

Media Uploading Systems with Hard Deadlines

Mu Zhang

Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA.
Email: zhangmu@cs.iastate.edu

Johnny Wong

Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA.
Email: wong@cs.iastate.edu

Wallapak Tavanapong *

Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA.
Email: tavanapo@cs.iastate.edu

JungHwan Oh

Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX 76019-0015, USA.
E-mail: oh@cse.uta.edu

Piet de Groen

Division of Gastroenterology and Hepatology
Mayo Clinic and Foundation
Rochester, MN 55905, USA

ABSTRACT

This paper introduces a new uploading system that automatically uploads multimedia files to a centralized server given client hard deadlines. Unlike systems supporting downloading applications such as video streaming and file sharing, systems that support uploading applications have received relatively little attention in the past decade. While not popularly studied, uploading systems with hard deadlines have several important applications in practice. For instance, such systems can be used in hospitals to gather videos generated from medical devices from various operating rooms for post-procedure analysis and in law enforcement to collect video recordings from police cars during routine patrolling. In this paper, we study an uploading system with hard deadlines in detail. We define a suitable performance measure for the system and evaluate the system performance via analysis and simulations.

KEYWORDS

Upload, Performance Modeling, Hard Real-time Systems

1 Introduction

Recent years have seen numerous designs and performance evaluations of downloading applications for multimedia files such as video streaming and file sharing applications. Few investigations have focused on systems that automatically upload multimedia files from a large number of clients to a centralized server given client hard deadlines. These deadlines are imposed by limited capability at client machines such as limited storage space and network bandwidth.

In this paper, we focus on deadlines imposed by client storage capacity. The client has software that repeatedly generates multimedia files on a local disk drive. When the disk is full, either a new video is not recorded or some

recorded videos are overwritten depending on the application that records the videos. In either case, the results are unacceptable. For instance, the uploading system can be used to collect video files generated from medical devices located in operating rooms for post-analysis. Each operating room has a client machine that captures a video signal from a medical device (say an endoscopy unit) into a file stored on a local disk. Within one operating room, several medical procedures are performed one after another (with a short break in between) for several hours per day, which results in several video files being created. These videos are later uploaded to a centralized server for automatic analysis and documentation of abnormality patterns. The uploading system is also useful for gathering video recordings in entirely different circumstances such as in police cars during routine patrolling [1] and for collecting voice recordings of conversations between customers and sale representatives for quality control purposes.

Unlike video streaming[8], continuous playback is not a required specification for the uploading system. However, the system must ensure that multimedia files are uploaded by hard deadlines as clients repeatedly create more files. The deadline requirement is not presented in existing systems such as video streaming, distributed backup systems (e.g., Amanda [6]), distributed file systems (e.g., NFS [9] and Andrew File System [7]), and file sharing applications (e.g., Kazaa [4], Gnutella [3], or Napster [2]). In our opinion, it is desirable that the client uploading software is able to work with different applications that generate multimedia files on different platforms or operating systems. Hence, the client software should not rely on low-level disk operations such as disk scheduling and disk block deallocation.

Several client-server models such as *server-pull*, *client push*, or *client-push-server-pull* can be considered to design an uploading system. In this paper, we investigate the server-pull paradigm since it allows the server to control critical activities in the system, such as scheduling file uploads and handling emergency situations. Our goal is to

*This work is partially supported by the National Science Foundation under Grant No. 0092914.

develop an uploading system with novel scheduling techniques that uses available resources effectively to prevent a given client from missing its deadline.

In this paper, we present the design of the uploading system that supports two uploading policies. Next we present performance analysis and assessment of each uploading policy. In addition, we discuss a system sizing tool that provides the lower bound of the maximum number of clients a specific client-server system can support such that no clients in the system miss uploading deadlines. The sizing tool can be used to determine the appropriate configuration of the uploading system when the approximate number of uploading clients is known.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the design of our uploading system. In Section 4, we present the performance analysis of the uploading system and discuss the system sizing tool. We validate our analysis with simulation results as well as demonstrate the performance comparison between the two uploading policies via simulations in Section 5. Our concluding remarks are provided in Section 6.

