P2P Volunteers for Reliable Server Farms

Ying Wang and Partha Dasgupta
Dept. of Computer Science and Engineering
Fulton School of Engineering
Arizona State University
Tempe, AZ 85287

Abstract

In recent years, there are more and more critical services being developed on the Internet with high requirements on availability and reliability. Distributed Denial-of-Service (DDoS) has become widespread and many web services have already been attacked. Server farms are one of the key approaches to improving the availability and reliability. However, due to the high maintenance cost and the "site disaster" problem of server farms, we are looking for cheaper infrastructure that is logically and geographical distributed to provide highly reliable services in spite of attacks and server failures.

In this paper we provide highly reliable, attack resilient servers by using P2P farms of volunteers. The volunteer group can be dynamically constructed, is scalable and is resistant to service failures resulting from a the loss of a few peers. In our approach, the system will first form a self-organizing group of peers who are interested in a service. Then statistical and operational data, service response delay, client request pattern, and so on are collected. Finally, the system applies this information to select new servers, and integrate into the server farm till the target level of service is met.

1 Introduction

The Internet is a shared resource, a cooperative network built out of millions of hosts all over the world [10]. The Internet is increasingly being used as access medium for a variety of critical services, for which very high requirements are set on availability and reliability [12]. Examples range from e-commerce sites, to per-pay media servers, to information servers in distributed mission critical systems, such as air-traffic control systems [12].

The obvious solution to the above problem is to make the service replicated. Suppose "acme.com" wants to build a replicated server. They can purchase large server farms at a few geographical locations. However, the network diversity (i.e. how dispersed they are topologically) is most likely to be small. In addition thousands of locations of server farms cannot be economically handled by Acme.com. Thus the high distribution requirements make it infeasible for one company to achieve, because it is very expensive to manage or purchase the distributed service centers.

The other approach that we are researching is for acme.com to "outsource" the service system to any volunteer (possibly paid volunteer) to provide the service. The volunteers are paid small amounts of money and can be home computer users or almost anyone. The volunteers are expected to provide their own system administration functions and are not under the control of acme.com. All they do is download some server scripts and data from acme.com and join the acme.com server system.

The resulting system is not a P2P system, but a client server system, constructed out of P2P based parts. The server, instead of being a set of dedicated, clustered machines, is a set of geographically dispersed volunteer hosts, coordinated through a P2P like collaboration framework. The distributed service network has a dynamically configurable structure using an overlay network that is discussed in detail in this paper.

The volunteer system is structured as a peer-to-peer system, i.e. the groups of volunteers self-configure into peer network, with certain structure and collaboratively provide the service. Each peer is not expected to be highly reliable, attack proof or even trustworthy. A set of protocols monitor the peers and steer the clients to the peers based on their reliability. The payment to the volunteers is based on how much "help" they provide in delivering the acme.com service. (Of course, the same peers could also be carriers of other service providers).

Thus, we argue, this approach has a potential of much further distributed reach and location diversity than a acme.com purchased server farm. In addition, the volunteer approach will be cheaper and have significantly lower management costs. A lot of the service costs will be paid out of "idle cycles" similar to the way grid computing is done. Also the client-server model embedded in the server farms can help us yield better performance, availability and scalability than peer-to-peer massively decentralized approaches as shown in [15].

In the following section, we outline the work conducted by the research community on peer-to-peer systems, application layer multicast, and an overlay peer utility service. We describe the peer-to-peer volunteer network design in section 3. In section 4, we analyze the results from our message-level peer-to-peer service simulator. We discuss future work and conclusions in section 5.

2 Related Work

2.1 Peer-to-Peer Systems

Peer-to-peer systems can be differentiated by the degree to which the overlay networks contain some structure or are created ad-hoc [1, 12]. The structure means the way in which the content of the network is located with respect to the network topology.

Unstructured: The main characteristic of unstructured peer-to-peer systems (such as Gnutella [6]) is that the placement of data is completely unrelated to the overlay topology. Since there is no information about which nodes are likely to have the relevant files, searching essentially amounts to random search, in which various nodes are probed and asked if they have any files matching the query. The advantage of such systems is that they can easily accommodate a highly transient node population. The disadvantage is that it is hard to find the desired files without distributing queries widely [13].

Structured: Structured systems were proposed (such as Chord [14], CAN [11], Tapestry [16]) in which the overlay network topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations. These systems provide a mapping between the file identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired file [6]. Structured systems offer a scalable solution for exact-match queries. The disadvantage of structured systems is that data locality is destroyed, data items form a single site are not usually co-located, meaning that opportunities for enhanced browsing, pre-fetching and efficient searching are lost.

