchical, using a Pentium 3 850 MHz PC proved inadequate acting as a supernode for the other 91 nodes in the overlay. The bottleneck was simply the JAVA application consuming 99% of the CPU due to the overhead associated with handling service requests

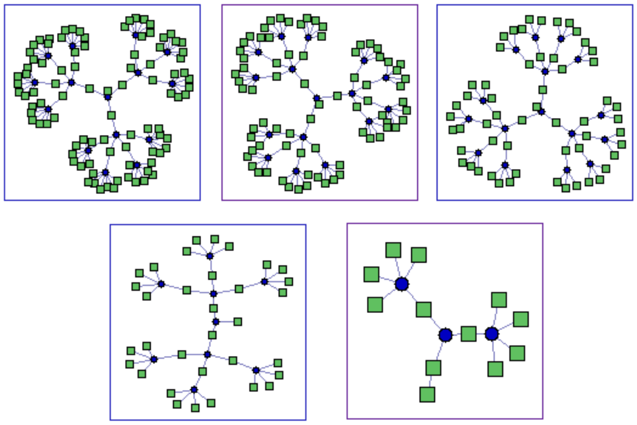


Figure 6.11: High level view of 92, 75, 54, 32, and 11 node hierarchical topologies.

for a large number of nodes. When deploying additional supernodes to divide the load however, the number of successful experiments increased considerably.

The second benefit provided using additional supernodes is redundancy. When a supernode fails in the P2P examples, the additional supernodes continue to operate to provide services. Each additional supernode is capable of distributing resolving queries for every node in the network. In the client/server model, a failed server or bottleneck link will completely disrupt the use of services. Initial placement of the supernodes indicates two hierarchial levels from the edge is optimal.

6.3.7 Benefits of Emulation

Current P2P network simulators [54][55] lack the realism found in an actual implementation of all the P2P protocol's unique behavior. For example, with emulation, we are able to evaluate with greater confidence than a simulator, the latency required to complete a service. Also, with emulation more practical issues are exposed such as the amount of CPU processing necessary for a supernode for support 100 nodes in a distributed environment.

One early result which came from the emulation and arguably would have came from a JXTA P2P network simulator (if it existed) was the importance of locality in nodes requesting services. When an arbitrary node requesting from any given neighbor the latency was almost as large as the client/server architecture. Making the assumption that most low

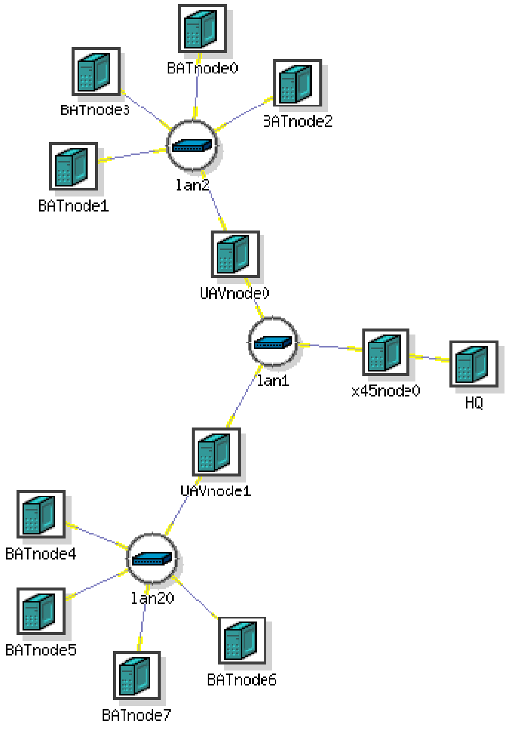


Figure 6.12: View of 11 node hierarchical topology.

flying nodes are generally interested in services from nearby nodes substantially increased the number of services completed and decreased the per service latency.

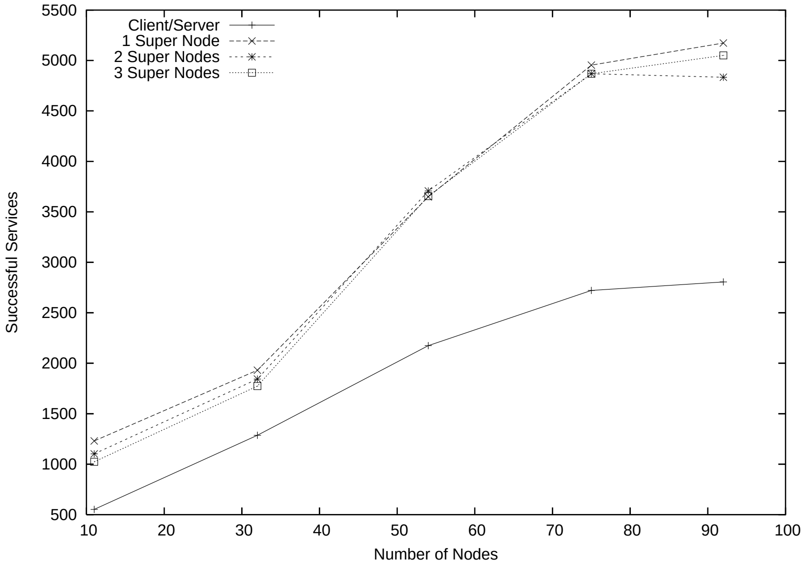


Figure 6.13: Successful number of services for localized communication hierarchical topologies.

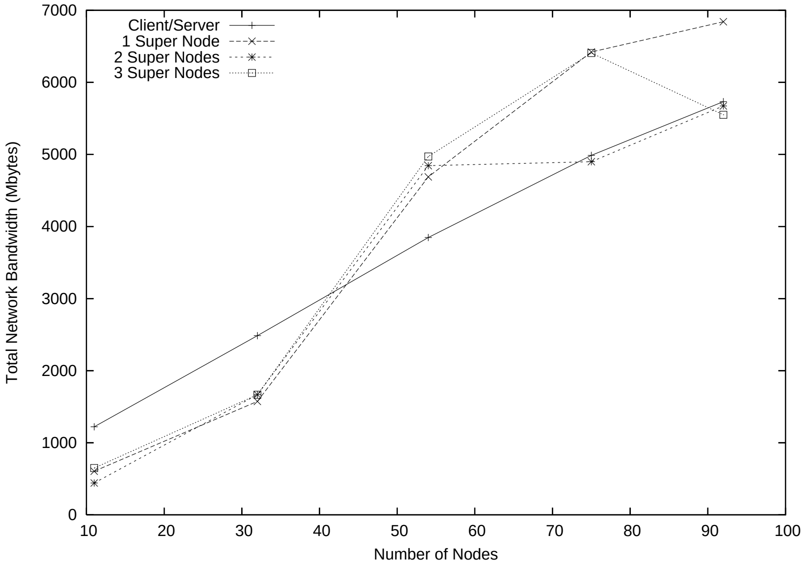


Figure 6.14: Total network traffic for hierarchical topologies.

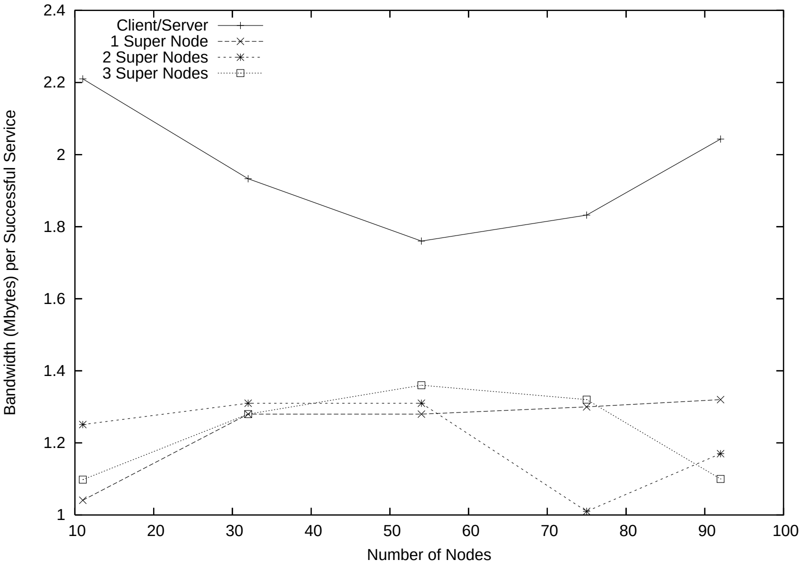


Figure 6.15: Network traffic per successful service for hierarchical topologies. The client/server approach requires more bandwidth for all sized topologies with a rising trend in the largest experiments.

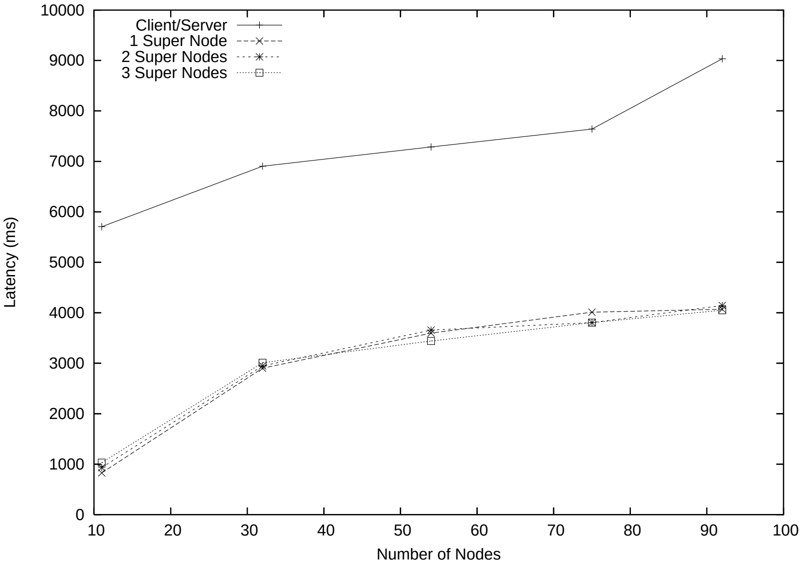


Figure 6.16: 700 Kbps UDP Stream Latency for hierarchical topologies. The P2P architectures discover services at least twice as fast as the client/server.