2 Related Work

Bistro [5] is a file uploading system for Wide Area Network upload applications such as submissions of income tax forms, of projects and homework in distance learning, and of grant proposals to a funding agency [11]. Bistro aims to support a large number of clients that typically upload small files. The clients do not have the deadlines imposed by their limited capacity. Bistro uploads submitted files in two steps. First, the client software uploads the submitted file to an intermediate node (called a *bistro node*) by a deadline. For example, the submitted federal income tax forms must be uploaded to the bistro nodes by the deadline determined by the government. Once the file is in a bistro node, the user is informed of the successful submission. Their algorithms focus on deciding which bistro node the client should upload its file, assuming that a sufficient number of bistro nodes are available. In the second step, the bistro nodes collaborate to upload the files to a server through some other bistro nodes along the network route towards the server. No deadlines are imposed in this step. To the best of our knowledge, Bistro system is the only work that investigates uploading applications in detail. Nevertheless, the research problem addressed in Bistro is different from the problem studied in this paper.

3 Design of Media Uploading Systems

We describe the designs of the server and client software. For simplicity, we assume that the server has very large storage space. This is possible given today's hard-drive hot swap technologies, which makes it possible to replace a full hard drive on the fly. We assume a wired Intranet environ-

ment and do not take advantage of any particular network topology in this paper. We introduce two important design concepts as follows.

- *Critical stage of a client* is the event in which the available disk space at the client is below a pre-defined *client critical threshold*. For example, for the client critical threshold of 100 MB, we say that the client has entered its critical stage at the time when its available disk space is below 100 MB.
- *Critical stage of a system* is the event in which the percentage of the clients already in their critical stage is above a pre-defined *system critical ratio*. For example, with the system critical ratio of 80%, the entire system is in its critical stage if more than 80% of the clients have entered their critical stage. A good uploading system should prevent the system from entering the critical stage.

3.1 Server Design

Figure 1 shows the software architecture and important modules of our multi-threaded server. The *initial thread* is in charge of establishing connections with clients. The *uploading thread* repeats the following steps.

- 1. Query:** The uploading thread uses its Query Handler to collect client status. Query Handler sends a query request to each client. The client returns its current status that consists of the amount of used disk space and the estimated rate that files are generated at the client. The updated client status is maintained by Client Information Manager.
- 2. Retrieval:** Based on the current system condition derived from the received client information, the uploading thread uses one of the uploading scheduling algorithms described in the next subsection to choose a client to retrieve a file from. Uploader sends the uploading request to this client and the client transfers a file to the server. However, if the received client information indicates that any client has entered its critical stage the uploading thread executes Emergency Handler. Several designs can be considered for handling the emergency situation. For example, a file can be retrieved from the client who is about to enter its critical stage and place in another client that is least likely to enter its critical stage. Due to limited space, we report different emergency handling techniques in a separate paper.

3.1.1 Scheduling Algorithms

We investigate Round Robin scheduling and introduce Vulnerability Based (VBS) scheduling.

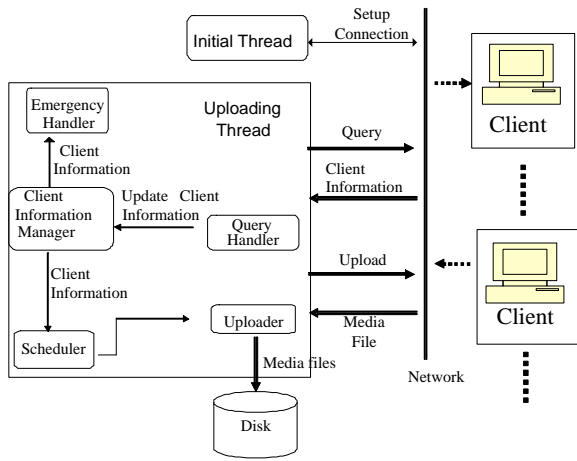


Figure 1. Server architecture

- In Round Robin scheduling, the server collects media files from the clients one client after another. The same client is not requested to upload its file again until all the clients in the system have been visited. The server simply skips the client that has no files to upload. This scheduling algorithm is simple and fair to all the clients, but does not take deadlines into account. This algorithm is used as the baseline algorithm in our performance comparison.
- VBS employs the concept of the Earliest Deadline First scheduling (EDF) [10]. The deadline in our system is determined by the vulnerability of the client. Specifically, how soon this client will enter its critical stage. The scheduler calculates the vulnerability of every client after the upload of one file. The client with the most vulnerability at the time (i.e. it has the shortest time left before entering the critical stage) is chosen. The file with the highest priority (determined by the capturing application of this client) is chosen to be uploaded next. This scheme gives the client with the highest risk of entering its critical stage the highest priority to upload its file first, which will prolong the entire system from entering its critical stage.

