Boston College
Computer Science Department

Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage: Archiving Over Networks
Prof. Elizabeth Borowsky

Distributed Data Storage: Archiving Over Networks

Hiroyuki Hayano
Computer Science Department
Boston College
Chestnut Hill, MA 02467
Email: hayanoh@bc.edu

1. INTRODUCTION

1.1 Background

In the age of information technology, computers have become one of the most indispensable components of our lives. We use computers at home, school, office and virtually anywhere else to take advantage of the fast computational power and the massive data storage abilities. Due to the efficiency and capabilities that computers can provide, we have become so dependent on them – in fact, too dependent that information stored in computers has become not only massive, but also vital. Loss of such information may cause irrecoverable damages to computer-regulated systems and databases and inflict catastrophic problems.

To avoid permanent loss of data, one may choose to back up the data on multiple media so that even if there is a data loss in one of them, she can recover from others. However, what would she do if all of the media get destroyed? Say, she stores all the disks and tapes in a safe place in her office so that she feels invincible, but an earthquake or an inferno destroys the office and everything within. She must have felt secure for having backed up her vital data, but even those backups are gone now – she is in a big trouble.

In such computer-dependent society as this, there are services that provide data storage and maintenance to ensure that no data loss occurs, but they are usually very expensive for they require pricey hardware and software modification for specific data types. A CEO of a .com company may look for a better solution for remotely backing up crucial information, and that's when the distributed data storage system comes into play.

1.2 System Overview

In essence, distributed data storage (DDS) takes the data that needs to be backed up, splits it into smaller packages and distributes them to the personal computers through the broad web of network. When the data needs to be recovered, the system retrieves the mirrored packages and rebuilds the exact copy of the data. DDS makes use of unused space that personal computers have; in current high-tech industry, computers manufactured for personal uses have at least tens of gigabytes, and unless the user is heavily multimedia-oriented, there are at least a few gigabytes to spare per computer. If the unused portions of hard disks in hundreds of thousands of personal computers are put together, that can create a virtual disk with terabytes of capacity.

The personal computers that are candidates for hosting data to be backed up must be linked together through a network via which the packages are delivered. Hence, the speed of the network becomes vital in establishing an efficient DDS. Depending on how many pieces into which DDS splits the data, the size of each package may be substantially large; therefore the system must take advantage of the high-speed networking media that are available today.

The data packages are distributed among and retrieved from remote computers that are serving DDS. Hence, even if the hard drive or the medium that contains the original data collapses, the data is still alive at remote locations and can be recovered as long as the data originator has the key to collect all the packages.

1.3 Outline of the Paper

This paper is a presentation of a suggestive design with which DDS may be implemented. The rest of the paper is organized as follows:

- **System Design** (Section 2) - This is the core of the paper describing a possible implementation of DDS. It presents assumptions that are made upon creation of the system as well as a target application, and the rest of the section is divided into subsections describing the more technical details of the system. The subsections include a description of the four basic components of DDS and a scenario of how

the system works, showing the actual interaction between the components for a normal operation of DDS.

- **Future Work** (Section 3) – DDS discussed in this paper is far from actual application and still faces countless problems. This section suggests possible solutions to such problems and addresses the issues that need to be further investigated.

- **Conclusions** (Section 4) – The project is still in progress, but this section covers anything that can be stated from what has been done so far.

- **Related Work** (Section 5) – There are previous works done in the similar field, and this section presents some of them that relate to DDS in one way or another.

** The DDS is coded in Java for this project and some notes on the actual codes as well as the entire listing of the program are attached at the end of this paper.

2. SYSTEM DESIGN

2.1 Assumptions

There are several assumptions that are made upon construction of DDS prototype in order to simplify the structure. These assumptions may be dismissed as the design of the system improves after revisions and further studies, but for now, they are as follows:

Assumption 1 – The network through which the packages are delivered is fairly reliable without any failures.

Assumption 2 – There are three types of servers in terms of reliability:

Very reliable – absolutely no downtime.

Somewhat reliable – almost no downtime, and even if it does go down, it comes back up in minutes.

Not reliable – frequent downtime, and once it goes down, there is no guarantee of coming back up. This can be a personal computer that shuts down (or goes off-line) frequently, unexpectedly and for a long duration (i.e. hours every day).

Assumption 3 – The data that has been backed up does not require to be updated. That is, no data re-write is necessary once it is distributed. As for now, DDS is intended for backup purposes only for simplification reasons.

Assumption 4 – The owner of the original data is the only one that requires access to the backup.

Assumption 5 – Parties other than the owner of the original data may have an access to the distributed packages and hence the entire data, but it is not necessarily intended by the system.

Assumption 6 – There is a sufficient number of personal computers with sufficient amount of available space participating in DDS to serve the data.

Assumption 7 – There is a sufficient number of "somewhat reliable" servers participating in DDS.

Assumptions 1 and 2 are made to block out any sort of network errors that may occur, 3 thru 5 to simplify the DDS prototype, and 6 and 7 to ensure that DDS will not encounter an interminable search for resources (i.e. the computers to which the packages are delivered). These assumptions may be removed as the research advances and the DDS becomes more immune to errors and network failures.

2.2 Target Application

The target application of DDS needs to be drawn with the above assumptions in the mind. The first two assumptions are made simply for reliability purposes, but the third one is critical for deriving the target application. Since the application cannot re-write the data once it is distributed, the application cannot update or modify the data. This narrows the target application to be something that serves simply for storing historical data that does not get altered once it is created; it must be a read-only application.

The fourth assumption disables the sharing of the data; hence the target application is intended for non-public information.

When packages are passed from a computer to another on a network, there are always security holes through which a hacker can access the information. This paper designs and builds

merely a prototype of DDS, and security issues are not the main focus. The target application, therefore, must not require a high degree of security.

Considering all the restrictions, one possible application of DDS is archiving. Back in the days when newspapers and magazine articles were stored only on microfiches, all they needed were thin plastic films to be bundled up and stored in a safe place. Today, as multimedia presentations and high-resolution graphics along with massive volumes of information become predominant means of communication, there often is information that cannot be stored on physical media such as sheets of paper and videotapes. The enormous data can, however, be stored in digital archives, retaining all the vital information necessary for future references. Businesses that are heavily multimedia-oriented may wish to backup the digital archive to prevent from permanent loss of information in case anything should happen to their electronic database, and such circumstances call for a great application of DDS.

2.3 System Components

Now that we have established the assumptions and possible target application, we dive right into the core of the system design. There are four basic components to DDS:

- **DataOwner** – The owner of the original data that uses the DDS service to backup data. It splits the data into a number of smaller packages and sends them to Managers upon creating a backup. It has the capability of collecting the packages from the Managers as well as terminating the DDS service, and it does not have to be connected to the network unless it is trying to distribute or collect the split packages.

- **DataHost** – A "not reliable" personal computer that receives and stores the split data. It can store packages for multiple DataOwners as long as its capacity is not met, and pings the Manager when it is up and running.

- **Manager** – A "somewhat reliable" server that serves as an intermediary between the DataOwner and a group of DataHosts. It keeps track of a number above threshold of DataHosts alive that host the packages it is responsible for, and while maintaining the distributed data, it pings the ServiceProvider to indicate that it is up and running.

- **ServiceProvider** – The "very reliable" administrator of DDS, and it maintains lists of DataOwners, DataHosts and Managers participating in the system. It provides pointers (or IP addresses) to necessary number of components whenever DataOwners or Managers request for Managers or DataHosts, respectively. It keeps track of how much drive space each DataHost has available for DDS, as well as which Managers are serving a DataOwner and which ones are not. It maintains three lists – one that contains information on all DataHosts and their available storage capacities, one that contains information on the Managers and the number of DataOwners they are serving, and one that contains information on DataOwners using the DDS service.

A single instance of data backup involves the ServiceProvider, one DataOwner, a number of Managers equaling the number of packages DataOwner generates, and as many DataHosts required to keep their alive count above a threshold at all times. We will give you a scenario describing how a data actually gets backed up in the next section.

2.4 DDS At Work

DDS has five overall states per instance of life cycle – that is, for every instance of DataOwner (Figure 2-1). Beginning with the Start state in which the ServiceProvider is listening for incoming connections, the system enters the Pre-Init state when a new Manager or a new DataHost makes a connection to the ServiceProvider for its initial participation in the DDS. The ServiceProvider continues to accept new Manager/DataHost participants until a DataOwner makes a contact to use the service. In the Init state, the DataOwner splits the file to be backed up and distributes the packages to the Managers. The Managers then distributes the packages to the DataHosts, driving the system into the Maintain state. In this state, the ServiceProvider, the Managers and the DataHosts work together to ensure that all the packages are kept alive and available, and when the number of either the Managers or the DataHosts that are alive goes down below a threshold, the system goes back to the Init state

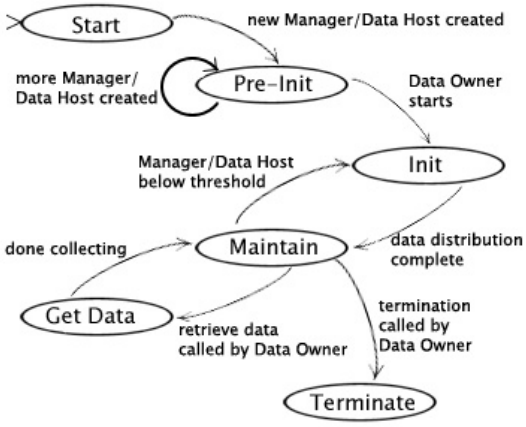

Figure 2-1: The states of DDS

to replace the "dead" Managers/DataHosts with ones that are alive.

While the system is in the Maintain state, the DataOwner can make a request to retrieve the distributed packages and rebuild the original data. The system goes into the GetData state in which the DataHosts sends out the packages and the Managers relay them back to the DataOwner. When the DataOwner feels that the DDS service is no longer needed, it can call to terminate the instance of the service. In the Terminate state, the DataHosts dump the packages that belong to the terminating DataOwner, and the Managers likewise make themselves available to manage packages for other DataOwners.

Now that you know how the system works in a nutshell, in the following subsections, we will explain what the DDS components are doing in each state and describe the actual interaction between the DataOwner, DataHosts, Managers and the ServiceProvider.

2.4.1 Start

The ServiceProvider must be running at all times regardless of the lifecycle of the system, and in the Start state, it is listening for incoming connections while maintaining the three lists described in the previous section.

2.4.2 Pre-Init

The system starts off and enters the Pre-Init state when either a new Manager or DataHost presents itself to ServiceProvider that it is ready to serve the DDS (Figure 2-2).

1. Manager contacts the ServiceProvider, giving its address and the number of DataOwners it is willing to serve.

2. ServiceProvider accepts the Manager as a new Manager, puts it in the list of Managers that is presorted by the number of DataOwners it can serve, and assigns a unique ID to it. The first Manager in the list has the greatest number of DataOwners it is willing to manage for.

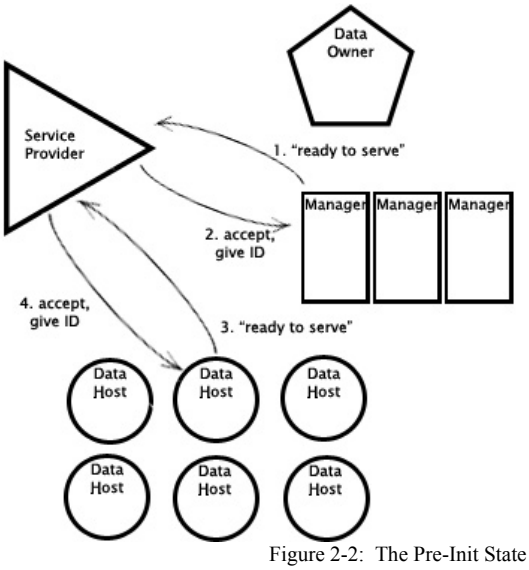

Figure 2-2: The Pre-Init State

3. DataHost contacts the ServiceProvider, giving its address as well as its initial capacity it is willing to provide for the service.

4. ServiceProvider accepts the DataHost as a new DataHost, puts it in the DataHost list according to the available capacity and assigns a unique ID to it. The first DataHost in the list has the most capacity.

2.4.3 Init

The system keeps on accepting new Managers and DataHosts until a DataOwner comes into the play, at which time the state changes to Init (Figure 2-3). In this state, the DataOwner splits the data to be backed up, requests the SP to give a certain number of Managers, and distributes the packages to the Managers. The number of Managers equals the number of packages the data is split into, and each Manager receives two consecutive packages. Each Manager is held responsible for

two packages, and this allows for two copies of a single package to be distributed to different Managers – this is done in protection against loss of data due to a failure in a Manager, just as the Petal system does [1]. If a Manager dies or loses the two packages it had, those packages can be retrieved from other Managers maintaining the same packages.

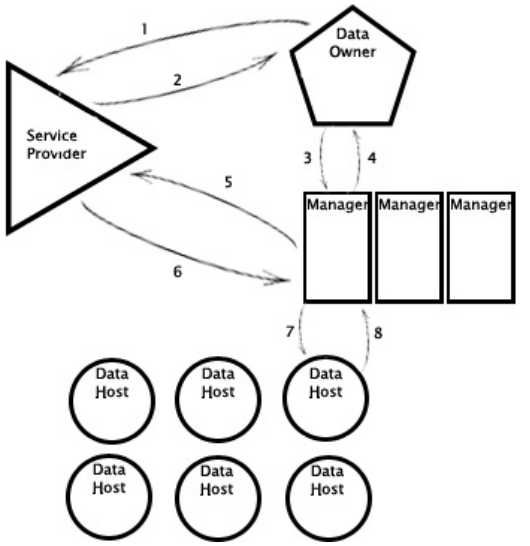

Figure 2-3: The Init State

In Init state, the Managers that have been assigned to a DataOwner make requests to get DataHosts from the ServiceProvider and distribute the packages to them. Here, the threshold defined by the Manager determines the number of DataHosts per package. DataHosts are not reliable as indicated in section 2.3, hence mirroring is used to make sure there are certain number of copies of packages available among DataHosts at all times. In essence, the threshold is the minimum number of mirrors available at any given time.

1. New DataOwner initializes by dividing the data into N packages. It gives the name of the original file to the ServiceProvider and asks for N Managers.

2. The ServiceProvider picks N Managers from the beginning of the Managers list and gives the addresses of the Managers to the DataOwner. The ServiceProvider resorts the Managers list using the binary search according to the number of DataOwners the Managers can still serve. Meanwhile, the ServiceProvider assigns a unique ID to the DataOwner.

3. The DataOwner, after receiving the IP addresses to N Managers, opens socket connections to them and sends two of the split packages to each of them.

4. When the two packages are successfully received, the Manager sends an ACK to the DataOwner.

5. The Manager asks the ServiceProvider for M DataHosts, where

   M = 2 * T, where T = threshold for the number of DataHosts per package.

6. The ServiceProvider gives the IP addresses of the top M DataHosts to the Manager. Note that based on Aussmption 6, there are enough DataHosts with sufficient drive space to accommodate the Manager. The ServiceProvider resorts the DataHost list using the binary search according to the available capacity that each DataHost has.

7. The Manager opens socket connections to the give DataHosts and distributes the packages to them. Each DataHost hosts one package for a single DataOwner.

8. Upon receipt of the package, the DataHost sends an ACK to the Manager, and this completes the Init state of the system.

2.4.4 Maintain

Once the distribution of the packages is complete, the system enters the Maintain state (Figure 2-4). The figure shows the state in which there are three packages total (P1, P2, P3) with the threshold value T = 3. The shown DataHosts all belong to the Manager managing P1 and P2. In this state, the DataHosts, the Managers and the ServiceProvider exchange messages to ensure that the distributed packages are available at all times.

DataHosts ping the Manager to indicate that they are alive while Manager keeps count of the number of pings that it receives from each DataHost in a set period of time. If the number of pings goes below the set threshold indicating that the number of living DataHosts is becoming critical, the system goes back to the Init state to give the Managers more DataHosts to fill the gaps.

