Rochester Institute of Technology

Distributed Systems-I

Final Project Report

Team Distributed Koders

Fairness in P2P Streaming Multicast

Team Members

Jason Winnebeck (jpw9607@cs.rit.edu)
John Kaeuper (jak8520@cs.rit.edu)
Kumar Keswani (krk1706@cs.rit.edu)

Project Web Site

http://www.gillius.org/distsys/

Professor Alan Kaminsky

Winter Quarter-20062

Table of Contents

Project Description

Most of the existing P2P networks face the challenges of maintaining a cooperative environment which forces clients to properly follow the protocols. Some clients act selfishly and misbehave in the network by not allowing their peers to obtain service from them. These clients are often called as “freeloaders” because they freeload on the system by refusing service to its peers. Freeloaders will refuse to forward the content to children and may refuse to accept children. We consider such fairness issues in the context of P2P multicast streaming applications such as ‘SplitStream’.

Hence in order to enforce fairness into the system for resource sharing we implement 3 different fairness mechanisms namely “Debt Maintenance”, “Ancestor Rating” and “Tree Reconstruction” as discussed in our research papers. Thus the main goal of our research is to enforce different fairness techniques into the system and evaluate our results using discrete event simulation.

There are several parameters that would not be simulated in this project.

- Network protocol messages to structure the network
- Routers and networks between the nodes
- Network congestion (tree algorithms will ensure no overloads)

In this research project we are going to simulate the individual packets being forwarded between nodes in the network. The nodes will have a parent child relationship and there is a direct connection between all the nodes. The simulation will include a percentage of packet loss on a per node basis.

The algorithms which we use for the simulation are mentioned below.

- Node fairness detection algorithms
- Tree construction algorithms

There are a few several **input requirements** for the simulation to work. These parameters are divided on the basis of nodes with their respective configuration, stripe information and simulation duration as shown below.

- List of nodes and their configuration
    - Behavior algorithm
    - Percentage of packet loss
    - Inbound and outbound bandwidth capacity
    - Time of entry into and time of departure from the multicast
- Stripe information
    - Number
    - Bits per second
    - Packet size
- Simulation duration (Time interval for the simulation to run)

The **outputs** that would be produced based on the input requirements are as follows.

- Percentage of packets received by node behavior group over time
- Cumulative distribution function (CDF) of nodes by behavior group and debt level at end of simulation
- CDF of nodes by behavior group and negative confidence at end of simulation
- Measurements of social welfare

Paper Analysis 1:
SplitStream: high-bandwidth multicast in cooperative environments

Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., and Singh, A

The main problem addressed by the SplitStream research paper is that, for a tree-based P2P multicast network, a single tree suffers from unacceptably poor qualities. There are several shortcomings. The first is that in a single tree, a small number of interior nodes must bear the forwarding burden for the entire network. This is unacceptable not only because it is unfair, but also because these nodes can only really handle such a burden if they are highly available, dedicated infrastructure routers. This is an expensive and usually unavailable resource. Most nodes would simply not be able to handle this forwarding. A second shortcoming is that the network will exhibit poor fault tolerance. If one node fails, then the ancestors of this node will receive none of the original content. A third shortcoming is that the network will offer poor scalability. The more nodes that join the network, the more overwhelmed the interior nodes will become. Their burden is compounded in a manner similar to a server in a server-client network in which there are too many clients.

The SplitStream design addresses these issues by striping the original content across k stripes, and multicasts each stripe in a separate tree. The nodes in the network only need to join the trees of the stripes they wish to receive. They may also specify how many children they are willing to accept. This forest of trees has two essential properties. One is that the trees are interior-node-disjoint. That is, each node is an interior node in only one tree and is a leaf node in all others. The other essential property of the forest is that it satisfies the node’s inbound and outbound bandwidth constraints. That is, each node must be able to receive all desired stripes and must not be forced to accept more children than it is willing. If the striping of the original content satisfies these properties, the shortcomings mentioned above are eliminated. The forwarding burden is now distributed over all nodes, rather than being placed on a constant set of interior nodes. The level of fault tolerance is now acceptable because if a node fails, its ancestors will only lose one stripe on average. And the network will now be scalable because as more nodes join the network, the new nodes will always pick up some of the increasing forwarding load. There is no constant set of interior nodes that must bear a larger and larger forwarding burden as the network grows.

There are two conditions that must be met if SplitStream is to be able to construction its striped forest satisfying the two essential properties mentioned above. The first is a necessary, but not sufficient condition. It is that the collective forwarding capacity of all nodes in the network must be greater than or equal to the collective desired indegree of all nodes. If this is not met, then some nodes must necessarily not receive all the stripes they desire, violating the essential forest property that all nodes’ bandwidth constraints are met. The second condition is sufficient to allow for a high probability of forest construction. This is that if any node in the network can forward more stripes than it wants to receive, it must receive (or originate, if node is a tree root) all k of the stripes. This basically prevents nodes from sending duplicate stripes. The essential forest property of satisfying a node’s inbound bandwidth will not be met if, for example, a node wants to receive stripes 1, 2, and 5, but only receives three copies of stripe 5. Each stripe received must be distinct. If these two conditions are met, then the probability that a striped forest of trees cannot be built in such a way as to satisfy the two essential properties of being interior-node-disjoint and meeting all node bandwidth constraints is negligible.

So how does SplitStream accomplish the construction of the striped forests that are interior-node-disjoint and satisfy node bandwidth constraints? It uses a Scribe group communication system that is built on top of the Pastry overlay protocol. Pastry provides a P2P routing mechanism. Nodes are assigned 128-bit node ids and messages are routed using 128-bit keys. The message is routed to the node whose id is numerically closest to the key, which is called the key's root. This will mean that the message is passed between nodes that have progressively longer prefix matches between their id's and message's key. If there is a tie in prefix match size, then the message is sent to the node whose id is simply numerically closer to the key.

The Scribe group communication system consists of multicast groups that are each assigned a pseudo-random Pastry key, called the groupId. In the SplitStream case, these groups are the trees of the forest, and the groupId key's root is the root node of that tree. So any message routed to a Scribe group (forest tree) is sent towards the groupId of that group (towards the root of that forest tree). Thus the multicast trees of the SplitStream forest are formed by combining the routes from new group members (tree nodes) to the groupId (to the tree root).

As long as the two conditions for the ability to build a striped forest are met, then this Pastry and Scribe implementation will allow for such a forest satisfying the two essential properties of meeting node bandwidth constraints and being interior-node-disjoint. The interior-node-disjoint property is satisfied by ensuring that all groupId's assigned to the tree roots differ in their most significant digit. Since the nodes along a path from a tree node to the tree root will have id's with progressively longer and longer prefix matches with the tree root's id, this will mean tree interior nodes must share the most significant digit with the groupId. And since these groupId's will differ in this most significant digit, the forest will be interior-node-disjoint. SplitStream may sometimes, although very rarely, violate the interior-node-disjoint property. This possibility involves the Spare Capacity Group, which is discussed below.

The essential property of meeting node bandwidth constraints is also satisfied by this Pastry and Scribe implementation. Nodes may join the trees of the stripes they wish to receive, so their inbound bandwidth constraints are met. In order to satisfy the outbound bandwidth constraint, the Scribe implementation of SplitStream allows for the orphaning of nodes from parent nodes whose outbound bandwidth exceeds its maximum. The algorithm is as follows. If a node attempts to be the child of a node with an exhausted outbound bandwidth, the child is accepted. Then, the parent attempts to orphan one of its children from a tree in which the parent's id shares no prefix match with that tree's groupId. (Note that in this case this parent must then be an interior node in a tree in which it shares no prefix match with the groupId, violating the property which establishes the interior-node-disjoint trees. This is extremely unlikely and involves the Spare Capacity Group, discussed below.) This will tend to restore the interior-node-disjoint property if it has been violated. If the child who just joined the tree is among these, it is chosen to be the orphan. Otherwise, a random node from this set is chosen. If there are no such nodes, then the parent is an interior node in only one tree, and the child with the shortest prefix match with that tree groupId is chosen to be the orphan. There may be several children with the same shortest prefix match. If so, and if the child who just joined the parent is among them, then it is selected. Otherwise, a random node from this set is selected.

Once the orphan is selected, a "push-down" process is followed in order to find a parent for the orphan. First, the orphan tries to become a child of its former siblings. If there are no such siblings, the orphan anycasts to the Spare Capacity Group (this is a Scribe tree consisting of all nodes having any spare forwarding capacity), and performs a depth-first search of this group to find a parent. Otherwise, the node attempts to attach to a random former sibling, and the same algorithm above is used to determine the new orphan. This process continues recursively down the tree. If a point is reached at which there are no former siblings of an orphan, or if there are no former siblings whose node id's share a prefix match with that sibling's tree's groupId, then the orphan must find a parent by anycasting to the Spare Capacity Group. A depth-first search is performed on this group to find a parent. As mentioned above, this is how the interior-node-disjoint property may be violated. The parent the orphan finds in the Spare Capacity Group may be an interior node in a second tree. However, since the Spare Capacity Group is only used as a last resort, this is very unlikely. It is also possible that no nodes in the Spare Capacity Group receive the required stripe, in which case a node cannot receive the desired stripe, and the application is notified. However, this is even more unlikely.

This paper provided us with the basic tree architecture to follow when implementing our RandomTreeManager class. That is, we implemented an algorithm for constructing the forest that tends to be interior-node-disjoint. This was done differently than the Pastry method because prefix matching was not used to construct the path from node to tree root. Instead, as the name of the class suggests, a node selects a random parent from the nodes in the network that forward the desired stripe and attaches to it. Each node is assigned a stripeId, which identifies the stripe that that node is required to forward (the stripe of the tree in which that node is required to be an interior node). So for each stripe that a node wishes to receive, it looks for a parent having the corresponding stripeId. Only after checking all nodes in the network for such a parent will a node attach to a parent that does not have the desired stripeId (but that receives the desired stripe). This parent should ideally be a leaf, but if required, it must be able to become an interior node. This process is analogous to the depth-first search of the Spare Capacity Group in the SplitStream implementation, in that it is possible for the interior-node-disjoint property to be violated as a last resort. However, this is unlikely. As in the SplitStream case, it is also possible that in our implementation a node may not be able to find a parent that receives the needed stripe and does not have exhausted outbound bandwidth. Our implementation will notify the simulation user with an error message in such cases. This too is unlikely.