Another approach is the peer-to-peer communities [8, 9] which turns groups of ad-hoc peer-to-peer systems into logical communities. They use communities as a more natural arrangement of distributed systems and show how they are helpful in pruning the search space. The community-based search technique allows search operations to be based on content rather than just filenames.

2.2 Application Layer Multicast

In order to improve the system performance and efficiency, we use application layer multicast for the group communications of the server farms formed by peers. Typically, the goal of application layer multicast algorithms is to build distribution trees that minimize the additional routing overhead compared to native IP multicast. There are several approaches that are close to our proposal, including overcast, End System multicast, Chord [14] and CAN [11], which we discuss below.

As one of the tree first solutions, Overcast [7] explicitly measures available bandwidth on an end-to-end path and builds a multicast tree that maximizes the available bandwidth from the source to the receivers. A distributed tree-building protocol is used to create this source specific tree.

End system multicast [4][5] is one of the tree-over-mesh solutions. It builds a mesh structure across participating end-hosts and then constructs source-rooted trees by running a routing protocol.

Some other recent projects (Chord [13], CAN [11] and Tapestry [16]) address the scalability issue by creating application layer overlays. The common idea of these approaches is to realize a scalable lookup service for objects where the responsibility for managing the object space is shared equally among a network of peer nodes. Because these peer-to-peer networks base their routing decisions on application semantics, the resulting distribution tree may be very inefficient with respect to end-to-end delay.

2.3 Overlay Peer Utility Service

Opus [3] is a large-scale overlay utility service that provides a common platform and the necessary abstractions for simultaneously hosting multiple distributed applications. Opus consists of a number of cooperating components: service level agreements (SLAs) specify the amount each customer is willing to pay for a given level of network service. Opus uses these SLAs to perform resource allocation among competing applications during times of peak demand. The application's SLA, current demand level and node allocation determines the target per-application overlay topology. A control overlay maintains connectivity and performance characteristics among all overlay nodes. The introspection module maintains low-level performance metrics among pairs of overlay nodes. Different from Opus which is aimed to provide general utilities for all kinds of network services, and our approach is specifically for high reliable service and so

we focus on solving the reliability problems of current Internet services.

3 Our Approach

3.1 Overview

In our approach, we build an infrastructure using peer-to-peer architecture. This infrastructure could dynamically construct and configure server farms to meet target levels of service based on the changing network characteristics and peers' statues.

The basic approach is : first, form the peers interested in a specific service into a group in a self-organizing fashion. Second, collect statistical and operational data, such as service response delay, client request pattern, and so on. Then, apply the information to select new servers, and integrate them into the server farm, till target level of service is met. Last, a "frequent flyer" program is used to encourage the peers to contribute resources and get bonus from the contribution.

3.2 Architecture

As shown in figure 3-1, the peer-to-peer services system is composed of three layers:

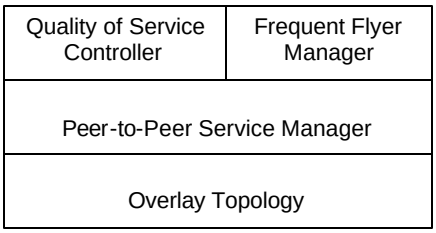

Figure 3-1. System Architecture

The bottom layer is the overlay topology layer, which is used to form and maintain the overlay topology of peer-to-peer service systems. It also provides protocols for multicast and aggregations.

The layer in the middle is the peer-to-peer service manager layer, which provides management functions of the peer-to-peer service.

The top layer includes two modules, QoS controller and "frequent flyer" manager. The QoS controller follows the feedback algorithm to adjust the server farm's configuration to improve the quality of service and maintain the quality of service at the target level. The "frequent flyer" manager works as the "frequent flyer" program of airline companies, the goal is to motivate peers to contribute resources by providing bonus to the contributors.

3.3 Overlay Topology

In order to propagate and aggregate data in an efficient way in the peer-to-peer service system, it requires an overlay build the top of the peer-to-peer network designed for better performance and easier management. In the first section, we address our overlay topology; second, the formation, maintenance and optimization protocols are stated.