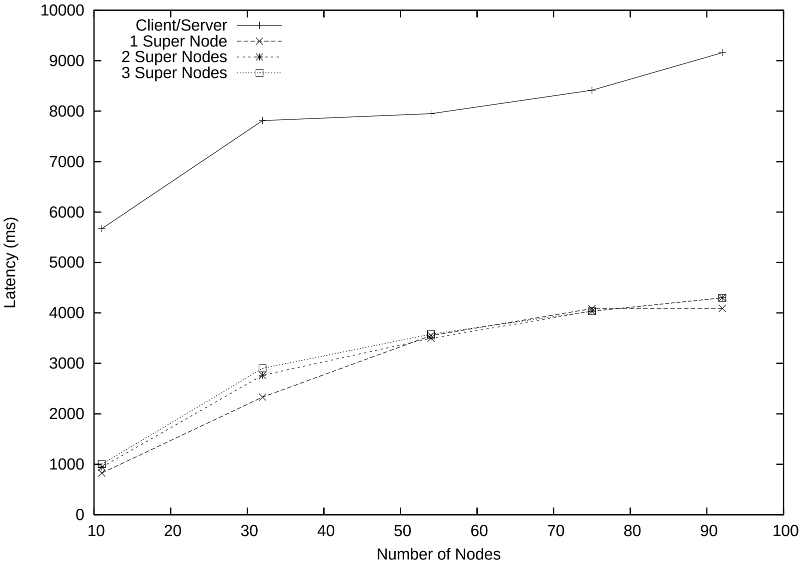


Figure 6.17: 40 Kbps UDP Stream Latency for hierarchical topologies. The P2P architectures service latency outperforms the client/server in requesting the UDP data stream.

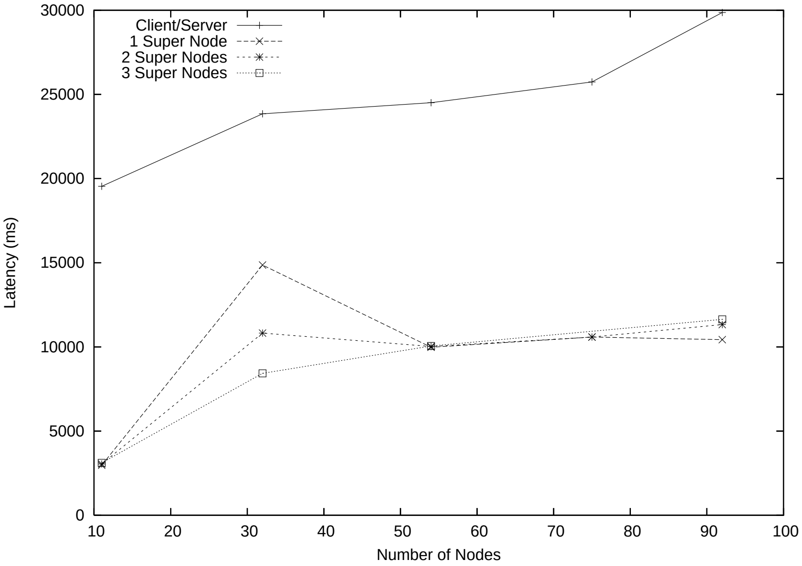


Figure 6.18: 100 Kbyte TCP Transfer Latency for hierarchical topologies. The client/server architecture continues to rise at an increasing rate for the largest experiments.

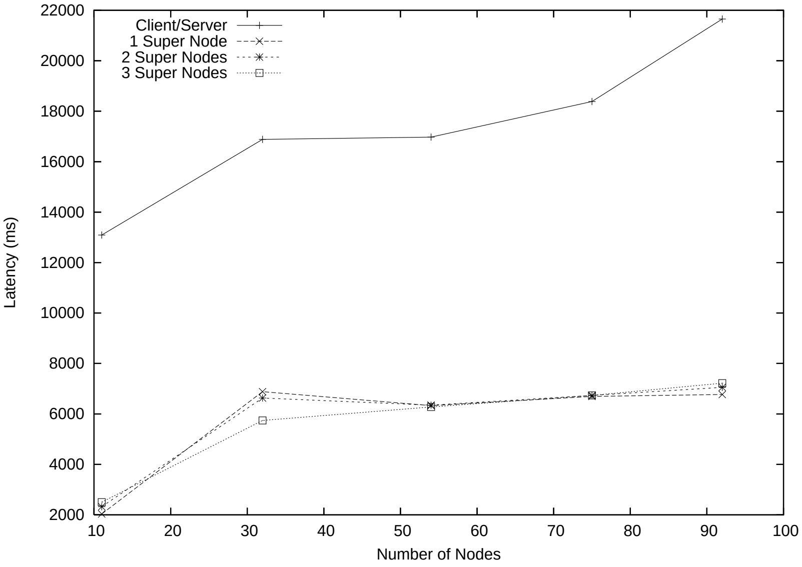


Figure 6.19: 10 Kbyte TCP Transfer Latency for hierarchical topologies with the P2P latencies scaling very well with larger topologies.

Chapter 7

Evaluation of Hardware Accelerated Supernodes

This chapter investigates the use of Field Programmable Gate Arrays (FPGAs) to improve the performance of a supernode (SN) operating in a avionic satellite network. This high performance SN is clustering data and forwarding traffic to nodes designated to receive the service. An architecture is proposed along with performance evaluations on the Emulab testbed. The material in this Chapter was originally published in IEEE Aerospace 2008 [47]. Portions of this chapter were written with my co-authors John Meier, Adam Covington, and John Lockwood.

7.1 Introduction

Multiple commercial and military aerospace platforms (aircraft, satellites, and trucks) use a diverse set of sensors (e.g. radar, infrared) to track targets. Distributed track fusion is also used for condition based maintenance, robotics, medical diagnosis and environmental monitoring [28]. The application layer single sensor data is partitioned into sets of observations, or tracks, that provide both time and distance history of targets sent by the network to a centralized location for fusion into a Common Operating Picture (COP). Having an accurate and timely COP improves the ability to perform an accurate Situation Assessment

(SA). Track observations are processed by the sensor system to uniquely identify the targets using key metrics such as velocity, future predicted position, and target type. Centralized fusion requires distributed sensors to send track information to a common aggregation point which currently creates significant latency due to limited bandwidth and processing challenges. Today large engagements can create more than 10 Gbps of distributed sensor data. The ability to route and and process this data is limited by the network architecture and processing capabilities on the platform. Many have suggested moving to a distributed or hybrid architecture [28],[9] may improve cross-range accuracy through combining data from multiple separated sensors while mitigating the risk of single fault failures. Track data (observations) are distributed using shared bandwidth between platforms to improve SA. Clustering improves SA by providing improved data association. Clustering results in prioritization of network data, conservation of bandwidth and lowering the track fusion latency. Real time SA requires improved dynamic exchange of sensor observations which often generates duplicate data for targets located in overlapping coverage which cause network overload. It is critical to select only the best information to send over the limited bandwidth. Lossless compression techniques are not effective in fitting all the information within the limited bandwidth for large scale systems. Clustering intelligently groups track messages autonomously in real-time to use available bandwidth with the highest priority track data. Intelligent grouping uses the algorithms described above with network data content to prioritize, aggregate and disseminate key information in real time. Our goal is to reduce the number of tracks exchanged while retaining information quality. The effectiveness of using our clustering algorithm operating in a distributed environment was proven using large-scale Emulab experiments.

Distributed sensor data has the risk of increasing the dissemination bandwidth exponentially as compared to centralized fusion. Adding a network layer that intelligently distributes only high value measurements (sensor data) is needed to mitigate that risk. We argue that flooding data to allow all sensors at each node to build separate COPs is not an intelligent use of network resources [83]. Informed selective collaboration at the network layer is needed to reduce latency and bandwidth. Improving bandwidth usage decreases latency which also improves track accuracy. Distributed fusion technology increases connectivity by sharing and processing more data locally. After the data arrives at designated aggregation nodes and is processed, the results (track reference table with unique IDs for all tracks) are disseminated to each sensor node to achieve the COP. Traditional methods to distribute track observations often load the existing bandwidth beyond the channel capacity resulting in information latency and loss. We discuss the relative merits of a new network architecture that is aware of the higher layer (application level) distributed data fusion to improve situation awareness using real time technology at the network layer.

This chapter presents a brief overview of distributed sensor system track fusion (application layer, intelligent network layer, and intelligent gateway), simulated and hardware experimental results, hardware designs and future research plans. We contrast filtering observations at the application with clustering at the network layer. Our simulation, emulation and hardware results indicate a distinct advantage of performing clustering in the network layer (node and gateway).

We assume the distributed fusion algorithms are operating on each wireless node and the track observation content is accessible by the network. For simplicity we only disseminate (not modify) new and update track message types provided to the network prior to transmission.

7.2 Related Work

Distributed track fusion (DTF) includes data association and track filtering. Data association receives observations and must assign them to existing tracks or identify them as new. Sensor measurements of the targets may be imprecise due to measurement resolution, noise, lack of other sensor information and other error sources. DTF has the potential to improve accuracy [83] provided wireless networks can support the data dissemination. Clustering applied to distributed fusion using avionic networks is a novel application. Image and text based clustering was the basis which formed our current approach. The two main forms of clustering are agglomerative (bottom-up) and divisive (top-down). Each approach utilizes a distance metric that measures the difference between two elements. Agglomerative clustering treats every data element as a separate cluster and merges clusters if they exhibit similar distance metrics. Divisive clustering starts with all data elements in the same cluster and partitions them into different clusters based on the distance metric. Both forms of clustering are described in [41] as it applies to hierarchical document clustering using the k-means clustering algorithm.

The original k-means algorithm was described by Duba and Hart [18] and has been used by Estlick et al. [21] and Lesser et al. [36] to implement a hardware approach of k-means clustering for hypersectral images. We leveraged the basic idea of partitioning the pixels composed of multiple spectral channels in N dimensions associated with a "center". Our solution partitions the observations composed from multiple sensors at varying times into N clusters associated with a center or centroid representing each track. One novel modification of the existing algorithm is the projection of the centroid to help assess the value of the data. They [36] compared three different distance metrics that included: Manhattan, Euclidean, and Max distances. Euclidian distance is rotationally invariant and minimizes within class variance but is more expensive requiring a multiplication for every distance vector to each centroid or track recorded. They determined that the Manhattan distance would be the best fit for their hardware. Our demonstration system uses only two dimensions and therefore does not demand rotational invariance provided by the Euclidian distance. Covington et al. [14] showed the implementation of k-means mapped into integer arithmetic. They utilized a Cosine Theta distance metric since this metric (also known as the spherical distance) provides a better distance in high dimensional sparse data.

