Project Overview

Distributed Network Traffic Controller

Revision Number: 1.1
Last date of revision: 5/11/05

22c:198

Johnson, Chadwick Hugh

Motivation

When a limited resource is shared between multiple people, the defining what is considered "fair" is a standard reaction to people who feel cheated. In my undergraduate at Wartburg College, I lived in the dorms with several hundred other students, and we shared 2 T1 lines, totaling 3 Mbps up and down. When a few dozen of those students were using P2P programs, the connection we all shared became so saturated it was unusable. Speed tests conducted showed there were times when a 56kbps dial up connection was about twice as fast as using the college's connection, and was less likely to be timed out.

My senior project was an attempt at giving the college a solution to their growing problem. My idea was to create logical pipes based on their past usage. The idea of a "Fair Share", and placing people on an "Open Pipe" or a "Slow Pipe" came into being. I had a successful project, demonstrating proof of concept, but the update process was very slow due to the flat files I used for data storage. With this design, all traffic was forced to go through a single computer, allowing for any single problem to bring down the Internet connection for everyone.

For a graduate level programming project I turned things up a notch. I wanted to have multiple gateways for the traffic to go through, communicating with a primary Database for quick storage and retrieval between the gateways. I also wanted to have a web-based administrative screen for changing of settings.

Project System Description

The current system has two gateways, each with two network cards. On each gateway one NIC acts as a WAN port, leaving the other to act as a LAN port. Each gateway has a own DHCP server running on the LAN port with a unique range of IP address to hand out. When a user connects to the network for the first time, they request an IP address via DHCP, and will randomly use one of the two gateways. This is an attempt at balancing the load. All traffic that is created for that IP address is logged.

Each gateway is also running an update program in the background that performs all the grunt work for this project. It updates the database with any new information such as new MAC address-IP address combinations and user traffic. Based on this information, it proceeds to place users in their earned pipes. The refresh rate for this update program is administratively set through a web interface.

Problems Faced and Solutions Proposed

Problem 1: Getting all necessary hardware was initially a problem. I needed a minimum of five computers; two as gateways, two for databases, and at least one more computer

for testing. The solution was to purchase the necessary parts needed to complete the computers I already had most of the parts for.

Problem 2: Installing operating systems was not a problem for DB1 and Gateway1. For some reason Gateway2 and DB2 would not install Fedora2. After several hours of trying on each computer, the solution was to try different versions of Linux. I was able to get Fedora3 to work on Gateway2, and Red Hat 7.3 on DB2. All the installs together should have taken about 8 hours total, but instead took closer to 20 due to the time needed for each, often unsuccessful, install.

Problem 3: When I used IPForwarding in my undergraduate using Red Hat 7.3, a kernel recompile was required. After several hours of attempting to recompile the Fedora 2 kernel and still not getting IPForwarding to work, I joked to myself about how it would be funny if the newer kernels didn't even need to be recompiled. Shortly after that "joke" I check, and it turns out I was wasting my time. No kernel recompile was needed, just a single additional command before I enabled IPForwarding.

Problem 4: When I started integrating the database into my project, I noticed that my results were not being stored. This also happened to be at about the same time that 22c:244 (Database System Implementation) covered "commits". For some reason, my Postgres database required I formally commit, otherwise it would rollback. This was likely the result of how I had installed Postgres. Although this only took a small amount of time to figure out, it was an interesting problem.

Problem 5: While I continued to use the system, traffic was created daily, stored in the database, and updated regularly. During this time, I was also creating the administrative pages for system configurations and administrative view of users statistics. As days passed, the screen for user statistics would take longer and longer to load. Finally, while presenting this project to a group of students, the page took an embarrassingly long time, which was the last piece of encouragement I needed for me to find and fix what was causing this performance degradation. I was able to trace the problem to a "WHERE macAddress IS NOTNULL" on a table with about 450 entries. I created a secondary index on non-null values, which boosted performance to acceptable. After telling Dr. Lawrence about this improvement, he also recommended I look into a "Vacuum" option. Vacuum recomputes the statistics on the database, and clears out empty space. This noticeably increased performance to a very satisfactory level when run regularly.

Problem 6: Time is always an issue for project with deadlines. Had the above problems not take up as much time as they did, the backup database may have been implemented, along with more detailed instructions on how to recreate this project.