3.3.1 System Topology Overview

In our peer-to-peer Service System, we build the server farms using multicast trees. All the servers of one service form one multicast tree with the root server as the root of the tree. The clients can choose only one server at a time and they can switch to other servers at any time. All the clients of one server form one group. As shown in figure 2, the clients of server A form one group, and the dash circle around A indicate the size of the client group, and same for server B. Since the statistics data like request pattern is required to be monitored at the client level, also client needs service information to select appropriate servers, and clients are organized in the overlay topology too. Clients of one server form a group and the topology of the group is also the multicast tree. In summary, the overall topology of the system is a tree of trees. All the servers form the main tree with the service source as the root, i.e. each node in the main tree is a server. And each server is the root of a tree for clients, whose nodes are all clients expect the root node.

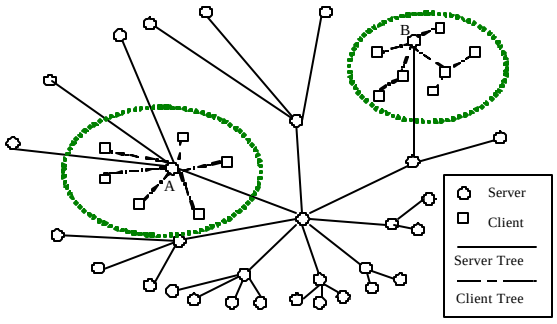

Figure 3-2. System Topology

Each tree in our peer-to-peer service system topology is a source specific overlay application multicast tree. These trees are used to organizing the peers in the system. The main functions of the trees are to propagate and aggregate data from/to the root of the tree. We adapted the ideas from application layer multicast research, e.g. [7] on building a tree based on the network condition, including latency, and bandwidth for better system performance and scalability.

3.3.2 Tree building

In our approach, we assume that each peer knows a list of other peers from bootstrap, and the peers in the list are called neighbors of the peer. If a peer does not know neighbors, it could advertise itself through other ways, like URL, email, and record the peers who response to its advertisement, and then set up its own neighbor list.

We achieve the tree building in the following three steps. First, the source sends out tree-creating-message to all of its neighbors. Second, for each peer that receives the tree-creating-message will broadcast the tree-create-message with TTL (Time To Live) equals to two, and if it is interested in the service, it will send ion message to its parent (it locate its parent through the information given in the tree-creating-message). Third, for each peer receives joining request will follow a tree-joining algorithm. This algorithm is used to deal with the received joining request and adjust the fan-out according to the capability of the peer. If the peer has the capability to accommodate one more children, then accepts it, otherwise, it will compare its worst child with the requesting one, and accept the better one, reject the worse one.

While after the first multicast tree of the service is setup, the tree will be split into multiple trees, and we address the process of split and organizing in section 3.4.3.

3.3.3 Maintenance

To maintain the system topology, we use two heartbeat timers at each peer in the tree. One timer is used to send out heartbeat message periodically and the other is for detecting connections to its neighbors (parent and children), if it does not receive its parent's heartbeat on time, then it will start selecting new parent. If it does not receive its child's heartbeat, delete the child from its children list.

While in peer-to-peer network environment, the peer-to-peer overlay networks are dynamic, heterogeneous and unpredictable. The peers who were offline or new to the service can join the system topology from the root of the tree. In our system topology, when the peer is unavailable, it will only have impact on its neighbors (parent and its children). *For its parent:* since it will query its children at each heartbeat, once it detects the child is not available, it will just delete the child from its children list. *For its children:* who lost its connection to its parent will rejoin the tree in the following way: since it stores a candidate parent list during tree building process, and it knows a set of peers who is in the tree, like its previous ancestors, its previous children, it could ask these peers or their decanters to be its new parent.

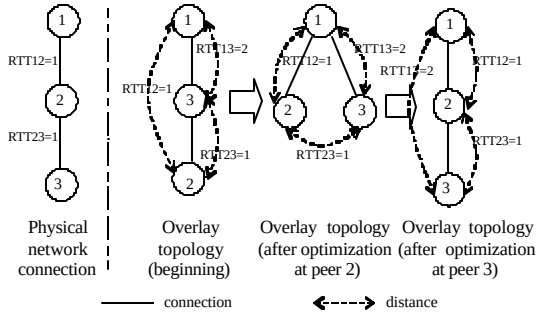

Figure 3-3. Example of Topology Optimization

3.3.4 Optimization

In the process of optimization, the overlay topology mimics the real physical network connections so that the difference between them is minimized. The peers execute the optimization algorithm periodically to adjust their positions in the tree dynamically to achieve better system performance and scalability. Peers in the grid receive heartbeat message from their parent. The heartbeat messages contain the children list of the parent, the ancestor list of the parent, etc. The peer follows the optimization algorithm to find a better parent upon receiving the heartbeat message from its parent.