In the Maintain state, Managers also ping the ServiceProvider to indicate that they are still up and running. The ServiceProvider, similar to the Managers, keeps track of the number of the pings received from the Managers, and if a Manager seems to have gone down, the system goes back to the Init state and the packages managed by the dead Manager are retrieved and redistributed to new Managers.

1. The DataHost pings its Manager every set period of time. The Manager keeps track of the pings it receives from each DataHost, and if the total number of pings from all of its DataHosts goes below the threshold T, then the Manager asks the ServiceProvider for more DataHosts. The Manager retrieves its two packages from the DataHosts that are alive and distributes them to the new DataHosts.

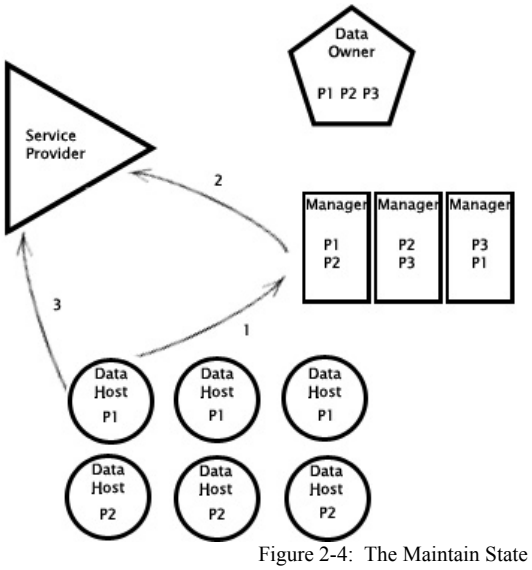

Figure 2-4: The Maintain State

2. The Manager pings the ServiceProvider every set period of time. The ServiceProvider keeps track of the pings it receives from each Manager, and if it is determined that a Manager is no longer alive, it retrieves the packages that were managed by the dead Manager and redistributes them to another Manager in the Manager list.

3. Periodically, the DataHost contacts the ServiceProvider and informs it of its current capacity. This helps the ServiceProvider maintain the updated capacity of all DataHosts, enabling a more accurate tracking of the DataHosts.

Whenever the pings are sent to the Managers and the ServiceProvider, the information on the IP address is updated. This dynamic update ensures that all of the Managers and the DataHosts are never lost in the broad web of network – they can be located as long as they are up and running.

2.4.5 Get Data

When the DataOwner requests retrieval of the distributed packages to reconstruct the original data, the DDS enters the Get Data state (Figure 2-5). DataOwner sends the retrieval request to the ServiceProvider, and the ServiceProvider relays the message to the appropriate Managers to collect the packages from their DataHosts. The Manager checks and sees if the two packages received from the DataHosts are intact, and if they are, sends them up to the DataOwner. When the DataOwner successfully receives the requested packages, the system goes back to the Maintain state.

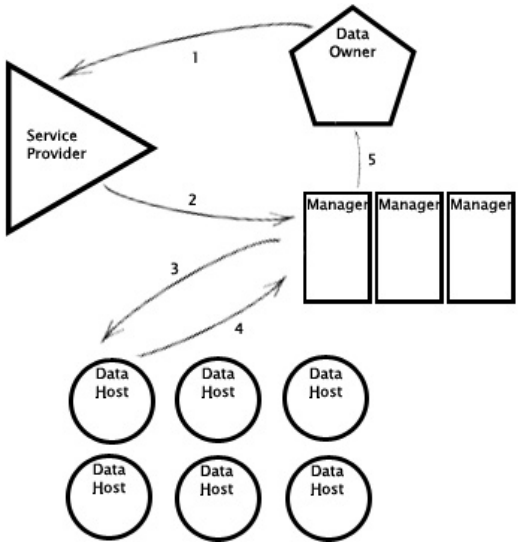

Figure 2-5: The Get Data State

1. The DataOwner contacts the ServiceProvider and makes a retrieval request. The DataOwner must wait while the packages are being delivered.

2. Upon receipt of the retrieval request from the DataOwner, the ServiceProvider determines which Managers are managing

packages for the DataOwner and tells those Managers to collect packages.

3. The Manager determines which DataHosts are hosting the requested packages and tells them to send the files.

4. The DataHost sends the appropriate package to the Manager.

5. If the received packages are intact, the Manager relays them up to the DataOwner that has been waiting for the package delivery. Once the DataOwner receives all the required packages, it proceeds to rebuild the original file.

2.4.6 Terminate

The DataOwner calls termination of an instance of DDS when it no longer needs a distributed data backup (Figure 2-6). It sends a terminal message to the ServiceProvider that triggers the termination of this particular instance of the service.

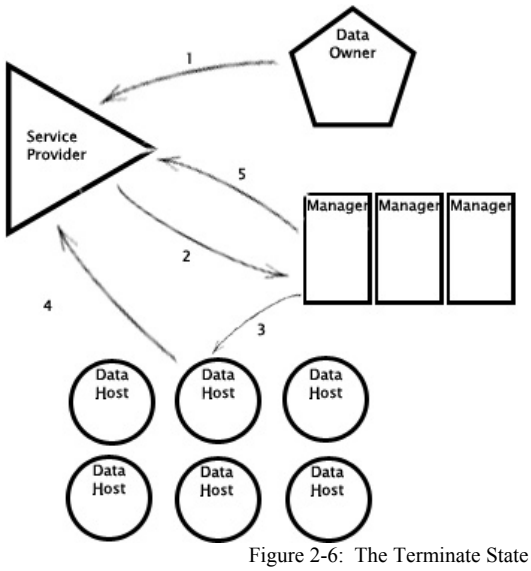

Figure 2-6: The Terminate State

In this state, the DataHost purges the package it was holding for the DataOwner and notifies the ServiceProvider with the new space capacity. The Manger also notifies the ServiceProvider that its duty to serve the DataOwner has been terminated and that it now has one more slot to manage for a new DataOwner.

Although this instance of the data management has been terminated, the DDS itself is not terminated. The ServiceProvider is still running and maintaining the three lists, and other DataOwners along with Managers and DataHosts are still active.

1. When the DataOwner is ready to terminate the data backup, it sends a termination request to the ServiceProvider. The ServiceProvider marks the DataOwner dead in its DataOwner list.

2. The ServiceProvider determines which Managers are managing packages for the terminating DataOwner and sends them a "terminate" message.

3. When the "terminate" message is received, the Manager likewise sends the "terminate" message to all of its DataHosts.

4. The DataHost purges the package that it was holding for the terminating DataOwner and notifies the ServiceProvider with its new drive space capacity. The ServiceProvider, upon receipt of the new capacity, updates the DataHosts list.

5. Manager updates its managing status by making one slot available for a new DataOwner. It notifies the ServiceProvider of its new capacity, and the ServiceProvider updates and resorts its Managers list accordingly.

We have just covered the basic lifecycle of the DDS ignoring any possible errors or any sort of problems that may arise. In the next section, we will look at some of the possible problems that DDS may encounter and how they may be prevented or fixed.

3. FUTURE WORK

3.1 Problems and Possible Solutions

The DDS presented here has only been idealized and nothing of great extent has been made concrete or even tested out. Again, the design needs much more reconstruction and revision before it can actually serve its roles, but here are some of the obvious problems that the system currently faces.

3.1.1 Message Passing Failure

The DDS spends most of its time in the Maintain state, and the message passing becomes vital in retaining all of the packages. The system requires the network to be error-free just as assumed in the earlier section (Assumption 1) because if the pings do not get passed on successfully, then more and more of the Managers and the DataHosts need to sacrificed to cover for the incompetent ones. Since the DDS is intended for a large volume data backup, the demand for frequent package transfers would consume a great amount of resources.

Now, let's say that the network is completely free of errors as assumed and that the message passing is always successful *as long as they are executed*. This seems to prevent any waste of the resources caused by unsuccessful message passing, but needless to say, things can still go wrong. Assumption is that the only "very reliable" component of the system is the ServiceProvider and that the Managers are not so reliable. What happens if the Manager goes down for a prolonged duration of time? If the ServiceProvider determines that the Manager is dead, then it will start retrieving all the packages that the dead Manager was responsible for and redistributes them to a new Manager and DataHosts. Again, this consumes a lot of resources, and in this world of scarce resources, such event is not favored at all.

The simplest solution to this problem is to increase the time the ServiceProvider or the Managers wait for the pings. This modification in both the ServiceProvider and the Managers allows for a multiple failures of message passing and a longer downtime of the servers. As long as the DataHosts or the Managers can send out pings before the wait time expires, they are deemed alive and hence no extra Managers/DataHosts are summoned for unnecessary service.

A downside to this solution is that it makes the DDS less reliable. What if the ServiceProvider is waiting for a long time for a ping from a Manager thinking that it is still alive, while in truth that Manager is dead and not going to come back up soon? While the ServiceProvider waits for the ping from the dead Manager, the DataOwner associated with the Manager may not be able to retrieve all of its packages.

The optimal wait time for responses differs depending the reliability of the network and how long each Manager or the DataHost goes down on average, and the optimization is possible only through testing and/or modification of the system to allow for dynamical update of wait time.

3.1.2 Frequent Downtime of the DataHosts

One of the purposes of DDS is to make a good use of unused drive spaces in any computer with network capability, but one must always remember that they can go down as frequently and as long as they want. While utilization of personal computers is what makes DDS unique and distinguishable from other distributed file systems that many predecessors have worked on, it is also one of the biggest challenges that the system faces.

The DDS must prioritize reliability to serve its purposes, and for that reason it creates multiple mirrors to which a package is distributed. One can expect the number of mirrors for a single package to be large in order to complement for the frequent downtime of the DataHosts, but once again, this can cause inefficiency in resource allocation. The size and the quantity of the packages would be substantially large, and if the system were to make many copies of them and distribute them to the DataHosts, the terabytes of virtual space may run out in no time. Again, experiments and simulations as well as a dynamically updating algorithm are necessary in order to minimize the number of mirrors allocated for a single package while achieving a superior reliability. The applicability of DDS is still questionable, and it must be considered if it is worth the hassle to use such unreliable servers to backup data.

3.1.3 Data Transfer Efficiency

When a Manager goes down for a long time, the ServiceProvider creates a replacement of the Manager and the system goes through the entire process of distributing the data to new DataHosts through the new Manager. Although the assumption is that the Managers are fairly reliable and do not go down frequently and for a long duration of time, multiple failures among them can cause transfers summing up to tens of hundreds of gigabytes through the network. The speed of the connections networking the DataHosts may not necessarily be 1 Gbps, and such an enormous data transfer is too much of a burden on the network that links the DataHosts to the Managers and therefore must somehow be lightened.

One solution is to begin the redistribution of the lost files with a lower threshold; that is, if the required number of DataHosts for a package was 5 with the dead Manager, cut it down to a lower

number like 2 when the packages are being redistributed. If the old Manager comes back up eventually, then the 2 new DataHosts can merge with the original DataHosts – which only takes an update in the information tracked by the Manager. This will increase the number of DataHosts associated with the Manager well above the threshold, giving a "grace resource" to the Manager. And, if the former Manager does not come back up for a long time after the redistribution of the packages, then the threshold can be increased gradually and thereby send the packages to more DataHosts.

In this solution, the timing at which the ServiceProvider increases the threshold becomes vital. If it increases too rapidly, it could be a great burden on the network, and if it is too slow, then the packages may become unavailable or get lost with the limited number of unreliable DataHosts. Further research must be conducted in order to optimize this solution.

3.1.4 Invasion of the System by Outsiders

So far, simple message passing keeps all the components of the DDS intact. The unique ID provided by the ServiceProvider upon initialization and pre-initialization establishes the identity of each of the components, and that facilitates the message passing and tracking of who is alive and who is not. While this allows for the simple design described in this paper, it leaves so many security holes for network hackers. For example, some random computer on the network can fabricate an ID that matches the one ServiceProvider generated, and in effect, enter the system without anyone knowing it. If the intruder is cynical enough, he may very well choose to execute the terminate command and wipe out all the backed-up data.

Doing anything on the network essentially creates an environment that is never 100% secure, yet at least some considerations must be given to security issues in order for the system to be realized. Some encryption methods involving public and private keys may be used to prevent any outsiders to enter the system, but this is an area that can be discussed in another paper dedicated solely to it. As of this point of the research, the ServiceProvider makes use of a unique key associated with an ID, and every time the ServiceProvider receives a command, it checks if the key provided within the command matches the ID. This, however, is a very weak form of security, and the security issue must be considered to a greater extent.

3.1.5 Recovery of Data

The DDS serves its purpose when it successfully rebuilds the data that the DataOwner loses. In order to rebuild the data, DataOwner only needs to call for retrieval of the distributed packages. But under certain circumstances a problem may arise; as mentioned in the introduction of this paper, the system should be able to recover the lost data even if everything is wiped out on the DataOwner's side. But then, how is the DataOwner supposed to re-link itself with the ServiceProvider to execute the retrieval call if they have lost everything, including the ID and the key?

The simplest solution is to have the unique ID and the key stored in a very safe place. As of now, all it takes for the ServiceProvider to recognize a DataOwner is through the ID and a matching key, so as long as these two are kept in tact, then the ServiceProvider can retrieve the packages for the Owner. However, just the fact that the ID and the key must *never* be lost defeats the purpose of the DDS, with which one should be able to rebuild the data *no matter what*. Moreover, a security issue arises. Similar to the problem discussed in 3.1.4, what if an outsider disguises as a DataOwner, requests the ServiceProvider to link the Managers to him and rob the entire data? If the ID and the key are so easy to carry around, then it would not be too difficult to replicate them. To prevent such intrusion, the ServiceProvider must be able to recognize and verify only the genuine DataOwner. A better method of secure client recognition is another area that needs to be researched further.

3.1.6 Dependency on the ServiceProvider

In the DDS, there is only one ServiceProvider that does the resource allocation and component tracking. Although, for this project, the assumption is made so that the ServiceProvider is free of errors, it does not work out that nicely in reality. If the ServiceProvider goes down for any reason, all the distributed data is lost and hence the system becomes nothing but a resource-eating ghost.

To prevent any permanent damage to the system, the ServiceProvider may be replicated or backed up for emergency needs. This, however, does not guarantee that all of the ServiceProviders will never go down, and moreover, results in more consumption of the limited resources. As the number of ServiceProviders increase, so does the amount of

overhead associated with it. The mirroring of the ServiceProvider may be too much of a burden to take, and this too must be considered in depth.

3.1.7 Network Speed

The system attempts to take a great advantage of today's technology that allows for faster transmission speed in the network. However, the DDS may be overestimating the networking capability – many computers are still using dial-up modems to connect to the network, and such slow connections may be too inadequate to handle the massive data transfers that the system requires.

The prototype of the DDS was tested out on the campus network connected by Ethernet, and the data transfer rate was right around 1 Mbps. With this speed, the system seemed to run without a problem, but if the DataHosts were to be connected through 54 Kbps modems, it would have taken too much time for the distribution and the retrieval of the packages.

Given that the system deals with a massive amount of data and that the network bandwidth is not always large, the worthiness of using the private computers as DataHosts may be questioned.

3.2 The Next Step

The first three problems presented above require repeated experiments and careful optimization to solve. In order to conduct such experiments and derive a more solid approximation of numbers, we must take the idealized model and the system prototype and keep on modifying and testing them with a larger, more diverse group of networked computers. Only then, we can start estimating the efficiency of the system and therefore determine the worthiness of the DDS.

Meanwhile, we must also construct a good algorithm that dynamically optimizes the system. Given that the performance of the DataHosts are unpredictable and that it changes from a second to the next, dynamic optimization will play a key role in making the DDS applicable or merely useful.

As far as the security issues are concerned, we may use preexisting encryption methods when it comes to the actual implementation of the system, but we may leave the details up to those who specialize in the filed.

The technological problems, such as those that are mentioned in 3.1.6 and 3.1.7, will always be there no matter how well we try to cope with them. Perhaps in the distant future there will be absolutely error-free machines and networks that can solve these problems, but as for now, we must work with what we have.

So there are numerous problems associated with the implementation of the system, but we must keep in our perfectionist minds that what is discussed in this paper is no more than a prototype and a suggested implementation of the backup system.