Our RandomTreeManager implementation also satisfies the other essential forest property of satisfying node bandwidth constraints. That is, the construction algorithm searches for a stripe parent for a node as long as that node has remaining inbound bandwidth. Again, as the name of the class suggests, the stripes are randomly selected, but all distinct. The outbound bandwidth constraints are also satisfied in our implementation. A parent will not accept a child unless it has remaining outbound bandwidth. If a parent cannot accept a child, then, unlike in the SplitStream implementation, the child simply searches for a new, random parent. So our RandomTreeManager implementation constructs a forest satisfying the two essential properties of the SplitStream design, that of satisfying node bandwidth constraints and being interior-node-disjoint.

Paper Analysis 2:
Incentives Compatible Peer to Peer Multicast

Tsuen-Wan-Ngan, Dan S. Wallach, Peter Druschel

The authors Ngan, Dan and Druschel in their paper address the fairness issues in the context of p2p multicast streaming services. Such fairness issue arises due to the existence of freeloaders in the network. Freeloaders are the misbehaving peers who refuse to provide service to other peers in the network. Hence in order to solve the fairness issues, this paper presents incentives compatible policies and mechanisms that would detect nodes with selfish behavior and thus reduce the quality of such freeloaders in the network.

The experiments performed by the authors are a representation using simulation for studying the various fairness mechanisms. The experiments are run using the instrumented version of 'SplitStream'. Several mechanisms are simulated in isolation and finally combinations of experiments are run for discriminating between selfish and normal nodes.

The authors assume that most of the p2p system designs use cooperative environments using multiple clients running the same software. Hence some clients can modify their software in order to get the maximum benefit in the network. The authors assume that the current tit for tat strategies are not suitable for multicast applications because they make use of static distribution trees which are constructed only once and are used forever. Also, they assume that a node can falsely claim that its outgoing bandwidth is fully utilized and would refuse to accept children.

Hence the authors presents several fairness mechanism and assume that by periodically rebuilding the multicast forest trees, the parent-child relationships can reverse and thus giving a chance to nodes to refuse service to the freeloaders. In one of their fairness technique called "Debt Maintenance" when a packet is forwarded between two nodes each node keeps a track of debts between each other. By choosing a threshold, nodes can refuse service if the debt goes beyond the threshold.

Their next fairness mechanism uses the periodic tree reconstruction technique as mentioned earlier. Here they assume that a node may suffer in an unfair position if the trees are built randomly. This is because an innocent nodes' parent could be a freeloader. Hence they built forest trees periodically to track down freeloaders in the system. The next mechanism they discuss is the "Reciprocal requests" where they suggest reversing the parent- child relationship if a node has always been a parent of another node. This would allow a child to pay off its debts and regain its reputation.

The "Ancestor Rating" is another mechanism the authors suggest which they assume is an extension to their "Debt Maintenance" technique. Here the debts/credits are applied not only to immediate parents but to all the nodes in the path to the root. Here the authors assume that selfish nodes have no incentive to provide correct information and hence they rely on the knowledge of ancestors. It uses the confidence value instead of debts. When the trees are reconstructed, falsely blamed nodes will average out and freeloaders will be detected. Another fairness mechanism they mention is preventing 'Sybil attacks'. Here they assume that if a node's reputation goes down, it can rejoin the network using new identities and get back its lost reputation. This is termed as 'Sybil attack'. This is avoided by allowing each node to start on a probation period and gain its reputation over a period of time by providing service to other nodes.

Thus for these various fairness mechanisms, the authors assume that its effectiveness will depend strongly on the choice of multicast applications, p2p routing substrates and network topologies.

The fairness mechanisms described by the authors helps in achieving their goal of identifying freeloaders and misbehaving peers in the multicast network. Also these freeloaders are denied service once they are identified. They demonstrate that periodically rebuilding forest trees helps in calculating the reputation on a per node basis.

Most of the peer to peer systems have no method of dealing with the selfish peers. All of these networks assume cooperative environments. The techniques mentioned by the authors helps in achieving fairness in the p2p multicast streaming services. These fairness mechanisms react severely to the freeloaders by denying service to them. The experiments performed by the authors showed a simulation where it identifies and refuses service to the freeloaders in the system.

The fairness mechanisms used in their research has network and computational overhead low helping in scaling large number of nodes. Also the parent-child reverse role plays a crucial role in the success of their fairness mechanisms.

The paper only addresses some interesting behaviors of nodes and does not address malicious behavior where freeloading nodes deliberately prevent distribution of streaming media. Also the results are shown using simulation. The fairness mechanisms should have been implemented using real application. Also they could implement robustness into the system so that it can tolerate very large number of freeloaders, but this is one of their future works.

Paper Analysis 3:

A Case for Taxation in Peer-to-Peer Streaming Broadcast

Yang-hua Chu, John Chuang, and Hui Zhang

In their paper, the authors Chu, Chuang and Zhang present an approach for overcoming the bit-for-bit contribution status quo for peer-to-peer (p2p) streaming multicast networks by introducing a taxation model. Using a linear taxation scheme, the publisher in the network sets a publicly published and fixed tax scale that allows the nodes in the network to maximize their utility (benefit minus cost) in a self-strategic manner.

The authors have experience working with p2p streaming systems, specifically the End System Multicast hosted by Carnegie Mellon University (http://esm.cs.cmu.edu/). According to its site at CMU, ESM is a single tree p2p multicast system; however, the authors present the material in the paper in the context of multiple tree multicast systems like that in SplitStream.

The authors state that protocol designers have settled with the bit-for-bit contribution model because it is simple and is inherently fair. One of their examples cites SplitStream. While nothing in SplitStream requires that nodes contribute equally to the network, it seems reasonable that a bit-for-bit model is a natural choice.

The primary fault of this model, according to the paper, is that in the modern Internet environment, typical users have asymmetrical bandwidths. Typically, users on DSL or cable connections have a higher capacity to download than upload. The bit-for-bit model would restrict the maximum content bitrate to the effective upstream capacity of these peers, the argument being that due to poor content (audio/video) quality, users would be discouraged from viewing the content. Another assumption is that the network contains also a number of high-bandwidth peers, perhaps such as those on university connections that are able to provide more bandwidth than what they receive.

The taxation model assumes that the publisher has the ability and the will to resolve this situation using the model. The will comes from the publisher’s desire to have their content viewed (a reasonable assumption). The ability comes from the unique situation of the p2p streaming situation, since the publisher has control over the content. Essentially, the end result is that the publisher subsidizes the upstream-poor peers with bandwidth from the upstream-rich peers, and in return gives the upstream-rich a higher quality stream. The paper assumes that peers are strategic, in that they make a trade-off of contribution versus quality based on their resources.

The conclusion provided with the work is that “taxation is a strongly dominating strategy over Bit-for-Bit when peers are heterogeneous.” By heterogeneous, this means that the network contains a mixture of peers with low and high bandwidth capacities. The data collected by the authors give a network composition with a 20 high / 80 low split.

The work done by the authors contributes a novel idea of using an asymmetric model of peer contribution to resolve the issue of the asymmetric connections commonly found in the modern Internet. While the authors state “As a proof of concept, we have implemented and deployed a p2p streaming system” in reference to ESM, they do not directly state that they implemented taxation in a real streaming system. Checking the technical descriptions of the software from the ESM site and noting that the data and conclusions were drawn from simulation, I do not believe that the authors implemented taxation in ESM – the text can be slightly misleading to some.

However, despite some unresolved issues, with some further work I believe that the idea can be implemented in an actual system and even be possibly freeloader resistant.

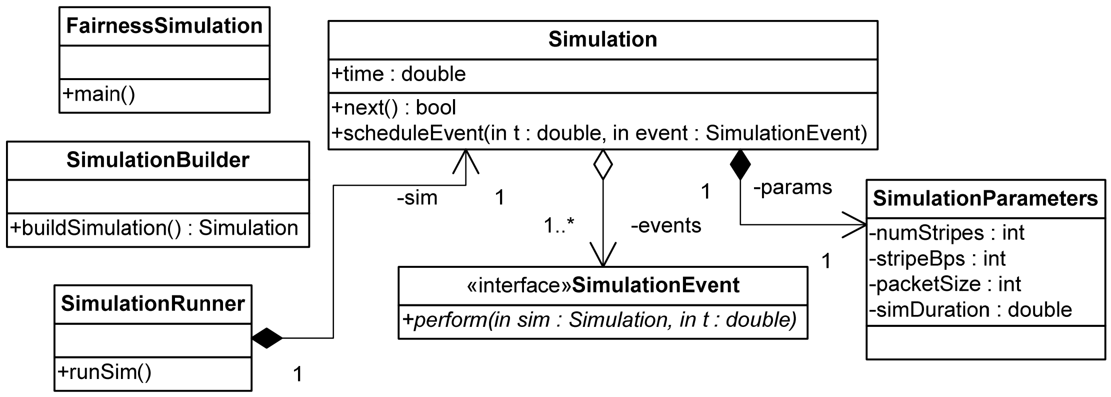
There are some limitations in the research and areas that can be improved, perhaps with some future research. One of the more apparent areas is the assumption that the publisher has the means to control the network to enforce that peers are paying their taxes properly. Essentially, the paper mentions simply that in this case the means consists of proprietary software that the publisher has developed. However, a solution that can work without trusting the software on the peers or relying on security through obscurity would be better. This is not a critical weakness of the work, because this can be solved through further research.

