Distributed Algorithms for the Placement of Network Services

Todd Sproull
Roger D. Chamberlain

Todd Sproull and Roger D. Chamberlain, “Distributed Algorithms for the Placement of Network Services,” in *Proc. of 2010 International Conference on Internet Computing*, July 2010.

Dept. of Computer Science and Engineering
Washington University in St. Louis

Distributed Algorithms for the Placement of Network Services

Todd Sproull and Roger D. Chamberlain
Computer Science and Engineering, Washington University in St. Louis, St. Louis, Missouri, USA

Abstract—*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. These services are often 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. In this work we develop distributed algorithms to assign nodes the role of a supernode. We first build on 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 then developed to assign nodes the supernode role.*

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 services 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 SN in these examples assumes some of the responsibilities of a traditional 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. Here, we focus on distributed algorithms for the placement of supernodes in P2P networks.

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. 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.

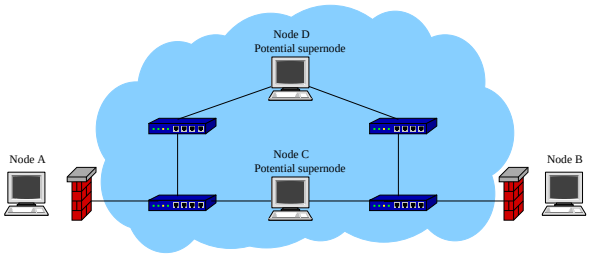

Fig. 1: Example topology of two nodes A and B communicating with VoIP using either supernode C or D to relay the conversation.

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.

2. Background

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 [1]. 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.

Placing SNs in a P2P network is more than just the k-median or facility location problem. In addition to 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.

2.2 Distributed Algorithm for Service Placement

Laoutaris et al. [2] describe a distributed algorithm for the k-median problem 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 (similar to a SN). 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 the rest of the nodes.

Laoutaris et al. performed Matlab simulations comparing the placement of the r-ball algorithm to the optimal placement of facilities. These simulation results serve as a starting point for this research. Our approach takes advantage of the limited horizon view similar to r-balls and extends it. Assumptions such as inferring node topology and communication with multiple SNs in a single r-ball are left unexplored in the previous work. This research embraces the limited horizon view of service placement and focuses on placement down to the node level.

3. Supernode Placement Problem

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 2. 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.

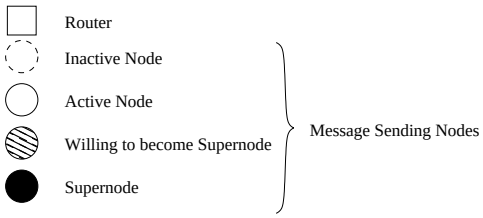

Fig. 2: Components in the network graph topology.

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. This topology contains three nodes A, B, and C and one router providing communication between the nodes. From Figure 3 we can see that Node A is one unit of distance from the router and two units from Node B.

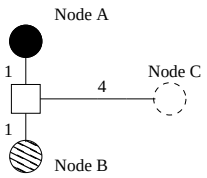

Fig. 3: Example topology with three nodes A, B, and C connected through a router. Nodes A and B are active and A is the SN.

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 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. In this initial example, 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 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 when 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) for a total cost of 7.

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.

4. 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

$$\forall S \in 2^{V_W}, |S| = k \rightarrow (\sum_{u \in V_A} t(u)d(u, V_k) \leq \sum_{u \in V_A} t(u)d(u, S)). \quad (1)$$

Finally we define a $TotalCost$ for any S where $|S| = k$ as the following

$$TotalCost = \sum_{u \in V_A} t(u)d(u, S) \quad (2)$$

This allows us to compare the cost of one SN assignment of S to the optimal assignment cost using SNs V_k.

5. 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. The SN placement completes once each SN reaches its locally optimal assignment, where any additional move does not lower the cost of the SN placement.

5.1 General Behavior

Nodes initially joining the network connect to a 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.