4. CONCLUSIONS

The system discussed in this paper is still very immature and there are far more topics and issues that need to be considered before much of significant conclusions can be drawn. However, the idealized model of the DDS we have created so far reminds us of difficulties of implementing a system that involves components that are completely unreliable and unpredictable. Use of dynamic optimization algorithms is recommended for the system, and it is probably one of the biggest areas that need to be further studied. The implementation of the system is theoretically possible, but given the restraints in the amount of available resources and technology we have today, the DDS may not be 100% reliable. Since backup media and technology are constantly improving from day to day, perhaps it is not worth archiving a large data remotely in networked computers.

Nonetheless, technology similar to the DDS may be used to serve as a file sharing system for large volume of data or as a highly secure method of warehousing sensitive data (it may take ages for a computer hacker to get his hands on the packages that are spread all over the world!). The applicability of the system is not limited, and it is only up for imagination to implement the system in marvelous ways.

Further research with tens of hundreds of testing is necessary to optimize the system to the fullest extent and to determine the usefulness of the system. However, it is my wish that this paper will serve as guidance to the next level of research and a cornerstone for the DDS and its future derivatives.

5. RELATED WORK

There are several papers that opened the doors to the idea for the basic structure of DDS. Jun Rao et al. discusses a way to automate data partitioning that ensures optimal performance

upon parallel execution of the data by multiple processes. Although no execution of the data occurs in DDS that may require efficient partitioning, the idea of making the system more efficient through parallel processing has given rise to the concept of implementing Managers. Since there are multiple Managers serving the DataOwner, the workload for the Managers and the DataOwner is cut down significantly [2].

David Lomet presents a way in which data can be backed up from "off-line" media that does not get modified while restoration is in progress. The method is very handy for any backup that involves write functionality, but since DDS assumes that the archived data is only read and not written into, it does not quite apply to the system [3].

Leslie Lamport's Paxos discusses how different processes in a distributed system can execute efficiently and without any inconsistency, but it does not apply to DDS that involves no execution of multiple processes that share a critical section [4].

Stefan Ludwig and Winfried Kalfa introduce Fairly Secure File System (FSFS) in which files are encrypted at the kernel level. This paper was the initial point at which encryption ideas for DDS, such as the need for multiple keys for access of the files by multiple users, originated. But since DDS is handled at the software level rather than at the kernel level and it assumes that only one party, namely the DataOwner, requires the access to the file, FSFS does not directly apply to DDS and other ways of encryption still needs to be sought [5].

Edward Lee and Chandramohan Thekkath present the Petal, a concept by which virtual disks are created from distributed systems. The foundation of DDS lies in the Petal system, for it has spawned the idea of having a global state manager (the ServiceProvider), of having multiple copies of a package in different servers for heightened reliability, of overhead reduction, of requirements for large storage space due to mirroring, and of message passing among the components of the system [1].

6. ACKNOWLEDGEMENTS

I would like to thank Professor Elizabeth Borowsky for her generous and patient assistance and guidance throughout the 2002-2003 academic year. I would also like to thank Phil Temples for his kind support with Unix when I was testing out the prototype. And finally, I would like to thank my family for their warm words of encouragement and especially my father for his brilliant idea that gave birth to this project.

7. REFERENCES

[1] Edward Lee and Chandramohan Thekkath. Petal: Distributed Virtual Disks. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages84-92, September 1996.

[2] Jun Rao, et al. Automating Physical Database Design in a Parallel Database. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 558-569, 2002.

[3] David Lomet. High Speed On-line Backup When Using Logical Log Operations. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 34-45, 2000.

[4] Leslie Lamport. The Part-Time Parliament. *ACM Trans. On Comp. Sys., Volume 16, Number 2*, pages 133-169, May 1998.

[5] Stefan Ludwig and Winfried Kalfa. File System Encryption with Integrated User Management. *ACM SIGOPS Op. Sys. Review, Volume 35, Issue 4*, pages 88-93, October 2001.

Appendix A – Notes on the Codes

A.1 HJSplit

The prototype of the DDS was coded in Java, and it implements HJSplit for Java 1.0, a class that splits a file of any size into smaller files and rebuilds them back to the original file. The original code was written by Henk Hagedoorn and it was rewritten for Java by Rhesa Rozendaal.

A.2 Classes

There are 25 classes total including the HJSplit class, and a brief description of each class is given in the header of the class source code. Each of the components in the DDS must have the following classes (the * indicates the classes that need to be executed to run the program):

ServiceProvider:
- Commands.java
- CommandSender.java
- DataHostData.java
- DataOwnerData.java
- FileTracker.java
- ListConsole.java
- ManagerData.java
- ServiceProviderDriver.java *
- SPConnect.java

DataOwner:
- Commands.java
- CommandSender.java
- DataOwnerData.java
- DataOwnerDriver.java *
- DOCommandListener.java
- DOConnect.java
- FileReceiver.java
- FileSender.java
- FileTracker.java
- HJSplit.java
- ManagerData.java

Manager:
- Commands.java
- CommandSender.java
- DataHostData.java
- DHInfo.java
- DHLocator.java
- DHTracker.java
- FileDistributer.java
- FileReceiver.java
- FileSender.java
- FileTracker.java
- ManagerData.java
- ManagerDriver.java *
- MNCommandListener.java
- MNConnect.java

DataHost:
- Commands.java
- CommandSender.java
- DataHostData.java
- DataHostDriver.java *
- DHCommandListener.java
- DHConnect.java
- FileReceiver.java
- FileSender.java
- FileTracker.java
- ManagerData.java

A.3 Current Implementation

Depending on where the ServiceProvider resides, the IP address information in the following classes must be changed accordingly:

- DataHostDriver.java
- DataOwnerDriver.java
- DHConnect.java
- FileDistributer.java
- ManagerDriver.java
- MNConnect.java

Currently, the Maintain and Terminate states have not been implemented in the codes. The program runs fine if there is only one split package to be distributed, but if there is more than one, the DataOwner fails to send out the packages.

The program is heavily multi-threaded, and there may be some violation of simultaneously accessing critical sections. Also, the synchronization in some classes may halt the program interminably in a number of instances (i.e. when a class unexpectedly quits running). The program is vulnerable to errors and does not handle Exceptions too well.

The algorithm that presorts the lists in ListConsole.java have some faults and may ignore the first element in the list.

A.4 Commands

The system relies heavily on message passing that is done by exchanging instances of the Commands class. Here is a brief summary of the numeric commands used by the current implementation. Service Provider generates all one-digit commands. Other components generate two-digit commands, and the ones

starting with 1 is generated by DataOwner, with 2 by Manager, and with 3 by DataHost.

Commands	Function
1	Check to see if Manager is alive, if it is, make it inactive
2	Send a list of Managers to DataOwner
3	Check to see if DataHost is alive, if it is, make it inactive
4	Send a list of DataHosts to Manager
5	Tell Manager to send packages to DataOwner
10	Initiate DataOwner
11	Give the original file name to ServiceProvider
12	Tell ServiceProvider that I'm about to distribute packages
13	Tell Manager to get ready to receive a package
14	Package sent, make the Manager active
15	Give a list of my Managers to ServiceProvider
16	Package not sent, make the Manager active
17	Tell ServiceProvider that I want to retrieve packages
20	Initiate Manager
21	N/A
22	Tell ServiceProvider that I'm active
23	Tell ServiceProvider that I'm about to distribute packages to DataHosts
24	Tell DataHost to get ready to receive a package
25	Package sent, make the DataHost active
26	Package not sent, make the DataHost active
27	Tell DataHost to send a package
28	Tell DataOwner to get ready to receive a package
30	Initiate DataHost
31	N/A
32	Tell ServiceProvider that I'm active
33	Tell manager to get ready to receive a package

A.5 References for the Codes

Café au Lait. Java Socket Programming. Retrieved November 3, 2002 from http://www.cafeaulait.org/slides/sd2003west/sockets/Java_Socket_Programming.html.

IBM. Writing Efficient Thread-Safe Classes. Retrieved February 14, 2003 from http://www-900.ibm.com/developerWorks/cn/java/threadsafe/index_eng.shtml.

Kobu.com. Java Code Samples. Retrieved February 3, 2003 from http://www.kobu.com/purejava/index-en.htm.

Linux Journal. Java and Client-Server. Retrieved February 5, 2003 from http://www.linuxjournal.com/article.php?sid=155.

Network Buyers Guide. White Paper – Storage Networking. Retrieved September 9, 2002 from http://www.networkbuyersguide.com/search/105112.htm.

Sun Microsystems. Hello Client-Server Example. Retrieved January 27, 2003 from http://java.sun.com/docs/books/tutorial/idl/hello/.

Sun Microsystems. Java Developer Connection – Copy File. Retrieved March 19, 2003 from http://forum.java.sun.com/thread.jsp?thread=28255&forum=17&message=69505.

Sun Microsystems. Reading and Writing Data Code Samples. Retrieved February 5, 2003 from http://developer.java.sun.com/developer/codesamples/rw.html.

Sun ONE Middleware Developer. Java Code Samples. Retrieved January 30, 2003 from http://developer.iplanet.com/docs/examples/java.html.

Appendix B – Code Listing

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider, DataOwner, Manager, DataHost - Commands.java

Object that gets passed around between the components as messages
*****************************************************************/

import java.io.*;
import java.util.*;

public class Commands implements Serializable
{
	private long myID = -1;
	private double myKey = -1;
	private int command = -1;
	private int num = -1;
	private int num2 = -1;
	private long lNum = -1;
	private String str = null;
	private String str2 = null;
	private Date time = null;
	private Object o = null;

	public Commands(long id, double key, int cmd)
	{
		time = new Date();

		myID = id;
		myKey = key;
		command = cmd;
	}

	public Commands(long id, double key, int cmd, int x)
	{
		time = new Date();

		myID = id;
		myKey = key;
		command = cmd;

		num = x;
	}

	public Commands(long id, double key, int cmd, long x)
	{
		time = new Date();

		myID = id;
		myKey = key;
		command = cmd;

		lNum = x;
	}

	public Commands(long id, double key, int cmd, int x, long y)
	{
		time = new Date();

		myID = id;
		myKey = key;
		command = cmd;

		num = x;
		lNum = y;
	}

	public Commands(long id, double key, int cmd, String s)
	{
		time = new Date();

		myID = id;
		myKey = key;
		command = cmd;

		str = s;
	}

	public Commands(long id, double key, int cmd, String s, int i)
	{
		time = new Date();

		myID = id;
		myKey = key;
		command = cmd;

		str = s;
		num = i;

	}
```

```
public Commands(long id, double key, int cmd, String s, String s2)
{
	time = new Date();

	myID = id;
	myKey = key;
	command = cmd;

	str = s;
	str2 = s2;
}

public Commands(long id, double key, int cmd, String s, long l)
{
	time = new Date();

	myID = id;
	myKey = key;
	command = cmd;

	str = s;
	lNum = l;
}

public Commands(long id, double key, int cmd, String s, long l, int i)
{
	time = new Date();

	myID = id;
	myKey = key;
	command = cmd;

	str = s;
	lNum = l;
	num2 = i;

}

public Commands(long id, double key, int cmd, Object ob)
{
	time = new Date();

	myID = id;
	myKey = key;
	command = cmd;

	o = ob;
}
```

```
public long getID()
{
	return myID;
}

public double getKey()
{
	return myKey;
}

public int getCommand()
{
	return command;
}

public int getNum()
{
	return num;
}

public int getNum2()
{
	return num2;
}

public long getLong()
{
	return lNum;
}

public String getString()
{
	return str;
}

public String getString2()
{
	return str2;
}

public Date getTime()
{
	return time;
}

public Object getObject()
```

```
	{
		return o;
	}

	public void clearObject()
	{
		o = null;
	}
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider, DataOwner, Manager, DataHost - CommandSender.java

Class that sends commands created with Commands class
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class CommandSender
{
	private String address = "";
	private Commands command;
	private int port = 3000;
	private final int TIMEOUT = 60 * 1000;

	public CommandSender(String ad, Commands cm)
	{
		address = ad;
		command = cm;
	}

	public CommandSender(String ad, int pt, Commands cm)
	{
		address = ad;
		port = pt;
		command = cm;
	}

	public boolean sendCommand() throws Exception
	{
	ObjectOutputStream oos = null;
	ObjectInputStream ois = null;
	Socket socket = null;

	try
	{
	// open a socket connection
	socket = new Socket(address, port);
	socket.setSoTimeout(TIMEOUT);
```

```
		// open I/O streams for objects
		oos = new ObjectOutputStream(socket.getOutputStream());
		ois = new ObjectInputStream(socket.getInputStream());

			//send the command
		oos.writeObject(command);
		oos.flush();

		//receive ACK
		Commands ack = (Commands) ois.readObject();

		oos.close();
		ois.close();

		if(ack.getID() == command.getID() && ack.getTime().equals(command.getTime()))
		{
			System.out.println("Command successful: " + command.getCommand());
			return true;
				}
		else
		{
					System.out.println("Command unsuccessful: " +
command.getCommand());
					return false;
				}
			}
		catch(Exception e)
		{
		System.out.println(e.getMessage());
		return false;
		}
		}
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider, Manager, DataHost - DataHostData.java

Object that stores information on a DataHost
*****************************************************************/

import java.io.*;
import java.util.*;

public class DataHostData implements Serializable
{
	private long id; //The ID
	private double key; //The Key
	private int capacity; //The total capacity of shared drive space
	private int used = 0; //Drive space used for the service
	private String addrs = ""; //My IP address
	private boolean active = true; //Available for service
	private Vector fTracker = null; //File tracker for each file hosted

	public DataHostData(long i, double k, int c, String s)
	{
		id = i;
		key = k;
		capacity = c;
		addrs = s.substring(1);
		fTracker = new Vector();
	}

	public long getID()
	{
		return id;
	}

	public double getKey()
	{
		return key;
	}

	public int getCapacity()
	{
		return capacity;
	}
```

```
	public int getUsed()
	{
		return used;
	}

	public int getAvailable()
	{
		return capacity - used;
	}

	public String getAddress()
	{
		return addrs;
	}

	public boolean isActive()
	{
		return active;
	}

	public void setAddress(String s)
	{
		addrs = s.substring(1);
	}

	public void incUsed(int a)
	{
		used += a;
	}

	public void setActive()
	{
		active = true;
	}

	public void setInactive()
	{
		active = false;
	}

	public void addTracker(FileTracker ft)
	{
		fTracker.add(ft);
	}
}
```

```
/**********************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataHost - DataHostDriver.java

Driver class for a DataHost
**********************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class DataHostDriver
{
	private final int MB = 100;//in MB
	private final int CAPACITY = 1024000*MB;//in bytes
	private final String FILE = "DataHost.dat";
//	private final String ADDRS = "127.0.0.1";
//	private final String ADDRS = "136.167.212.93";
	private final String ADDRS = "136.167.117.84"; //IP Address of the
ServiceProvider
	private final int SPPORT = 3000;
	private final int TIMEOUT = 60 * 1000;

	private DataHostData data;

	public DataHostDriver()
	{
	}

	public static void main(String argv[]) throws Exception
	{
		////////////////////////////////////////////////////////////////////////
		System.out.println("Boston College");
		System.out.println("Computer Science Department");
		System.out.println();
		System.out.println("Senior Thesis 2003");
		System.out.println("Distributed Data Storage: Data Backup Over
Networks");
		System.out.println("(c) Hiroyuki Hayano");
		System.out.println("Prof. Elizabeth Borowsky");
		System.out.println();
		System.out.println();
		////////////////////////////////////////////////////////////////////////
```

```
        DHCommandListener cl = new DHCommandListener();
        cl.start();

        DataHostDriver dh = new DataHostDriver();
        dh.startSession();
        }

        private void startSession() throws Exception
        {
        readData();

        if(data==null)
        {
                        System.out.println("Data file not found or corrupt, setting up
DataHost...");
                        setupDataHost();
                }