Another area that could be improved is the equations that the authors chose for cost and benefit. The authors acknowledge the fact that the equations are based on intuition. They seem reasonable – and the most important property that the benefit function is concave and the cost function is convex when evaluating for the bandwidth rate is maintained. Probably the fuzziest area in the equation is how the cost function works, because it will differ based on whether the connection is shared, and even when shared it is questionable what number to use for total bandwidth and how to rate cost. For example a university network may have a 100mbps connection for 10,000 users, but using 100mbps for the total bandwidth parameter seems unreasonable. Also, in such a situation essentially the user considers the cost of bandwidth to be “free” since he or she has no way of measuring the impact of forwarding a certain number of stripes. A future topic of research to address this would be to see how users judge benefit and cost in an actual streaming system. Such research would also provide additional data sets beyond the single data set for network composition that the authors used.

Fairness Simulation Design

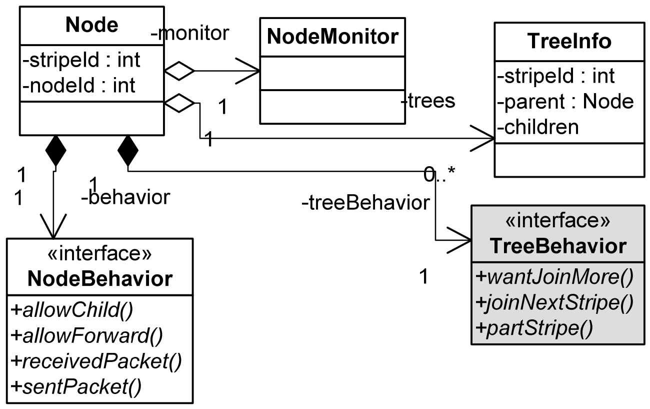
This section of our report will deal with the design of the classes and the design decisions that were made. The UML shown in the following descriptions is not exhaustive; many elements have been simplified or left out so that the concepts and relationships can have the focus. The generated documentation will have document the complete method specifications and more detailed information on how each class works.

Core Simulation Classes



The core classes of the simulation process the input and output from the Simulation. The Simulation class itself holds a priority queue of events and processes events in time order. The perform method of the SimulationEvent is called when it is that events turn to run. The simulation is run until there are no more events or until the simulation reaches the end of the desired time range (this is one of the simulation parameters).

Node Class



The Node class is the center of the program design. It contains all of the information for its participation in the trees, for which there is one tree per stripe. A Node is typically in multiple trees at the same time, depending on its amount of inbound bandwidth. The NodeBehavior class is utilized to implement a freeloader or a freeloader detector of our two different types (ancestor rating or debt maintenance). The TreeBehavior was to be used as part of the taxation system but this part of the system was not implemented. The NodeMonitor contains methods to receive events for things happening in Node and is used to collect statistics for the “packets received” graph output at the end of the simulation. Nodes also route packets to their children. The amount of time it takes to transfer is calculated based on the bandwidth of the source and destination node and a NodeTransferEvent is created to deliver the packet to the destination at the correct time.

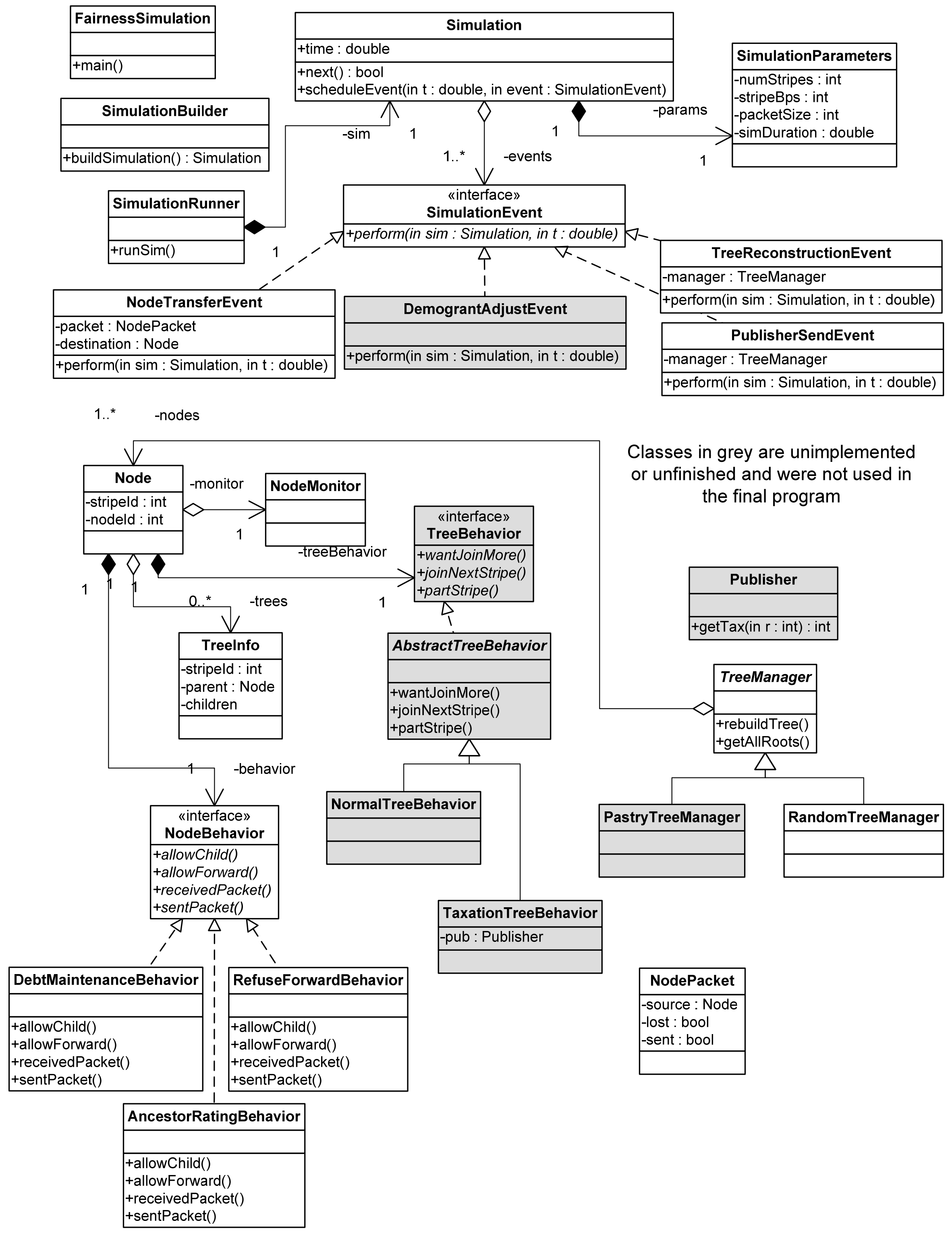
TreeManager

The TreeManager manages the construction of the stripe trees, and is also responsible for assigning node IDs and stripe IDs. The TreeManager also reports the stripe root nodes so that the PublisherSendEvent can create the initial stripe packets that will be routed in the network. The TreeReconstructionEvent tells the TreeManager when to rebuild the tree. When the trees are rebuilt, all pending NodeTransferEvents are cancelled to prevent errors with routing and confusion in the nodes by not having them receive packets from invalid parents.

There are two types of tree managers. The random tree manager constructs a tree randomly by giving each node a single turn to join a tree. The manager searches the nodes in the desired stripe tree randomly by visiting nodes in that stripe tree until it finds an appropriate parent for the joining node.

The second type of tree manager implements the tree building algorithm from Pastry, but unfortunately we were unable to get this manager working.

Complete Fairness Design UML



Classes in grey are unimplemented or unfinished and were not used in the final program

Fairness Simulation User Guide

This user guide will explain how to configure and run the fairness simulation, and explain the outputs from the program.

Prerequisites

In order to run the fairness simulation program, Java JRE 1.5 or later is required as well as the Computer Science Class Library found at http://www.cs.rit.edu/~ark/cscl.shtml. A cut down version of the CSCL built on 7-Feb-2007 is included in the package in case the original source is not available. The CSCL is available under the terms of the GNU General Public License. You may obtain a copy of the GNU General Public License on the World Wide Web at http://www.gnu.org/licenses/gpl.html or by writing to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

To run the unit tests, JUnit version 4 or later is required. JUnit can be found at http://www.junit.org/.

Compiling the Program

From the raw source code, on a UNIX-like machine run the `compile.sh` script. You can also download the executable version, which includes a jar file with code already precompiled. Assuming the CSCL is installed in the base of the fairness directory, one can compile with the following command:

```
javac -d classes -cp cscl.jar -sourcepath src
src/Fairness/FairnessSimulation.java
```

Note: The command above is all on one line but here is wrapped due to space. Compilation of the tests can be done in a similar way but by choosing files in the tests directory instead of src.

Invoking the Program

The name of the main class is `Fairness.FairnessSimulation`. The program takes two parameters:

1. Input file or input directory
   - If the first parameter is a file, loads a single file and runs the simulation.
   - If the first parameter is a directory, runs a simulation on each file in the directory ending in “`.txt`”
2. Output directory

An example is shown below, assuming classes were compiled with `javac Fairness.FairnessSimulation.java`

```
java –cp cscl.jar;classes Fairness.FairnessSimulation
data/inputs/2freeloaders.txt data/outputs
```

Note: The command above is all on one line but here is wrapped due to space. For convenience, there are two wrapper scripts that will work on Microsoft Windows or UNIX-like systems, `runSim.sh` and `runSim.bat`. These exist only in the executable version of the distribution and can be used as the following:

```
./runSim.sh data/inputs/2freeloaders.txt data/outputs
```

Program Output

At the start of each run the simulation outputs what file it is processing, and when it is over it writes how long the process took. An example transcript is shown below for a single run:

```
Running simulation with C:\ fairness\data\inputs\AncManyChild.txt
  Simulation complete, writing output
  Finished in 98 seconds
```

Output files were generated as a result of this process, and their content is described later in this guide.