System Specification and Software Used

Computers:

DB1 – AMD 2800+, 160 GB ATA133 HDD, 512 MB DDR

Gateway1 – AMD Duron 1800, 30 GB ATA133 HDD, 384 MB DDR
DB2 – Intel Celeron 667, 40 GB ATA100 HDD, 384 MB SDRAM
Gateway2 – Intel Celeron 733, 40 GB ATA133 HDD, 128 MB SDRAM
Web Server – P3 550, 10 GB HDD, 128 MB SDRAM
Laptop/Test Computer – AMD Athlon 900, 40 GB HDD, 320 MB SDRAM

Operating Systems:
Red Hat 7.3, Fedora 2, Fedora 3

Software:
Java 1.5
Python 2.3.3
Jakarta (Tomcat) 5.0.28 – Web Pages
Jude 1.4.3 – For ER Diagrams
CBQ – For throttling
Postgres – Database
DHCP – distributing IP addresses

Other:
Several home-made network cables
8-port 10/100 switch
4-port 10/100 switch

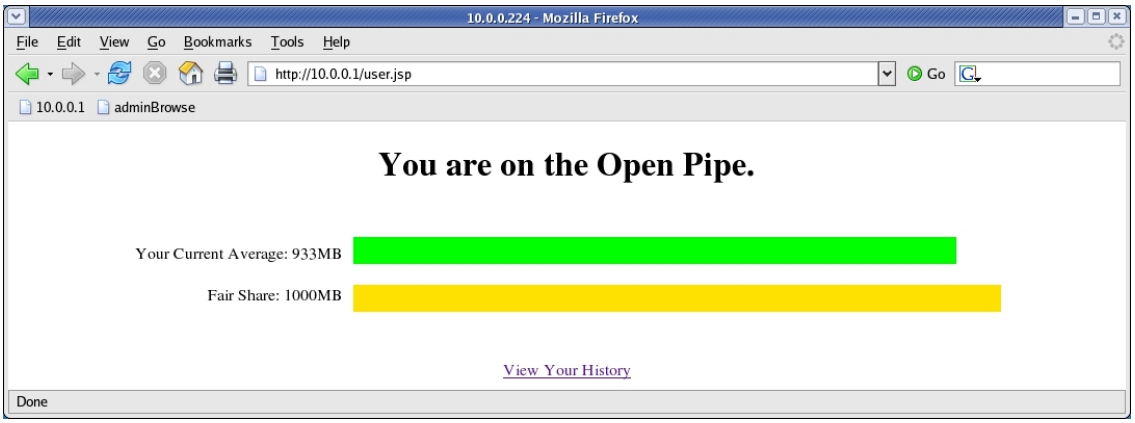
Installation Instructions

1. Get all hardware together. Each gateway will need two NICS, while everything else will only require one. Then put together the computers and network. Note: A general rule for the gateways is 8MHz per expected user for 100 Mbps connections. Testing showed that 450 users on an 1800MHz gateway maxed the connection out at 55 Mbps, while 225 users on the same gateway had 100 Mbps.
2. Install Fedora 2 on each of the computers. I did a full install with no firewalls.
3. Install Java 1.5 on the gateway computers.
4. Install Tomcat on the gateway computers. In the tomcat directory/conf/server.xml, the port should be set to 80 instead of 8080.
5. Setup the relations on the Database computers. Insert values for every possible IP address and gateway computer.
6. Copy over necessary files:
    a. /etc/sysconfig/cbq/cbq-0002.abusers
        i. Maintains all users, but more importantly, specifies flow rates
    b. /usr/local/tomcat/webapp/ROOT/*
        i. /usr/local/tomcat will change depending on where it was installed
        ii. The contents of this directory should be all the JSP pages created.
    c. /usr/local/tomcat/common/lib/postgresql.jar
        i. Allows tomcat to talk to the Postgres Database.
    d. commands.py
        i. contains all the commands for initializing the system, and does regular updates.