                //Tell ServiceProvider that I'm online
                data.setActive();
                writeFile(data, FILE);
                Commands c = new Commands(data.getID(), data.getKey(), 32, data);
                CommandSender cs = new CommandSender(ADDRS, c);
                cs.sendCommand();
        }


        private void readData() throws Exception
        {
                //attempt to read data from DataHost.dat
                try
                {
                        System.out.println("Reading " + FILE);
                        FileInputStream fis = new FileInputStream(FILE);
                        ObjectInputStream ois = new ObjectInputStream(fis);
                        data = (DataHostData) ois.readObject();

                        ois.close();
                        fis.close();
                }
                catch(Exception e)
                {
                        data = null;
                }
        }

        //Method to write to an object to a file
        private void writeFile(Object o, String fName)
        {
try
{
                        FileOutputStream fos = new FileOutputStream(fName);
                        ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                        oos2.writeObject(o);
                        oos2.flush();

                        System.out.println(fName + " written");

                        oos2.close();
                        fos.close();
                }
                catch(Exception e)
                {
                        System.out.println("Cannot write to file: " + fName);
                }
        }

        //Method to setup a DataHost first time it is run
        private void setupDataHost() throws Exception
        {
        ObjectOutputStream oos = null;
        ObjectInputStream ois = null;
        Socket socket = null;

        Commands command = new Commands(-1, -1, 30, CAPACITY);

        try
        {
        // open a socket connection
        socket = new Socket(ADDRS, SPPORT);
        socket.setSoTimeout(TIMEOUT);

        // open I/O streams for objects
        oos = new ObjectOutputStream(socket.getOutputStream());
        ois = new ObjectInputStream(socket.getInputStream());

        //send the command
        oos.writeObject(command);
        oos.flush();

        //receive the unique id
        data = (DataHostData) ois.readObject();

        oos.close();
        ois.close();
```

```
		//write the data to DataHost.dat
		synchronized(data)
		{
			FileOutputStream fos = new FileOutputStream(FILE);
					ObjectOutputStream oos2 = new ObjectOutputStream(fos);
					oos2.writeObject(data);
					oos2.flush();

					System.out.println(FILE + " created");

					oos2.close();
					fos.close();
				}

				System.out.println("DataHost setup complete");
				System.out.println("ID: " + data.getID());
				System.out.println("Maximum capacity: " + CAPACITY + " bytes");
			}
		catch(Exception e)
		{
		System.out.println(e.getMessage());
		}
		}
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider, DataOwner - DataOwnerData.java

Object that stores information on a DataOwner
*****************************************************************/

import java.io.*;
import java.util.*;

public class DataOwnerData implements Serializable
{
	private long id; //The ID
	private double key; //The Key
	private String file = null; //Name of the original file
	private String addrs = ""; //My IP address
	private Vector managers = null; //A list of my Managers
	private int numFiles = 0; //Number of packages

	public DataOwnerData(long i, double k, String s)
	{
		id = i;
		key = k;
		addrs = s.substring(1);

		managers = new Vector();
	}

	public long getID()
	{
		return id;
	}

	public double getKey()
	{
		return key;
	}

	public String getFileName()
	{
		return file;
	}

	public String getAddress()
```

```
	{
		return addrs;
	}

	public Vector getManagers()
	{
		return managers;
	}

	public int getNumFiles()
	{
		return numFiles;
	}

	public void setFileName(String s)
	{
		file = s;
	}

	public void setNumFiles(int i)
	{
		numFiles = i;
	}

	public void setAddress(String s)
	{
		addrs = s.substring(1);
	}

	public void addManager(long m)
	{
		Long l = new Long(m);
		managers.add(l);
	}
}
```

```
/**********************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataOwner - DataOwnerDriver.java

Driver class for a DataOwner

HJSplit for Java 1.0
(c)1997-2000 Henk Hagedoorn <hjh@usa.net>
(c)1997-2000 Java version Rhesa Rozendaal <rhesa@usa.net>
**********************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class DataOwnerDriver
{
	private final String FILE = "DataOwner.dat";
//	private final String ADDRS = "127.0.0.1";
//	private final String ADDRS = "136.167.212.93";
	private final String ADDRS = "136.167.117.84"; //IP Address of the
ServiceProvider
	private final int SPPORT = 3000;
	private final int MNPORT = 3002;
	private final int TIMEOUT = 60 * 1000;
	private final int TRIALS = 5;

	private DataOwnerData data = null;

	public DataOwnerDriver()
	{
	}

	public static void main(String argv[]) throws Exception
	{
		//////////////////////////////////////////////////////////////////////
		System.out.println("Boston College");
		System.out.println("Computer Science Department");
		System.out.println();
		System.out.println("Senior Thesis 2003");
		System.out.println("Distributed Data Storage: Data Backup Over
Networks");
		System.out.println("(c) Hiroyuki Hayano");
```

```
                System.out.println("Prof. Elizabeth Borowsky");
                System.out.println();
                System.out.println();
                ////////////////////////////////////////////////////////////////////////

        DataOwnerDriver dod = new DataOwnerDriver();
        dod.startSession(argv);
        }

    //Read the parameter and take an appropriate action
    private void startSession(String[] param)
    {
        try
        {

                //initiate DataOwner----------------------------------------------------
                if(param[0].equals("-i"))
                {
                        readData();

                        if(data==null) //If the DataOwner has not been initialized
                        {
                                System.out.println("Initializing DataOwner...");
                                setupDataOwner();
                        }
                        else
                        {
                                System.out.println("DataOwner has already been
initialized");
                                System.exit(1);
                        }
                }

                //split file-----------------------------------------------------------
                else if(param[0].equals("-s"))
                {
                                readData();

                                String size;
                                File f;

                                if(data==null) //If the DataOwner has not been initialized
                                {
                                        System.out.println("DataOwner has not yet been
initialized");
                                        System.out.println("Type 'java DataOwnerDriver
-?' for help");
                                        System.exit(1);
                                }

                                else
                                {
                                        //Specify the size of each package
                                        try
                                        {
                                                size = "-s" + param[2];
                                                System.out.println("size = " +
param[2] + "kB");
                                        }
                                        catch(Exception e)
                                        {
                                                size = "-s";
                                                System.out.println("size =
1440kB");
                                        }

                                        try
                                        {
                                                f = new File(param[1]);

                                                if(!f.exists()) //If the original
file does not exist
                                                {

        System.out.println("The file " + param[1] + " cannot be read.  Please check the
file name.");
                                                        System.exit(1);
                                                }

                                                data.setFileName(param[1]);
                                                writeFile(data, FILE);

                                                //Give the name of the
original file to the ServiceProvider
                                                Commands c = new
Commands(data.getID(), data.getKey(), 11, param[1]);
                                                CommandSender cs = new
CommandSender(ADDRS, c);
                                                if(!cs.sendCommand())
                                                {

        System.out.println("The communication to server failed.  Try again later");
                                                        System.exit(1);
                                                }
```

```
                                        String newFName = "" + data.getID();
                                        File newFile = new File(newFName);
                                        copyFile(f, newFile);
                                        newFile.deleteOnExit();

                                        String command[] = {size,
newFName};
                                        System.out.println("splitting " +
param[1] + "...");


                                        //Now split the original file
                                        HJSplit.main(command);
                                }
                                catch(Exception e)
                                {
                                        System.out.println("Required
parameter missing");
                                        System.exit(1);
                                }
                        }
                }

        //distribute files-----------------------------------------------------
        else if(param[0].equals("-d"))
        {
                readData();

                DOCommandListener dl = new DOCommandListener();
                dl.start();

                        if(data==null) //If the DataOwner has not been initialized
                        {
                                System.out.println("DataOwner has not yet been
initialized");
                                System.out.println("Type 'java DataOwnerDriver
-?' for help");
                                System.exit(1);
                        }

                        String fPrefix = "" + data.getID();
                        boolean fileExists = true;
                        int i = 0;
                        int j = 0;
                        int k = 1;
                        int numFiles = 0;
                        Vector fileList = new Vector();

                        //Determine the number of packages to distribute
                        while(fileExists)
                        {
                                String s = fPrefix + "." + i + j + k;
                                File f = new File(s);
                                f.deleteOnExit();

                                if(f.exists())
                                {
                                        fileList.add(f);
                                        numFiles ++;
                                        k++;
                                        if(k>=10)
                                        {
                                                k=0;
                                                j++;
                                                if(j>=10)
                                                {
                                                        j=0;
                                                        i++;
                                                }
                                        }
                                }

                                else fileExists = false;

                        }//end while

                        if(numFiles == 0) //If there is no package to
distribute
                        {
                                System.out.println("The split files
could not be found.  Please choose option -s.");
                                System.exit(1);
                        }

                        data.setNumFiles(numFiles);
                        writeFile(data, FILE);


                        int numComplete = 0;
                        int fileIndex1 = 0;
                        int fileIndex2 = 0;
                        int outstanding = 0;
                        int attempts = 0;

                        while(numComplete < numFiles) //While all
packages are being delivered
                        {
```

```
ServiceProvider
data.getKey(), 12, outstanding);
CommandSender(ADDRS, c);


could not be grabbed.  Try again later");


//Temp file containing the Managers


readFile("reqman.dat");
Manager is given from the ServiceProvider


Managers could not be grabbed.  Try again later");


//While the packages are being sent to the Managers


(ManagerData) managers.get(a); //Grab a Manager
fileList.get(fileIndex1); //Grab a package
(int) (Math.random() * 1000.0);

to receive a package
new Commands(data.getID(), 0, 13, f1.getName(), f1.length(), tPort);
cs1 = new CommandSender(mn.getAddress(), MNPORT, c1);
```

```
outstanding = numFiles - numComplete;

//Get a number of Managers from

Commands c = new Commands(data.getID(),

CommandSender cs = new


if(!cs.sendCommand())
{
        System.out.println("The Managers

        System.exit(1);
}

else
{
        File reqman = new File("reqman.dat");

        reqman.deleteOnExit();
        Vector managers = (Vector)

        if(managers.size() == 0) //If no

        {
                System.out.println("The

                System.exit(1);
        }
        for(int a = 0; a<outstanding; a++)

        {
                try
                {
                        ManagerData mn =

                        File f1 = (File)

                        int tPort = 3004 +

                        //Prepare Manager

                        Commands c1 =

                        CommandSender
```

```
if(cs1.sendCommand())
                                                                {

//Send a package

FileSender fs1 = new FileSender(f1, mn.getAddress(), tPort);

if(fs1.sendFile())

{

fileIndex2 = fileIndex1 + 1;

if(fileIndex2 >= numFiles)

        fileIndex2 = 0;


//Send a second package

File f2 = (File) fileList.get(fileIndex2);

if(!f2.equals(f1))

{

        int tPort2 = 3004 + (int) (Math.random() * 1000.0);


        //Prepare Manager to receive another package

        Commands c2 = new Commands(data.getID(), 0, 13, f2.getName(),
f2.length(), tPort2);

        CommandSender cs2 = new CommandSender(mn.getAddress(),
MNPORT, c2);

        if(cs2.sendCommand())

        {

                //Send the second package

                FileSender fs2 = new FileSender(f2, mn.getAddress(),
tPort2);

                if(fs2.sendFile())
```

```
                {
                        //Packages are delivered, now tell Manager to be active again
                        Commands c3 = new Commands(data.getID(), 0, 14, f1.getName(),
f2.getName());
                        CommandSender cs3 = new CommandSender(mn.getAddress(),
MNPORT, c3);
                        if(cs3.sendCommand()) //If the delivery process is complete
                        {
                                fileIndex1++;
                                if(fileIndex1 >= numFiles)
                                        fileIndex1 = 0;
                                //Tell ServiceProvider who is managing for me
                                Commands c4 = new Commands(data.getID(),
data.getKey(), 15, mn.getID());
                                CommandSender cs4 = new CommandSender(ADDRS,
c4);
                                cs4.sendCommand();
                                numComplete++;
                        }
                }
                else //if 2/2 files not sent
                {
                        //Packages are not delivered, but tell Manager to be active again
                        Commands c5 = new Commands(data.getID(), 0, 16);
                        CommandSender cs5 = new CommandSender(mn.getAddress(),
MNPORT, c5);
                        cs5.sendCommand();
                        attempts++;
                        if(attempts<TRIALS * numFiles) //If package
delivery failed too many times
                                System.out.println("Distribution of
files failed, trying alternatives...");
                        else
                        {
                                System.out.println("Distribution of
files failed.  Try again later");
                                System.exit(1);
                        }
                }
        }
        else //if there is only 1 package total
        {
                //Package delivered, not tell the Manager to be active again
                Commands c6 = new Commands(data.getID(), 0, 14, f1.getName());
                CommandSender cs6 = new CommandSender(mn.getAddress(),
MNPORT, c6);
                if(cs6.sendCommand())
                {
                        //Tell ServiceProvider who is managing for me
```

```
                Commands c7 = new Commands(data.getID(), data.getKey(), 15, mn.getID());
                CommandSender cs7 = new CommandSender(ADDRS, c7);
                cs7.sendCommand();
                numComplete++;
        }
        }
                                                                        }
                                                                        else //if
no file sent
                                                                        {
        //Packages are not delivered, but tell Manager to be active again
        Commands c6 = new Commands(data.getID(), 0, 16);
        CommandSender cs6 = new CommandSender(mn.getAddress(), MNPORT, c6);
        cs6.sendCommand();

        attempts++;
        if(attempts<TRIALS * numFiles) //If package delivery failed too many times
        System.out.println("Distribution of files failed, trying alternatives...");
        else
        {
        System.out.println("Distribution of files failed.  Try again later");
        System.exit(1);
        }
                                                                        }
                                                                }
                                                        }//end try
                                                        catch(Exception e) //If
something fataly wrongs happenes in distributing packages
                                                        {
                                                                attempts++;
        if(attempts<TRIALS * numFiles)
        System.out.println("Distribution of files failed, trying alternatives...");
                                                                                else
                                                                                {
        System.out.println("Distribution of files failed.  Try again later");
        System.exit(1);
                                                                                }
                                                                        }
                                                                }//end for
                                                        }//end else
                                                }//end while

                                                System.out.println("Distribution successful");
                                                System.exit(1);
                                        }
                //retrieve packets----------------------------------------------------------
                else if(param[0].equals("-r"))
                {
                                        readData();

                                DOCommandListener dl = new DOCommandListener();
                                dl.start();

                                        if(data==null) //If the DataOwner has not been
initialized
                                        {
                                                System.out.println("DataOwner has not
yet been initialized");
                                                System.out.println("Type 'java
DataOwnerDriver -?' for help");
                                                System.exit(1);
                                        }

                                        else
                                        {
                                                //Send ServiceProvider a package
retrieval request
                                                Commands c = new
Commands(data.getID(), data.getKey(), 17);
                                                CommandSender cs = new
CommandSender(ADDRS, c);
```

```
            cs.sendCommand();
}
int numFiles = data.getNumFiles();
String prefix = "" + data.getID();
boolean filesGrabbed = false;

while(true) //While all the packages are being delivered
{
            int i = 0;
            int j = 0;
            int k = 1;

            for(int a=0; a<numFiles; a++)
            {
                        String s = prefix + "." + i + j + k;
                        File f = new File(s);

                        if(!f.exists())
                        {
                                    a = numFiles;
                                    filesGrabbed = false;
                        }

                        else
                        {
                                    filesGrabbed = true;

                                    k++;
                                    if(k>=10)
                                    {
                                                k=0;
                                                j++;
                                                if(j>=10)
                                                {
                                                            j=0;
                                                            i++;
                                                }
                                    }
                        }
            }

            if(filesGrabbed)
                        break;
}

int o = 0;
int p = 0;
int q = 1;
                                                String fName = data.getFileName();

                                                //Rename the packages to match the original file
                                                for(int b=0; b<numFiles; b++)
                                                {
                                                            String s = prefix + "." + o + p + q;
                                                            File f = new File(s);
                                                            f.deleteOnExit();

                                                            File fOriginal = new File(fName + "."
+ o + p + q);

                                                            f.deleteOnExit();
                                                            f.renameTo(fOriginal);