Note on a Bug

Unfortunately there can be a bug in the program where the RandomTreeManager can generate an invalid tree, and an error “the source is not valid for the packet” will appear. If this occurs, it can be possible to fix by changing the random number seed. This error happens much more often if the number of nodes is small. Unfortunately we were unable to fix this bug in time for submission.

Input File Format

The input file format is a text file with at least two lines. The first line specifies global simulation parameters, and the second and following lines specify groups of nodes to instantiate.

Global Parameters Line

The global parameters line contains 7 items, separated by spaces:

1. Number of stripes
2. Stripe kilobits per second
3. Node packet size (bytes)
4. Simulation Duration (seconds)
5. Random Seed
6. Tree Manager Implementation (class name) – right now only `Fairness.Trees.RandomTreeManager` is working.
7. Tree Manager reconstruction rate (seconds)

Node Group Line

Every line beyond the first is a node group line. The node group line specifies a certain number of nodes tied under a common group name. Each group is monitored separately when the packet reception, debt level, and confidence level graphs are generated. There are 7 items on each node group line:

1. Number of nodes to create
2. Name of the node group

3. NodeBehavior implementation, possibilities include:
    - Fairness.Nodes.NodeBehaviors.RefuseForwardBehavior: always refuses to send packets
    - Fairness.Nodes.NodeBehaviors.DebtMaintenanceBehavior: performs debt maintenance, and if given a parameter, stops forwarding to freeloaders with debt over that parameter
    - Fairness.Nodes.NodeBehaviors.AncestorRatingBehavior: performs confidence rating, and if given a parameter, refuses service to freeloaders with confidence less than the parameter
    - Fairness.Nodes.NodeBehaviors.SimpleNodeBehavior: Always forwards to all nodes
4. NodeBehavior parameter (optional), or `null` if not specified. Usually this is a threshold value for the freeloader detector types.
5. Inbound bandwidth in kilobits per second
6. Outbound bandwidth in kilobits per second
7. Average percentage of packets lost when sending or receiving

Example Input File

An example input file is given below (unfortunately there are line breaks due to the size of the page):

```
8 16 512 1530.0 1234567 Fairness.Trees.RandomTreeManager 5.0
49 DebtM Fairness.Nodes.NodeBehaviors.DebtMaintenanceBehavior 550 128 128
0.000
2 Freeloaders Fairness.Nodes.NodeBehaviors.RefuseForwardBehavior null 128 128
0.000
49 AncestorR Fairness.Nodes.NodeBehaviors.AncestorRatingBehavior -550 128 128
0.000
```

There are 4 lines in this file. The first line specifies the 7 global parameters: 8 stripes each at 16 kbps per stripe in 512 byte packets. The simulation will run for 1530 seconds, and a RandomTreeManager manages the network and rebuilds every 5 seconds. A random number seed 1234567 is provided.

The following three lines create groups of nodes, 49 debt maintenance nodes, 49 ancestor rating nodes, and 2 freeloaders who never forward packets. Every node has an incoming and outgoing bandwidth of 128 kbps, so they will join 8 stripes and have 8 children. The thresholds for debt level are set to 550, and for confidence, -550. The detectors will refuse service to nodes that have more than 550 debt or less than -550 confidence.

Output File Format

After the simulation ends, the simulation will generate several files. The graphs are images in PNG format, and the raw data samples are in CSV (comma separated values) format that can be loaded with a program like Excel. The CSV files may contain more columns of data than the graph.

- Packets received graph and CSV data file for each group of nodes. The x-axis is the time in number of tree reconstructions, and the percentage is the fraction of packets received in that time period. The files are named after the name of the group.
- Combined packets received graph and CSV data file for all groups of nodes. The files are named with “combined”.
- If there were any ancestor rating nodes, a “ConfCDF” showing a confidence CDF for all nodes was generated. The graph plots the cumulative distribution function with percent on the y axis and confidence on the x axis. The related CSV file contains the numbers used.
- If there were any debt maintenance nodes, a “DebtCDF” showing the debt level CDF for all nodes was generated. The graph plots the cumulative distribution function with percent on the y axis and debt level on the x axis. The related CSV file contains the numbers used.

From the example input shown above, the simulation would output 12 files. If the file was named “2Freeloaders.txt,” the outputs would be named:

- 2Freeloaders-AncestorR.csv (received packets for node group AncestorR)
- 2Freeloaders-AncestorR.png
- 2Freeloaders-combined.csv (received packets for all node groups)
- 2Freeloaders-combined.png
- 2Freeloaders-ConfCDF.csv (node confidence rating CDF)
- 2Freeloaders-ConfCDF.png
- 2Freeloaders-DebtCDF.csv (debt level rating CDF)
- 2Freeloaders-DebtCDF.png
- 2Freeloaders-DebtM.csv (received packets for node group DebtM)
- 2Freeloaders-DebtM.png
- 2Freeloaders-Freeloaders.csv (received packets for node group Freeloaders)
- 2Freeloaders-Freeloaders.png

Results

The results we accumulated from our simulations involved the varying of several input parameters. These were the rate of tree reconstructions (as seconds between each reconstruction), the node outbound bandwidth (same value for each node), the number of nodes and their type (freeloader, those implementing ancestor rating behavior, those implementing debt maintenance behavior) distribution, and the threshold values used for debt maintenance and ancestor rating. The other input parameters were constant for our simulations. We used forests consisting of ten stripes, a bandwidth of 100 kbps for each stripe, a packet size of 512 bytes, a 1560 second simulation duration, a node inbound bandwidth of 1000 kbps, a packet loss rate of 0.5% (or 0% for a "baseline" simulation, discussed below), and a RandomTreeManager implementation for the tree reconstructions.

The results from our simulation consisted of graphs of Cumulative Distribution Functions (CDF) and percentage of packets versus the number of total tree reconstructions (which measures simulation time since tree reconstructions are done at constant intervals) made since the simulation start time. The CDF graphs show the distribution among normal and freeloader nodes at the very end of the simulation. That is, the x-axis value is a debt value or ancestor rating, and the y-axis value is the percentage of nodes of that type (freeloader or normal) that are at or below that debt value or ancestor rating. We used these CDF graphs to help us pick an appropriate debt or ancestor rating threshold value. The threshold is chosen such that a separation of the two types of nodes may be achieved. That is, a threshold should be chosen such that at that debt value, all or almost all of the freeloader nodes are at or above (or at or below for ancestor rating confidence values) that value, and none or almost none of the normal nodes are at or below (or at or above for ancestor rating confidence values) that value by the end of the simulation.

The percentage of packets graphs are the real measure of success of a fairness mechanism. Over time (tree reconstructions), one would ideally like to see the freeloader nodes percentage of packets received to drop quickly to 0, and the normal nodes percentage to stay constant above 90%. So there are really two elements to the measure of success. The freeloader nodes should be cut out of the network quickly, and the normal nodes should not lose a significant number of packets. Of course, the normal nodes will not receive 100% of the packets just because of the very presence of the freeloaders and their refusal to forward packets and because of the 0.5% packet loss rate. However, this percentage will be less than 100% also because the nodes will be falsely blamed for the freeloaders' behavior. Normal nodes will sometimes refuse service to other normal nodes, thinking they are freeloaders. The careful choice of the input parameters should minimize this false blaming.

Before running simulations of networks containing freeloaders, we first tested a network with no freeloaders (only debt maintenance nodes or only ancestor rating nodes), and then a network with no freeloaders and also no packet loss. This was done to obtain a "baseline" reading of the network for both debt maintenance and ancestor rating. The baseline percentage of packets received with no packet loss is shown in Figure 1 for debt maintenance and in Figure 2 for ancestor rating. The corresponding graphs for a baseline network with 0.05% packet loss are shown in Figure 3 for debt maintenance and Figure 4 for ancestor rating.