Many data association algorithms have been presented [23] [22] for group target clustering but few leverage recent advancements in data clustering technology to improving bandwidth usage. Clustering has been used in two ways for tracking in the literature. Target clustering groups similar data elements or observations together to form a track (usually without prior knowledge). Group clustering typically computes a location and velocity PT position of a large number of closely spaced targets moving in the same direction in order to reduce the track data transferred, system loading, and miscorrelation. Older track fusion methods use Nearest Neighbor (NN) algorithms to make decisions as the data arrives while newer methods delay decisions by storing the data. Our clustering methods rely on making decisions as the data arrives. We extend the current research [36] for clustering dynamic DTF data sets. We project a representation of the centroid (a projected target (PT)) to calculate the distance from all known tracks stored. This novel method enables us to map static data clustering techniques to dynamic data association required for DTF.

The traditional K means algorithm requires specifying the number of clusters. Since we never know exactly the number of targets being tracked, applying traditional clustering algorithms is impossible. Most clustering algorithms operate on a fixed set of data and are iterative. Every cycle a data element is selected and is moved to the cluster that is determined to be the best fit by the distance metric. The clustering algorithm used to cluster track data differs from these cyclical algorithms. Since the clusters represent known tracks that could move, the incoming track data can only be clustered once. The track clustering also utilizes a cluster threshold to determine if a track is close enough to be included in a cluster.

Clustering algorithms use similarity measurements known as the "distance." The distance is computed from observations to centroids. Normally, centroids represent the average of grouped data elements however we project the last observation to the temporal-space for the discrete time interval needed to match the bandwidth available. Others [36] have evaluated different distance algorithms (Manhattan, Euclidian, and Chebyshev) for use with the k-means clustering algorithm. Although there are numerous distance metrics that can be used for clustering, our work utilizes the Manhattan distance metric [36]. We can implement the spatial clustering algorithm (SCA) using Manhattan distance with less (50 percent) number of gates in an FPGA and the accuracy is adequate.

Our approach compares the value of the information being intelligently clustered using active networking with normal application layer data disseminated using traditional networking. We propose a simple method to project the target based on velocity and use distance methods to assess the value of each observation. Our method simply compares the distance to the PT of each track to make the clustering and transmission decisions. Improvements to lower level queuing using higher level geometric methods Shannon [66]have been observed. The challenge is to provide the information required for accurate tracking while selectively eliminating redundant or unnecessary information to match the bandwidth capacity available. Our solution adds intelligence between the source and receiver for difficult real time decision making at the network level to increase the value of information and improve network quality of service (QoS).

Messages differing by only a slight variation of measurement (to a limited extent) represent nearly the same information and offer an opportunity to eliminate redundant information. This may also reduce the number of dimensions in the current message space. Track observations which are considered equivalent by the destinations (receivers of track observations) can be grouped together and treated as one point or a reduced set of key observations. Equivalency is evaluated by assessing the relative closeness of the observation to the PT. We propose that this grouping method requires fewer messages to specify one of these equivalence classes defined as we cluster the track observations rather than sending sequential non-prioritized observations. Distance methods, such as Manhattan or Euclidean distance, compare the distance from the PT to the observations. If all observations are evenly spaced on a circle around the PT, they can be regarded as equivalent, and theoretically can be reduced to a one-dimensional space or point. We extend this specific example representing points (track observations) within the circle as key messages to be sent at higher priority. The radius of this circle is defined by the clustering threshold. We preserve state by storing each target's most recent track observation sent for every PT.

7.3 System Overview

The future situational assessment (SA) capability is considering distributed fusion to increase track accuracy and reduce latency in acquiring a unified common operating picture (COP). Distributed fusion challenges are difficult due to the many constraints such as unreliable wireless transport, limited processing power at the network edge, exponential increased demand for bandwidth and the use of multiple legacy wireless communication links with low throughput.

Sensor networks exhibit similar scalability and wireless network challenges when deploying many tiny inexpensive sensors that integrate sensor data. These small sensors can be deployed on platforms or ”sprayed” onto roads or other surfaces to monitor lights or heat or fires or highway traffic [83]. The large number of small sensors connected using very limited wireless bandwidth is a key challenge when disseminating sensor data. In either avionic or distributed sensor networks the kinematic parameters (position, velocity, etc.) are key to data fusion. Early Multiple Target Tracking (MTT) systems only used basic kinematic quantities to track objects. Today Bayesian formulation is based on multiple attributes from multiple sensors to improve target tracking and identification.

MTT data from multiple sensor was used to evaluate the bandwidth required for distributed fusion using network clustering at the node and gateway. We selected a large number of sensors (75 nodes) exchanging a maximum of 50 track observations from each node which generated an upper bound of approximately 100 Mbytes or 800 Mbps. Avionic or wireless sensor network architectures today limit interconnecting numerous moving sensors each tracking multiple targets. The targets (T1...T3) are within range of two distributed sensors relaying the track observations to the gateway node as shown in Figure 7.1. Target T1 is static (not moving), T2 has random movement and T3 is flying in a defined pattern. The gateway node is shown choosing the best track information and removing the redundant T2 track information at the gateway (GW) supplied by N1 and N2 before passing it up the hierarchial network chain. The scenario illustrates the integration of application with the network layer.

Avionic and sensor network capacity is typically less than 100 Kbps which creates a major bandwidth bottleneck. This bottleneck adds significant latency using normal network queuing structures. We insert an intelligent gateway as shown in Figure 7.1 to reduce the data at key bottleneck points by adjusting for the available bandwidth. We also have the ability to eliminate the gateway queue to decrease data latency.

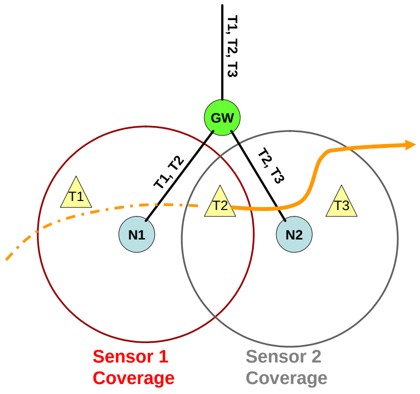


Figure 7.1: Target (T1-3) are tracked by two sensors and the gateway (GW) node eliminated the redundant target 2 data.

7.3.1 Application layer distributed track fusion dissemination

Existing MTT detection, classification, and tracking algorithms work well on a single platform. Multiple platform sensor MTT systems need to accurately correlate a target's parameters (position, range rate, velocity, and acceleration) [9]. Today the application layer normally decides which MTT data will be distributed and passes it to the network layer. It is not realistic for every node to be sent and process every track observation in a large operation to develop exactly the same COP. The distributed application must evaluate the data and determine what should be sent to enhance the COP.

The major challenge faced by distributed multi-target track (MTT) fusion is choosing the right information to send at the right time over severely limited bandwidth links to construct a scalable unified picture that will enhance situational awareness (SA). MTT employs one or more sensors, together with computing resources, to interpret the environment based on a series of measurements.

Our Matlab experiments compared traditional application layer dissemination techniques with adding an intelligent network layer. Traditional "Queuing" network methods are contrasted with our new ”Clustering” method by evaluating the pattern generated (percentage of area). All targets maintained known patterns (geometric shapes) and the observations used normal network buffering operation. Bursts of data congested the link and resulted in data losses due to finite length buffers dropping packets. The objective was to evaluate the changes in the geometric shapes due to loss of the data (shown by the red line - ”received target observations”) compared to the actual target pattern (shown in blue - ”ideal target generation”) illustrated in Figure 7.2. The dropping of critical data prevents regeneration of the ideal target pattern. The difference in the area of the two geometric shapes was adopted as a measure of the value of the observations received, as suggested by Shannon [66]. Significant value is lost (area reductions) using traditional application layer dissemination depending on the randomness of the queue, available bandwidth, burstyness of the data, and the changes in the target pattern.

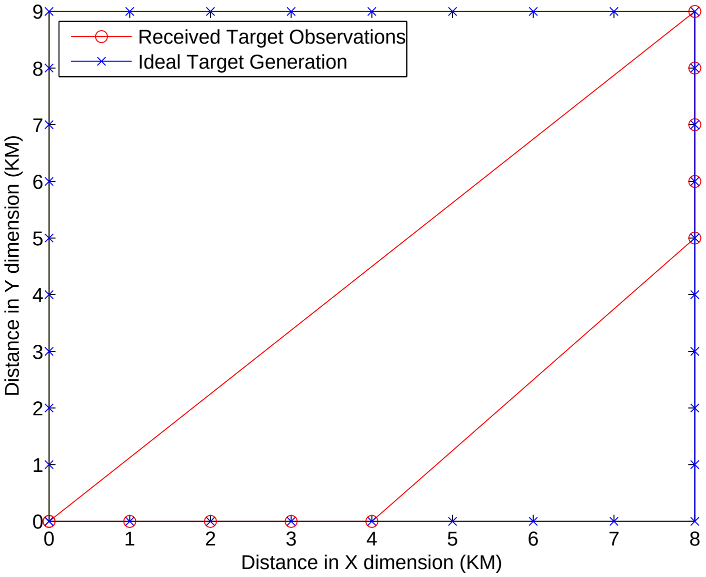


Figure 7.2: Decreased area represents reduced information value

Figure 7.3 represents the ability to recreate the ideal target pattern based on available bandwidth. Our results indicate that a more accurate representation of the target pattern is recreated with less bandwidth using clustering. The target pattern remains distorted using ”Queueing” until a large percentage of the bandwidth is available as illustrated in Figure 7.3. The ”Clustering” solution is shown to provide increased information content which reduces required bandwidth and latency.

Our clustering method can eliminate the majority of the delay by replacing the traditional queuing mechanisms with parallel state machines in FPGA hardware to make real time decisions. Figure 7.3 illustrates that clustering can recreate the target pattern better at much lower bandwidths than traditional methods.