                                                            q++;
                                                            if(q>=10)
                                                            {
                                                                        q=0;
                                                                        p++;
                                                                        if(p>=10)
                                                                        {
                                                                                    p=0;
                                                                                    o++;
                                                                        }
                                                            }
                                                }

                                                String command[] = {"-j", fName};
                                                //Rebuild the original file
                                                HJSplit.main(command);
                                    }

                                    else if(param[0].equals("-z"))
                        {
                                    System.out.println("This function has not yet
been programmed");
                                    }

                        //print help if no parameter-----------------------------------------------
                                    else
                                    {
                                                printHelp();
                                                System.exit(1);
                                    }
                        }
                        catch(Exception e)
                        {
```

```
                printHelp();
                System.exit(1);
            }
        }

        private void printHelp()
        {
            System.out.println("usage:  java DataOwnerDriver [command]");
            System.out.println();
            System.out.println("commands:");
            System.out.println("-i                    initialize DataOwner");
            System.out.println("-s [filename] [size]    split file into packets of size kB");
            System.out.println("                        (if no file size is specified, the");
            System.out.println("                        default is 1440kB)");
            System.out.println("-d                    distribute the split files");
            System.out.println("-r                    retrieve and recover original file");
            System.out.println("-z                    delete the distributed packets and
terminate");
            System.out.println("                        DataOwner");
            System.out.println("-?                    help (this text)");
        }

        //Method to copy (duplicate) a file
        private void copyFile(File f, File newFile)
        {
            try
            {
                FileReader in = new FileReader(f);
                FileWriter out = new FileWriter(newFile);
                int c;

                while ((c = in.read()) != -1)
                {
                    out.write(c);
                }

                out.flush();
                in.close();
                out.close();
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }

        //Method to read an object from a file
        private Object readFile(String fName) throws Exception
        {
            //attempt to read a file
            try
            {
                System.out.println("Reading " + fName);
                FileInputStream fis = new FileInputStream(fName);
                ObjectInputStream ois = new ObjectInputStream(fis);
                Object o = ois.readObject();

                ois.close();
                fis.close();

                return o;
            }
            catch(Exception e)
            {
                System.out.println(fName + " could not be read.");
                return null;
            }
        }

        //Method to write an object to a file
        private void writeFile(Object o, String fName)
        {
try
{
                FileOutputStream fos = new FileOutputStream(fName);
                ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                oos2.writeObject(o);
                oos2.flush();

                System.out.println(fName + " written");

                oos2.close();
                fos.close();
            }
            catch(Exception e)
            {
                System.out.println("Cannot write to file: " + fName);
            }
        }

        //Read the info on this DataOwner
        private void readData() throws Exception
        {
            //attempt to read data from DataOwner.dat
            try
            {
```

```
                System.out.println("Reading " + FILE);
                FileInputStream fis = new FileInputStream(FILE);
                ObjectInputStream ois = new ObjectInputStream(fis);
                data = (DataOwnerData) ois.readObject();

                ois.close();
                fis.close();
        }
        catch(Exception e)
        {
                data = null;
        }
}

//Initialize this DataOwner through ServiceProvider
private void setupDataOwner() throws Exception
{
ObjectOutputStream oos = null;
ObjectInputStream ois = null;
Socket socket = null;

Commands command = new Commands(-1, -1, 10);

try
{
// open a socket connection
socket = new Socket(ADDRS, SPPORT);
socket.setSoTimeout(TIMEOUT);

// open I/O streams for objects
oos = new ObjectOutputStream(socket.getOutputStream());
ois = new ObjectInputStream(socket.getInputStream());

        //send the command
oos.writeObject(command);
oos.flush();

//receive the unique id
data = (DataOwnerData) ois.readObject();

oos.close();
ois.close();

//write the data to DataOwner.dat
synchronized(data)
{
        FileOutputStream fos = new FileOutputStream(FILE);
                        ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                oos2.writeObject(data);
                oos2.flush();

                System.out.println(FILE + " created");

                oos2.close();
                fos.close();
        }

        System.out.println("DataOwner setup complete");
        System.out.println("ID: " + data.getID());
    }
catch(Exception e)
{
System.out.println(e.getMessage());
}
}
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataHost - DHCommandListener.java

Listens for commands sent by CommandSener class
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class DHCommandListener extends Thread
{
	private final int MYPORT = 3003;
	private ServerSocket dhServer;

	public DHCommandListener(){}

	public void run()
	{
	try
	{
				dhServer = new ServerSocket(MYPORT);
		System.out.println("CommandListner listening on port " + MYPORT);
		}
		catch(Exception e){}

	while(true)
	{
		try
		{
		System.out.println("Waiting for incoming commands...");
		Socket incoming = dhServer.accept();
		System.out.println("Accepted a connection from " +
incoming.getInetAddress());
		DHConnect c = new DHConnect(incoming);
		}
		catch(Exception e) {}
	}
	}
}
```



```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataHost - DHConnect.java

Class that processes incoming commands
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

class DHConnect extends Thread
{
	private Socket incoming = null;
	private ObjectInputStream ois = null;
	private ObjectOutputStream oos = null;
	private Commands command = null;
	private final String FILE = "DataHost.dat";
//	private final String ADDRS = "127.0.0.1";
//	private final String ADDRS = "136.167.212.93";
	private final String ADDRS = "136.167.117.84"; //IP Address of
ServiceProvider
	private final long SVID = 761985762;
	private final double SVKEY = 856019204;
	private final int MNPORT = 3002;
	private final int TIMEOUT = 60 * 1000;
	private Vector v = null;

	public DHConnect() {}

	public DHConnect(Socket incomingSocket)
	{
	try
	{
		incoming = incomingSocket;
		incoming.setSoTimeout(TIMEOUT);

		ois = new ObjectInputStream(incoming.getInputStream());
		oos = new ObjectOutputStream(incoming.getOutputStream());
	}
	catch(Exception e1)
	{
	try
```

```
	{
	incoming.close();
	}

	catch(Exception e)
	{
		System.out.println(e.getMessage());
	}

	return;
	}

	this.start();
	}


	public void run()
	{
	try
	{
	int flag = 0;
	command = (Commands) ois.readObject();

			int c = command.getCommand();
			System.out.println("Incoming command: " + c);

	//Commands from SeviceProvider
	if(command.getID() == SVID && command.getKey() == SVKEY)
	{
				//Check if I'm alive, and set myself inactive
				if(c==3)
				{
					DataHostData dhd = (DataHostData)
readFile(FILE);

					if(dhd.isActive() && dhd.getAvailable() >=
command.getLong())
					{
						dhd.setInactive();
						flag = updateFile(FILE, dhd);
					}
					else flag = -1;
				}

			}

			//Get ready to receive a package from the Manager
			else if(c==24)
```

```
				{
					FileReceiver fr = new
FileReceiver(incoming.getInetAddress(), command.getNum2(), command.getString(),
command.getLong());
					fr.start();
					flag = 1;
				}

				//Confirm the receipt of a package
				else if(c==25)
				{
					String f = command.getString();
					FileTracker ft = new FileTracker(f,
command.getID());

					DataHostData dhd = (DataHostData)
readFile(FILE);
					dhd.addTracker(ft);
					dhd.incUsed(command.getNum());
					dhd.setActive();

					flag = updateFile(FILE, dhd);

					//Update my information at ServiceProvider
					Commands cm = new Commands(dhd.getID(),
dhd.getKey(), 32, dhd);
					CommandSender cs = new
CommandSender(ADDRS, cm);
					cs.sendCommand();

				}

				//Package not received, make myself active again
				else if(c == 26)
				{
					DataHostData dhd = (DataHostData)
readFile(FILE);
					dhd.setActive();

					flag = updateFile(FILE, dhd);

					//Update my information at ServiceProvider
					Commands cm = new Commands(dhd.getID(),
dhd.getKey(), 32, dhd);
					CommandSender cs = new
CommandSender(ADDRS, cm);
					cs.sendCommand();
				}
```

```
                        //Package retrieval request
                        else if(c == 27)
                        {
                                File f = new File(command.getString());
                                String mnAdd =
(""+incoming.getInetAddress()).substring(1);

                                int tPort = 3004 + (int) (Math.random() * 1000.0);

                                //Tell Manager to get ready to receive a package
                                Commands c1 = new Commands(0, 0, 33,
command.getString(), f.length(), tPort);
                                CommandSender cs1 = new CommandSender(mnAdd,
MNPORT, c1);
                                if(cs1.sendCommand())
                                {
                                        //Send a package to the Manager
                                        FileSender fs = new FileSender(f, mnAdd, tPort);
                                        if(fs.sendFile())
                                                flag = 1;
                                        else flag = -1;
                                }
                                else flag = -1;
                        }


                        //send ACK or NAK
                        if(flag == 1)
                        {
                                oos.writeObject(command);
                                oos.flush();
                        }
                        else
                        {
                                oos.writeObject(new Commands(0,0,0));
                                oos.flush();
                        }
                }
                catch(Exception e){}
        }

        //Method to read an object from a file
        private Object readFile(String fName) throws Exception
        {
                //attempt to read a file
                try
                {
                        System.out.println("Reading " + fName);
                        FileInputStream fis = new FileInputStream(fName);
                        ObjectInputStream ois = new ObjectInputStream(fis);
                        Object o = ois.readObject();

                        ois.close();
                        fis.close();

                        return o;
                }
                catch(Exception e)
                {
                        System.out.println(fName + " could not be read.");
                        return null;
                }
        }

        //Method to write an object to a file
        private void writeFile(Object o, String fName)
        {
try
{
                        FileOutputStream fos = new FileOutputStream(fName);
                        ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                        oos2.writeObject(o);
                        oos2.flush();

                        System.out.println(fName + " written");

                        oos2.close();
                        fos.close();
                }
                catch(Exception e)
                {
                        System.out.println("Cannot write to file: " + fName);
                }
        }

        //Method to update the object contents of a file
        private int updateFile(String fOld, Object nContent)
        {
                synchronized(nContent)
                {
                        try
{
                                File f = new File(fOld);
                                File tmp = new File("temp.tmp");
```

```
                        FileOutputStream fos = new FileOutputStream(tmp);
                        ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                        oos2.writeObject(nContent);
                        oos2.flush();

                        oos2.close();
                        fos.close();

                        f.delete();
                        tmp.renameTo(f);

                        System.out.println(fOld + " updated");

                        return 1;
                }
                catch(Exception e)
                {
                        System.out.println(e);
                        System.out.println(fOld + " could not be updated");
                        return -1;
                }
        }
    }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Manager - DHInfo.java

Object that stores IP address of a DataHost
*****************************************************************/

import java.io.*;

public class DHInfo implements Serializable
{
        private long id = -1;
        private String address = "";

        public DHInfo(long i, String ad)
        {
                id = i;
                address = ad;
        }

        public long getID()
        {
                return id;
        }

        public String getAddress()
        {
                return address;
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Manager - DHLocator.java

Class that keeps track of DataHosts
*****************************************************************/

import java.util.*;
import java.io.*;

public class DHLocator implements Serializable
{
	Vector dh = null;

	public DHLocator()
	{
		dh = new Vector();
	}

	public void update(DHInfo dhi)
	{
		add(dhi, 0, dh.size()-1);
	}

	public DHInfo getDHI(long id)
	{
		DHInfo dhi;

		synchronized(dh)
		{
			dhi = findDHI(id, 0, dh.size()-1);
		}

		return dhi;
	}

	public String getAddress(long id)
	{
		DHInfo dhi = getDHI(id);
		return dhi.getAddress();
	}

	private DHInfo findDHI(long id, int i, int j)
	{
		long indexID;

		if(j < 0)
		{
			return new DHInfo(-1, "");
		}

		else if(i == j)
		{
			DHInfo dhi = (DHInfo)(dh.get(i));
			if(dhi.getID() == id)
				return dhi;
			else return new DHInfo(-1, "");
		}

		indexID = ((DHInfo)(dh.get(1+(i+j)/2))).getID();

		if(indexID > id)
			return findDHI(id, i, (i+j)/2);

		else
			return findDHI(id, 1+(i+j)/2, j);
	}

	private void add(DHInfo dhi, int i, int j)
	{
		long indexID;

		if(j < 0)
		{
			dh.add(i, dhi);
			return;
		}

		else if(i == j)
		{
			if(((DHInfo)(dh.get(i))).getID() == dhi.getID())
			{
				dh.remove(i);
				dh.add(i, dhi);
			}
			else if(((DHInfo)(dh.get(i))).getID() < dhi.getID())
				dh.add(i+1, dhi);
			else
				dh.add(i, dhi);

			return;
		}
```

```
                indexID = ((DHInfo)(dh.get(1+(i+j)/2))).getID();

                if(indexID < dhi.getID())
                        add(dhi, i, (i+j)/2);

                else
                        add(dhi, 1+(i+j)/2, j);

                return;
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Manager - DHTracker.java

Object that keeps track of which DataHosts are responsible for
a DataOwner
*****************************************************************/

import java.util.*;
import java.io.*;

public class DHTracker implements Serializable
{
        private Vector children1 = null;
        private Vector children2 = null;

        public DHTracker()
        {
                children1 = new Vector();
                children2 = new Vector();
        }

        public void setChildren1(Vector v)
        {
                children1 = v;
        }

        public void setChildren2(Vector v)
        {
                children2 = v;
        }

        public Vector getChildren1()
        {
                if(children1 == null)
                        return new Vector();
                return children1;
        }

        public Vector getChildren2()
        {
                if(children2 == null)
                        return new Vector();
                return children2;
```

```
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataOwner - DOCommandListener.java

Listens for commands sent by CommandSender class
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class DOCommandListener extends Thread
{
        private final int MYPORT = 3001;
        private ServerSocket doServer;

        public DOCommandListener(){}

        public void run()
        {
        try
        {
                        doServer = new ServerSocket(MYPORT);
                System.out.println("CommandListner listening on port " +
MYPORT);
                }
                catch(Exception e){}

        while(true)
        {
                try
                {
                System.out.println("Waiting for incoming commands...");
                Socket incoming = doServer.accept();
                System.out.println("Accepted a connection from " +
incoming.getInetAddress());
                DOConnect c = new DOConnect(incoming);
                }
                catch(Exception e) {}
        }
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataOwner - DOConnect.java

Class that processes incoming commands
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

class DOConnect extends Thread
{
	private Socket incoming = null;
	private ObjectInputStream ois = null;
	private ObjectOutputStream oos = null;
	private Commands command = null;
	private final long SVID = 761985762;
	private final double SVKEY = 856019204;
	private final int TIMEOUT = 60 * 1000;
	private Vector v = null;

	public DOConnect() {}

	public DOConnect(Socket incomingSocket)
	{
	try
	{
		incoming = incomingSocket;
		incoming.setSoTimeout(TIMEOUT);

		ois = new ObjectInputStream(incoming.getInputStream());
		oos = new ObjectOutputStream(incoming.getOutputStream());
	}
	catch(Exception e1)
	{
	try
	{
	incoming.close();
	}

	catch(Exception e)
	{
		System.out.println(e.getMessage());
	}

	return;
	}

	this.start();
	}


	public void run()
	{
	try
	{
	int flag = 0;
	command = (Commands) ois.readObject();

			int c = command.getCommand();
			System.out.println("Incoming command: " + c);

	if(command.getID() == SVID && command.getKey() == SVKEY)
	{
				if(c==1)
				{
					flag = 1;
				}

				if(c==2)
				{
					v = (Vector) command.getObject();
					command.clearObject();
					writeFile(v, "reqman.dat");
					flag = 1;
				}
			}

			else if(c==28)
			{
				File f = new File(command.getString());
				if(f.length() >= command.getLong())
					flag = -1;
				else
				{
					FileReceiver fr = new
FileReceiver(incoming.getInetAddress(), command.getNum2(), command.getString(),
command.getLong());
					fr.start();
					flag = 1;
```

```
                        }
                    }

                    //send ACK or NAK
                    if(flag == 1)
                    {
                        oos.writeObject(command);
                        oos.flush();
                    }
                    else
                    {
                        oos.writeObject(new Commands(0,0,0));
                        oos.flush();
                    }
                }
                catch(Exception e){}
            }

            private void writeFile(Object o, String fName)
            {
        try
        {
                    FileOutputStream fos = new FileOutputStream(fName);
                    ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                    oos2.writeObject(o);
                    oos2.flush();