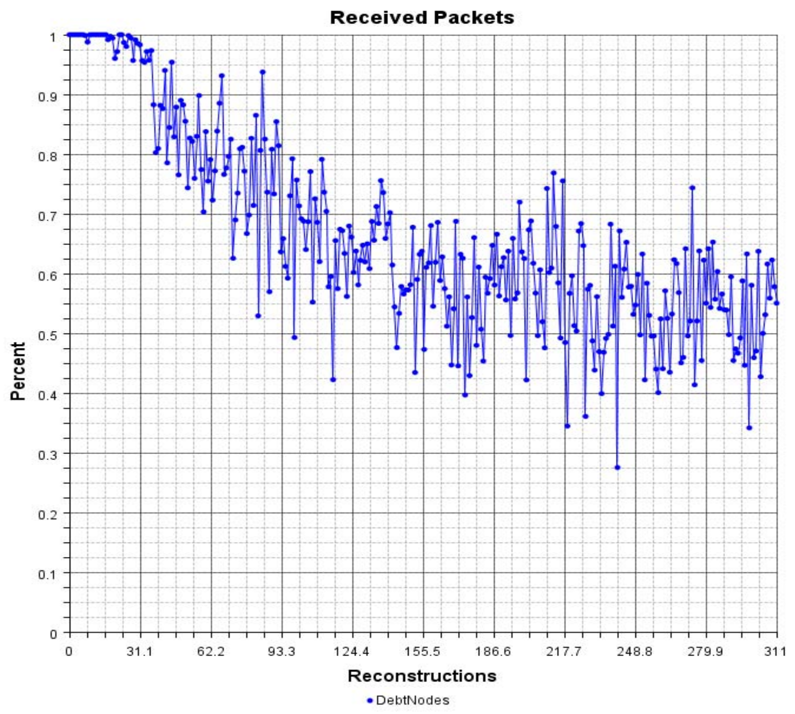


Figure 1

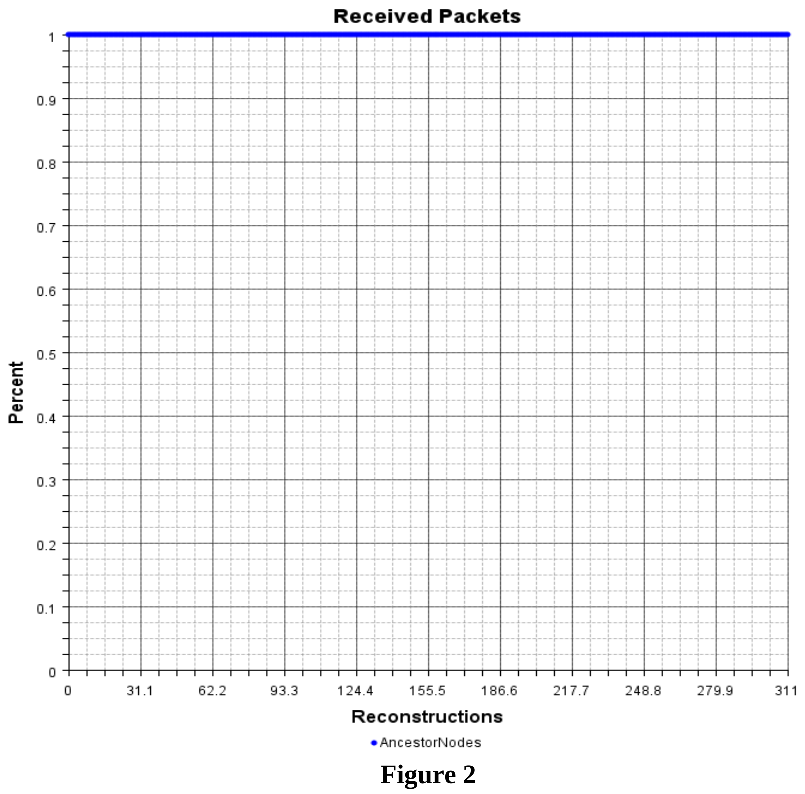


Figure 2

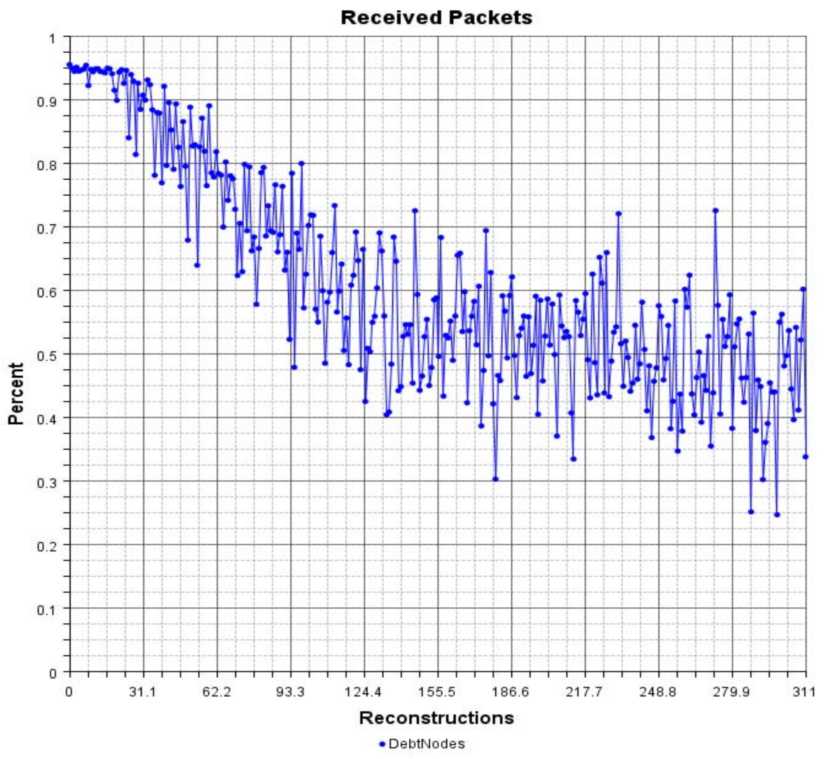


Figure 3

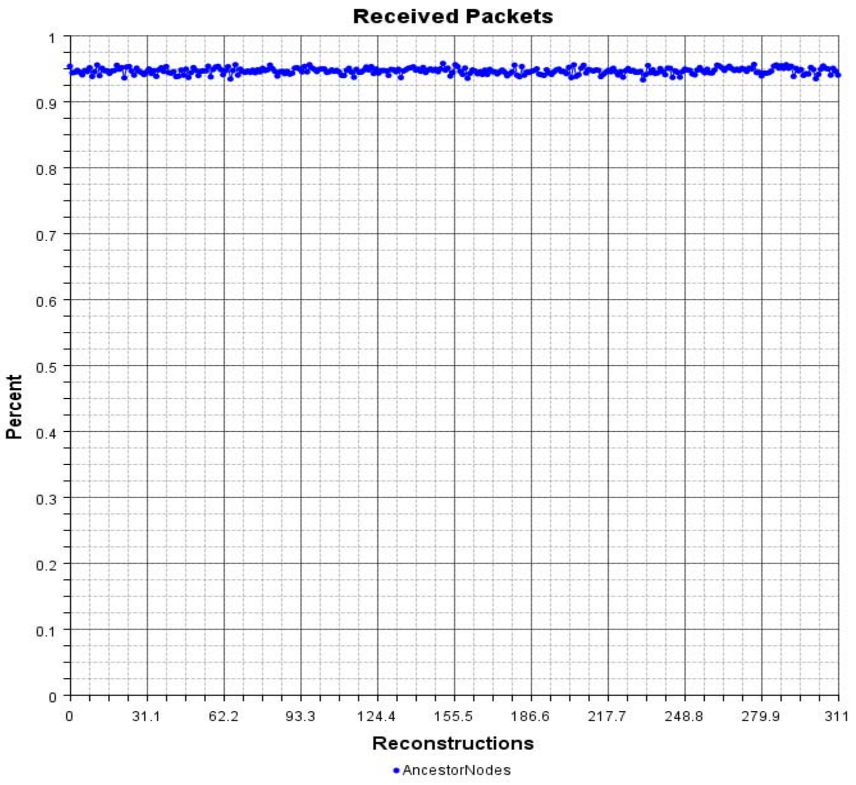


Figure 4

The baseline simulations consisting of only debt maintenance nodes yielded a puzzling and unfortunate result. As may be seen in the figures, roughly half the packets are lost by the debt maintenance nodes, even in the case with no packet loss. We propose two possible explanations for this result. One is that, unlike in ancestor rating, every time a packet is sent under a debt maintenance behavior a debt is accumulated. Therefore, nodes may be falsely blamed as freeloaders even when there is no packet loss. If the RandomTreeManager tree reconstructions do not reverse a particular parent-child relationship often enough, then debts will not be paid off, and nodes will be falsely blamed as freeloaders. One remedy for this possible pitfall (something that could be done as future work) is to implement a reciprocal request fairness mechanism, as mentioned above in the second research paper analysis. This would force the reversal of the parent-child relationships, which would allow the paying off of debts, and the minimizing of false blaming of normal nodes. Another possible source of this packet loss could simply be an incorrect implementation of our design. This is always a possibility.

The baseline simulations for the ancestor rating nodes yielded the desired results. There is no packet loss whatsoever for the 0% packet loss baseline simulation. This makes sense for the ancestor rating mechanism because with no packet loss, no nodes ever suffer any damage to their network reputation based on their confidence levels. Only when a packet is not received by a node do the nodes above it in a tree suffer any damage to their confidence levels. There is only a minimal packet loss in the 0.5% packet loss baseline simulation graph of Figure 4. This is obviously due to the 0.5% loss itself, but there is more that may be concluded from this graph. This is that none of the graph's packet loss should be due to any false blaming of nodes because there really is no change in the packet loss as the simulation continues. At the beginning of a simulation, no nodes will be falsely blamed as freeloaders, so if they will be so blamed, the effect of this must be seen in a change in packets received with increasing time. This is not seen in Figure 4. Ancestor rating is therefore much less susceptible to false blaming of nodes than debt maintenance, and it must take more than a 0.5% packet loss for this to occur. This makes sense with the ancestor rating mechanism. When an ancestor rating node suffers a false confidence level damage from the 0.05% packet loss, it will be a simple matter of successfully sending another packet to regain that confidence. There are not even any tree reconstructions required to make this occur if there are no freeloaders. However, with debt maintenance, tree reconstructions must randomly reverse a specific parent-child relationship. This is less likely to occur.

The first results we obtained for simulations involving freeloaders were of networks containing 95 freeloader nodes and 5 debt maintenance nodes. The tree reconstruction rate was once every 5 seconds, and the outbound bandwidth was chosen such that each node could have four children. Figures 5, 6, and 7 show the received packets graph for such a simulation with debt thresholds of 200, 550, and 750, respectively.

