CS240B Project Proposal

Hui Dai, Mike Lee, and Steve McGhee

February 16, 2003

1 Overview of topic

Storage clusters allow systems to view many physical storage repositories as one coherent "virtual disk." This abstraction allows administrators to add storage, implement redundancy, and perform repairs to damaged nodes without interfering with the users. Since each data-housing node is connected via standard networking, files must be transferred across these links to the end user. Simply using TCP to transfer files in an FTP fashion is not efficient for smaller files, as the overhead for the three-way handshake might dwarf the data being transferred. Because of this, we propose a system that will allow the reliable, efficient transfer of both large and small files at high speeds and at high network loads. This system can be used to improve the performance of storage clusters and any application that utilizes them.

2 Summary of Project

We plan on developing and running tests to determine what might slow down the transfer of different sized files. If scenarios are found where UDP is faster than TCP, we will attempt to discover if these two protocols can be run in parallel. We plan to address congestion and reliability issues with UDP, to ensure that all files are delivered completely and on time. Once the new protocol is developed, we will implement and test it. We will transfer files of different sizes and compare the transfer rates to those of normal TCP systems. We hope to achieve higher transfer rates than TCP. Once we have completed this initial test, we will test our system with the file system trace from the CS network in order to develop an idea of how it will perform in a real-world scenario. We will use the Sprite file system trace to perform the evaluation. We hope this research will result in conclusions helpful to the development of storage clusters.

3 Summary of Experiments

3.1 Initial testing

1. Under what circumstances is TCP inefficient?

- We will try different TCP payload sizes to determine at which point UDP's efficiency outweighs TCP's reliability.
- We will determine how altering the number of open TCP connections affects traffic flow.

2. Under what circumstances is UDP inefficient?
   - We will try different UDP payload sizes to determine at what point dropped packets outweigh the benefits of bypassing TCP's overhead.
   - We will congest network with different amounts of external traffic to determine how TCP's congestion control improves transmission rates over UDP.
3. How well can TCP and UDP work together? We will run both protocols at the same time to determine if UDP will dominate TCP to the point that it is rendered useless.

3.2 Testing of protocol implementation

Given the initial testing data, we will try to determine how closely our algorithm can approximate the behavior of the better of the two algorithms under given conditions. First, we will run our system with various sized files, ranging from bytes to megabytes. The performance of this will be compared to pure TCP and pure UDP systems. Next, we will run our system with the file system trace provided by Hong. This will give us a good understanding of how our system will work under realistic load.

4 Timeline

Dates	Duration	Description
2/13		The project proposal is due
2/13 - 2/17	5 days	initial testing
2/18 - 2/20	3 days	protocol development
2/21 - 3/04	12 days	protocol implementation
3/05 - 3/09	5 days	analysis and final testing/report writing
3/10		Report is due.
3/12		Demo/presentation.

5 Group Logistics

5.1 Division of Tasks

Initially, tests on existing systems will be done on an individual basis. Each group member will assume a test and report his findings to the group by 2/17. Once these tests are completed, we will meet to develop the protocol, considering the results from previous tests. We will then divide up implementation and

schedule times to merge our code together. We will test the completed system together, collecting the results for the final report.

5.2 Meeting Schedule

We plan on meeting after every lecture period and additionally on weekends when necessary. During these meetings, we will discuss progress and assign tasks.

6 Related Work Survey

6.1 Size-based Adaptive Bandwidth Allocation: Optimizing the Average QoS for Elastic Flows[?]

6.1.1 Introduction

This is an overview of the paper *"Size-based Adaptive Bandwidth Allocation: Optimizing the Average QoS for Elastic Flows,"* by Shanchieh Yang and Gustavo de Veciana at the University of Texas. This overview will summarize the results of that paper and attempt to relate its conclusions to our CS240b project.

6.1.2 The Problem of Perceived Throughput

The paper deals with the unique problem of managing a set of network flows in a dynamic network, such as the Internet as it applies to web browsing.

The paper begins by describing the behavior of current bandwidth allocation schemes and their impact on perceived throughput. While throughput in terms of bytes/second may stay reasonably constant on a network link thoughout a long period of time, another important metric is the time it takes to finish a transfer. Users with many small transfers to complete expect their transfers to complete quickly, especially in an interactive setting such as web browsing. On the other hand, there is almost an expectation that some transfers, like the download of a large multimedia file in the background, will take a long time.

The quality of service metric used throughout the paper to compare different bandwidth allocation schemes is Bit Transmission Delay, or BTD, and is defined as delay/file size.