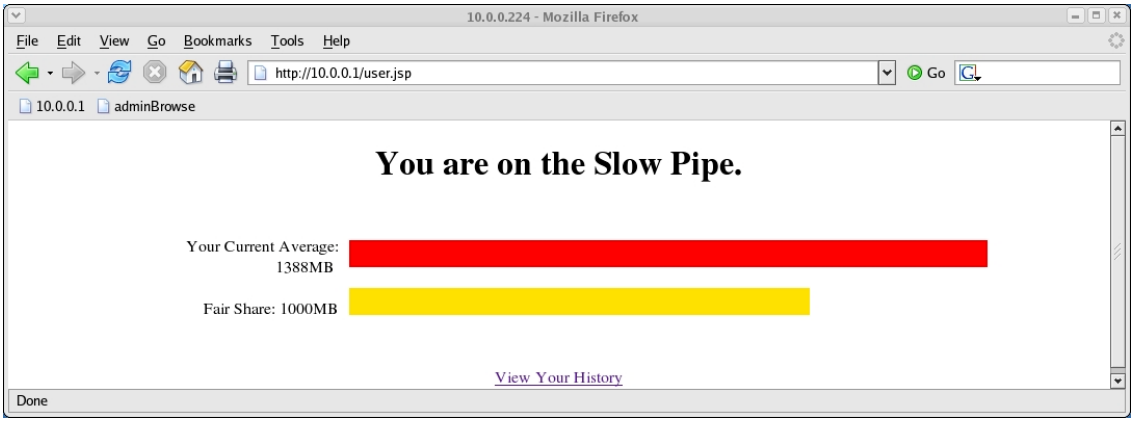
ii. At the top of the file, thisGatewayAddress should be changed to the IP address of that gateway.

7. Open a terminal window, and execute commands.py as su. This terminal window must be left open for python to continue running.

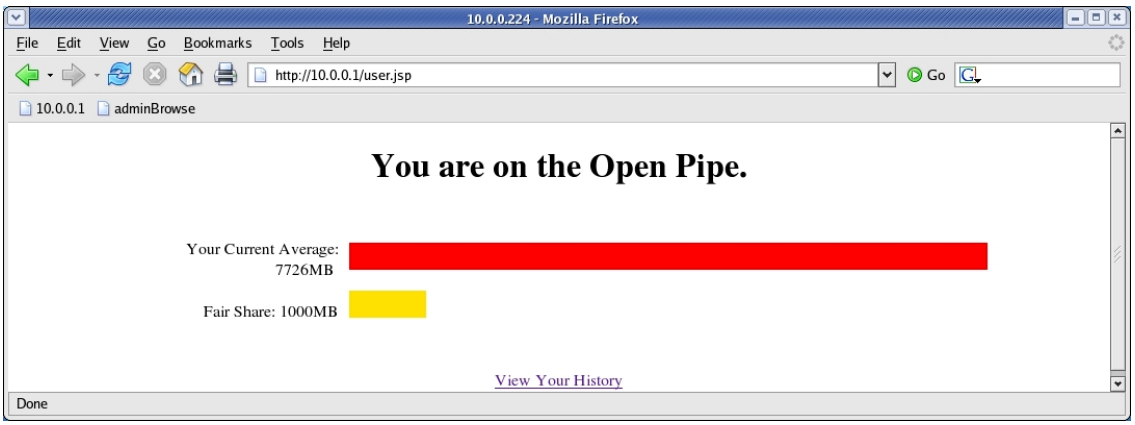
Highlights of the Algorithm and Code



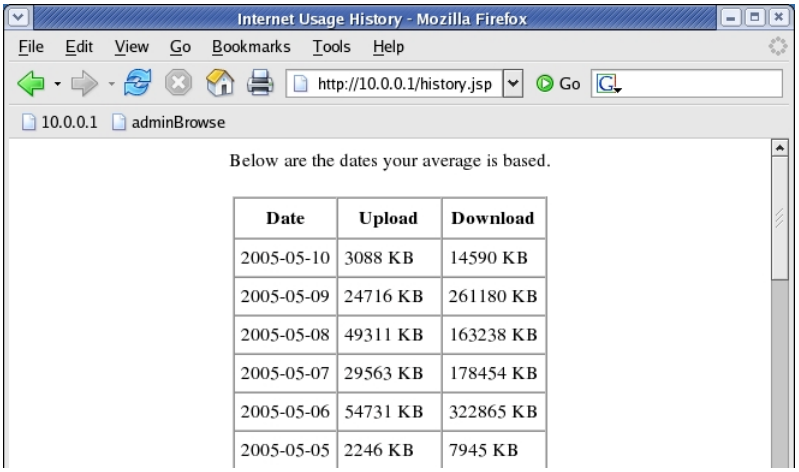
When a user points their web browser at any of the gateways, a screen comes up showing their current average usage, and comparing it to that of a fair share. The information is automatically related only the IP address of the user. The size of the bars are automatically sized to depict the actual ratio.