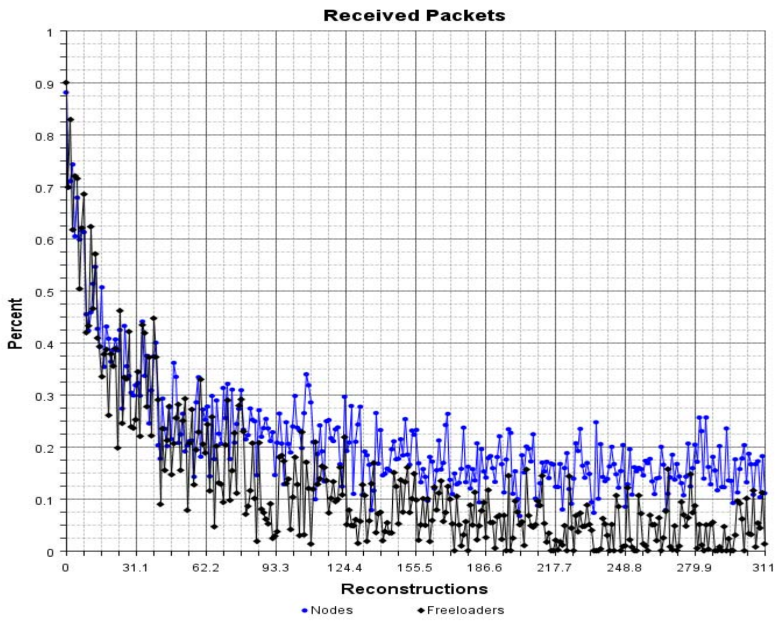


Figure 5

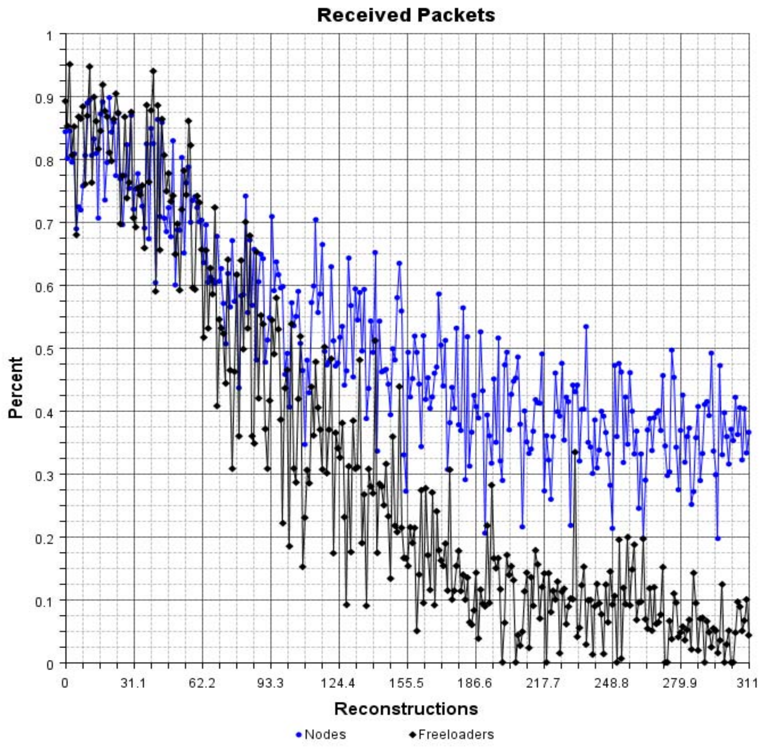


Figure 6

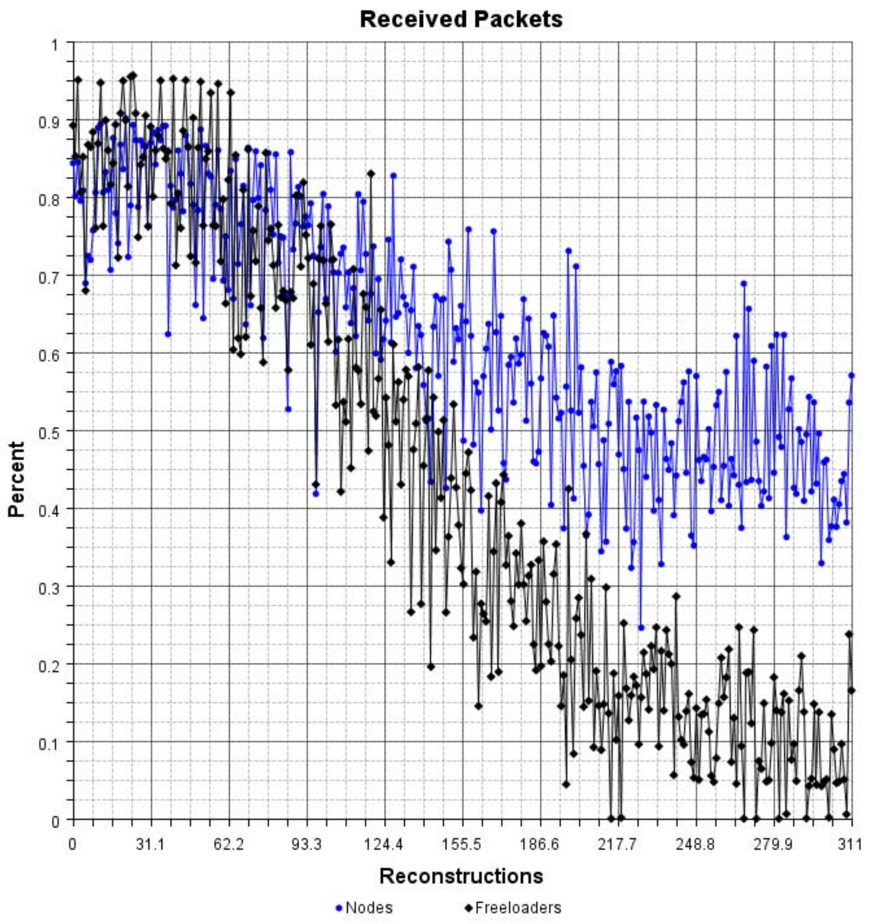


Figure 7

As may be seen in these Figures, the undesired packet loss of the normal nodes is still present. None of these results separated the normal nodes from the freeloader nodes very effectively. However, what may be concluded is that there is a debt threshold tradeoff. This is important to consider in choosing input parameters regardless of the fairness mechanism used. If the threshold is too low, then it becomes too easy for the normal nodes to be falsely blamed as freeloaders. This effect is seen in Figure 5, since the normal nodes receive only minimally more packets than the freeloaders by the end of the simulation. In Figures 6 and 7, the normal nodes start to receive more packets with the higher threshold. It becomes harder to blame normal nodes as freeloaders. However, the tradeoff may be seen if we examine the rate at which the freeloaders are denied service. This rate is fastest for Figure 4 with the lowest threshold. So there is a tradeoff between normal nodes receiving the packets they should be receiving and the rate at which the freeloader nodes are denied service. Both are important considerations in a P2P network, but neither can be achieved simultaneously with any satisfaction, at least in terms of threshold choice.

The next result we obtained was for the same simulation input parameters as Figure 6 above, except that a faster tree reconstruction rate of once per second was used. The packets received graph for this simulation is shown in Figure 8 below.

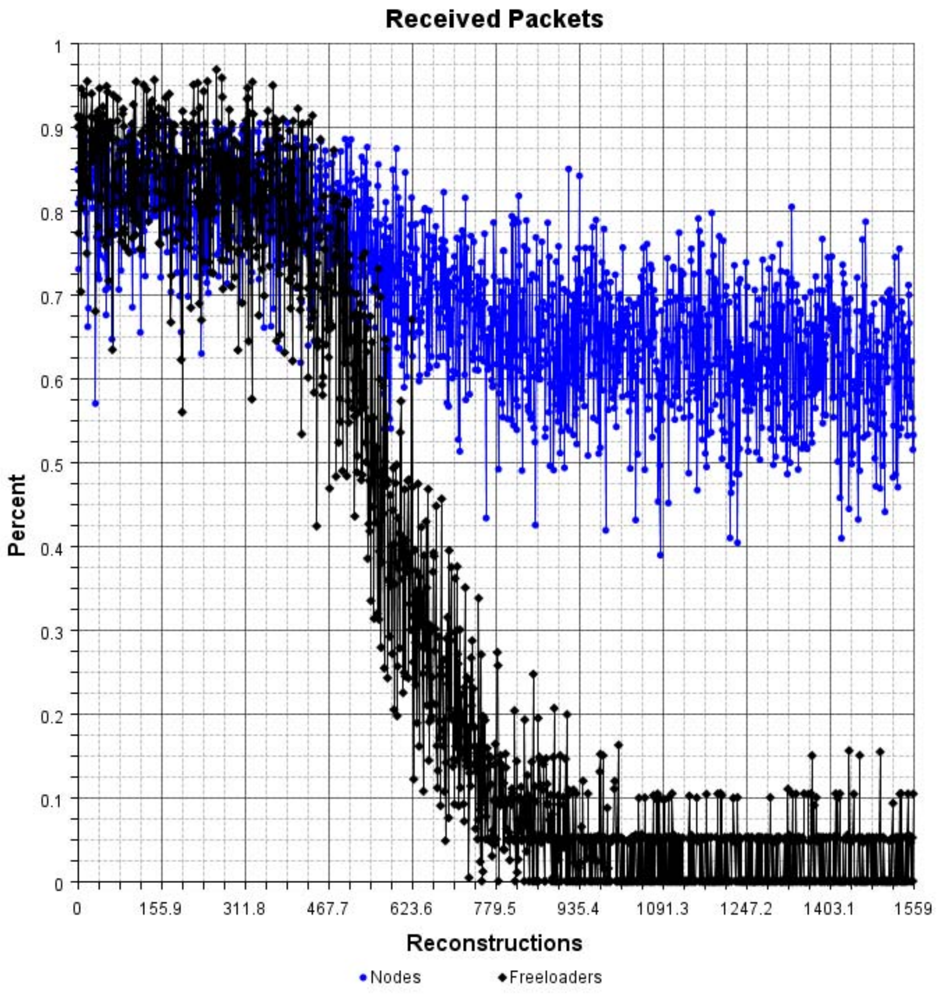


Figure 8

The simulation for this graph is the most successful debt maintenance result we achieved. There is a significant separation of normal and freeloader nodes, albeit with the usual dip to a roughly 65% packet rate for the normal nodes. This finding leads to a conclusion that will also be discussed for ancestor rating below. This is that the tree reconstruction rate seems to be a very influential factor in determining the success of a simulation in separating the freeloaders from the normal nodes, perhaps more important than the threshold. As compared to Figure 6, Figure 8 shows that both benefits of the threshold tradeoff are improved. The rate at which the freeloaders are denied service is much faster, and also the normal nodes improve their packets received by roughly 30%.