6.1.3 Optimizing BTD on a Single Link

The paper first deals with managing flows on a single network link. The central point of the paper is presented as the following lemma:

> Consider a set of flows contending for capacity on a single link. Any bandwidth allocation policy that allocates positive bandwidths to more than one flow at a time is not BTD-optimal.

The idea is that smaller flows or flows with less residual work can be allowed to finish first without penalizing larger flows, and this will reduce BTD. A logical conclusion is that bandwidth allocation schemes which consider flows' residual sizes may reduce BTD.

From the above lemma, it is clear that we must choose a single flow at any time and give it the full bandwidth of the link. The paper proposes several ways of choosing such a flow, which take into consideration the remaining work of the flows. The best of these schemes provide performance increases of 50% over fair sharing on a link with two thirds saturation.

6.1.4 BTD on other Networks

The paper also deals with more complex networks, such as a linear series of m hops, each having the same bandwidth. Some flows on this network may span up to m hops, and other flows may only need one hop each. In this case, the algorithm compares the overall difference in BTD between allocating bandwidth for routes of different lengths. For example, if many 1-hop flows will complete within a short time window, then the option to allocate bandwidth for 1-hop flows will have a very high weight.

The paper then introduces a class of bandwidth allocations called Size-based Adaptive Bandwidth Allocations (SABA). They test an implementation of SABA, and present their results.

6.1.5 Analysis

The central idea presented in this paper seems like the common "printer queue scheduling" problem, in which a printer is given several tasks of differing magnitudes, and must chooes the order in which to complete them. One solution is to perform the shortest jobs first, which is in essence what this paper suggests. The difference is that a network link can transmit several different streams at once, or at least simulate that behavior, but the authors show that it is still beneficial to only service one of the flows at a time.

The paper briefly mentions a common problem in such situations: namely that of starvation. Conceivably if there are enough small jobs, then a large job will never be able to finish because the smaller jobs get priority. However, the paper claims that in a real-world network with flows of different weights, this is not a problem. The paper refers the reader to other works but does not discuss it in depth.

The paper gives the quantitative results of several tests that the authors performed. These include real-world performance of their proposed allocation schemes over a single link as well as different multi-link networks and help show that their solutions have a significant real-world advantage over traditional schemes.

6.1.6 Conclusions

There are certain elements of this paper that would be useful for us in our adaptive FTP project. Specifically, the idea that smaller transfers should be given priority is important. However, as the paper focuses on dynamically allocating bandwidth among flows and not dynamically choosing a transport protocol with which to transmit data, it is not entirely applicable.

6.2 Fair Bandwidth Sharing among Adaptive and Non-Adaptive Flows in the Internet[?]

Anjum & Tassiulas tackle the problem of unfairness between adaptive (TCP) flows and non-adaptive (CBR-UDP) flows through a gateway. Several existing gateway algorithms are presented, including *drop-tail*, *drop-front*, Early Random Drop (*ERD*), Random Early Detection (*RED*), and Longest Queue Drop (*LQD*). All of these algorithms attemtp to find the best way to handle packets that arrive at an overloaded gateway, whose buffers are full. While *RED* & *LQD* have their advantages, they do not differentiate between adaptive and non-adaptive flows. This is important because as congestion builds, adaptive flows decrease their sending rate. Non-adaptive flows do not self-regulate and end up dominating the channel. As a result, using non-adaptive flows actually become more attractive to applications than adaptive ones. This could result in the network being dominated by congestion, allowing no work to be done.

The authors present *BRED*, a variation on *RED*, as an algorithm developed to solve this problem. The simulation of a network running *BRED* shows that while non-adaptive flows still dominate, adaptive flows are not completely cut off as they were with previous algorithms. By taking into account which flow every packet was associated with and making judgements in a way that ensures fairness for both adaptive and non-adaptive flows.

While this work is interesting from a network infrastructure point of view, only a few points are relevant to our topic of adaptive file transfer. First of all, we must take into consideration the domination of UDP flows over TCP flows. If we intend to have many flows of both types over the same channel, it is possible that we will shut the TCP flows out entirely. To avoid this, we may have to build some sort of congestion control on top of UDP, or abandon the idea of using both protocols over the same channel and instead develop and use only a reliable UDP, for example. Another interesting point to note is that front-dropping was initially proposed because it would yield faster detection of packet drops by TCP, but as a side-effect, it deteriorates non-TCP flows. If we choose UDP-only flows, we might consider the possibility of being harmed by this.