3.2 Client Design

The client software is configured with the name of the directory to store the generated multimedia files and the maximum amount of storage space allocated for this purpose. Figure 2 shows our multi-threaded client with two important threads.

- **Disk monitoring thread** monitors the availability of the client disk space in the desired file system. If the client enters a critical stage while the server is busy uploading a file from another client, the disk monitoring thread contacts the server to handle the emergency situation.

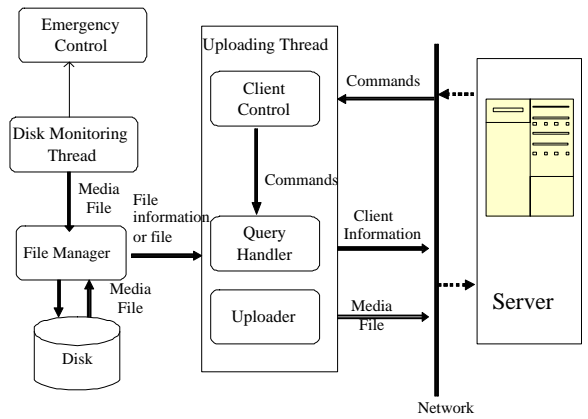


Figure 2. Client architecture

- **Uploading thread** uses Client Control to respond to the server request. In response to the server query request, the thread executes Query Handler that sends a message to the server indicating the available disk space, the file generation speed, and the size of the next file to be delivered. When the server requests the client to upload a file, the thread executes Uploader to transmit one media file. Once the entire file has been uploaded, the thread removes the file from the disk. The available space can be used to store a new file. Currently, the uploading thread uses TCP to transmit data. To improve security, other secure transport protocols can be used.

4 Performance Analysis of Media Uploading Systems

We describe the analysis of the uploading system using the Round Robin scheduling in this section. Due to the dynamic nature of VBS, it is not possible to quantify the performance difference between Round Robin and VBS via analysis. However, we can prove that VBS outperforms Round Robin scheduling. The idea of the proof is to show that any uploading schedule handled by Round-Robin can be handled by VBS, but not vice versa. Due to limited space, interested readers are referred to the proof in [12]. We assume that each client generates a file at the same constant rate and has the same storage space. The system sizing tool based on Round Robin scheduling algorithm is presented. Table 1 summarizes the notations used in the analysis.

4.1 Performance Analysis

To estimate when a client will enter its critical stage, we need to model the amount of data stored at the client disk at any time t since the system starts. This amount ($\Delta S(t)$) is the difference between the amount of data generated by the file generation application at time t ($S_i(t)$) and the

Symb.	Explanation	Unit
N	Number of clients the system can support	
gr	File generation rate	Mbps
wt	Client working duration per day	Sec.
bw	Effective network bandwidth	Mbps
fs	Average file size	Mbit
d_{gen}	Average time taken to generate one file	Sec.
d_{rnd}	Average time taken between two consecutive uploads by the same client	Sec.
d_{xf}	Average time taken to transfer one file	Sec.
$S_i(t)$	Cumulative amount of data generated at a client since the system starts until time t	Mbit
$S_o(t)$	Cumulative amount of data a client has uploaded since the system starts until time t	Mbit
S	Total storage space at a client	Mbit
S_{th}	Client critical threshold (see Section 3)	
T	The time when the system enters its critical stage	
Δt	Time period a client has to wait before beginning its first upload	Sec.
GR	Maximum file generation rate at a client	Mbps
UR	Smallest file uploading rate for one client (including the server processing time for other clients' requests)	Mbps

Table 1. Notations

amount of data removed from the client disk once the data has been uploaded at time t ($S_o(t)$). Figure 3 depicts the value of $\Delta S(t)$ over time.

$$\Delta S(t) = S_i(t) - S_o(t), \text{ where} \quad (1)$$

$$S_i(t) = gr \times \left[\frac{t \times wt}{60 \times 60 \times 24} \right] + gr \times \left[\min\left(60 \times 60 \times \left(\frac{t}{60 \times 60} \% 24\right), wt\right) \right] \quad (2)$$

$$S_o(t) = bw \times \left[\frac{t - \Delta t}{d_{rnd}} \times d_{xf} + \min((t - \Delta t) \% d_{rnd}, d_{xf}) \right] \quad (3)$$