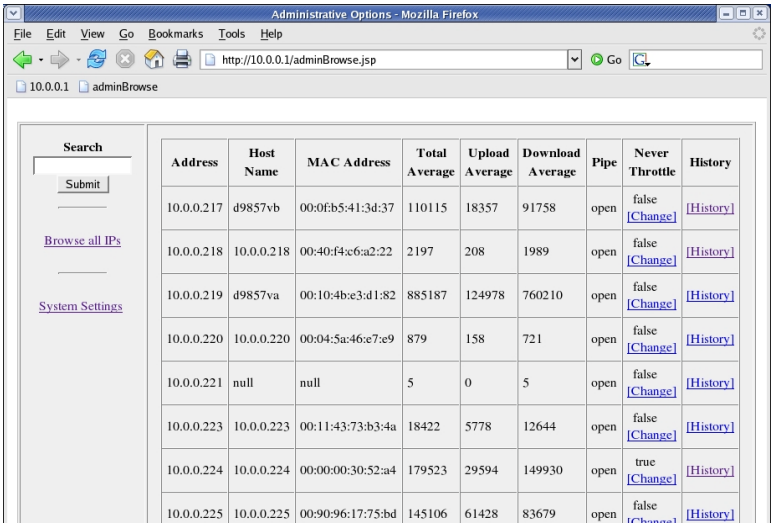
Should the user's average usage go over their "Fair Share" the top bar changes to red, and they are told they are placed on the "Slow Pipe".



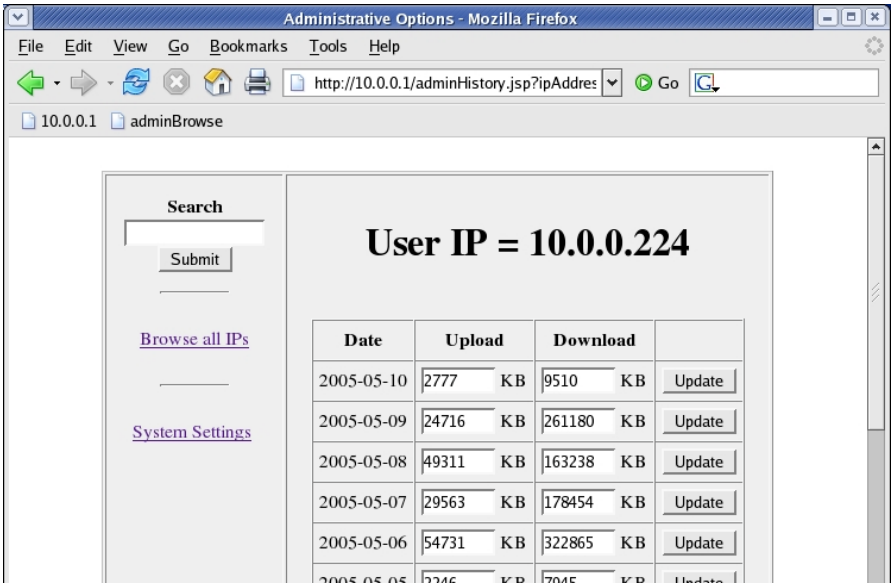
The administrator has the ability to grant individual IP addresses immunity from the slow pipe. Should that happen, the user's average can still be see by the user in the same fashion, and the pipe placement will be specified as Open.