6.3 Smooth Loss With Variability Smoothing Factor to Increase Bandwidth Utilization[?]

This paper describes a method for applying a smoothing factor to adaptive transmission control which results in higher transmission speeds. When a loss rate of a transmission is recorded, it is filtered through an equation to determine the "smoothing value." Next, the congestion rate of the network is determined. This is compared to the smoothing value to determine if the network is currently in a *CONGESTED*, *LOADED*, or *UNLOADED* state. Once this is determined, the transmission rate is adjusted accordingly. By using a "smoothed" value to compare against, instead of using the raw loss report, transmission rates are improved.

While the title and abstract to this paper were promising, its contents were disappointing. Despite its short length, there were numerous grammatical errors which made the reading difficult. This aside, the idea submitted is interesting and might help us to find an ideal transmission rate when transferring files. Hopefully we can utilize the smoothing equation to give us better transmission rates.

6.4 RFC 1350[?]

This RFC detailed the TFTP protocol, a very simple and limited FTP protocol that, for matching the simplicity of its design, is implemented over UDP instead of TCP, although the latter is possible. This protocol adds very little overhead to that of UDP: there are only 5 operations supported, read request, write request, data send, acknowledgement, and error. Data sizes range from 0 to 512 bytes, and 3 different data modes can be specified. The state information on the file transfers are kept through ack packets, and time-outs are used to determine if a packet should be retransmitted in the case that an ack packet is lost.

Since 2 bytes are used for the block number to keep track of order for reassembly at the client side, a maximum size of $2\hat{1}6 \;\; 512 \;\; 1$, or 33554431 bytes, or almost 32 MB of data may be transmitted using an unchanged model of this protocol. This is suitable for our purposes, since for larger files, TCP is desirable due to many of its reliability features. However, given a reliable network with low dataloss rate, a modified TFTP protocol over UDP may still be preferable to that of TCP for higher throughput.

The virtual circuit information that is available to the TFTP client and server are the port numbers, which are chosen randomly. There is no scheme within TFTP itself for the detection of bit errors, although duplicate packets, lost packets can be dealt with the block number and the ack packets. The header does not require 2 bytes, but for alignment purposes, along with the block number, it is more convenient.

The protocol does not provide for user authentication, which does not adversely affect our project, which aims to develop an adaptive but necessarily secure protocol. Another possible drawback of this protocol is that it does not provide for the listing of directories, but for efficiency testing purposes, this is

also not a problem. The biggest problem of this protocol comes from the fact that the window size is limited to one packet: each sent packet must be acknowledged before the next packet is sent, and time-outs are used to determine resends. While this keeps the overhead very small: only one packet needs to be cached on the server side for retransmission, one can only imagine what this, coupled with a small packet size, does to the throughput of the transfer.

Although the efficiency of this protocol is in question, it does propose simple schemes for somewhat application based reliability control on top of UDP, the alternative protocol to be examined in our project. Some ideas, such as simplicity of design over functionality. The retransmission scheme is similar to that of TCP.

6.5 Real Time Video and Audio in the World Wide Web[?]

This paper introduces VDP, presumably Video Datagram Protocol, a protocol used to send and receive continuous media over the internet. While this does not apply to our project directly, as the FTP server deals with files currently existent in the file system, which means the transfers are indeed full file transfers, it does examine bandwidth usage optimization issues.

VDP itself is not necessarily an adaptive protocol, but it is resource sensitive: it would drop packets based on current network congestion and CPU load. And I would explain the difference between to the two to be whether a protocol uses past trends in determining future actions, i.e. whether certain TCP/UDP connection ratio has done well in the past in certain times of the day, of the week, under certain load conditions. "The server must ensure that it has enough network bandwitdth and processing power to maintain service qualities to current requests. The criteria used to evaluate requests may be based on the reqeuested bandwidth, server available bandwidth, and system CPU load" Dropping packets is not an option for our system, since it equates to retransmission somewhere down the line, and that fault in files are not tolerable. However, our server should definitely base its decisions on network and CPU utilization, as well as past performance evaluations.

One interesting aspect of the VDP protocol is that it uses two channels, one is an unreliabled data transmission channel and the other is a reliable control information transmission channel. In a streaming media environment, packet drops are tolerated, and control information must be received to correct the stream for future sends. It is possible to adopt this in our protocol. Using UDP to send data, and TCP to relay control info and acknowledgements back, this might do better than using timeouts in the TFTP for retransmission purposes, and might do better than TCP for data transmission, even for large files.

;