Our results for ancestor rating similarly indicated the importance of the threshold and tree reconstruction rates. It is less clear in this case, however, which of these factors is more important. These results are shown in the graphs of Figures 9-12 below. The four simulations of these figures are all of networks consisting of 95 normal nodes and 5 freeloader nodes. The nodes all have an outbound bandwidth allowing four children per node. The simulations of Figures 9 and 10 use a confidence level threshold of -200, with that of Figure 9 having a tree reconstruction every 30 seconds, and that of Figure 10 having a tree reconstruction every 5 seconds. The input parameters for the simulations of Figures 11 and 12 are identical to those of Figures 9 and 10, respectively, except that they use a confidence level threshold of -550.

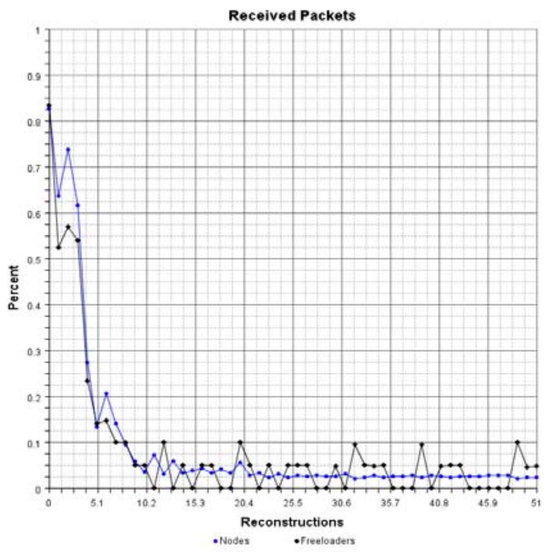


Figure 9

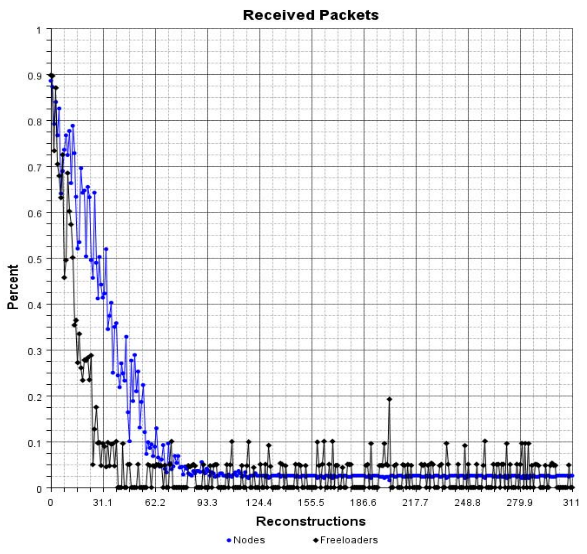


Figure 10

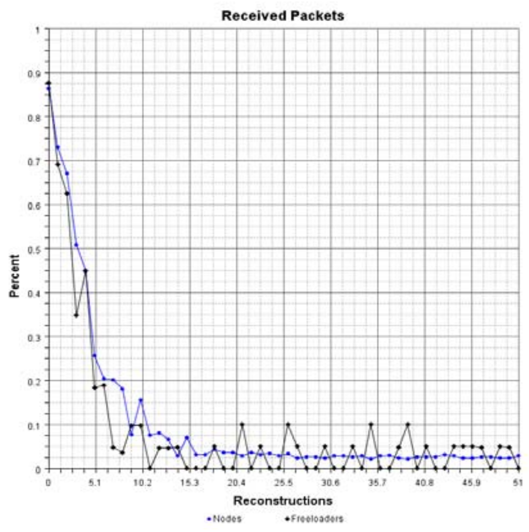


Figure 11

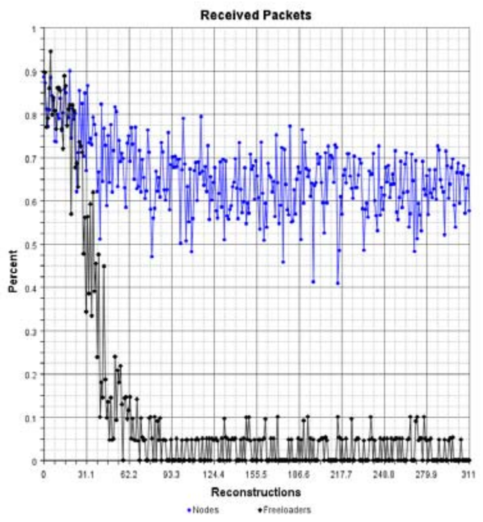


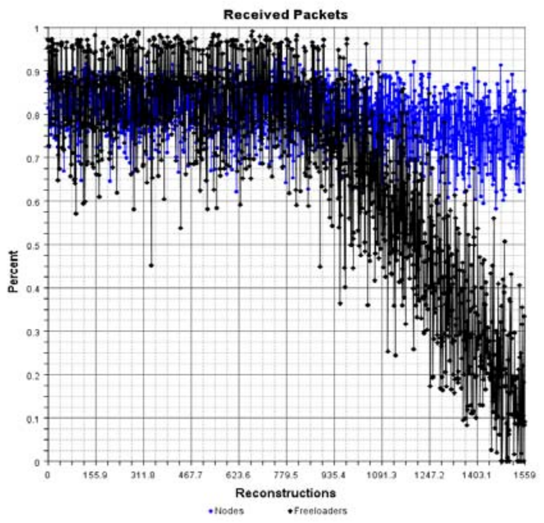
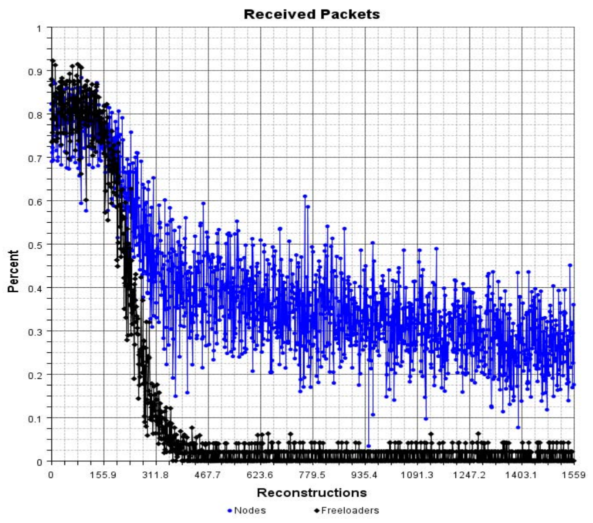
Figure 12

As can be seen in Figures 9-12, there is a drastic improvement in the separation of normal and freeloader nodes in changing from either the input parameters of the simulation in Figure 11 to that of Figure 12, or from that of Figure 10 to Figure 12. In the first case, we are only changing the tree reconstruction rate from once every 30 seconds to once every 5 seconds. In the second case, we are only changing the threshold from -200 to -550. However, the change in moving from Figure 11 to Figure 12 is slightly more drastic than in moving from Figure 10 to Figure 12. So this lends some support to the conclusion that tree reconstruction rate is more important than threshold in the ancestor rating case, as well as in the debt maintenance case.

The reasons for the powerful influence of threshold value are similar, we believe, to those reasons for its powerful influence in debt maintenance. When it is too low (in absolute value) at -200, it is too easy to falsely blame normal nodes as freeloaders, so they are denied service along with the freeloaders in Figures 9 and 10.

It seems, however, that the success of freeloader detection is even more sensitive to the tree reconstruction rate for ancestor rating than for debt maintenance. This is supported by the fact that the difference in the success of the simulations of Figures 11 and 12 is more drastic than the difference in the success of the simulations of Figures 6 and 8. The only difference between these pairs of simulations is that the first of each pair has a slower tree reconstruction rate. As indicated previously, we believe that this is due to the fact that tree reconstructions are more beneficial in helping innocent nodes to restore their network reputations as ancestor rating confidence values than they are in restoring innocent nodes' reputations in terms of debt values. After a tree reconstruction, it is much less likely that a particular parent-child pair will have their roles reversed than it is that a new node pathway to a certain node that was not receiving packets previously will now contain no freeloaders.

We also obtained results for simulations in which the node outbound bandwidths are varied. In the debt maintenance case, we ran two simulations with the exact same input parameters as those of the simulation of Figure 8, but in the first case the outbound bandwidths of all nodes only allow them to accept at most two children, and in the second case they may accept as many as ten children. The packets received graphs for these two simulations are shown in Figures 13 and 14, respectively.

 **Figure 13**	 **Figure 14**

There is a similar tradeoff in these simulations as in the threshold case. With fewer children, there is the benefit that normal nodes do not receive as few packets, but then it takes much longer for the freeloaders to be denied service. In the case of ten children per node, the nodes receive fewer packets, but the freeloaders are denied service very quickly. There is a difference, however, in that in Figure 13 the normal nodes receive more packets than even the normal nodes in Figure 8. In some sense, the results of Figure 13 are even more successful than those of Figure 8 because of this increased number of packets received by normal nodes. And it does appear that the percentage of packets received by freeloaders is headed towards 0. The results of Figure 14 are less promising, since the normal nodes receive few packets. However, there is also a positive result of Figure 14, and that is that the freeloaders are denied service the fastest of all the debt maintenance simulations we ran. It could also be argued that since the freeloaders are denied service so quickly, they will leave the network quickly, and the normal nodes will begin receiving a higher percentage of packets. However, given the low numbers of packets receive in Figure 1, and also given that we did not simulate freeloaders leaving the network, this conclusion would be shaky at best. Therefore, we do not make this claim. These simulations are left to future work.

The difference between the two children per node and ten children per node cases is even more drastic in the ancestor rating case in terms of the success of separating the normal and freeloader nodes. The results are shown in Figures 15 and 16.