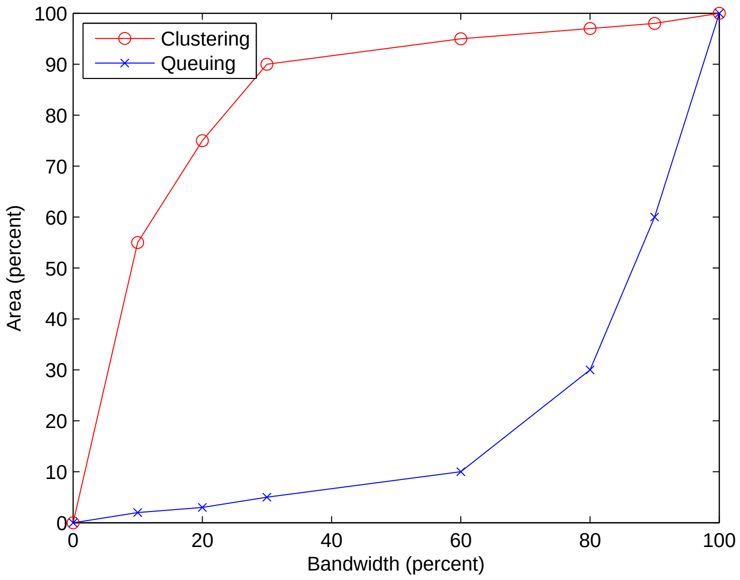


Figure 7.3: Clustering increases information content

7.3.2 Intelligent network layer clustering

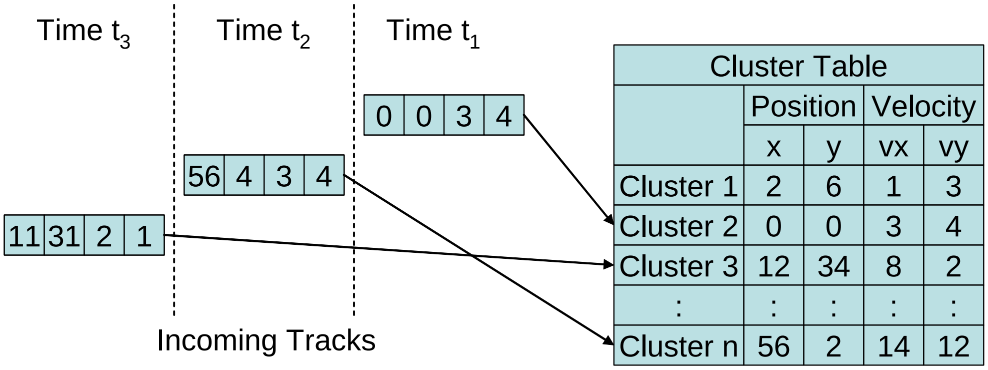
MTT partitions sets of measurements (observations) or tracks for object representations in space and time. Target prioritization (missiles, aircraft, trucks, ships) is performed to correctly assess the environment however this information is not normally passed on to the network layer. Quality of Service (QoS) at the network layer normally relies on this type of prioritization to route data more efficiently. MTT application developers have recently proposed application layer solutions to address the QoS issues related to distributing the sensor data. One solution uses overlays to perform network layer functions. This approach creates real time performance issues, demands significant increases in processing, significantly increases the bandwidth, limits key access to network management control parameters, and adds latency due to large queues buffering network packets. We evaluated this approach using JXTA peer to peer (P2P) overlay technology in Emulab (see simulation results) and found it increased the bandwidth by a factor of 17 while adding significant latency.

Our research evaluates performing intelligent distributed fusion data dissemination at the network layer. We use clustering at the network layer to minimize the problems associated with application layer techniques.

We implemented advanced techniques at the network layer to improve dissemination of track fusion data. The techniques move a portion of the application layer logic into the network. The logic was implemented using software on a general purpose CPU and on new network hardware. Predicting target position either at the application or network layer is difficult. Moving key application logic such as target projection is key to enable decisions at the network layer. The real time hardware accelerates the decision making and distribution logic. Our implementation simply uses kinematic data for the target projection. Reduction of the data while selecting the right data to send in real time is critical to developing both a realistic and scalable solution.

Two main distributed track fusion messages (initialize and update tracks) containing kinematic data were provided to the network layer. Our generated distributed track messages simply contain position, velocity and relative time of the measurement information. We assume the network layer has the ability to parse the messages to assess the distance and time characteristics.

Our distance and time dissemination algorithms attempt to map the target prediction and dissemination application layer functions efficiently to the network layer. The target prediction and dissemination are broken into distance and time functions. The distance (spatial) algorithms (target projection, distance of observation to each track observation, and track association) assess the value of each track observation by adding logic to make



Cluster Table

	Position		Velocity	
	x	y	vx	vy
Cluster 1	2	6	1	3
Cluster 2	0	0	3	4
Cluster 3	12	34	8	2
:	:	:	:	:
Cluster n	56	2	14	12

Figure 7.4: Tracks mapped into L dimensional vectors are clustered into groups of current tracks

real time data dissemination decisions. Our time algorithms (prioritization of targets, update rate decisions) evaluate the required update frequency for dissemination based on key target characteristics accessible at the network layer.

The distance or spatial clustering algorithm (SCA) requires three steps. The algorithm for track clustering operates as follows:

1. Calculate distances from incoming observation to predicted target position for each track.

2. Determine if there is a projected track PT_{min} close enough to have a distance below the cluster threshold.

   (a) If PT_{min} exists, assign track $\vec{t}$ to centroid and update the position of PT_{min} based on the velocity.

   (b) If PT_{min} does not exist, add the track $\vec{t}$ as a new track.

We first project the target (PT) using the stored kinematic observation track data, as shown in Figure 7.4, for each track (cluster) using following equation:

$$PT = stored(X, Y) + (timeinc) * (currentvelocity) \tag{7.1}$$

We use the target projection similar to the centroid (mean value for all the objects in the cluster). Second, the distance from the observation to the projected target is calculated.

Each observations (received at different times t1, t2, t3,...) is compared to every track (cluster) using the PT equation. Third, if the distance is less than the clustering threshold, the observation is selected to send. The clustering threshold is currently set based on empirical data.

Our temporal clustering algorithm (TCA) evaluates velocity of the observation then prioritizes it to select the correct update rates. The update rate is varied based on the priority level of the target. We cluster the data using three priority levels; high (e.g. missiles traveling at Mach 2 or higher), medium (e.g. aircraft traveling between Mach 1 and Mach 2) and low (e.g. trucks, ships traveling below Mach 1). We currently use static thresholds for the prioritization. Based on experience, the update rates for the high priority is set at 6 hertz, medium priority is set to 4 hertz and low priority is set to 2 hertz. TCA next compares the relative system time associated with last observations sent to decide whether the current observation should be sent. The observation won't be sent unless the update rate set for the target has been exceeded, even if the observation is below the clustering threshold (close to the projected target). Our experimental software and hardware required threshold to set key clustering and dissemination parameters. The three main thresholds include the clustering threshold which determines the minimum acceptable distance from the PT for association with a know track, the time increment determines the next time the track observation should be sent and the two velocity thresholds partition the track observations into high, medium and low priorities. The velocity thresholds determine the frequency of disseminating the track observations. The thresholds were used to provide clustering design and development constraints used in simulation, emulation and hardware evaluation.

Currently the SCA and TCA algorithms operate independently but are sequentially linked. The sequentially linked SCA and TCA determines if the observation should be used only at the local node or disseminated to other nodes.

7.3.3 Intelligent Gateway Node Clustering

The SCA or TCA algorithms were implemented in both hardware and software. To minimize the latency required to cluster data and maximize throughput, we parallelize the processing of the TCA and SCA algorithms using Field Programmable Gate Array (FPGA) technology in the Intelligent Gateway Node (IGN). The IGN implemented the SCA and TCA algorithms above using the NetFPGA platform. The NetFPGA is an open source project that allows researchers to develop network applications and systems [40]. Many projects including Secure Switches (Ethane) [43], Routers, and Rate Control Protocol (RCP) [20] are also using this hardware.

The main three hardware modules are Track Cluster, Time Compare, and Update. The Track Cluster module calculates the Manhattan distance to the PT and maintains a list of current clusters/tracks. The Time Compare module determines the priority and whether to send or drop the data. The Update module computes the projection of the target (PT) for clustering the track data and passes the PT to the Track Cluster module. The prototype modules are implemented and demonstrated on an open platform called the NetFPGA. The NetFPGA processes the MTT data as it is received in real time over a network.

For the development of our network layer solutions, the NetFPGA contains a Xilinx Virtex-II Pro FPGA. Real time decision making is made possible leveraging the board's Double Data Rate (DDR2) SDRAM device, two SRAM devices, two serial ATA (SATA) connectors, and quad-port physical-layer transceiver. The NetFPGA library provides a Verilog template for design that interfaces to the memory devices and the network interfaces [40] for ease of design.

Deploying an intelligent gateway at strategic locations will significantly improve the overall network performance for distributed fusion. Advanced SA requires the use of high speed networks with increased bandwidth to implement centralized track fusion to improve tracking performance. Distributed track fusion can process the data locally and only exchange key observations to improve SA. It is critical to select the correct distance and clustering methods to ensure the right data arrives at the right time to the distributed nodes.

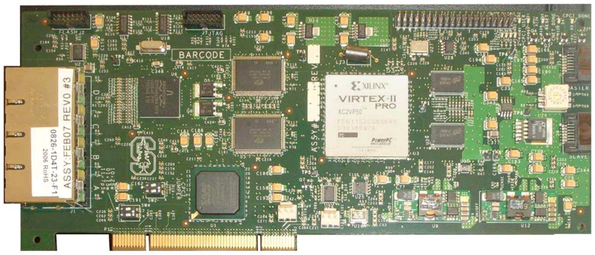


Figure 7.5: The NetFPGA platform used to implement Track Clustering

The Emulab experiments indicated that the general purpose computer could not keep up with the network traffic. The intelligent gateway node hardware demonstrated real time performance at wired line rates was possible. The hardware design implemented each of the functions (target projection, distance of observation to each track observation, and track association,prioritization of targets, update rate decisions)in modules that leverages parallel operations to achieve very high rate decision making performance.

The IGN achieved a significant improvement over traditional software processing methods implemented at the application layer. Our parallel hardware design can perform 4 simultaneous distance metric measurements with up to 100 simultaneous tracks operating at a clock speed of 125 MHz. The total time required for distance calculations, assignment determination, and updates requires 0.904 μs for each incoming track which creates a real time throughput of approximately 1.1 million packets per second.

7.4 Hardware Design

The track clustering algorithm performs four primary operations (1) calculating the distances between observations and centroids, (2) identifying if the incoming observation maps to an existing centroid, (3) updating the centroid or creating a new cluster, and (4) determining if the track should be dropped or sent based on the timetable. The hardware implementation is comprised of three primary modules: Track Cluster, Time Compare, and Update. In addition to these modules, control modules were needed to load and run the hardware clustering system. Figure 7.6 shows the architecture of the track clustering system.