The basic idea of the optimization algorithm is that: first, if there is no penalty for the peer to get the message from its sibling, that's the physical path from the parent to the peer goes through the sibling, then we use the sibling as the peer's new parent, then we can reduce the link stress (the number of duplicated packets go through the physical link) between the parent to the sibling, that means we saved network resources. Second, if it is possible for the peer to move closer to the source in the tree, that is if the capacity of its ancestor is not full, it's more efficient to add the peer under its ancestor than under its parent. A threshold is set to adjust the computing time of the peer to forward a message. Peers receive join requests from siblings or decanters will follow tree-joining algorithm.

An example is shown in figure 3-3. At the beginning, the overlay topology is different from the physical network connection. The messages from peer 1 to peer 2 always go through peer 3 in the overlay topology, but according to the underlying network connection, the message has to first pass peer 2 and then peer 3 and then back to peer 2. In the first step of the optimization, peer 2 will compare the RTT 13 plus RTT23 with RTT12, and since RTT 12 is shorter, peer 2 will move directly under peer1. Then in the optimization of peer 3, it compares RTT12 plus RTT23 with RTT13, and since there is no difference between the two paths, peer 3 will

move down to be child of peer 2 reduce the like stress between peer 1 and peer 2.

3.4 Peer-to-Peer Service Manager

The peer-to-peer Service Manager is the layer used to setup, and maintains the peer-to-peer services. We address how it works on peers in different roles, general server, client, and root server.

3.4.1 Servers

In our approach, the servers refer to the peers who provide peer-to-peer service to other peers. A peer may be a server in one peer-to-peer service, and a client is another peer-to-peer service. In our overlay topology, the server belongs to the tree for servers of the specific service, and the server has a group of clients reporting aggregation data to it.

3.4.1.1 Data Propagation

The data propagation includes the data multicasting from the server to its clients and also the data propagation from the server to all the other servers. If there is any message to sends to all the servers, the server will propagate the message through the tree for servers. . If there is any message to sends to all the clients of this server, the server will propagate the message through tree for clients. If there is any message that needs to sends to all the clients, the server will first propagate the message to all the other servers, and then each server propagate the message to its clients.

3.4.1.2 Data Collection

Every server periodically collects data from its clients for decision-making usage. Also the server records its own data and aggregates the data from its decanters servers in server farm, and then report to its parent.

At each aggregation time, the server first calculates the local data by aggregating data messages during this aggregation period received from its clients in the tree for clients starting from the server. Then, the server calculates the aggregated data from the servers who are its children in the tree for servers, and last, it report its aggregated data, including its own local data and all the data from its children servers, to its parent in the tree fro servers.

3.4.1.3 Failure Recovery

When a server is online again, it first contacts its previous parent in tree for servers, and joins the tree. Then it will download the new updated service from its parent if there is any. Followed, the server will update all the service related information, like the updated server list.

3.4.1.4 Network partition

Network partition happens only when the root server is offline, since the other type of partitions can be easily fixed by rejoining from the root. When the root server is offline, a new root server will be elected though root server election algorithm, and the elected root server will perform the responsibilities of the root server, until the original root server is back.

3.4.1.5 Service Quitting

When a server is going to be offline permanently or is willing to stop providing the service, it will quit the service and propagate its leaving to all the other servers and all the clients. On the other hand, if there is no or only a very small number of service request for that server for a period of time, the server is considered not useful, since the cost for its existence including control overhead, communication cost on server updates is larger than the benefit it can provide. For this type server, the server will stop its service to imp rove overall system efficiency.

3.4.2 Clients

In our peer-to-peer service, due to the statistics data like request pattern is required to be monitored at the client level. To do that, clients of one server forms a group and the topology of the group is also the multicast tree. In our overlay topology, the client always belongs to one of the trees for clients and the clients could switch to other trees for clients at any time.

3.4.2.1 Joining

There is only one server of a specific service at the every beginning; all the clients joins same server. Later, the system will add servers for the service gradually, and each time a new server is added, the message will be broadcast to all the clients. Then, the clients compare the distance (RTT) to new server with the old server, and choose the closer one to join. Anytime when the server is offline or overloaded, the clients of it will switch to the second close server among the available servers.

3.4.2.2 Data Gathering

According to the service requirement, the client may be asked to collect some service related data. The most important data includes the request pattern for the service, and the evaluation of the servers, etc. As we stated before, all the clients of peer-to-peer services are in the trees for clients. Each client has several neighbors in the overlay topology, including one parent and their children. The client is responsible for propagate system

data/request to its children, and collect required data from its children, and summarized them with its own data, and then report it to its parent.