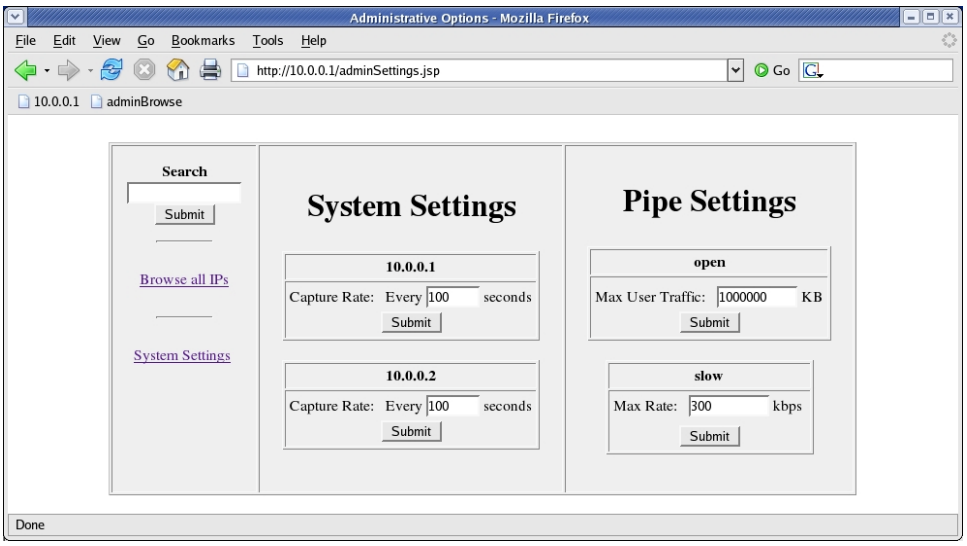
Users also have the ability to view their history by clicking on the history link at the bottom of the user's page. This shows a complete history of daily usage for both Up and Down traffic. These values are what the averages are based.



The administrator has the ability to see every user's information and history. The search field on the left hand side allows for a quick search of all instances of a value inside Address, Host Name, MAC Address, and Pipe, all at the same time. For instance, if I do a search for "Open", get every user on the open pipe. From this screen the administrator can also specify if a user has throttling immunity. There is also a link to the system settings.



When the administrator clicks on the history for any user, a screen displays all values for Upload and Download that user has created, and the ability to adjust them.



When the administrator click on "System Settings", they can view and change the refresh rate for each of the gateways. They can also state what the fair share is for the Open Pipe, and the speed of the Slow Pipe.

Future Work

This project allows for several options for future work. One of which would be the use of a redundant database that would take over incase of a system failure in the main database. This could be implemented in several ways. One option would be to have the main database perform a complete database dump, and the redundant database copy and replace everything it has every time. This would be a simple but costly solution. Another option would be to have the main database keep track of all new transactions, and store those to update the secondary database with. This would likely be complicated but limit needed network traffic. A third option would involve the gateways updating both databases every time. This would be very inefficient on both the gateway side, and the database side, but would effectively keep both databases completely up to date.

The thing I would like to have most done with this project would have been a more detailed "HOWTO", giving full instructions on every step needed to recreate my project, and any insight I may be able to give. With a few more weeks, this would have been an option, but unfortunately this is something I was unable to do.

To expand on my current design, it would be very easy to allow for more than two gateways. This was never tested, but in theory, it should simply be a matter of recreating a third, fourth, etc, gateways, and plugging them in. The only thing that would need to be done would be to redistribute IP addresses for the DHCP.

Now that two pipes have been shown effective, the flexibility to allow for more than an "Open Pipe" and a "Slow Pipe" would be an interesting expansion. Although much of my code required only the two pipes, it should be capable of being modified to allow for levels of pipes, and each pipe with it's own maximum rate and fair share.

For this project to be implemented by die hard security advocates, MAC verification and administrative passwords would need to be implemented. Because this was more of a research project for proof of concept, these details were ignored to allow for more development.

In places where users have the ability to have multiple computers, such as the dorms, it may be desirable to have throttling based on user's usage instead of usage per IP address. This would group IP or MAC addresses together for form average usage.

While designing this project, performance was a consideration at most steps, but the mail goal was completion of a successful project. Some areas of this project have left optimization as a job for later.

Conclusion

This project has been both fun and educational. I enjoyed the self-paced aspect of the project, but what probably acted as the major driving force for me was how results could be seen almost immediately at each stage, giving me something to get excited over. With many other types of projects, interesting results are not seen until towards the end of the project.

On the educational side, I learned that a quiet place to work makes a huge difference in work performance. I also learned that there is a balance between experimenting and researching. If something is going to be done, over-researching can waist time, and take away from the excitement of getting things to work. Simply trying things without researching (under-researching) can sometimes prove to be a waste of time also. I also learned the importance of keeping good logs, giving me the ability to look back quickly at things I had done in the past. This allowed for recreating the functionality in a fraction of the time.

Most importantly, I learned to appreciate databases, and gain experience at interacting with them in different programming languages. The performance increase cause by using the database was more than I had expected, and still lends itself to future optimizations.