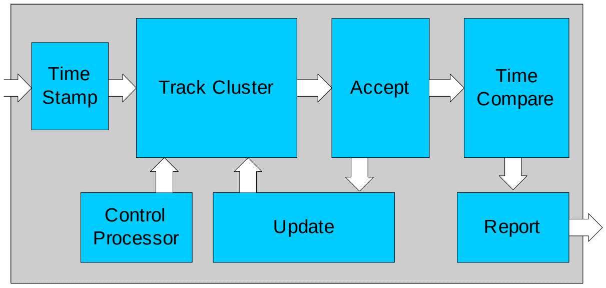


Figure 7.6: Hardware Track Clustering Block Diagram

7.4.1 Time Stamp

As the track data enters the system it is provided with a time stamp. The time compare module uses the 32-bit time stamp to determine if the track should dropped or sent based on the defined update rates set by the time threshold.

7.4.2 Track Cluster

A set of smaller modules comprise the track cluster function. The first module calculates the distance and is designed in a modular fashion so that it can be replaced with diverse metrics implemented in hardware. The Manhattan distance calculation is replicated in parallel to evaluate multiple distances simultaneously.

The second component is the cluster table. The cluster table maintains a list of current clusters/tracks locally on the FPGA chip. The centroids of the clusters represent the projected position of the track for prioritization. This projected position determines if an incoming track matches a cluster. The calculated parallel distances are passed to the accept module to make the critical decision to forward the data.

7.4.3 Accept

The accept module compares all parallel distances to a cluster threshold. If the distance is less than the cluster threshold, the module will assign the incoming track to the specified cluster. The module will not assign the incoming track if the distances are greater than the cluster threshold. If this occurs, the system currently decides the incoming track is a new cluster and will specify that a new entry needs to be added to the cluster table.

7.4.4 Update

After the accept module determines if an assignment is accepted, the update module then updates the cluster table. The update is computed from the velocity information contained in the track data and determines the projected position of the track. The projected position is passed to the cluster table (in the track cluster module) for storage. The accept module determines whether the track is a new track and needs to be added to the set of known clusters. The update module will calculate the projection of the track and then send the information to the cluster table to be stored.

7.4.5 Time Compare

The time compare module determines if the track should be dropped or forwarded to downstream modules/systems. The module is comprised of a timetable that maintains two time values: the last time the cluster/track was reported and the last time a track message was received. A time threshold determines if a track message should be sent or aggregated. If the difference between the last sent time and the current time is greater than the threshold, the data is sent from the node. However, if the difference is less than the time threshold the data updates the cluster table however the data is not sent from the node.

The timetable also contains information regarding the velocity and the accuracy of the incoming track data. This data is used in conjunction with the time data to determine whether the system sends or drops the data. The velocity information is used to determine the three time categories or priorities of traffic: high, medium and low.

7.5 Clustering Observation Data

The track data is part of the network traffic flowing to and from each clustering node. Each track observation has a current position and velocity in the xy dimension. The clustering system can be expanded into a larger dimensional space, although the dimensionality is set to two currently.

The clustering nodes are interconnected using a network where multiple sensors on different platforms report the same track information but may have different accuracies. These tracks are similar representations of the same information, but are received at multiple clustering nodes at different times due to network traversal. The clustering nodes maintain information on the current clusters (active tracks) and cluster centroids. In addition to the position of the track, the system uses time stamps to determine the last time a cluster was reported.

The clustering system uses a modified algorithm similar to k-means. In traditional k-means, the number of clusters (K) is set to a specified number. We allow the number of clusters to start from zero and expand as new tracks are found.

The algorithm utilizes two tables to achieve the desired functionality. The first table, the cluster table, is comprised of the current clusters that are projections of where the track should be in the next track message. As tracks are received, they are compared to the clusters using the Manhattan distance. If the distance is less than a threshold (cluster threshold), the track message is assigned to the specified cluster. The velocity of the track message updates the projection of the cluster. This projection update allows the system to account for tracks that are not static. If the distance from the input track is greater than the threshold for all the current clusters, a new cluster is added.

The second table, the track time table, maintains two times for each cluster: the last time a message was sent about that cluster and the last time a message was received about that cluster. Since the track data is time stamped when the clustering system receives the track, the system compares the time received to the time sent. If this time is greater than the timing threshold the system will allow the message to pass and update the cluster received time in the table. However, if the time is less than the time threshold, the system updates the cluster received time and removes the message from the outgoing traffic.

7.6 Hardware Fabrication

The implementation on the NetFPGA platform was able achieve a clock frequency of 125 MHz. The hardware utilization of a track clustering algorithm with four parallel distance metrics is shown in Table 7.1. With a slice utilization of 44% the number of distance metrics can be increased. This increase in parallel distance calculation reduces the latency to compare incoming tracks to all known clusters.

Resources	**XC2VP50 Utilization**	**Utilization Percentage**
Slices	10533 out of 23616	44%
4-input LUTS	14318 out of 47232	30%
Flip Flops	12958 out of 47232	27%
Block RAMs	82 out of 232	35%
External IOBs	353 out of 692	51%

Table 7.1: Device utilization for XC2VP50 Hardware Track Clustering with four concepts

Given our implementation with four parallel distance metrics with 100 total tracks in the cluster table at any one time, we can estimate the total time for clustering an incoming track. The parallel distances can be calculated in four cycles. The total number of cycles required to produce all 100 distances would be 100. Since the operation of the clustering circuits are pipelined, the accept module would only require three cycles for determining the correct action to perform. The update module would require three cycles to perform an update or creation of a new cluster. The time table compare takes three cycles. Add in an additional four cycles for header processing and we have a total of 113 cycles. Since we are running the hardware at 125 Mhz, the total time required for distance calculations, assignment determination, and updates requires 0.904 μs for each incoming track. This gives us an approximate throughput of 1.1 million packets per second.

7.7 Simulated Experiments

Our large scale experiments help to quantify the distributed track fusion dissemination bandwidth load and the latency related to available link capacity. We use software to cluster the track observations in the large scale experiment to reduce cost and compare the IGN with general purpose CPU performance.

The first set of experiments involved measuring the amount of bandwidth created in transmitting track data to a distributed set of nodes using application layer (JXTA) and network layer (Multicast, broadcast) protocols. The protocols tested were Ethernet Broadcast, IP Multicast, and a P2P Multicast called JXTA. Four different sized star topologies were deployed; 10, 25, 50, and 75 nodes. The experiments were performed in an emulation environment called Emulab, on an emulation testbed with over 300 nodes available for researchers at the University of Utah.

The first set of experiments investigated the amount of latency experienced by a single node receiving track data. This experiment inserted a time stamp on packets leaving the sender. The receiver then compares the packet time stamp against its current time. The network time protocol (NTP) synchronized clocks on all machines in the experiment.

Figure 7.8 illustrates a sample topology of the initial experiment. In this figure, the ten node experiment deployed an additional Gateway Node (GW) that enabled multicast routing in the Emulab environment. Each node sent sensor track data to all other nodes in the network using IP Multicast. A Boeing track generator at each node created the tracks. The generator creates targets of interest and records their position as they change over time. Modifications were made to the track data generation software to provide simple X, Y coordinates and X,Y velocities. The track data also contained an identifier field describing the type of track data (initialize or update) and packet length. Each experiment lasted approximately 20 minutes.

Figure 7.7 contains the graph demonstrating an increased latency experienced at the GW node disseminating track data. As the networks grows from 10 to 68 nodes, the amount of track data increases as well as the latency for a single track to be processed. The maximum latency varied from 109 to 260 ms as the number of nodes increased. The average latency ranged only from 103 to 109ms. This illustrates application layer protocols significantly load bandwidth and increase latency making scalability challenging.

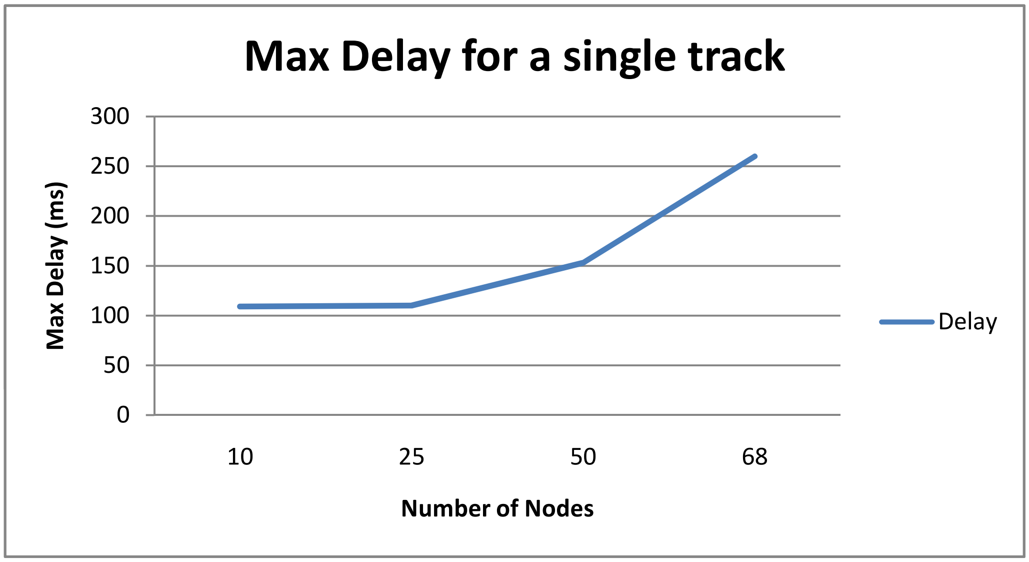


Figure 7.7: Maximum Latency to Receive a packet over a 100 ms link.

In the next experiment, we evaluate using clustering algorithms to to identify similar track data and match the available bandwidth. Each packet that arrives at the gateway node is clustered and compared to packets previously clustered. If a packet is similar to a one previously clustered and recently forwarded to the neighboring network, it is considered redundant and discarded. The nodes topologies consisted of 25, 50 and 75 nodes, each sending track data to all nodes in the network, including a gateway clustering node.