To reduce the impact of the blocking of data transfer channel for any reason, we include the past system data into the reporting data. In this way, the system could measure and response to the blocking/attack in a more precise way, so system performance could be better and the system resource could be saved too. Our reporting data is calculated as show following:

Statistical data = 0.5*history data+0.5*collected data

History data = statistical data

3.4.2.3 Server Selection

Since there may be thousands of servers for one specific peer-to-peer service, how to select the appropriate one is an important issue for the client. In our approach, each client maintains a list of servers, and their performance records. Before each service call, the client will sort the servers according to their performance, and select the best one, and after each service call, the client will evaluate the server performance, and put it in the records.

3.4.2.4 Server Promotion

All the clients of a peer-to-peer service are the potential servers of that service. There are two possible ways to promote clients to servers. First, the client can volunteer to provide service. Second, the client is selected by the system to be the best new server and the client agrees to become a server.

3.4.2.5 Service Volunteer

A client of a peer-to-peer service can express its willing to become a server to its current server, and the server will transmit the service to it upon receiving the request.

3.4.2.6 Placement of New Server

In order to improve the scalability and efficiency, we follow the following steps to choose a new server. First, the server chooses the worst average metrics from its client children list based on the data collected. And this selected client also chooses the worst average metrics from itself and its children list, and so on so far, the client received the worst quality of service could be identified. And that client will be the new server. The quality of service around the past worse service area will improve because of the adding of the new server, and also, in this way, the servers are distributed logically and geographically, so that the "site disaster" [12] problem could be minimized.

3.4.2.7 Server Transmit

Besides the root server, a peer becomes a server when another server transmits the service to that peer. Let the server sends service as peer A, and the server receives service as peer B. Before the transmission of service, peer A sends the recourse requirement of the service to peer B, if peer B confirms that B has that much resource, then A start transmitting, otherwise, the transmission is abort, After the service transmitted from A to B, the peer A becomes the parent of peer B in the tree for servers. Peer A will set peer B as its children. On the other hard, peer B will set peer A as its parent. After peer B joined the tree for service following above settings, peer B will use underlying overlay topology to propagate the adding of new server to all the other servers, and the servers received the message will propagate the message to all their clients. Clients may switch to peer B's tree for clients if peer B is a closer server comparing to their current server.

3.4.3 Root Servers

The root servers refers to the peers that are willing to provide itself created service to the other peers and propagate the service to other peers to share its workload. The root server is the root of the tree for server of the specified service, and it plays the most important role in the peer-to-peer server.

3.4.3.1 Service Creation

When there is a new service is setup on a peer in the network, the peer could choose to create a peer-to-peer service, and then the peer becomes the root server of the peer-to-peer service. To do that, first, the peer just need to call the underlying overlay topology layer to build a multicast tree with itself as the root. Then the peers interested in the service will join the tree, and call the server for service. After this, the server collect enough data on the client request pattern, can then can start select the right place for the replicas of its service in the peer-to-peer network. So on and so far, the peer-to-peer service is setup.

3.4.3.2 Service Update

Each time there is an update to the service, the root server propagate the new service through the tree for service. Every time the server is recovered from failure, it will ask it parent for new update if there is any. So, all the servers of one specified service all provide the same version of service.

3.4.3.3 Service Dismiss

When the root server is going to get offline permanently or give up the service it created, it will notify all the servers about it. Then all the servers

receives the message will stop the service and notify the clients of the service.

3.5 "Frequent Flyer" Program Manager

In our peer-to-peer service system, we design the "frequent flyer" program to motivate the peers to contribute resource. As in the airline company, the customer will get bonus by accumulating fight miles with this airline. Similar in our "frequent flyer" program, the peers contribute to the system (be a server for any peer-to-peer service), will get its bonus, money from the root service provider. The more it contributes, the more it could get. In this section, we discuss the three roles the peers may play in the frequent flyer program and the detailed rewarding procedure.

In our approach, the total available time of the peer-to-peer server is used as the metrics for rewarding. Usually, the root server is responsible for recording the bonus of all the servers, but in case the root server is down, the temporary root server of the peer-to-peer service will play this role, and transfer the data to the root server when it's up. The root server maintains a list of all the servers, for each server, the total available time and its bonus is recorded. The root server periodically accesses the servers in the list, and computes their available time and bonus. For each month or year, the root server gives out the bonus, money, to the servers according to the list and then reset the list.