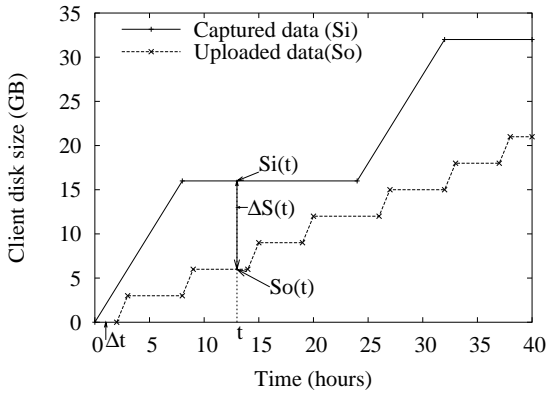


Figure 3. Amount of data in a client disk at time t

In other words, at time t seconds since the system has started, the file generation application has been generating data for $\frac{t}{60 \times 60 \times 24}$ days and $\min(60 \times 60 \times (\frac{t}{60 \times 60} \% 24), wt)$ seconds in the last day. For each day, the data are generated for wt seconds

at the rate of gr Mbps. The client cannot start its upload until its first file has been generated, which is assumed to be Δt seconds since the system has started. Within $t - \Delta t$, the client has uploaded $\frac{t - \Delta t}{d_{rnd}}$ files and some $\min((t - \Delta t) \% d_{rnd}, d_{xf})$ fraction of a file. On average, the amount of the data in each file is $bw \times d_{xf}$ Mbps. We simplify Equations (2) and (3) as

$$S_i(t) = gr \times \frac{t \times wt}{60 \times 60 \times 24} \quad (4)$$

$$S_o(t) = bw \times d_{xf} \times \frac{t - \Delta t}{d_{rnd}} \quad (5)$$

We estimate that $\Delta t \in [d_{gen}, d_{gen} + d_{rnd}]$ since for the first client scheduled for uploading, Δt is as short as the time it takes for the client to finish generating one file (d_{gen}), but for the last client scheduled for uploading, Δt is as long as the time taken to generate one file and the wait time for all the other clients to upload. We estimate the average Δt as $\Delta t = d_{gen} + \frac{d_{rnd}}{2}$. We estimate d_{gen} as the expected duration to generate one file.

$$d_{xf} = \frac{fs}{bw} = \frac{d_{gen} \times gr}{bw} \quad (6)$$

$$\begin{aligned} d_{rnd} &= d_{xf} \times (N - 1) + d_{xf} \\ &= N \times d_{xf} \\ &= N \times \frac{d_{gen} \times gr}{bw} \end{aligned} \quad (7)$$

After we substitute the value of Δt and Equation (7) into Equation (5), we have Equation (8).

$$\begin{aligned} S_o(t) &= \frac{t - \Delta t}{N \times d_{xf}} \times d_{xf} \times bw \\ &= \left[t - d_{gen} - N \times d_{gen} \times \frac{gr}{2 \times bw} \right] \times \frac{bw}{N} \end{aligned} \quad (8)$$

Suppose the client enters its critical stage at time T , we have $\Delta S(T) \geq S - S_{th}$. From Equations (1), (4) and (8), we have

$$\begin{aligned} \Delta S(T) &= \frac{T \times wt \times gr}{60 \times 60 \times 24} - \frac{(T - d_{gen} - \frac{d_{gen} \times gr \times N}{2 \times bw}) \times bw}{N} \\ &\geq S - S_{th} \end{aligned}$$

After some arithmetic manipulations, we have

$$T \geq \frac{S - d_{gen} \times \frac{bw}{N} - d_{gen} \times \frac{gr}{2} - S_{th}}{\frac{wt \times gr}{60 \times 60 \times 24} - \frac{bw}{N}} \quad (9)$$

4.2 System Sizing Tool

We provide the system sizing tool using the Round Robin scheduling. The tool determines the lower bound of the maximum number of clients that the system can support given certain server and client configurations such that no clients in the system enter their critical stage. If the file generation application generates data at the rate faster than the rate the uploading software can upload the data from this client, this client will eventually enter its critical stage. Suppose the system can support N clients. To prevent any client from entering its critical stage, we need to keep $GR \leq UR$, where $GR = \frac{gr \times wt}{24 \times 60 \times 60}$. That is, the client generates video data continuously during its working

hours. The smallest file uploading rate is the case where all the clients concurrently upload their data; each client has the uploading rate of $\frac{bw}{N}$. Hence, we have the following result.