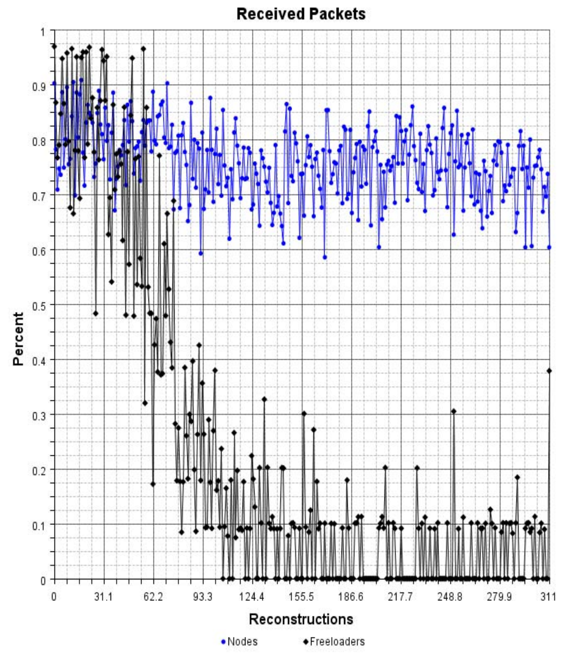


Figure 15

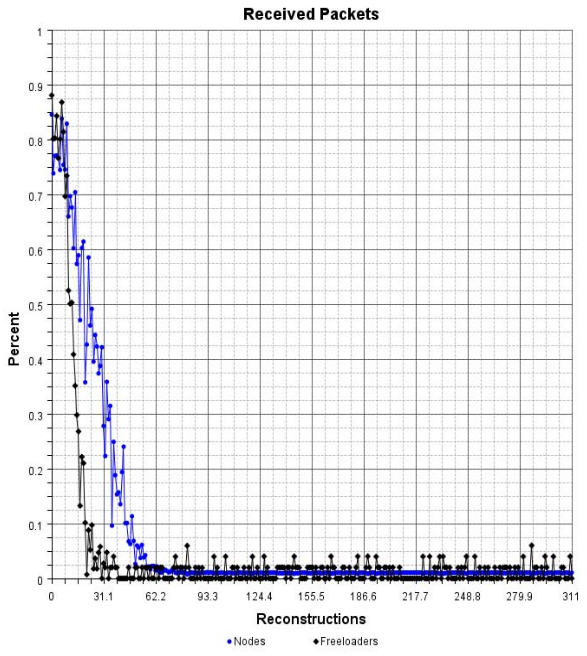


Figure 16

The simulation of Figure 15 is in some senses even more successful than that of Figure 12 because the normal nodes receive more packets. However, in Figure 12 the freeloader nodes are denied service more quickly. In the cases of Figures 15 and 16 there is a similar kind of tradeoff as in Figures 13 and 14. This is the tradeoff between speed of freeloader denial and percentage of packets received by normal nodes. However, this tradeoff is less meaningful in the ancestor rating case because clearly the results of Figure 16 are unacceptable. The normal nodes receive almost no packets within a very short period of time.

We also obtained results for simulations involving a much larger number of nodes, 500 normal and 25 freeloaders. The debt maintenance simulation with this node distribution had its other input parameters identical to those of the simulation of Figure 8. These results are shown in Figure 17. The ancestor rating simulation with this node distribution had its other input parameters identical to those of the simulation of Figure 12. These results are shown in Figure 18.

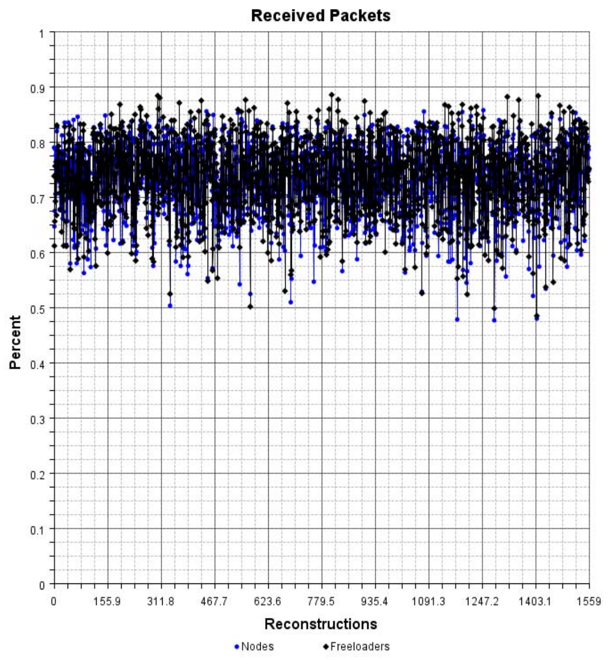


Figure 17

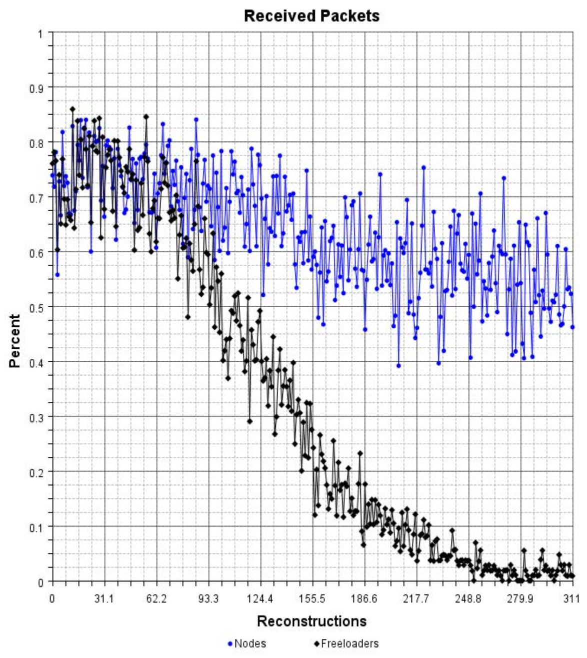


Figure 18

The results of Figure 17 are not very conclusive since there really is no separation whatsoever between normal and freeloader nodes. This is probably due simply to the fact that the network is so large that not enough information about the reputation of nodes has been propagated. The ancestor rating case is more successful, though not as successful as in Figures 12 and 15. Again, this should be due simply to the fact that it is harder to gather all the reputation information for a larger number of nodes.

Complete result files can be downloaded from the project web site http://www.gillius.org/distsys/.

Conclusions and Lessons Learned

We achieved our goal of enforcing fairness into the system by using these different fairness mechanisms described in our research paper 2. From the results obtained from our discrete event simulation we draw our conclusions. First of all, we demonstrated that periodically rebuilding the forest trees helps in identifying freeloaders from the normal nodes. Nodes get a chance to track the behaviors of its peers and if it finds its freeloading on the system it may deny service to them.

By running our various test cases by changing our input parameters we got different kinds of outputs. From these outputs we conclude that the success of detecting freeloaders is highly sensitive to the rate of tree reconstruction and the threshold values.

Also when we compare our results between Debt Maintenance and Ancestor Rating we conclude that if the tree reconstruction rate and the threshold values are optimal, then Ancestor Rating is much more effective than Debt Maintenance. As we can see from the graph when there are no freeloaders in the network, no packet loss, and using ancestor rating all the nodes in the network receive packets. This is because a node's reputation never suffers unlike Debt Maintenance.

A node's outbound bandwidth as one of our input parameter also shows some interesting patterns of outputs. Outbound bandwidth refers to the number of children per node. From our results we imply that if the nodes have more children the freeloaders are detected earlier and are denied service. In this situation, normal nodes receive fewer packets as the freeloaders have more children to which they deny service. Similarly, if the freeloaders have fewer children, then their detection takes time.

Number of nodes present in the network is also influential. If there is large number of nodes in the network then the task of detecting the freeloaders becomes hard and cumbersome. For example, when we ran our simulation using 525 nodes using Ancestor Rating node behavior, it took longer to detect freeloaders in the network. Similarly using the same number of nodes for Debt Maintenance, the simulation time was not long enough to detect any freeloaders.

Future Work

There are certain features which could be added into our simulation project but because of time constraints we could not implement them. First of all, we need to figure out the problems of accuracy of detecting a freeloader and improve its response. Currently, if a node in the network finds out that its immediate parent is a freeloader then it still continues to be its child until the tree reconstruction mechanism takes place.

Also we could implement the publisher taxation algorithm as discussed in our third research paper.

Currently we have implemented three fairness mechanisms in our research project. We can implement other different fairness mechanisms discussed in our second research paper. The other fairness mechanisms which could be implemented as discussed in the papers are “Parental Availability”, “Reciprocal Requests” and prevention of “Sybil Attacks”.

Currently we detect the freeloaders who refuse to forward to its children. There is another behavior where a freeloader can refuse to accept its children in the network. So in future work we could implement such a freeloader and a type of detection for this.

We could also implement the taxation scheme under the fairness mechanism which would detect freeloaders based on a tax schedule.

We could also finish implementation of the Pastry tree manager.

Finally, we can try various experiments using different kinds of input parameters such as changing the packet size, number of stripes, and the simulation duration, and by adding new parameters such as node entry and exit times to research things such as how the network will behave when freeloaders enter and leave during the middle of a simulation.

References

1. Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., and Singh, A. 2003. SplitStream: high-bandwidth multicast in cooperative environments. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM Press, New York, NY, 298-313. DOI= http://doi.acm.org/10.1145/945445.945474

2. T. W. J. Ngan, D. S. Wallach, and P. Druschel. Incentives-Compatible Peer-to-Peer Multicast. In The Second Workshop on the Economics of Peer-to-Peer Systems, July 2004. http://citeseer.ist.psu.edu/ngan04incentivescompatible.html

3. Chu, Y. 2004. A case for taxation in peer-to-peer streaming broadcast. In Proceedings of the ACM SIGCOMM Workshop on Practice and theory of incentives in Networked Systems (September 2004). ACM Press, New York, NY, 205-212. DOI= http://doi.acm.org/10.1145/1016527.1016535

4. Professor Alan Kaminsky’s Computer Science Course Library (CSCL)