3.6 Quality of Service Controller

The QoS controller provides functions to achieve more specific requirements of the services. We used two metrics to measure the system's performance, one is Availability, - number of satisfied requests over total number of requests. The service is available when there is at least one server available for providing service. The number of satisfied request of one client is the times when the client calls the servers in the server list and gets the service (at least one server is available at that time). Then the average of number of satisfied request of clients of the whole service indicates the availability of the service. The other metrics is Efficiency, - delay of service response. The delay of service response is the time between a service requests is issued and the service response is received. The server's CPU power, memory size and network condition jointly impact this metrics. The shorter the average delay indicates the better service performance, and so as the stronger servers' capability.

In our approach, we use feedback to control the QoS. As shown in figure 3-4, the QoS requirement is set up when the service is created, and data will be collected for decision making, after the action is taken, goes back to the data collection to make the second decision, and so on so far, QoS is improved step by step, and maintained at that level.

Depends on the requirement item, the corresponding metrics will be collected either through data client's data gathering. After the data collection, the root server knows the overall average of the metrics, and the server knows the average of the metrics of the tree for clients starting from it.

In our approach, the control center is on every server, they compare the their statistics metrics with the requirement, if the statistics data is worse, then they start adding new servers as their children following placement of new server and service transmit. If the statistics data is already better than the requirement, then do nothing.

4 Simulation

In this section we present results from our message level simulator. Our simulator is written in C#, and can simulate the entire peer-to-peer service system with thousands of nodes. Through our simulation experiments, we used Georgia Tech [17] random graph generators to generate topologies of the peer-to-peer network. We designed and implemented experiments under four kinds of situations.

Targeted attack: a number of servers in the server farms are attacked.

Random attack: a number of peers, including both servers and clients are attacked.

Unstable environment: all the peers go on and off randomly.

Low capability server: the servers can only response to a limited number of requests at a time.

4.1 Targeted Attack

We simulate targeted attack by turned off a number of random picked servers periodically, and the servers recover from the attack at a random time in a range.

First, we compare of the actual data collected by the simulator and the statistics data collected by the system, and their relation to the number of servers. We did our simulation in a 100-peers network, with the quality of the peer-to-peer service is set as availability equals to 0.95, and targeted attacks with 10 peers been attacked. The results show that with the increasing of the number

of servers, both the actual data and statistics data are getting closer and closer to the system quality of service requirement, i.e. availability equals to 0.95. The results show that the number of servers increases exponentially until the target level of service achieved, and then the number of servers keeps around the idea number of servers.

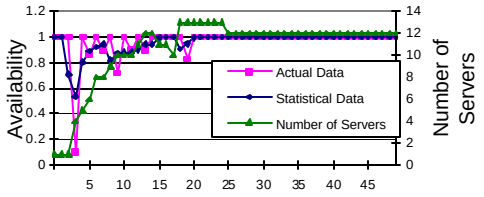

Figure 4-1. Actual Data vs Statistical Data in targeted attacks

Next, we compare the system performance in networks of difference sizes. We simulated network with 10, 100, 1000 peers with the quality of the peer-to-peer service is set as availability equals to 0.95, and targeted attacks with 10% peers been attacked.

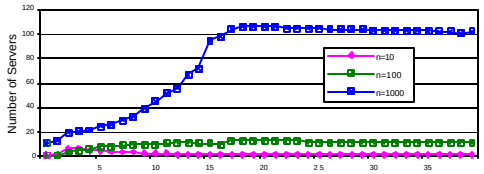

Figure 4-2. Num of Server vs Network Size in targeted attacks

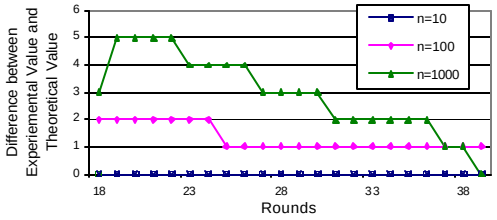

Figure 4-3. Difference between Experimental Value and Theoretical Value in targeted attacks

Figure 4-2 shows that the larger size the network is, the more time it takes to stabilize the number of servers required for the system. The reason is that it takes time for the system to collect data and add servers to adapt to the changes of the system. Since in this simulation, we simulated targeted attacks with 10% peers been attacked, the larger size the network, the more number of servers been attacked and the more number of servers needed to add, and the more time it takes to reach to requirement.

In figure 4-3, we show the difference between the experimental value of the number of servers while the system got stabilized and the ideal number of servers. The larger size the network is, the bigger difference between the experimental and ideal value. For our system, the difference is managed fewer than 5%, which shows the efficiency of our system resource usage.