                    System.out.println(fName + " written");

                    oos2.close();
                    fos.close();
                }
                catch(Exception e)
                {
                    System.out.println("Cannot write to file: " + fName);
                }
            }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Manager - FileDistributer.java

Class that distributes packages to DataHosts
*****************************************************************/

import java.io.*;
import java.lang.*;
import java.util.*;

public class FileDistributer extends Thread
{
        private final int DHPORT = 3003;
        private final int MINHOST = 1;
        private final String FILE = "Manager.dat";
        private final String DHINFO = "DHInfo.dat";
//      private final String ADDRS = "127.0.0.1";
//      private final String ADDRS = "136.167.212.93";
        private final String ADDRS = "136.167.117.84"; //IP Address of
ServiceProvider
        private final int TRIALS = 10;

        private int numFiles = -1;
        private int numHosts = -1;
        private String[] f = new String[2];
        private File[] file = new File[2];
        private long[] leng = new long[2];
        private Vector[] children = new Vector[2];
        private long parent = -1;
        private DHTracker dht = new DHTracker();
        private DHLocator dhl = null;

        public FileDistributer(long p, String ff, int x)
        {
            parent = p;

            f[0] = ff;
            f[1] = null;

            numFiles = 1;
            file[0] = new File(f[0]);
            file[1] = null;
            leng[0] = file[0].length();
```

```
		leng[1] = -1;
		numHosts = x;

		children[0] = new Vector();

		this.start();
}

public FileDistributer(long p, String ff1, String ff2)
{
		parent = p;

		f[0] = ff1;
		f[1] = ff2;
		numHosts = MINHOST;

		if(f[1] == null)
		{
			numFiles = 1;
			file[0] = new File(f[0]);
			leng[0] = file[0].length();
			children[0] = new Vector();
		}
		else
		{
			numFiles = 2;
			file[0] = new File(f[0]);
			file[1] = new File(f[1]);
			leng[0] = file[0].length();
			leng[1] = file[1].length();
			children[0] = new Vector();
			children[1] = new Vector();
		}

		this.start();
}

public void run()
{
		try
		{
			System.out.println("Reading " + DHINFO);
			FileInputStream fis = new FileInputStream(DHINFO);
			ObjectInputStream ois = new ObjectInputStream(fis);
			dhl = (DHLocator) ois.readObject();

			ois.close();
			fis.close();
		}
		catch(Exception e)
		{
			dhl = new DHLocator();
		}

		int numComplete;
		int outstanding = 0;
		int attempts = 0;
		boolean exit = false;

		try
		{
			ManagerData mn = (ManagerData) readFile(FILE);


			for(int nf=0; nf<numFiles; nf++)
			{
				numComplete = 0;

				while(numComplete < numHosts && !exit)
				{
					outstanding = numHosts -
numComplete;

					//Ask ServiceProvider for DataHosts
					Commands c = new
Commands(mn.getID(), mn.getKey(), 23, outstanding, leng[nf]);
					CommandSender cs = new
CommandSender(ADDRS, c);
					if(!cs.sendCommand())
					{
						System.out.println("The
DataHosts could not be grabbed");
						attempts++;

	if(attempts>=TRIALS*numFiles)
							exit = true;
					}
					else //if a number of data hosts are
grabbed
					{
						Vector datahosts = (Vector)
readFile("reqdh.dat");
						if(datahosts.size() == 0)
						{

	System.out.println("The DataHosts could not be grabbed");
```

```
                                                attempts++;
        if(attempts>=TRIALS*numFiles)
                                                        exit = true;
                                                }
                                                else for(int a = 0; a<outstanding; a++)
                                                {
                                                        try
                                                        {
                                                                DataHostData dhd
= (DataHostData) datahosts.get(a);
                                                                int tPort = 3004 +
(int) (Math.random() * 1000.0);

                                                                //Tell DataHost to
get ready to receive a package
                                                                Commands c1 =
new Commands(mn.getID(), 0, 24, f[nf], leng[nf], tPort);
                                                                CommandSender
cs1 = new CommandSender(dhd.getAddress(), DHPORT, c1);
        if(cs1.sendCommand())
                                                                {

        FileSender fs = new FileSender(file[nf], dhd.getAddress(), tPort);
        if(fs.sendFile())
                                                                        {

        //Package delivered, make the DataHost active again
        Commands c2 = new Commands(mn.getID(), 0, 25, f[nf], (int)leng[nf]);
        CommandSender cs2 = new CommandSender(dhd.getAddress(), DHPORT, c2);
        if(cs2.sendCommand())
        {
        children[nf].add(new Long(dhd.getID()));
        dhl.update(new DHInfo(dhd.getID(), dhd.getAddress()));
        writeFile(dhl, DHINFO);
        numComplete++;
        }
                }
        else //if the file not sent
        {
        //Failed to send a package, but make the DataHost active again
        Commands c3 = new Commands(dhd.getID(), 0, 26);
        CommandSender cs3 = new CommandSender(dhd.getAddress(), DHPORT,
c3);
        cs3.sendCommand();

        attempts++;
        if(attempts<TRIALS * numFiles)
                System.out.println("Distribution of files failed, trying
alternatives...");
        else
        {
                System.out.println("Distribution of files failed");
                exit = true;
        }
        }
                                                                }//end
if(cs1.sendCommand())
                                                        }//end try
                                                        catch(Exception e)
                                                        {
        attempts++;
        if(attempts<TRIALS * numFiles)
        System.out.println("Distribution of files failed, trying alternatives...");
                                                                else
                                                                {
```

```
                System.out.println("Distribution of files failed");
                                                                                exit =
true;
                                                                                }
                                                                        }
                                                                }//end for
                                                        }//end else
                                                }//end while
                                                if(numComplete >= numHosts)
                                                        System.out.println(f[nf] + " distributed
successfully");
                                                else
                                                        System.out.println(f[nf] + " not distributed
successfully");
                                }//end for
                                File reqdh = new File("reqdh.dat");
                                reqdh.delete();

                                dht.setChildren1(children[0]);
                                dht.setChildren2(children[1]);

                                writeFile(dht, parent + ".dat");
                        }//end try
                        catch(Exception e){}
                }

                private Object readFile(String fName) throws Exception
                {
                        //attempt to read a file
                        try
                        {
                                System.out.println("Reading " + fName);
                                FileInputStream fis = new FileInputStream(fName);
                                ObjectInputStream ois = new ObjectInputStream(fis);
                                Object o = ois.readObject();

                                ois.close();
                                fis.close();

                                return o;
                        }
                        catch(Exception e)
                        {
                                System.out.println(fName + " could not be read.");
                                return null;
                        }
                }

                private void writeFile(Object o, String fName)
                {
        try
        {
                                FileOutputStream fos = new FileOutputStream(fName);
                                ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                                oos2.writeObject(o);
                                oos2.flush();

                                System.out.println(fName + " written");

                                oos2.close();
                                fos.close();
                        }
                        catch(Exception e)
                        {
                                System.out.println("Cannot write to file: " + fName);
                        }
                }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataOwner, Manager, DataHost - FileReceiver.java

Class that receives a file sent by FileSender class
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class FileReceiver extends Thread
{
	private int port;
	private ServerSocket frServer;
	private InetAddress client = null;
	private String fName = "";
	private long fLeng = -1;
	private final int TIMEOUT = 60 * 1000;

	public FileReceiver(InetAddress cl, int p, String name, long leng)
	{
	try
	{
			client = cl;
			port = p;
			fName = name;
			fLeng = leng;

		frServer = new ServerSocket(port);
		frServer.setSoTimeout(TIMEOUT);
		}
		catch(Exception e)
		{
			System.out.println(e.getMessage());
		}
	}

	public void run()
	{
	try
	{
	boolean connected = false;
		Socket incoming = null;

			System.out.println("Server listening on port " + port + " for file
transfer...");

		while(!connected)
		{
			incoming = frServer.accept();
			incoming.setSoTimeout(TIMEOUT);

			InetAddress iAD = incoming.getInetAddress();

			if(client.equals(iAD))
			{
					connected = true;
					System.out.println("Accepted a
connection from " + iAD);
				}
				else
				{
					System.out.println("Denying a
connection from " + iAD);
					incoming.close();
				}
			}

			try
			{
				OutputStreamWriter osw = new
OutputStreamWriter(incoming.getOutputStream());

				File file = new File(fName);
				FileWriter out = new FileWriter(file);
				InputStreamReader isr = new
InputStreamReader(incoming.getInputStream());
				BufferedReader br = new BufferedReader(isr);
				long length = 0;
				int c;

				while((c=br.read()) != -1)
				{
					out.write(c);
					length ++;
				}

				out.flush();
				out.close();
```

```
                                        int success;

                                        if(length == fLeng)
                                        {
                                                System.out.println("File received successfully: "
+ fName);
                                                System.out.println("File size: " + fLeng);
                                                success = 1;
                                        }
                                        else
                                        {
                                                System.out.println(file.length() + "File corrupt,
removing: " + fName);
                                                success = 0;
                                                file.delete();
                                        }

                                        osw.write(success);
                                        osw.flush();
                                        osw.close();
                        }
                        catch(Exception e1)
                        {
                                try
                                {
                        incoming.close();
                        }

                        catch(Exception e)
                        {
                                System.out.println(e.getMessage());
                        }
                        }
                }
                catch(Exception e){}
                }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

DataOwner, Manager, DataHost - FileSender.java

Class that sends a file through a specified port
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class FileSender
{
        private int port;
        private Socket fsSocket;
        private File file = null;
        private String addrs = "";
        private final int TIMEOUT = 60 * 1000;

        public FileSender(File f, String a, int p)
        {
        try
        {
                        file = f;
                        addrs = a;
                        port = p;

                fsSocket = new Socket(addrs, port);
                fsSocket.setSoTimeout(TIMEOUT);
                }
                catch(Exception e)
                {
                        System.out.println(e.getMessage());
                }
        }

        public boolean sendFile()
        {
                try
                {
                        System.out.println("Sending " + file.getName() + " to " +
addrs);
                        FileReader in = new FileReader(file);
```

```
                                OutputStreamWriter osw = new
OutputStreamWriter(fsSocket.getOutputStream());
                                int c;

                                while ((c = in.read()) != -1)
                                {
                                        osw.write(c);
                                }

                                osw.flush();
                                fsSocket.shutdownOutput();
                                in.close();

                                InputStreamReader isr = new
InputStreamReader(fsSocket.getInputStream());

                                int success = (int) isr.read();

                                fsSocket.close();

                                if(success == 1)
                                {
                                        System.out.println("File " + file.getName() + " sent
successfully to " + addrs);
                                        return true;
                                }
                                else
                                {
                                        System.out.println("File " + file.getName() + " not sent to "
+ addrs);
                                        return false;
                                }
                        }
                        catch(Exception e)
                        {
                                System.out.println("File " + file.getName() + " not sent to " + addrs);
                                System.out.println(e);
                                return false;
                        }
                }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Service Provider, DataOwner, Manager, DataHost - FileTracker.java

Keeps track of to whom the files belong
*****************************************************************/

import java.io.*;
import java.util.*;

public class FileTracker implements Serializable
{
        private Vector files = new Vector();
        private long parentID = -1;
        private Date created = null;

        public FileTracker(String f1, String f2, long id)
        {
                files.add(f1);
                if(f2 != null)
                        files.add(f2);
                parentID = id;
                created = new Date();
        }

        public FileTracker(String f, long id)
        {
                files.add(f);
                parentID = id;
                created = new Date();
        }

        public long getParent()
        {
                return parentID;
        }

        public Vector getFiles()
        {
                return files;
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider - ListConsole.java

Class that manipulates the lists owned by ServiceProvider
*****************************************************************/

import java.io.*;
import java.util.*;

public class ListConsole
{
	private final String MANAGERS = "ManagerList.dat";
	private final String DATAHOSTS = "DataHostList.dat";
	private final String DATAOWNERS = "DataOwnerList.dat";

	private Vector mnList;
	private Vector dhList;
	private Vector doList;

	public ListConsole() throws Exception
	{
		//attempt to read data from ManagerList.dat
		try
		{
			System.out.println("Reading " + MANAGERS);
			FileInputStream fis = new FileInputStream(MANAGERS);
			ObjectInputStream ois = new ObjectInputStream(fis);
			mnList = (Vector) ois.readObject();

			ois.close();
			fis.close();
		}
		catch(Exception e)
		{
			System.out.println("Unable to read " + MANAGERS + ", updating
Manager List");
			mnList = new Vector();

	writeMNList();
		}

		//attempt to read data from DataHostList.dat
		try
		{
			System.out.println("Reading " + DATAHOSTS);
			FileInputStream fis = new
FileInputStream(DATAHOSTS);
			ObjectInputStream ois = new ObjectInputStream(fis);
			dhList = (Vector) ois.readObject();

			ois.close();
			fis.close();
		}
		catch(Exception e)
		{
			System.out.println("Unable to read " + DATAHOSTS + ",
updating DataHost List");
			dhList = new Vector();

	writeDHList();
		}

		//attempt to read data from DataOwnerList.dat
		try
		{
			System.out.println("Reading " + DATAOWNERS);
			FileInputStream fis = new
FileInputStream(DATAOWNERS);
			ObjectInputStream ois = new ObjectInputStream(fis);
			doList = (Vector) ois.readObject();

			ois.close();
			fis.close();
		}
		catch(Exception e)
		{
			System.out.println("Unable to read " + DATAOWNERS +
", updating DataOwner List");
			doList = new Vector();

	writeDOList();
		}
	}

	private void writeMNList() throws Exception
	{
		try
		{
	FileOutputStream fos = new FileOutputStream(MANAGERS);
			ObjectOutputStream oos = new ObjectOutputStream(fos);
			oos.writeObject(mnList);
```

```
                    oos.flush();

                    System.out.println(MANAGERS + " updated");

                    oos.close();
                    fos.close();
                }
                catch(Exception e)
                {
                    System.out.println("Failed to update " + MANAGERS);
                }

                //print the resulting ManagerList
                System.out.println("Currently there are " + mnList.size() + " Managers:");
                for(int i=0; i<mnList.size(); i++)
                {
                    ManagerData mnd = (ManagerData) mnList.get(i);
                    System.out.println("ManagerID " + mnd.getID() + " has " +
mnd.getNumOwner() + "/" + mnd.getCapacity() + " owners");
                }
        }

        private void writeDHList() throws Exception
        {
                try
                {
        FileOutputStream fos = new FileOutputStream(DATAHOSTS);
                    ObjectOutputStream oos = new ObjectOutputStream(fos);
                    oos.writeObject(dhList);
                    oos.flush();

                    System.out.println(DATAHOSTS + " updated");

                    oos.close();
                    fos.close();
                }
                catch(Exception e)
                {
                    System.out.println("Failed to update " + DATAHOSTS);
                }

                //print the resulting DataHostList
                System.out.println("Currently there are " + dhList.size() + " DataHosts:");
                for(int i=0; i<dhList.size(); i++)
                {
                    DataHostData dhd = (DataHostData) dhList.get(i);
                    System.out.println("DataHostID " + dhd.getID() + " has " +
dhd.getAvailable() + "/" + dhd.getCapacity() + " bytes available");
                }
        }

        private void writeDOList() throws Exception
        {
                try
                {
        FileOutputStream fos = new FileOutputStream(DATAOWNERS);
                    ObjectOutputStream oos = new ObjectOutputStream(fos);
                    oos.writeObject(doList);
                    oos.flush();

                    System.out.println(DATAOWNERS + " updated");

                    oos.close();
                    fos.close();
                }
                catch(Exception e)
                {
                    System.out.println("Failed to update " + DATAOWNERS);
                }