Figure 4 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

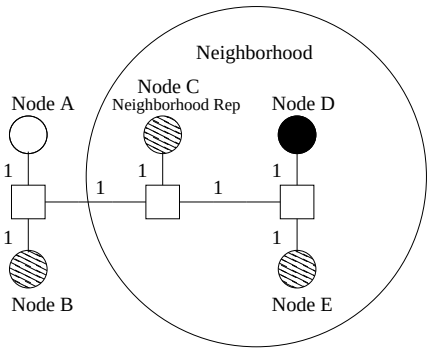

Fig. 4: Example neighborhood with SN at node D and neighborhood representative at node C with an r value of 3.

neighborhood (with distances of 3 and 2 to the SN, node D, respectively). Nodes A and B are too far from the SN and are 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. Any node not in the willing to become an SN state can not join the neighborhood 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.

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 problem while not completely ignoring nodes outside of the neighborhood by considering aggregate demand information.

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 introduced in the Background Section and will be referred to as r-mod (Algorithm 1).

We identify 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 $N^{(m)} = \bigcup_{v_i \in V_k^{(m)}} N_i^{(m)}, U^{(m)} = \bigcup_{v_i \in V_k^{(m)}} U_i^{(m)}$.

5.2 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 1 Placing SNs in a P2P Network (r-mod)

```
1:  ConnectToBootstrapNode()
2:  while V_k ≠ ∅ do
3:      v_i ∈ V_k
4:      DiscoverLocalTopology(v_i)
5:      G_J ← TestSupernodeNeighborMerge(v_i)
6:      V_k^(m) ← OptimizeSupernodeGroup(G_J)
7:      V_k ← V_k \ (J ⋂ V_k^-) {Remove processed supernodes}
8:      UpdateBootstrapNode(V_k)
9:      if V_k = ∅ then
10:         if V_k^(m) ≠ V_k^- then
11:             V_k ← V_k^(m), V_k^- ← V_k^(m)
12:         end if
13:     end if
14: end while
```

The 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 topology. In r-mod, nodes are able to join the topology at any time. The second 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.

6. Improving the $r-$mod algorithm

Based on an understanding of the r-mod algorithm, an improved algorithm was developed. This section describes those improvements and presents the new algorithm r-SPOT in Algorithm 2.

6.1 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 reduce the overall importance of poorly chosen initial SNs. In the

first iteration of r-mod, nodes may not initially connect to the best SN due to the order of nodes joining and assigning SNs. 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.

6.2 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. The value of r is increased as needed until at least one interior node is found. This strategy improves the mobility of SNs and promotes better placement scores by increasing the information available within a neighborhood and reducing the chances of a stranded initial SN.

6.3 Preventing Loops in SN Placement

Currently, r-mod compares the cost of placing an SN at a new location against the cost 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 cost of the new SN is less than the current cost. A problem can arise if the metric to compute distance creates a Triangle Inequality Violation. Savage et al. [3] 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 oscillating between two SNs. In order to correct this a *true neighborhood cost*, one for all nodes using that SN, is calculated. 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.

7. Evaluation of Algorithms

Both algorithms were implemented in a distributed system called the Supernode Placement in Overlay Topologies (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 Equation 2. SPOT is designed as a service for other applications to utilize. It was 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 used for the local k-medians problem is the GNU Linear Programming Kit (GLPK) [4]. This open source solver performs provides reasonable performance compared to other solvers [5].

In order to determine the distance, SPOT supports the use of ICMP ping messages for round trip time (RTT) or a packet hop count (using the time to live (TTL) field). A more advanced measurement scheme is enabled through the support of the Harvard Pyxida Coordinate System [6].