In Figure 7.8, the GW node connects to a neighboring network or node. Our experiments next deployed a single node connected to the other end of the GW with a 1 Mbit/sec link had a latency of 100 ms. Figure 7.9 illustrates the number of packets that the gateway node receives and clusters packets that are forwarded. The amount of packets it forwards varies from 1130 to 9400 packets. This is a considerable reduction in the network load because the amount of packets it receives and clusters ranges from 187,000 to 435,000 packets. Figure 7.10 displays the amount of bandwidth used in terms of megabytes processed in and out of the gateway node. Again, significant reductions in bandwidth are demonstrated

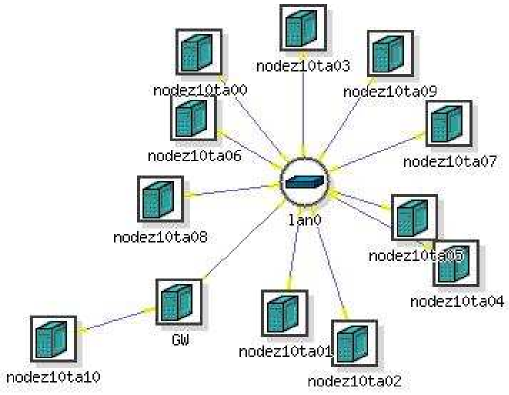


Figure 7.8: 10 node Emulab Experiment with a Gateway Cluster Node and Neighbor Link.

using clustering algorithms. The incoming amount of data varies from 40 to 90 Mbytes with the clustering gateway node reducing the traffic to 1.9 to 2.1 Mbytes. This provides an 18 to 48x reduction in bandwidth before traversing to the low capacity neighboring link.

The last experiment performed investigates the amount of packet loss experienced by the software clustering implementation. The clustering was developed in Java and operated on a 3 GHz Pentium 4 CPU using a Linux operating system.

Packet loss is due to the slow sequential processing required for the clustering algorithm. Processing compares the clustered hash value of each incoming packet against the know hash values of the packets in memory. As bursts of packets arrived, the CPU was unable to investigate all of the packets before buffers overflowed. Since the track data is assuming the use of an unreliable communication protocol, no retransmission is issued for track data and it is lost. The hardware implementation of this clustering algorithm will not suffer the same speed limitations of software. Hardware maps the clustering algorithm onto an FPGA, that allows for significant parallelism. The results of the packet loss at the gateway node are presented in Figure 7.11. The number of packets that are unable to be clustered varies from 27,000 to 209,000 packets.

Figure 7.9 illustrates the amount of traffic in and out of the gateway node. The average packet size received at the gateway is 217 bytes. Each node generates 9,090 packets.

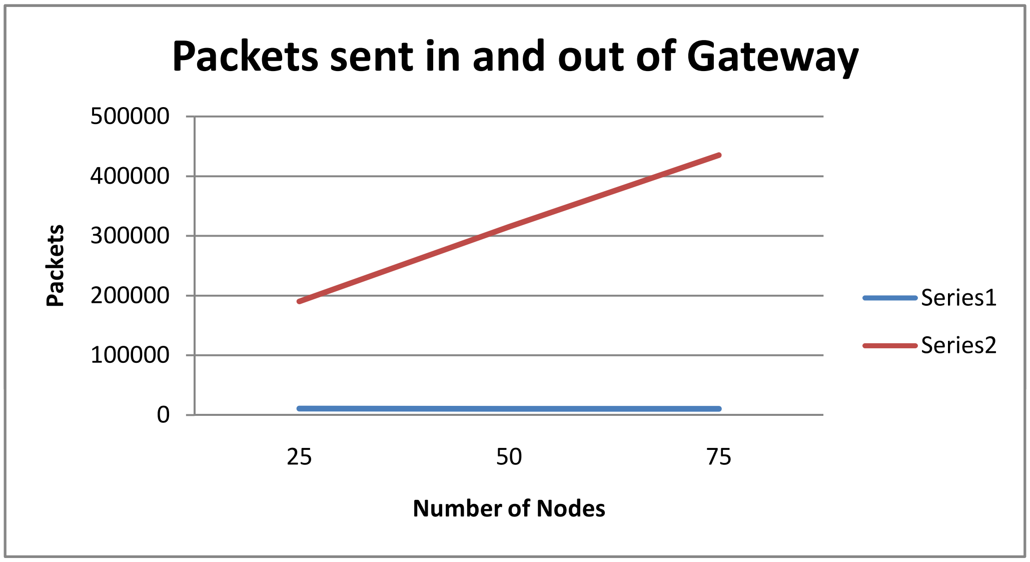


Figure 7.9: Performance of Clustering Algorithm in Software (Packets).

In the 75 node example, 9090 x 75 = 681,750 packets are destined for the gateway node or 138,087,788 bytes. The gateway was only able to receive 434,995 packets, and the others were dropped due to an overloaded CPU. The total traffic received at the gateway was 217 * 434,995 = 94,393,915 bytes or approximately 94 Mbytes. Testing confirms that even a 3 Ghz CPU could not keep up with receiving and clustering of the smaller multicast packets which necessitates the use of FPGA or ASIC technology.

7.8 Conclusions

The results show that the real time hardware using the K-mean algorithm for clustering can accurately locate redundant track data for aggregation, identify new tracks, and select the critical tracks to be forwarded to other distributed sensor nodes for fusing. Improvements in the information quality and latency for distributed track fusion were demonstrated using advanced clustering.

Track data distributed using multicast is shown to generate 2.3 Gigabytes of traffic for large scale (75 nodes) fusion. Experiments revealed state-of-the-art CPUs could not

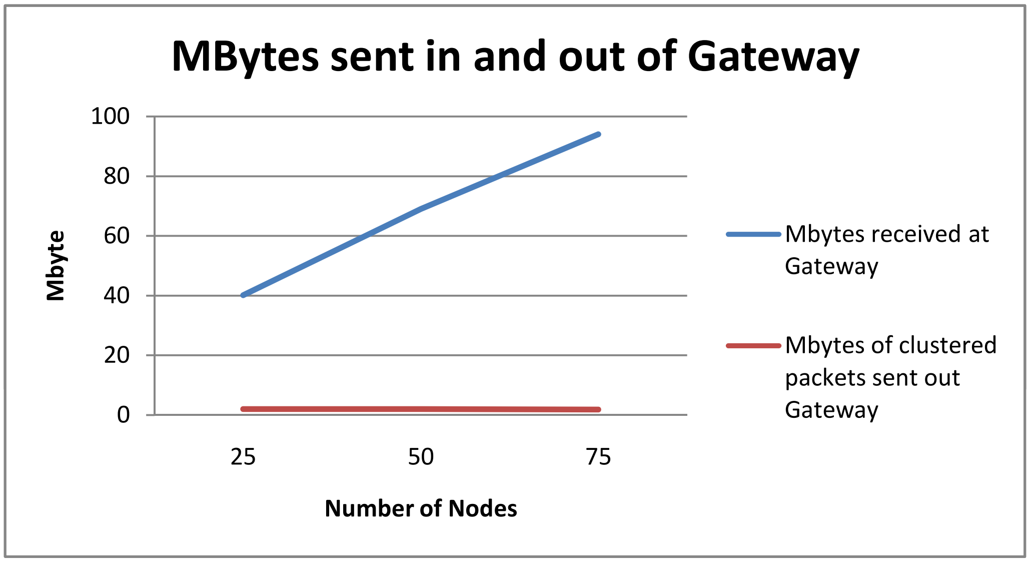


Figure 7.10: Performance of Clustering Algorithm in Software (Mbytes).

handle bursts of packets received which resulted in loss of packets due to network buffer overflow. The clustering gateway node was shown to provide an 18 to 48x reduction in bandwidth through eliminating redundant data. The results were tested in a distributed environment called Emulab that used a software version of the clustering algorithms. State-of-the-art Emulab CPUs could not keep up with the track data resulting in network buffer overflow and loss of packets. This illustrates the need for our special processing solution using FPGA technology.

Real time clustering is implemented in the network layer (OSI layer 3) to reduce the bandwidth intelligently while maintaining high information content. Our pipelined hardware design calculates four parallel distance metrics for 100 total tracks in 100 cycles. The total time required for distance calculations, assignment determination, and updates is only 0.904 μs for each incoming track. The hardware solution is prototyped using the Stanford NetFPGA in the Boeing Center for Intelligent Networked Systems (CINS) lab.

Our real time solution preserves Layer 7 resources and decreases latencies. The ability to add real time hardware in the network layer improves MTT performance in bandwidth limited environments ultimately preserving legacy avionic network resources.

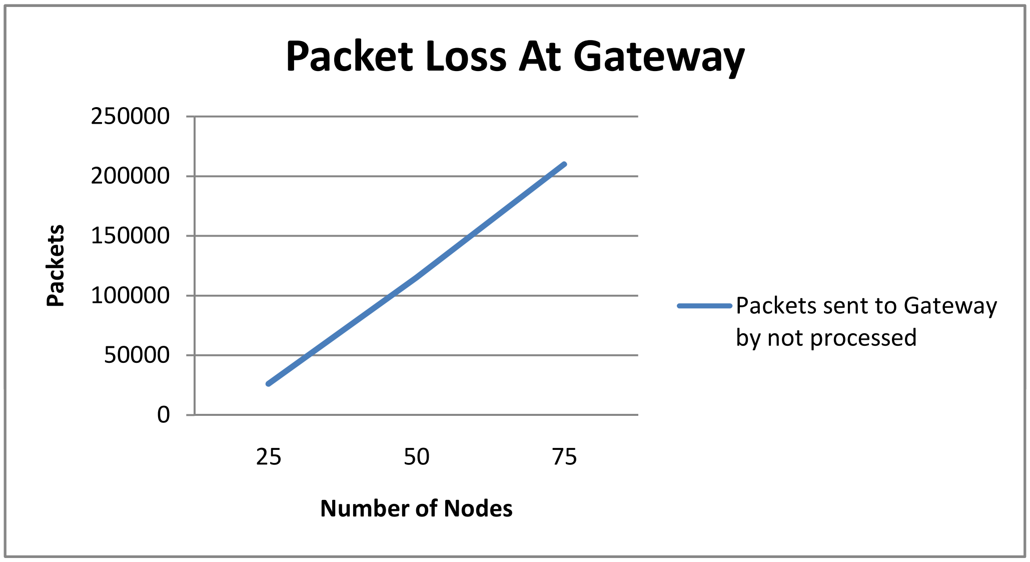


Figure 7.11: Packet Loss experienced at Gateway Cluster Node due to Software Clustering.