Last, we compared the impact of the attacks in difference severity. Our simulation use a network with 100 peers, where the quality of the peer-to-peer service is set as availability equals to 0.95, and targeted attacks with 0, 5, 10, 15, 20 peers been attacked each time. The more serious the attack, (the more number of servers been attacked), the more time the system needs to reach the ideal number of servers, the more number of servers needed in the server farm and the larger range of the number of servers could wave around the ideal number of server. The reason is similar as we described in previous experiment. Basically, the more number of servers to be added, the longer time it takes to achieve the system requirement.

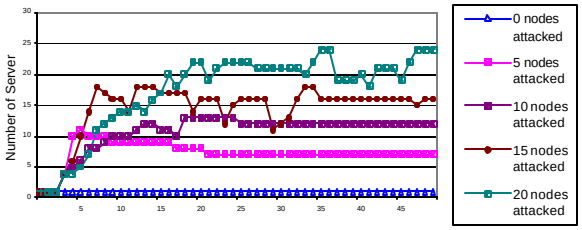

Figure 4-4. Attack Severity vs Number of Servers in targeted attacks

4.2 Random Attack

We simulate random attack by turned off a number of random picked peers periodically, and the servers recover from the attack at a random time in a range.

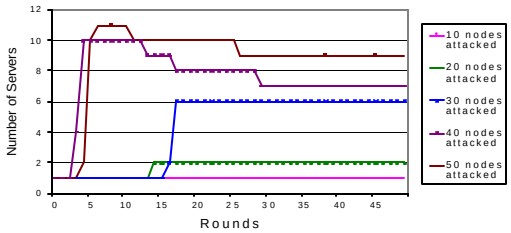

Figure 4-5. Attack Severity vs Number of Servers in random attacks

We compared the impact of the attacks in difference severity. Our simulation use a network with 100 peers, where the quality of the peer-to-peer service is set as availability equals to 0.95, and targeted attacks with 0, 5, 10, 15, 20 peers been attacked each time. The results show that once the root server is attacked, the number of servers increases exponentially, and then the number of servers keep around the idea number of servers. The more serious the attack, (the more number of peers been

attacked), the more servers added into the system. Comparing to the same setting for targeted attack simulation, the less servers required in random attack then in targeted attack, since system performance reduce, only when the servers been attack.

4.3 Instable Environment

We simulate the unstable environment by turned on and off all of peers periodically with configured peer on and off interval ranges.

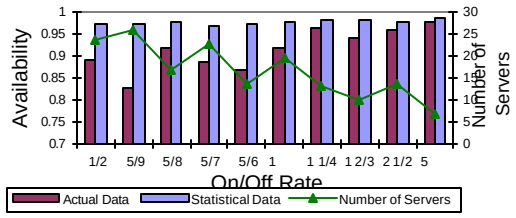

Figure 4-6. System Performance in Instable Environment

We compared the impact of the on interval/off interval rate on system performance. Our simulation use a network with 100 peers, where the quality of the peer-to-peer service is set as availability equals to 0.95, and the on/off interval rates are 5/10, 5/9, through 5/1.We did the same simulation 5 times, and calculated the average of the actual data, statistical data and number of servers of the 5 simulation results. The final results show that with the increasing of the rate, i.e. decreasing the off interval while keeping the on interval the same, the number of servers required decreased and the system performance increased.

4.4 Stable Environment

We assume in the stable environment, there is no attacks and turn downs on any peers in the network.

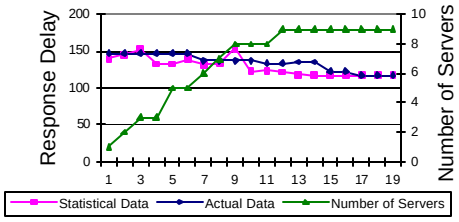

Figure 4-7. Response Delay vs Number of Servers in stable environment

First, we show that the system could be used to improve the service response delay. We did our simulation in a network with 100 peers, the RTT between two peers is randomly distributed between 0 to 99 and the server response to the client in 100 + 1.002^the number of clients the server is serving, and we set the required response delay as 120. The results show that the number of servers increase linearly, and with the increasing of the number of servers, the system statistical data and actual data collected by the simulator decrease gradually until they reached the requirement.

Second, we did our simulation in different sizes of networks, network with 10 peers with the RTT between two peers is randomly distributed between 0 to 18, network with 100 peers with the RTT between two peers is randomly distributed between 0 to 198, and network with 1000 peers with the RTT between two peers is randomly distributed between 0 to 1998. We set the response delay as the server response time plus 10% of the average RTT between two peers. The results show that the larger the network, the less percentage of servers needed to reach the requirement. Also, the larger the network, the more rounds needed to reach the requirement.