Algorithm 2 Improved placement of Supernodes in a P2P Network (r-SPOT)

```
1:  for iter = 1 to PlacementIter do
2:    ConnectToBootstrapNode()
3:    while V_k ≠ ∅ do
4:      v_i ∈ V_k
5:      DiscoverLocalTopology(v_i)
6:      G_J ← TestSupernodeNeighborMerge(v_i)
7:      V_k^(m) ← OptimizeSupernodeGroup(G_J)
8:      V_k^(m) ← EvaluateSupernodeGroup(V_k^(m))
9:      V_k ← V_k \ (J ⋂ V_k^-) {Remove processed SNs}
10:     UpdateBootstrapNode(V_k)
11:     if V_k = ∅ then
12:       if V_k^(m) ≠ V_k^- then
13:         V_k ← V_k^(m), V_k^- ← V_k^(m)
14:       end if
15:     end if
16:   end while
17: end for
```

7.1 Experimentation Details for r-mod and r-SPOT

The initial testbed for performing experiments is Emulab [7]. We are interested in the how well the algorithms perform in finding low cost SN placement solutions. The first metric of interest is the $TotalCost$ as defined in Equation 2. This calculates the sum of the products of each node's distance and demand to its chosen SN.

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. The initialization is randomized via a sleep function on the nodes.

The network topologies consisted of hierarchical networks of size 100, 200, 300, and 400 nodes using virtualization in Emulab. The virtualization placed 10 virtual nodes on a single 3 GHz Pentium 4 CPU with 2 GBytes of RAM.

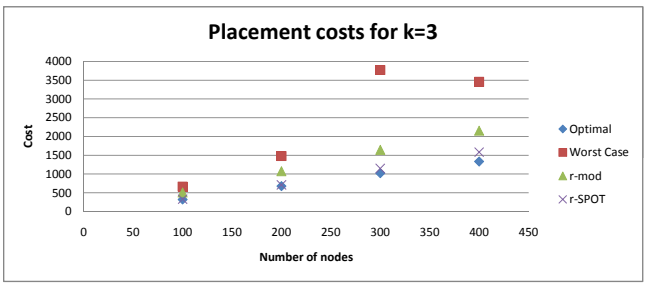

Fig. 5: Results of placement for three SNs with various sized network topologies.

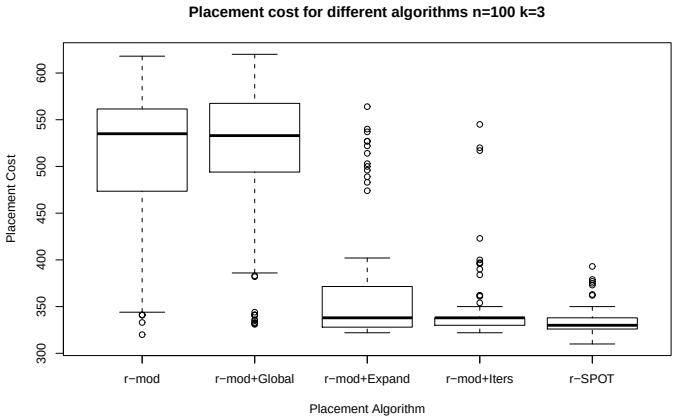

Fig. 6: Performance of the r-mod algorithm with enabling different improvements along with the r-SPOT algorithm topologies.

The radius value (r) was set to 2 in all of the experiments, consistent with related work [2]. The experiments were executed with the number of SNs (k) set to 1 and 3.

7.2 Evaluation Results

Experimental results with r-mod and r-SPOT are now presented. Due to space constraints we only present the k=3 placement results. In Figure 5 the results for k=3 show the performance of r-mod and the 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. Again, these results are the average scores out of 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.

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 6 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$\times$IQR less than the lower quartile or 1.5$\times$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+Global), r-mod plus the expanding neighborhood (r-mod+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 6 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. Even more improvements are made when the r-mod algorithm executes twice and uses the first set of SNs as an input for the next 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.

Additional experiments are run to understand some of the tradeoffs between r-SPOT and an optimal placement strategy. In Figure 7 the total time to locate three SNs were computed for various topology sizes in comparison to a global solution. The global solution required full 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 8. From the figure, at 400 nodes 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.

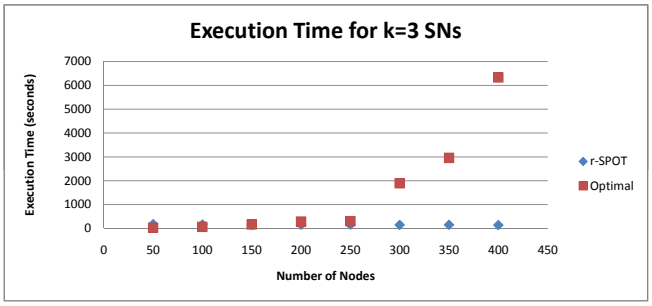

Fig. 7: Total time to place three SNs in r-SPOT algorithm compared to the optimal placement.