7.9 Future Work

Novel methods to illustrate the increased value of information using clustering over normal queuing methods was constructed based on spatial methods. Our use of clustering algorithms was fairly limited with k-means however there are plans to implement solutions based on N-means algorithms in the future. We realize selection of the correct clustering thresholds must be dynamic based on the separation of targets and is highly dependent on accurately projecting the centroid. The relationship between our temporal and spacial clustering algorithms must be integrated into a single solution. We plan to develop dynamic clustering thresholds for more accurate prediction of target paths while selectively reducing the distribution bandwidth. Real time assessment of the information value allows us to dynamically adjust the thresholds for preserving the key observations. We theorize that changing the update rates proportionally with the target velocity will provide improved information value. The proximity of multiple tracks and the sensor accuracy must determine the optimal value of the dynamic threshold setting. We also plan to add a weighting factor to take into account proximity, sensor accuracy and threat level will provide a more intelligent solution. Interconnecting the time increment used in the projection of the clustering centroid with the temporal target rate will simplify the algorithms and hardware.

Reliable transfer of data using wireless is available in several avionic systems and will be evaluated as a separate test case. Latency will definitely increase if all dropped packets are transferred as required by reliable transport.

Information content is improved by identifying key target characteristics such as turning ratio, threat level and multi-target separation. Increasing information content while decreasing bandwidth is the goal of the Boeing Intelligent Gateway (BIG) being developed. BIG is an intelligent gateway that uses a highly parallel state machine to implement a set of distributed services such as intelligent data association for improving the quality of information with reduced bandwidth.

Chapter 8

Conclusions and Future Research

8.1 Summary

The SPOT system was developed out of a need to assign SN functionality in avionic networks, however it became apparent that other applications stand to benefit from it as well. Leveraging latency sensitive applications helped to clearly motivate the benefits of such a system. Applications such as game servers for First Person Shooters and relays for Voice over IP communication seem well positioned to benefit from this work. Also, with minor changes to the constraints associated with SPOT, other bandwidth sensitive applications can take advantage of dynamic distributed SN placement.

This research embraced prior work, refined the ideas to a slightly different application and then improved upon that work to create a better system. The dynamic aspects of *r*-SPOT serve an important role for ever complex networks. The development of SPOT allows for further advancement in distributed systems research. In undertaking this task, real world issues (such as Triangle Inequality Violations) were observed and addressed. Also the SPOT system stands as a tool to serve other researchers interested in building on top of it for more advanced experimentation.

Though the initial emulation experiments the technical and intellectual contributions flourished. Emulation demonstrated the ability for this distributed algorithm to truly scale for hundreds of nodes in a distributed system. The use of Planetlab also helped illustrate the capabilities of a heterogeneous system with a range of computation and communication capabilities. Using data gathered from Planetlab nodes provided insight into how an application can successfully leverage the placement service that SPOT provides. For these latency sensitive applications such as a First Person Shooter game server, SPOT allows can provide an improved game play experience.

A description of the formal problem of SN placement was described in Chapter 3. Here a generic placement model was developed with a network consisting of nodes and routers. Using this simple model various properties of the placement problem were identified, a cost model was presented along with bounds on the cost as the network increases in size. Finally a dynamic distributed placement algorithm called r-mod was presented.

In Chapter 4 the dynamic distributed placement algorithm was evaluated with software system called SPOT. Using emulation, initial results demonstrated an ability to improve the cost of placement even further when multiple SNs were considered. Enhancements to the original algorithm were then proposed and analyzed demonstrating reduced costs for multiple SNs.

Chapter 5 detailed the deployment of improved algorithms of r-SPOT in diverse environments. Larger topologies were explored using an Internet topology generator along with SPOTSim, the Java simulator that mimics the behavior of SPOT. The creation of the SPOTSim simulator allows researches to discover the impact of different placement algorithms on a much wider set of network topologies. By building off of the open source GLPK solver, a system exists to experiment with SN placement on a range of environments.

In Chapter 6, the dissertation investigated the performance of P2P service deployment in contrast to a more traditional client/server service model. Here an avionic satellite network was evaluated using the Emulab testbed environment. From those experiments, reductions in latency and network traffic were observed as the location of the network services were located on nodes utilizing the service.

Finally in Chapter 7 a hardware accelerated SN was presented and analyzed. The SN was developed on an FPGA and provided a clustering service used to recognize targets of interest in an avionic network environment. The hardware accelerated SN along with a software SN were evaluated to determine the performance capabilities of each. The hardware version demonstrates much greater throughput and reduced latency compared to the software version.

8.2 Future Work

A few different areas of research emerge for future work. First, the study of more dynamic aspects of SPOT. Provided now are a few bullet points on different areas of research.

- Explore additional dynamic aspects of the system
  - Investigate nodes joining and leaving (churn)
  - Evaluate the effects of dynamic neighborhood sizing to determine an appropriate size per network topology
- Develop incremental SN placement
  - Current results motivated through the amount of time required to assign SNs in a large topology
  - Develop algorithms to determine when a portion of the network needs reconfigured
- Deploy SPOT as a generic service
  - Allow SPOT to support more than one application using services
  - Implement a richer set of constraints such such CPU load and available memory

While SPOT currently supports a dynamic nature in regards to the topology growing in size, additional dynamic aspects are of interest. Additional research into nodes joining and leaving the network (churn) will improve our understanding of this large dynamic system to deal with rapid changes. Also, the study of incremental updates to the SN placement shows promise. From the current research experiments helped establish the amount of time necessary to reconfigure the entire network. Depending on the application, this time requirement may become a bottleneck. We need to investigate strategies to break up the problem and reconfigure just a single neighbor or set of neighborhoods may be appropriate. Mechanisms to detect the churn associated with nodes joining and leaving the network can be used to determine which portions of the network stand to benefit from a reassignment of SNs.

An additional area of research is the interaction between applications and SPOT is also of interest. SPOT is a useful open-source tool for application developers to build on top of. Deploying SPOT as a generic service for multiple applications creates a shared infrastructure supporting a range of applications.

References

[1] Shilpi Agarwal, Joel Sommers, and Paul Barford. Scalable network path emulation. In *MASCOTS '05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 219–228, Washington, DC, USA, 2005. IEEE Computer Society.

[2] Grenville Armitage and Philip Branch. Distribution of first person shooter online multiplayer games. *International Journal of Advanced Media and Communication*, 1(1):59–75, October 2005.

[3] Grenville Armitage, Mark Claypool, and Philip Branch. *Networking and Online Games:Understanding and Engineering Multiplayer Internet Games.* John Wiley & Sons, June 2006.

[4] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. In *Proceedings of 33rd ACM Symposium on Theory of Computing, 2001*, 2001.

[5] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In vini veritas: realistic and controlled network experimentation. *SIGCOMM Comput. Commun. Rev.*, 36(4):3–14, 2006.

[6] Terry Benzel, Robery Braden, Anthony Joseph, Keith Sklower, Ron Ostrenga, and Stephen Schwab. Experience with DETER: A Testbed for Security Research. In *Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom2006)*, Barcelona,Spain, March 2006.

[7] B. Beverly Yang and H. Garcia-Molina. Designing a super-peer network. pages 49–60, March 2003.

[8] Danny Bickson and Dahlia Malkhi. The Julia Content Distribution Network. In *The 2nd Usenix of Real World Distributed Systems (WORLDS'05)*, 2005.

[9] Samual S. Blackman. Multiple-target tracking with radar applications. In *Artech House*, pages 357–395, 1986.

[10] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Exploiting network proximity in distributed hash tables. In Ozalp Babaoglu, Ken Birman, and

Keith Marzullo, editors, *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 52–55, June 2002.

[11] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[12] Moses Charikar, Sudipto Guha, Eva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 1–10, 1999.

[13] Julia Chuzhoy and Yuval Rabani. Approximating k-median with non-uniform capacities. In *SODA '05: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 952–958, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[14] G. Adam Covington, Charles L.G. Comstock, Andrew A. Levine, John W. Lockwood, and Young H. Cho. High speed document clustering in reconifigurable hardware. In *16th Annual Conference on Field Programmable Logic and Applications (FPL)*, pages 411–417, Madrid, Spain, August 2006.

[15] John DeHart, Fred Kuhns, Jyoti Parwatikar, Jonathan Turner, Charlie Wiseman, and Ken Wong. The Open Network Laboratory. In *SIGCSE '06: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 107–111, New York, NY, USA, 2006. ACM Press.

[16] John D. DeHart, William D. Richard, Edward W. Spitznagel, and David E. Taylor. The smart port card: An embedded Unix processor architecture for network management and active networking. Technical Report WUCS-01-18, Applied Research Laboratory, Department of Computer Science, Washington University in Saint Louis, August 2001.

[17] B. Doshi, L. Benmohamed, and A. DeSimone. A Hybrid End-to-End QoS Architecture for Heterogeneous Networks (Like The Global Information GRID). In *Military Communications Conference*, October 2005.

[18] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, March 1973.

[19] Jonathon Duerig, Robert Ricci, Junxing Zhang, Daniel Gebhardt, Sneha Kasera, and Jay Lepreau. Flexlab: A realistic, controlled, and friendly environment for evaluating networked systems. In *Record of the Fifth Workshop on Hot Topics in Networks (HotNets V)*, pages 103–108, Irvine, CA, November 2006.

[20] Nandita Dukkipati, Glen Gibb, Nick McKeown, and Jianying Zhu. Building a RCP (Rate Control Protocol) Test Network. In *Hot Interconnects*, pages 91–98, Stanford, CA, August 2007.

[21] Mike Estlick, Miriam Leeser, James Theiler, and John J. Szymanski. Algorithmic transformations in the implementation of k- means clustering on reconfigurable hardware. In *FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 103–110, New York, NY, USA, 2001. ACM.

[22] A Tchamova et al. Target Tracking with generalized Data Assocoation based on the General DSM Rule of Combinaton. In *Proceedings of Fusion 2004*, pages 91–98, Stockholm, Sweden, 2004.

[23] M. R Chummun et al. Fast data association using multidimensional assignment with clustering. In *IEEE Transactions On Aerospace and Electronic Systems*, pages 898–912, Vol. 37, No 3, 2001.

[24] GLPK - GNU Linear Programming Kit. `http://www.gnu.org/software/glpk/`.

[25] Gnutella: Distributed Information Sharing. `http://www.gnutella.com`.

[26] Li Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, 2001.

[27] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, may 1964.

[28] D. L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[29] G. Handler and P. Mirchandani. *Location on Networks Theory and Algorithms.* The MIT Press, 1979.