Network size	Response delay requirement	Num of Servers	Stabilized response delay	Rounds
10	102	3 (30%)	102	6
100	120	9 (9%)	117	13
1000	300	8 (0.8%)	292	84

Table 4-1. System Performance vs Network Size

Last, we did our simulation with different requirement in a network of 100 peers. The response delay are set to 150, 140, 130, 120 115 respectively. The results show that the higher the requirement, the more number of servers needed to reach the requirement, the more rounds needed to reach the requirement.

Response delay requirement	Num. of Servers	Stabilized response delay	Rounds to reach the requirement
150	1	147	1
140	3	135	4
130	5	123	7
120	9	117	13
115	11	114	19

Table 4-2. System Performance vs QoS Requirement

5 Conclusion

In this paper, we addressed the problem of providing peer-to-peer service for high reliability. We use the peer-to-peer architecture for the attack resilient services, and we argue that this structure has advantages over the replicated approach used conventionally.

Through various simulations using our message-level simulator, we show that our infrastructure can provide high reliable service in various environments. The peer-to-peer service is resilient to targeted attacks, random attacks, and it also shows good performance in unstable environment where peers are turned on and off frequently. Also, in the normal stable environment, the peer-to-peer service could provide high quality of service by automatically configuring the server farms.

We believe that our approach is a novel way of providing high reliable service on internet, especially in service-critical environments.

References

[1] Karl Aberer, Manfred Hauswirth, *An Overview on Peer-to-Peer Information Systems*, Swiss Federal Institute of Technology (EPFL), Switzerland, 2002

[2] K. Aberer, M.Hauswirth, M. Punceva, and R. Schmidt, Improving Data Access in P2P Systems, IEEE Internet Computing, 6(1), Jan./Feb. 2002

[3] R. Braynard, D. Kostic, A. Rodriguez, J. Chase and A. Vahdat, Opus: an Overlay Peer Utility Service, Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH), June 2002.

[4] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan and Hui Zhang, Enabling Conferencing Application on the Internet using an Overlay Multicast Architecture, In Proceedings of ACM SIGCOMM 2001.

[5] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang, A Case for End System Multicast, in Proceedings of the ACM SIGMETRICS, June 2000.

[6]. Q.Lv, S.Ratnasamy, and S.Shenker, Can heterogeneity make gnutella scalable? , In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002

[7]. J. Jannotti, D.K. Gifford, K. L. Jognson, M. F. Kaashoek and J. w. O'Toole Jr, Overcast: Reliable Multicasting with an Overlay Network, Operating System Design and Implementation (OSDI), Oct. 2000

[8] M. Khambatti, K. Ryu and P. Dasgupta, Peer-to-Peer Communities: Formation and Discovery. In 14th IASTED International Conference on Parallel and Distributed Computing Systems (PDCS), Cambridge, MA, USA, November 2002.

[9] M. Khambatti, K. Ryu and P. Dasgupta, Structuring Peer-to-Peer Networks using Interest-Based Communities? International Workshop On Databases, Information Systems and Peer-to-Peer Computing (P2PDBIS), Humboldt University, Berlin, Germany, September 2003.

[10] Andy Oram, Peer-to-Peer: Harnessing the Power of Disruptive Technologies, O' Reilly, March 2001

[11] S. Ratnasamy, P. Francix, M. Handley, R. Karp, and S. Shenker, A scalable content-addressable network, In proceedings of ACM Sigcomm, Aug 2001.

[12] G. Shenoy, S. K. Satapati, R. Bettati, Hydranet-Ft: Network Support for Dependable Services, Department of Computer Science, Texas A&M University.

[13] Stephanos Androutsellis-Theotokis, A Survey of Peer-to-Peer File Sharing Technologies, Athens University of Economics and Business, Greece, 2002

[14] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek and H. Balakrishnan, Chord: A scalable peer-to-peer bokup service for internet applications, In Proceedings of ACM Sigcomm, Aug. 2001

[15] A. Vahdat, J.Chase, R. Braynard, D. Kostic, P.Reynolds and A. Rodriguez, Self-Organizing Subsers: From Each According to His Abilities to Each According to His Needs, Duke University.

[16] B.Y. Zhao, J. Kubiatowicz and A. Jeseph, Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, University of California, Berkeley, CA, USA, Apr. 2001

[17] Modeling Topology of Large Internetworks http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html