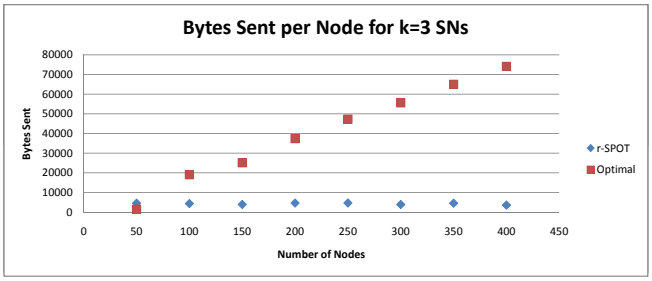

Fig. 8: Total amount of network traffic sent per node in placing three SNs compared to an optimal solution requiring global knowledge.

8. Game Servers

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. Multi-player first person shooters [8] 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 [9]. Therefore, it is important to choose the game server carefully. In order to experiment with Internet conditions the Planetlab [10] distributed testbed was utilized.

SPOT was run on 50 nodes in Planetlab with k=1 and it chose 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 9. The optimal value is selecting node 6 as the SN with an average RTT of 40 ms and 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. This shows the importance of carefully selecting a game server to reduce latency for all players.

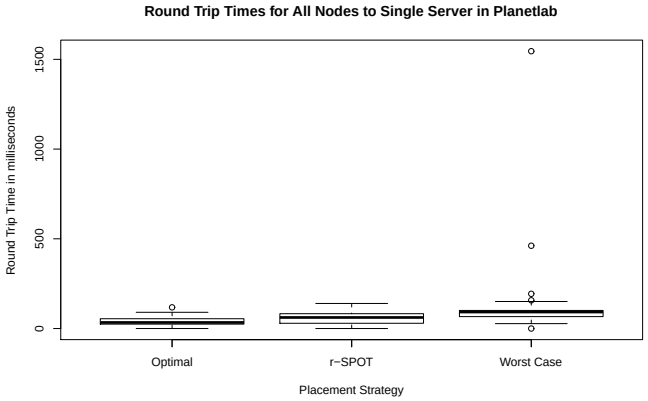

Fig. 9: Round Trip Times from each node to a single server selected based on the particular placement algorithm times.

9. Conclusion

This paper has presented the SPOT system, along with the two placement algorithms r-mod and r-SPOT. This research demonstrates a fully functioning P2P system that provides SN placement for a range of applications. An example application with SPOT locating a game server showcases the benefits of this network application.

References

[1] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Operations Research*, vol. 12, no. 3, pp. 450–459, May 1964.

[2] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros, "Distributed placement of service facilities in large-scale networks," in *IEEE Infocom 2007*, Anchorage, AK, May 2007.

[3] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of Internet path selection," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 289–299, Oct. 1999.

[4] "GLPK - GNU Linear Programming Kit." [Online]. Available: http://www.gnu.org/software/glpk/

[5] S. Thorncraft, H. Outhred, and D. Clements, "Evaluation of open-source LP optimization codes in solving electricity spot market optimization problems," in *Mini-Euro Conference on Operation Research Models and Methods in the Energy Sector*, Sept. 2006.

[6] "Pyxida: An open source network coordinate library and application." [Online]. Available: http://www.pyxida.sourceforge.net

[7] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of 5th Symp. on Operating Systems Design and Implementation*, Dec. 2002, pp. 255–270.

[8] G. Armitage, M. Claypool, and P. Branch, *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. John Wiley & Sons, June 2006.

[9] G. Armitage and P. Branch, "Distribution of first person shooter online multiplayer games," *International Journal of Advanced Media and Communication*, vol. 1, no. 1, pp. 59–75, October 2005.

[10] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 59–64, 2003.