                //print the resulting DataOwnerList
                System.out.println("Currently there are " + doList.size() + "
DataOwners:");
                for(int i=0; i<doList.size(); i++)
                {
                    DataOwnerData dod = (DataOwnerData) doList.get(i);
                    System.out.println("DataOwnerID " + dod.getID());
                }
        }

        public void addManager(ManagerData mn) throws Exception
        {
                synchronized(mnList)
                {
                    //insert the Manager at the appropriate location in the List
                    try
                    {
                        removeMN(mn.getID());
                    }
                    catch(Exception e)
                    {
                        System.out.println("Adding Manager to the
list...");
                    }

                    insertManager(mn, 0, mnList.size()-1);
```

```
                //write the List to a file
                writeMNList();
            }
        }

        private void insertManager(ManagerData mn, int i, int j)
        {
            int indexAvailable;

            if(j < 0)
            {
                mnList.add(i, mn);
                return;
            }

            else if(i == j)
            {
                if(((ManagerData)(mnList.get(i))).getAvailable() >
mn.getAvailable())
                    mnList.add(i+1, mn);
                else
                    mnList.add(i, mn);

                return;
            }

            indexAvailable = ((ManagerData)(mnList.get(1+(i+j)/2))).getAvailable();

            if(indexAvailable < mn.getAvailable())
                insertManager(mn, i, (i+j)/2);

            else
                insertManager(mn, 1+(i+j)/2, j);

            return;
        }

        public void addDataHost(DataHostData dhd) throws Exception
        {
            synchronized(dhList)
            {
                //insert the DataHost at the appropriate location in the List
                try
                {
                    removeDH(dhd.getID());
                }
                catch(Exception e)
                {
                    System.out.println("Adding DataHost to the
list...");
                }

                insertDataHost(dhd, 0, dhList.size()-1);

                //write the List to a file
                writeDHList();
            }
        }

        private void insertDataHost(DataHostData dh, int i, int j)
        {
            int indexCapacity;

            if(j < 0)
            {
                dhList.add(i, dh);
                return;
            }

            else if(i == j)
            {
                if(((DataHostData)(dhList.get(i))).getCapacity() >
dh.getCapacity())
                    dhList.add(i+1, dh);
                else
                    dhList.add(i, dh);

                return;
            }

            indexCapacity =
((DataHostData)(dhList.get(1+(i+j)/2))).getCapacity();

            if(indexCapacity < dh.getCapacity())
                insertDataHost(dh, i, (i+j)/2);

            else
                insertDataHost(dh, 1+(i+j)/2, j);

            return;
        }

        public void addDataOwner(DataOwnerData dod) throws Exception
        {
            synchronized(doList)
```

```
		{
			//insert the DataOwner at the appropriate location in the List
			insertDataOwner(dod, 0, doList.size()-1);

			//write the List to a file
			writeDOList();
		}
	}

	private void insertDataOwner(DataOwnerData dod, int i, int j)
	{
		long indexID;

		if(j < 0)
		{
			doList.add(i, dod);
			return;
		}

		else if(i == j)
		{
			if(((DataOwnerData)(doList.get(i))).getID() < dod.getID())
				doList.add(i+1, dod);
			else
				doList.add(i, dod);

			return;
		}

		indexID = ((DataOwnerData)(doList.get(1+(i+j)/2))).getID();

		if(indexID < dod.getID())
			insertDataOwner(dod, i, (i+j)/2);

		else
			insertDataOwner(dod, 1+(i+j)/2, j);

		return;
	}

	public DataOwnerData getDO(long id)
	{
		DataOwnerData dod;

		synchronized(doList)
		{
			dod = findDO(id, 0, doList.size()-1);
		}
		return dod;
	}

	public ManagerData getMN(long id)
	{
		ManagerData mn = null;

		synchronized(mnList)
		{
			mn = findMN(id);
		}

		return mn;
	}

	public Vector getMN(int req, int offset)
	{
		try
		{
			int grabbed = 0;
			int index = 0;

			Vector mnGrabbed = new Vector();

			while(grabbed < req)
			{
				ManagerData mn = (ManagerData)
mnList.get((index + offset)%mnList.size());
				mnGrabbed.add(mn);
				grabbed++;
				index++;
			}

			return mnGrabbed;
		}
		catch(Exception e)
		{
			System.out.println("Requested number of managers not
grabbed.");
			return new Vector();
		}
	}

	public Vector getDH(int req, int offset)
	{
		try
		{
```

```
                int grabbed = 0;
                int index = 0;

                Vector dhGrabbed = new Vector();

                while(grabbed < req)
                {
                        DataHostData dhd = (DataHostData) dhList.get((index +
offset)%dhList.size());
                        dhGrabbed.add(dhd);
                        grabbed++;
                        index++;
                }

                return dhGrabbed;
        }
        catch(Exception e)
        {
                System.out.println("Requested number of data hosts not grabbed.");
                return new Vector();
        }
    }

    //verification of components
    public boolean checkDOIdentity(long id, double key)
    {
        DataOwnerData dod;

        synchronized(doList)
        {
                dod = findDO(id, 0, doList.size()-1);
        }
        if(dod.getKey() == key)
                return true;
        else
        {
                System.out.println("DataOwner not identified");
                return false;
        }
    }

    private DataOwnerData findDO(long id, int i, int j)
    {
        long indexID;

        if(j < 0)
        {
                return new DataOwnerData(-1, -1, "");
        }

        else if(i == j)
        {
                DataOwnerData dod = (DataOwnerData)(doList.get(i));
                if(dod.getID() == id)
                        return dod;
                else return new DataOwnerData(0,0,"");
        }

        indexID = ((DataOwnerData)(doList.get(1+(i+j)/2))).getID();

        if(indexID > id)
                return findDO(id, i, (i+j)/2);

        else
                return findDO(id, 1+(i+j)/2, j);
    }

    public DataOwnerData removeDO(long id)
    {
        DataOwnerData dod;

        synchronized(doList)
        {
                dod = findDO(id, 0, doList.size()-1);
                doList.remove(dod);
        }

        return dod;
    }

    public boolean checkMNIdentity(long id, double key)
    {
        ManagerData mn;

        synchronized(mnList)
        {
                mn = findMN(id);
        }
        if(mn.getKey() == key)
        {
                return true;
        }
        else
        {
                System.out.println("Manager not identified");
                return false;
```

```
		}
}

private ManagerData findMN(long id)
{
		ManagerData mn;
		try
		{
			int index = 0;

			while(true)
			{
				mn = (ManagerData) mnList.get(index);
				if(mn.getID() == id)
					return mn;
				index++;
			}
		}
		catch(Exception e)
		{
			System.out.println(e);
			System.out.println("The specified Manager is not in the list");
			ManagerData nmn = new ManagerData(-1, -1, -1,"");
			return nmn;
		}
}

public ManagerData removeMN(long id)
{
		ManagerData mn = null;

		mn = findMN(id);
		mnList.remove(mn);

		return mn;
}

public boolean checkDHIdentity(long id, double key)
{
		DataHostData dhd;

		synchronized(dhList)
		{
			dhd = findDH(id);
		}
		if(dhd.getKey() == key)
			return true;
		else
		{
			System.out.println("DataHost not identified");
			return false;
		}
	}

	private DataHostData findDH(long id)
	{
		DataHostData dhd;
		try
		{
			int index = 0;

			while(true)
			{
				dhd = (DataHostData) dhList.get(index);
				if(dhd.getID() == id)
					return dhd;
				index++;
			}
		}
		catch(Exception e)
		{
			System.out.println(e);
			System.out.println("The specified DataHost is not in the
list");
			DataHostData ndhd = new DataHostData(-1, -1, -1,"");
			return ndhd;
		}
	}

	public DataHostData removeDH(long id)
	{
		DataHostData dhd = null;

		dhd = findDH(id);
		dhList.remove(dhd);

		return dhd;
	}
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider, DataOwner, Manager, DataHost - ManagerData.java

Object that stores information on a Manager
*****************************************************************/

import java.io.*;
import java.util.*;

public class ManagerData implements Serializable
{
	private long id = 0;
	private double key = 0;
	private int numOwner = 0;
	private int capacity = 0;
	private String addrs = "";
	private Vector fTracker = null;
	private boolean active = true;

	public ManagerData(long i, double k, int n, String s)
	{
		id = i;
		key = k;
		capacity = n;
		addrs = s.substring(1);
		fTracker = new Vector();
	}

	public long getID()
	{
		return id;
	}

	public double getKey()
	{
		return key;
	}

	public int getNumOwner()
	{
		return numOwner;
	}

	public int getCapacity()
	{
		return capacity;
	}

	public int getAvailable()
	{
		return capacity - numOwner;
	}

	public String getAddress()
	{
		return addrs;
	}

	public Vector getFiles(long ID)
	{
		try
		{
			FileTracker ft = null;
			int index = 0;

			while(true)
			{
				ft = (FileTracker) fTracker.get(index);
				if(ft.getParent() == ID)
				{
					return ft.getFiles();
				}

				index++;
			}
		}
		catch(Exception e)
		{
			System.out.println(e);
			return new Vector();
		}
	}

	public boolean isActive()
	{
		return active;
	}

	public void incNumOwner()
	{
		numOwner++;
```

```
	}

	public void decNumOwner()
	{
		numOwner--;
		if(numOwner < 0)
			numOwner = 0;
	}

	public void updateTracker(Vector v)
	{
		fTracker = v;
	}

	public void setAddress(String s)
	{
		addrs = s.substring(1);
	}

	public void addTracker(FileTracker ft)
	{
		fTracker.add(ft);
	}

	public void setActive()
	{
		active = true;
	}

	public void setInactive()
	{
		active = false;
	}
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Manager - ManagerDriver.java

Driver class for a Manager
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class ManagerDriver
{
	private final int CAPACITY = 10;
	private final String FILE = "Manager.dat";
//	private final String ADDRS = "127.0.0.1";
//	private final String ADDRS = "136.167.212.93";
	private final String ADDRS = "136.167.117.84"; //IP Address of
ServiceProvider
	private final int SPPORT = 3000;
	private final int TIMEOUT = 60 * 1000;

	private ManagerData data;

	public ManagerDriver()
	{
	}

	public static void main(String argv[]) throws Exception
	{
		////////////////////////////////////////////////////////////////////////
		System.out.println("Boston College");
		System.out.println("Computer Science Department");
		System.out.println();
		System.out.println("Senior Thesis 2003");
		System.out.println("Distributed Data Storage: Data Backup Over
Networks");
		System.out.println("(c) Hiroyuki Hayano");
		System.out.println("Prof. Elizabeth Borowsky");
		System.out.println();
		System.out.println();
		////////////////////////////////////////////////////////////////////////
```

```
        MNCommandListener cl = new MNCommandListener();
        cl.start();

        ManagerDriver mn = new ManagerDriver();
        mn.startSession();
        }

        private void startSession() throws Exception
        {
        readData();

        if(data==null)
        {
                        System.out.println("Data file not found or corrupt, setting up
Manager...");
                        setupManager();
                }

                //Tell ServiceProvider that I'm online
                data.setActive();
                writeFile(data, FILE);
                Commands c = new Commands(data.getID(), data.getKey(), 22, data);
                CommandSender cs = new CommandSender(ADDRS, c);
                cs.sendCommand();
        }


        private void readData() throws Exception
        {
                //attempt to read data from Manager.dat
                try
                {
                        System.out.println("Reading " + FILE);
                        FileInputStream fis = new FileInputStream(FILE);
                        ObjectInputStream ois = new ObjectInputStream(fis);
                        data = (ManagerData) ois.readObject();

                        ois.close();
                        fis.close();
                }
                catch(Exception e)
                {
                        data = null;
                }
        }

        private void writeFile(Object o, String fName)
        {
try
{
                        FileOutputStream fos = new FileOutputStream(fName);
                        ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                        oos2.writeObject(o);
                        oos2.flush();

                        System.out.println(fName + " written");

                        oos2.close();
                        fos.close();
                }
                catch(Exception e)
                {
                        System.out.println("Cannot write to file: " + fName);
                }
        }

        private void setupManager() throws Exception
        {
        ObjectOutputStream oos = null;
        ObjectInputStream ois = null;
        Socket socket = null;

        Commands command = new Commands(-1, -1, 20, CAPACITY);

        try
        {
        // open a socket connection
        socket = new Socket(ADDRS, SPPORT);
        socket.setSoTimeout(TIMEOUT);

        // open I/O streams for objects
        oos = new ObjectOutputStream(socket.getOutputStream());
        ois = new ObjectInputStream(socket.getInputStream());

                //send the command
        oos.writeObject(command);
        oos.flush();

        //receive the unique id
        data = (ManagerData) ois.readObject();

        oos.close();
        ois.close();

        //write the data to Manager.dat
        synchronized(data)
```

```
        {
                FileOutputStream fos = new FileOutputStream(FILE);
                        ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                        oos2.writeObject(data);
                        oos2.flush();

                        System.out.println(FILE + " created");

                        oos2.close();
                        fos.close();
                }

                System.out.println("Manager setup complete");
                System.out.println("ID: " + data.getID());
                System.out.println("Maximum number of Owners: " + CAPACITY);
        }
        catch(Exception e)
        {
        System.out.println(e.getMessage());
        }
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Manager - MNCommandListener.java

Listens for commands sent by CommandSender
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

public class MNCommandListener extends Thread
{
        private final int MYPORT = 3002;
        private ServerSocket mnServer;

        public MNCommandListener(){}

        public void run()
        {
        try
        {
                        mnServer = new ServerSocket(MYPORT);
                System.out.println("CommandListner listening on port " +
MYPORT);
                }
                catch(Exception e){}

        while(true)
        {
                try
                {
                System.out.println("Waiting for incoming commands...");
                Socket incoming = mnServer.accept();
                System.out.println("Accepted a connection from " +
incoming.getInetAddress());
                MNConnect c = new MNConnect(incoming);
                }
                catch(Exception e) {}
        }
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

Manager - MNConnect.java

Class that processes incoming commands
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

class MNConnect extends Thread
{
	private Socket incoming = null;
	private ObjectInputStream ois = null;
	private ObjectOutputStream oos = null;
	private Commands command = null;
	private final long SVID = 761985762;
	private final double SVKEY = 856019204;
	private final String FILE = "Manager.dat";
	private final String DHINFO = "DHInfo.dat";
//	private final String ADDRS = "127.0.0.1";
//	private final String ADDRS = "136.167.212.93";
	private final String ADDRS = "136.167.117.84"; //IP Address of ServiceProvider
	private final int DOPORT = 3001;
	private final int DHPORT = 3003;
	private final int TIMEOUT = 60 * 1000;

	public MNConnect() {}

	public MNConnect(Socket incomingSocket)
	{
	try
	{
		incoming = incomingSocket;
		incoming.setSoTimeout(TIMEOUT);

		ois = new ObjectInputStream(incoming.getInputStream());
		oos = new ObjectOutputStream(incoming.getOutputStream());
	}
	catch(Exception e1)
	{
	try
	{
	incoming.close();
	}

	catch(Exception e)
	{
		System.out.println(e.getMessage());
	}

	return;
	}

	this.start();
	}


	public void run()
	{
	try
	{
	int flag = 0;
	command = (Commands) ois.readObject();

			int c = command.getCommand();
			System.out.println("Incoming command: " + c);

	if(command.getID() == SVID && command.getKey() == SVKEY)
	{
				if(c==1)
				{
					ManagerData mn = (ManagerData)
readFile(FILE);

					if(mn.isActive() && mn.getAvailable()
> 0)
					{
						mn.setInactive();
						flag = updateFile(FILE, mn);
					}
					else flag = -1;
				}

				if(c==4)
				{
					Vector v = (Vector)
command.getObject();
					command.clearObject();
					writeFile(v, "reqdh.dat");
```

```
                    flag = 1;
                }

                if(c==5)
                {
                    try
                    {
                        ManagerData mn = (ManagerData)
readFile(FILE);
                        DHLocator dhl = (DHLocator)
readFile(DHINFO);


                        String address = command.getString();
                        long doID = command.getLong();

                        DHTracker dht = (DHTracker)
readFile(doID + ".dat");
                        Vector file = mn.getFiles(doID);

                        Vector dhIDf1 = dht.getChildren1();
                        Vector dhIDf2 = dht.getChildren2();

                        while(true)
                        {
                            long dhID =
((Long)dhIDf1.remove(0)).longValue();
                            String dhADD =
dhl.getAddress(dhID);