[30] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the Emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

[31] Juniper M7i Intrusion Detection System. `http://www.juniper.net/products/-mseries`.

[32] Kamal Jain and Vijay V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[33] Brad Karp, Sylvia Ratnasamy Sean Rhea, and Scott Shenker. Spurring Adoption of DHTs with OpenHash, a Public DHT Service. In *In IPTPS*, 2004.

[34] Nikolaos Laoutaris, Georgios Smaragdakis, Konstantinos Oikonomou, Ioannis Stavrakakis, and Azer Bestavros. Distributed placement of service facilities in large-scale networks. In *IEEE Infocom 2007*, Anchorage, AK, May 2007.

[35] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. Measuring bandwidth between planetlab nodes. In *Passive and Active Measurement Workshop (PAM 2005)*, Boston, MA, March 2005.

[36] Miriam E. Leeser, James P. Theiler, Michael Estlick, Natalya V. Kitaryeva, and John J. Szymanski. Effect of data truncation in an implementation of pixel clustering on a custom computing machine. *Reconfigurable Technology: FPGAs for Computing and Applications II*, 4212(1):80–89, 2000.

[37] Shiding Lin, Aimin Pan, Rui Guo, and Zheng Zhang. Simulating Large-Scale P2P Systems with the WiDS Toolkit. In *MASCOTS '05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 415–424, Washington, DC, USA, 2005. IEEE Computer Society.

[38] Virginia Lo, Dayi Zhou, Yuhong Liu, Chris GauthierDickey, and Jun Li. Scalable supernode selection in peer-to-peer overlay networks. *hot-p2p*, 00:18–27, 2005.

[39] John W Lockwood. An open platform for development of network processing modules in reprogrammable hardware. In *IEC DesignCon'01*, pages WB–19, Santa Clara, CA, January 2001.

[40] John W. Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, and Jianyin Luo. Netfpga - an open platform for gigabit-rate network switching and routing. In *International Conference on Microelectronic Systems Education*, 2007.

[41] Moshe Looks, Andrew Levine, G. Adam Covington, Ronald P. Loui, John W. Lockwood, and Young H. Cho. Streaming hierarchical clustering for concept mining. In *Aerospace Conference (AERO)*, pages 1–12, March 2007.

[42] Cristian Lumezanue. Triangle inequality and routing policy violations in the internet. In *Passive and Active Network Measurement (PAM)*, 2009.

[43] Jianyin Luo, Justin Pettit, Martin Casado, John Lockwood, and Nick McKeown. Prototyping Fast, Simple, Secure Switches for Ethane. In *Hot Interconnects*, Stanford, CA, August 2007.

[44] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbors neighbor: the power of lookahead in randomized P2P networks. In *36th ACM Symposium on Theory of Computing (STOC)*, pages 54–63, 2004.

[45] Carlo Mastroianni, Domenico Talia, and Oreste Verta. A super-peer model for building resource discovery services in grids: Design and simulation analysis. In *Advances in Grid Computing - EGC 2005*, pages 132–143, 2005.

[46] Alberto Medina and John Beyers. Brite: an approach to universal topology generation. *9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, August 2001.

[47] John Meier, Todd Sproull, Adam Covington, and John Lockwood. Intelligent avionics with advanced clustering. In *IEEE Aeropace*, Big Sky, MT, March 2008.

[48] Shashidhar Merugu, Sridhar Srinivasan, and Ellen Zegura. p-sim: A simulator for peer-to-peer networks. *MASCOTS*, 00:213, 2003.

[49] John Montgomery. The orbiting internet fiber in the sky. *BYTE*, 22(11):58–72, 1997.

[50] Moni Naor and Udi Wieder. Know thy neighbor's neighbor: Better routing for skip-graphs and small worlds. In *Proceedings of IPTPS, 2004*, pages 269–277, 2004.

[51] Napster. `http://www.napster.com`.

[52] T. S. Eugene Ng, Yang hua Chu, Sanjay G. Rao, Kunwadee Sripanidkulchai, and Hui Zhang. Measurement-based optimization techniques for bandwidth-demanding peer-to-peer systems. In *Proceedings of IEEE INFOCOM*, pages 2199–2209, 2003.

[53] The Network Simulator ns-2. `http://www.isi.edu/nsnam/ns`.

[54] P2PSim. `http://pdos.csail.mit.edu/p2psim/`.

[55] PeerSim. `http://peersim.sourceforge.net/`.

[56] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[57] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, New York, NY, USA, 1997. ACM Press.

[58] Pyxida: An open source network coordinate library and application. `http://www.-pyxida-.sourceforge.net/`.

[59] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker. On the placement of web server replicas. In *INFOCOM*, pages 1587–1596, 2001.

[60] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, University of California, Berkeley, Berkeley, CA, 2000.

[61] J. Reese. Solution methods for the p-median problem: An annotated bibliography. *Netw.*, 48(3):125–142, 2006.

[62] Shansi Ren, Lei Guo, and Xiaodong Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 70, Washington, DC, USA, 2006. IEEE Computer Society.

[63] Charles Revelle, David Marks, and Jon C. Liebman. An analysis of private and public sector location models. *Management Science*, 16(11):692–707, July 1970.

[64] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[65] Savage. The end-to-end effects of internet path selection. *SIGCOMM Comput. Commun. Rev.*, 29(4), 1999.

[66] Claude E Shannon. Communication in the presence of noise. *Proceedings of the IEEE, Vol. 86, No. 2*, pages 447–457, 1998.

[67] David B. Shmoys, Eva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[68] Skype Users Online. http://seekingalpha.com/article/-48825-Ebay-Update-Skype-Still-Growing/.

[69] Skype Official Website. http://www.skype.com.

[70] Georgios Smaragdakis, Vassilis Lekakis, Nikolaos Laoutaris, Azer Bestavros, John W. Byers, and Mema Roussopoulos. The EGOIST Overlay Routing System. In *Proceedings of ACM CoNEXT 2008*, Madrid, Spain, December 2008.

[71] Todd Sproull, John Meier, and John Lockwood. Management and service discovery in satellite and avionic networks. In *IEEE Aeropace*, Big Sky, MT, March 2007.

[72] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, pages 149–160, 2001.

[73] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *IEEE/ACM Trans. Netw.*, 12(2):205–218, 2004.

[74] S. Tao, K. Xu, A. Estepa, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z. Zhang. Improving VoIP quality through path switching. In *Proceedings of IEEE INFOCOM*, March 2005.

[75] Stuart Thorncraft, Hugh Outhred, and David Clements. Evaluation of open-source lp optimization codes in solving electricity spot markey optmization problems. In *Mini-Euro Conference on Operation Research Models and Methods in the Energy Sector*, September 2006.

[76] Tools for Peer-to-Peer Network Simulation. http://tools.ietf.org/irtf/-draft-irtfp2prg-core-simulators-00.txt.

[77] Trauma POD. http://www.darpa.mil/DSO/thrust/biosci/traumpod.htm.

[78] Bernard Traversat. Project JXTA 2.0 Super-Peer Virtual Network. http://www.-jxta.org/projects/.

[79] Wisconsin Advanced Internet Laboratory. http://wail.cs.wisc.edu.

[80] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[81] George Xylomenos, George Polyzos, Petri Mahonen, and Mike Saaranen. Tcp performance issues over wireless links. *IEEE Communications Magazine*, pages 52–58, April 2001.

[82] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment, 2003.

[83] Reich Zhao, Shin. Information-driven dynamic sensor collaboration for tracking applications. In *IEEE Signal Processing Magazine*, pages 1–9, 2002.

Vita

Todd S. Sproull

Education

Ph.D. Computer Engineering, Washington University, August 2009
M.S. Computer Science, Washington University, May 2002
B.S. Electrical Engineering, Southern Illinois University at Edwardsville, May 2000

Professional Societies

Institute of Electrical and Electronics Engineers
Eta Kappa Nu Electrical and Computer Engr. Honor Society

Conference Papers

John Meier, Todd Sproull, G. Adam Covington, John Lockwood. Intelligent Avionics with Advanced Clustering. *Proceedings of IEEE Aerospace Conference*, (Big Sky, MT), March 1-8, 2008.

Todd Sproull, John Meier, John Lockwood. Management and Service Discovery in Satellite and Avionic Networks. *Proceedings of IEEE Aerospace Conference*, (Big Sky, MT), March 3-10, 2007.

Todd Sproull, John Lockwood. Extensible Network Configuration and Communication Framework. *Proceedings of Seventh Annual International Working Conference on Active and Programmable Networks (IWAN 05)*, (Nice, France), November 21-23, 2005.

Todd Sproull, Gordon Brebner, Christoper Neely. Mutable Codesign for Embedded Protocol Processing. *Proceedings of Field-Programmable Logic and Applications (FPL)*, (Tampere, Finland), August 24-26, 2005.

Haoyu Song, Todd Sproull, Michael Attig, John Lockwood. Snort Offloader: A Reconfigurable Hardware NIDS Filter. *Proceedings of Field-Programmable Logic and Applications (FPL)*, (Tampere, Finland), August 24-26, 2005.

Todd Sproull, John Lockwood. Distributed Intrusion Prevention in Active and Extensible Networks. *Proceedings of International Working Conference on Active Networking (IWAN 04)*, (Lawrence, KS), October 27-29, 2004.

Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, John Lockwood. Deep Packet Inspection Using Parallel Bloom Filters. *Proceeding of Hot Interconnects 11 (HotI-11)*, (Stanford, CA), August 2003.

David E. Taylor, John W. Lockwood, Todd S. Sproull, Jonathan S. Turner, David B. Parlour. Scalable IP Lookup for Programmable Routers. *Proceeding of 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, (New York, NY), June 2002.

Todd S. Sproull, John W. Lockwood, David E. Taylor. Control and Configuration Software for a Reconfigurable Networking Hardware Platform. *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, (Napa, CA), April 2002.

Journal Papers

David E. Taylor, Jonathan S. Turner, John W. Lockwood, Todd S. Sproull, David B. Parlour. Scalable IP Lookup for Internet Routers. *IEEE Journal on Selected Areas in Communications (JSAC)*, Volume 21, Number 4, May 2003.

Patents

Methods and Apparatus for Detecting Predefined Signatures in Packet Payload using Bloom Filters. Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, John Lockwood. U.S. Patent #7,444,515, Issued October 28, 2008.

Service Placement, Sproull, Ph.D. 2009