                            Commands c1 = new
Commands(0, 0, 27, (String)file.get(0));
                            CommandSender cs1 = new
CommandSender(dhADD, DHPORT, c1);

                            if(cs1.sendCommand())
                            {
                                File f = new
File((String)file.get(0));

                                int tPort = 3004 +
(int) (Math.random() * 1000.0);
                                Commands c2 =
new Commands(0, 0, 28, (String)file.get(0), f.length(), tPort);
                                CommandSender
cs2 = new CommandSender(address, DOPORT, c2);

                                {
                                    if(cs2.sendCommand())
                                    int trial = 0;

                                    while(trial < 10)

                                    {

                                    FileSender fs = new FileSender(f, address, tPort);

                                    if(fs.sendFile())

                                        trial = 11;

                                    else

                                        trial++;

                                    }

                                    break;
                                                                        }
                                                                        else
                                    break;
                                                                    }
                                                                }

                                                                if(!dhIDf2.isEmpty())
                                                                {
                                                                    while(true)
                                                                    {
                                                                        long
                                    dhID2 = ((Long)dhIDf2.remove(0)).longValue();
                                                                        String
                                    dhADD2 = dhl.getAddress(dhID2);

                                        Commands c1 = new Commands(0, 0, 27, (String)file.get(1));

                                        CommandSender cs1 = new CommandSender(dhADD2, DHPORT, c1);

                                        if(cs1.sendCommand())
                                                                                {

                                        File f = new File((String)file.get(1));

                                        int tPort = 3004 + (int) (Math.random() * 1000.0);
```

```
Commands c2 = new Commands(0, 0, 28, (String)file.get(1), f.length(), tPort);

CommandSender cs2 = new CommandSender(address, DOPORT, c2);

if(cs2.sendCommand())
                                                                                {

int trial = 0;

while(trial < 10)

{

FileSender fs = new FileSender(f, address, tPort);

if(fs.sendFile())

        trial = 11;

else

        trial++;

}

break;
                                                                                }
else
break;
                                                                                        }
                                                                                }
                                                                        }
                                                                        flag = 1;
                                                                }
                                                                catch(Exception e)
                                                                {
                                                                        System.out.println(e);
                                                                        flag = -1;
                                                                }
                                                        }
                                                }

                                                else if(c == 13)
                                                {
                                                        FileReceiver fr = new
FileReceiver(incoming.getInetAddress(), command.getNum2(), command.getString(),
command.getLong());
                                                        fr.start();
                                                        flag = 1;
                                                }

                                                else if(c == 14)
                                                {
                                                        String f1 = command.getString();
                                                        String f2 = command.getString2();

                                                        FileTracker ft = new FileTracker(f1, f2,
command.getID());

                                                        ManagerData mn = (ManagerData)
readFile(FILE);
                                                        mn.addTracker(ft);
                                                        mn.incNumOwner();
                                                        mn.setActive();

                                                        flag = updateFile(FILE, mn);

                                                        Commands cm = new Commands(mn.getID(),
mn.getKey(), 22, mn);
                                                        CommandSender cs = new
CommandSender(ADDRS, cm);
                                                        cs.sendCommand();

                                                        //now distribute the files to data hosts
                                                        FileDistributer fd = new
FileDistributer(command.getID(), f1, f2);
                                                }

                                                else if(c == 16)
                                                {
                                                        ManagerData mn = (ManagerData)
readFile(FILE);
                                                        mn.setActive();

                                                        flag = updateFile(FILE, mn);

                                                        Commands cm = new Commands(mn.getID(),
mn.getKey(), 22, mn);
                                                        CommandSender cs = new
CommandSender(ADDRS, cm);
                                                        cs.sendCommand();
                                                }

                                                else if(c == 33)
                                                {
```

```
                                        FileReceiver fr = new
FileReceiver(incoming.getInetAddress(), command.getNum2(), command.getString(),
command.getLong());
                                        fr.start();
                                        flag = 1;
                                }

                                //send ACK or NAK
                                if(flag == 1)
                                {
                                        oos.writeObject(command);
                                        oos.flush();
                                }
                                else
                                {
                                        oos.writeObject(new Commands(0,0,0));
                                        oos.flush();
                                }
                        }
                        catch(Exception e){}
                }

                private Object readFile(String fName) throws Exception
                {
                        //attempt to read a file
                        try
                        {
                                System.out.println("Reading " + fName);
                                FileInputStream fis = new FileInputStream(fName);
                                ObjectInputStream ois = new ObjectInputStream(fis);
                                Object o = ois.readObject();

                                ois.close();
                                fis.close();

                                return o;
                        }
                        catch(Exception e)
                        {
                                System.out.println(fName + " could not be read.");
                                return null;
                        }
                }

                private void writeFile(Object o, String fName)
                {
        try
        {
                                FileOutputStream fos = new FileOutputStream(fName);
                                ObjectOutputStream oos2 = new ObjectOutputStream(fos);
                                oos2.writeObject(o);
                                oos2.flush();

                                System.out.println(fName + " written");

                                oos2.close();
                                fos.close();
                        }
                        catch(Exception e)
                        {
                                System.out.println("Cannot write to file: " + fName);
                        }
                }

                private int updateFile(String fOld, Object nContent)
                {
                        synchronized(nContent)
                        {
                                try
                {
                                        File f = new File(fOld);
                                        File tmp = new File("temp.tmp");

                                        FileOutputStream fos = new
FileOutputStream(tmp);
                                        ObjectOutputStream oos2 = new
ObjectOutputStream(fos);
                                        oos2.writeObject(nContent);
                                        oos2.flush();

                                        oos2.close();
                                        fos.close();

                                        f.delete();
                                        tmp.renameTo(f);

                                        System.out.println(fOld + " updated");

                                        return 1;
                                }
                                catch(Exception e)
                                {
                                        System.out.println(e);
                                        System.out.println(fOld + " could not be
updated");
                                        return -1;
```

```
                }
            }
        }
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider - ServiceProviderDriver.java

Driver class for ServiceProvider
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;

public class ServiceProviderDriver
{
        private final int MYPORT = 3000;
        private ServerSocket spServer;

        public ListConsole list = new ListConsole();

        public static void main(String argv[]) throws Exception
        {
                ////////////////////////////////////////////////////////////////////
                System.out.println("Boston College");
                System.out.println("Computer Science Department");
                System.out.println();
                System.out.println("Senior Thesis 2003");
                System.out.println("Distributed Data Storage: Data Backup Over
Networks");
                System.out.println("(c) Hiroyuki Hayano");
                System.out.println("Prof. Elizabeth Borowsky");
                System.out.println();
                System.out.println();
                ////////////////////////////////////////////////////////////////////

                new ServiceProviderDriver();
        }

        public ServiceProviderDriver() throws Exception
        {
        spServer = new ServerSocket(MYPORT);
        System.out.println("Server listening on port " + MYPORT);
        startServer();
        }

        public void startServer()
```

```
	{
	while(true)
	{
		try
		{
		System.out.println("Waiting for connections...");
		Socket incoming = spServer.accept();
		System.out.println("Accepted a connection from " +
incoming.getInetAddress());
		SPConnect c = new SPConnect(incoming, list, new Date());
		}
		catch(Exception e) {}
	}
	}
}
```

```
/*****************************************************************
Senior Thesis 2003
Hiroyuki Hayano
Distributed Data Storage:  Data Backup Over Networks
Prof. Elizabeth Borowsky

ServiceProvider - SPConnect.java

Class that processes incoming commands
*****************************************************************/

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;

class SPConnect extends Thread
{
	private final int DOPORT = 3001;
	private final int MNPORT = 3002;
	private final int DHPORT = 3003;
	private Socket incoming = null;
	private ObjectInputStream ois = null;
	private ObjectOutputStream oos = null;
	private Commands command = null;
	private ListConsole list = null;
	private Date dt;
	private final long SVID = 761985762;
	private final double SVKEY = 856019204;
	private final int TIMEOUT = 60 * 1000;

	public SPConnect() {}

	public SPConnect(Socket incomingSocket, ListConsole l, Date date)
	{
	try
	{
		incoming = incomingSocket;
		incoming.setSoTimeout(TIMEOUT);
		list = l;
		dt = date;

		ois = new ObjectInputStream(incoming.getInputStream());
		oos = new ObjectOutputStream(incoming.getOutputStream());
	}
	catch(Exception e1)
	{
	try
```

```
{
incoming.close();
}

catch(Exception e)
{
	System.out.println(e.getMessage());
}

return;
}

this.start();
}


public void run()
{
try
{
int flag = 0;
command = (Commands) ois.readObject();

		int c = command.getCommand();
		System.out.println("Incoming command: " + c);
if(command.getID() == -1)
{
			setup();
		}

else if(c-c%10 == 10 && list.checkDOIdentity(command.getID(), command.getKey()))
{
			if(c==11)
			{
				flag = doNewFile();
			}

			if(c==12)
			{
				Vector managers = grabManagers();
				Commands c2 = new Commands(SVID,
SVKEY, 2, managers);
				CommandSender cs = new
CommandSender((""+incoming.getInetAddress()).substring(1), DOPORT, c2);
				if(cs.sendCommand())
				{
					flag = 1;
				}
			}

			if(c==15)
			{
				flag = doNewManager();
			}

			if(c==17)
			{
				flag = retrieveFiles();
			}
		}

	else if(c-c%10 == 20 && list.checkMNIdentity(command.getID(),
command.getKey()))
	{
			if(c==22)
			{
				flag = mnConnect();
			}

			if(c==23)
			{
				Vector datahosts = grabDataHosts();
				Commands c3 = new
Commands(SVID, SVKEY, 4, datahosts);
				CommandSender cs3 = new
CommandSender((""+incoming.getInetAddress()).substring(1), MNPORT, c3);
				if(cs3.sendCommand())
				{
					flag = 1;
				}
			}

		}

	else if(c-c%10 == 30 && list.checkDHIdentity(command.getID(),
command.getKey()))
	{
			if(c==32)
			{
				flag = dhConnect();
			}
		}

		//send ACK or NAK
		if(flag == 1)
		{
```

```
                                oos.writeObject(command);
                                oos.flush();
                        }
                        else
                        {
                                oos.writeObject(new Commands(0,0,0));
                                oos.flush();
                        }

        // close streams and connections
        ois.close();
        oos.close();
        incoming.close();
        }
        catch(Exception e) {}
        }

        private void setup() throws Exception
        {
                if(command.getCommand() == 10)
                {
                        System.out.println("Setting up DataOwner...");
                        Date dt = new Date();
                        DataOwnerData dataOwner = new DataOwnerData(dt.getTime(),
Math.random(), "" + incoming.getInetAddress());
                        oos.writeObject(dataOwner);
                        oos.flush();

                        //add dataOwner to DataOwnerList
                        list.addDataOwner(dataOwner);


                        System.out.println("DataOwner set up with ID " + dt.getTime());
                }

                else if(command.getCommand() == 20)
                {
                        System.out.println("Setting up Manager...");
                        ManagerData manager = new ManagerData(dt.getTime(),
Math.random(), command.getNum(), "" + incoming.getInetAddress());
                        oos.writeObject(manager);
                        oos.flush();

                        //add manager to ManagerList
                        list.addManager(manager);


                        System.out.println("Manager set up with ID " + dt.getTime());
                }

                else if(command.getCommand() == 30)
                {
                        System.out.println("Setting up DataHost...");
                        Date dt = new Date();
                        DataHostData dataHost = new DataHostData(dt.getTime(),
Math.random(), command.getNum(), "" + incoming.getInetAddress());
                        oos.writeObject(dataHost);
                        oos.flush();

                        //add dataHost to DataHostList
                        list.addDataHost(dataHost);


                        System.out.println("DataHost set up with ID " +
dt.getTime());
                }

                else
                {
                        System.out.println("Unrecognized command from " +
incoming.getInetAddress());
                }
        }

        //Actions for DataOwner-------------------------------------------------------------------
        private int doNewFile()
        {
                try
                {
                        DataOwnerData dod = list.removeDO(command.getID());
                        dod.setFileName(command.getString());
                        list.addDataOwner(dod);
                        return 1;
                }
                catch(Exception e)
                {
                        System.out.println(e);
                        return -1;
                }
        }

        private int doNewManager()
        {
                try
                {
                        DataOwnerData dod = list.removeDO(command.getID());
```

```
                        dod.addManager(command.getLong());
                        list.addDataOwner(dod);
                        return 1;
                }
                catch(Exception e)
                {
                        System.out.println(e);
                        return -1;
                }
        }

        private int retrieveFiles()
        {
                try
                {
                        Vector managers = null;
                        long dodID = -1;
                        String dodADD = "";
                        DataOwnerData dod = list.getDO(command.getID());

                        managers = dod.getManagers();
                        dodID = dod.getID();
                        dodADD = (""+incoming.getInetAddress()).substring(1);

                        for(int i = 0; i<managers.size(); i++)
                        {
                                long mID = ((Long)(managers.get(i))).longValue();
                                ManagerData mn = list.getMN(mID);
                                Commands c = new Commands(SVID, SVKEY, 5,
dodADD, dodID);
                                CommandSender cs = new
CommandSender(mn.getAddress(), MNPORT, c);
                                cs.sendCommand();
                        }

                        return 1;
                }
                catch(Exception e)
                {
                        System.out.println(e);
                        return -1;
                }
        }

        private Vector grabManagers()
        {
                try
                {
                        int numReq = command.getNum();
                        int numGotten = 0;
                        int offset = 0;
                        Vector managers = new Vector();
                        Vector temp = new Vector();

                        while(numGotten < numReq)
                        {
                                int numIndex = numReq - numGotten;

                                temp = list.getMN(numIndex, offset);

                                for(int i = 0; i<numIndex; i++)
                                {
                                        ManagerData mn = (ManagerData)
temp.get(i);
                                        Commands c = new Commands(SVID,
SVKEY, 1);
                                        CommandSender cs = new
CommandSender(mn.getAddress(), MNPORT, c);
                                        if(cs.sendCommand())
                                        {
                                                managers.add(mn);
                                                numGotten ++;
                                        }
                                }
                                offset += numReq;
                                if(offset > (numReq * 10))
                                        return new Vector();
                        }

                        return managers;
                }
                catch(Exception e)
                {
                        return new Vector();
                }
        }

        //Actions for Manager-------------------------------------------------------------------
        private int mnConnect()
        {
                try
                {
                        ManagerData mn = (ManagerData) command.getObject();
                        command.clearObject();

                        //routine update of information on Manager
```

```
                mn.setAddress("" + incoming.getInetAddress());

                list.addManager(mn);
                return 1;
            }
            catch(Exception e)
            {
                System.out.println(e);
                return -1;
            }
        }

        private Vector grabDataHosts()
        {
            try
            {
                int numReq = command.getNum();
                int numGotten = 0;
                int offset = 0;
                Vector datahosts = new Vector();
                Vector temp = new Vector();

                while(numGotten < numReq)
                {
                    int numIndex = numReq - numGotten;

                    temp = list.getDH(numIndex, offset);

                    for(int i = 0; i<numIndex; i++)
                    {
                        DataHostData dhd = (DataHostData) temp.get(i);
                        Commands c = new Commands(SVID, SVKEY,
3, command.getLong());
                        CommandSender cs = new
CommandSender(dhd.getAddress(), DHPORT, c);
                        if(cs.sendCommand())
                        {
                            datahosts.add(dhd);
                            numGotten ++;
                        }
                    }
                    offset += numReq;
                    if(offset > (numReq * 10))
                        return new Vector();
                }

                return datahosts;
            }
            catch(Exception e)
            {
                return new Vector();
            }
        }

        //Actions for DataHost-----------------------------------------------------------------
        private int dhConnect()
        {
            try
            {
                DataHostData dhd = (DataHostData)
command.getObject();
                command.clearObject();

                //routine update of information on Manager
                dhd.setAddress("" + incoming.getInetAddress());

                list.addDataHost(dhd);
                return 1;
            }
            catch(Exception e)
            {
                System.out.println(e);
                return -1;
            }
        }
}
```