$$UR \geq GR = \frac{gr \times wt}{24 \times 60 \times 60} \quad (10)$$

Substituting UR with $\frac{bw}{N}$ in Equation (10) and performing some arithmetic manipulations we have the maximum number of clients that the server can support as follows.

$$N \leq \frac{bw \times 24 \times 60 \times 60}{gr \times wt} \quad (11)$$

5 Performance Evaluation

First, we validate the analytical results with the simulation results for the uploading system with Round Robin scheduling. Next, we present the performance comparison between Round Robin and VBS. We measure a *non-critical period* defined as the total time since the simulation starts until the system enters the critical stage. A good scheduling algorithm should prolong the system from entering its critical stage. Note that other performance metrics for downloading applications such as service delays and system throughput do not apply to the uploading system.

5.1 Simulation Model

We implemented the simulator in C. We simulated one server with unlimited disk capacity and varied the number of clients, the network bandwidth, the client disk capacity, and the client working hours. To clearly observe the effectiveness of the two scheduling algorithms, we did not simulate any emergency control mechanisms in this study. The system enters its critical stage when a client enters its critical stage. The simulation is terminated after the system enters its critical stage. We generate a file size as a function of a file duration according to a Zipf distribution. We performed simulations on two configurations.

Homogeneous configuration: All clients have the same disk space and same client critical threshold. Files are generated at each client at the same rate. Between consecutive files generated by the same client, a random idle time between 0 and 5 minutes is introduced to reflect a practical scenario.

Heterogeneous configuration: Clients generate the files at the rate of 0.5, 1.0, 1.5, and 2 MB/s. We simulate the same number of clients for the different file generation rates. Besides these changes, the clients behave as in the homogeneous configuration.

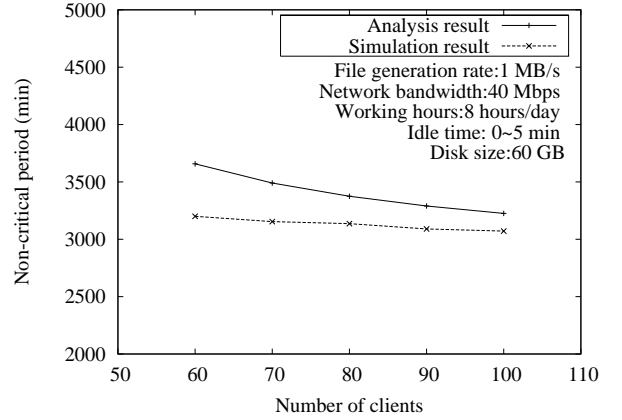


Figure 4. Analytical and simulation results for Round Robin scheduling

5.2 Validation of the Analysis

The homogeneous configuration is used to compare the analytical and simulation results. The expected file generation duration for the Zipf distribution is used in the analysis since the simulations generate file durations using the Zipf distribution. Figure 4 shows the closeness of the analytical and simulation results. The analytical results yield higher non-critical period because the analysis is simplified by not taking into account of the amount of the data generated in the last day (i.e., Equations (2) and (3) are simplified to Equations (4) and (5), respectively).

5.3 Simulation Results

Due to limited space, we only show the simulation results for the heterogeneous configuration. Each data point is an average of non-critical periods of five simulation runs. Figure 5 clearly shows that VBS consistently outperforms Round-Robin under various workloads. The uploading system using VBS enters its critical stage later than the system using Round Robin. The improvement of VBS over Round Robin is between 6% and 88%. The non-critical period for both scheduling algorithms becomes smaller with more clients as shown in Figure 5(a). Figure 5(b) shows that as the network bandwidth increases, with either scheduling algorithm, the system enters its critical stage later as the time to upload a file is shorter. When the network bandwidth is over 60 Mbps, the difference in performance between VBS and Round Robin becomes very significant. The similar trend is observed when the client disk space increases as depicted in Figure 5(c). Figure 5(d) shows that the increase in the working hours per day results in a smaller non-critical period since more data are generated.

6 Concluding Remarks

We have presented a new media uploading system that allows collection of large media files given deadline con-

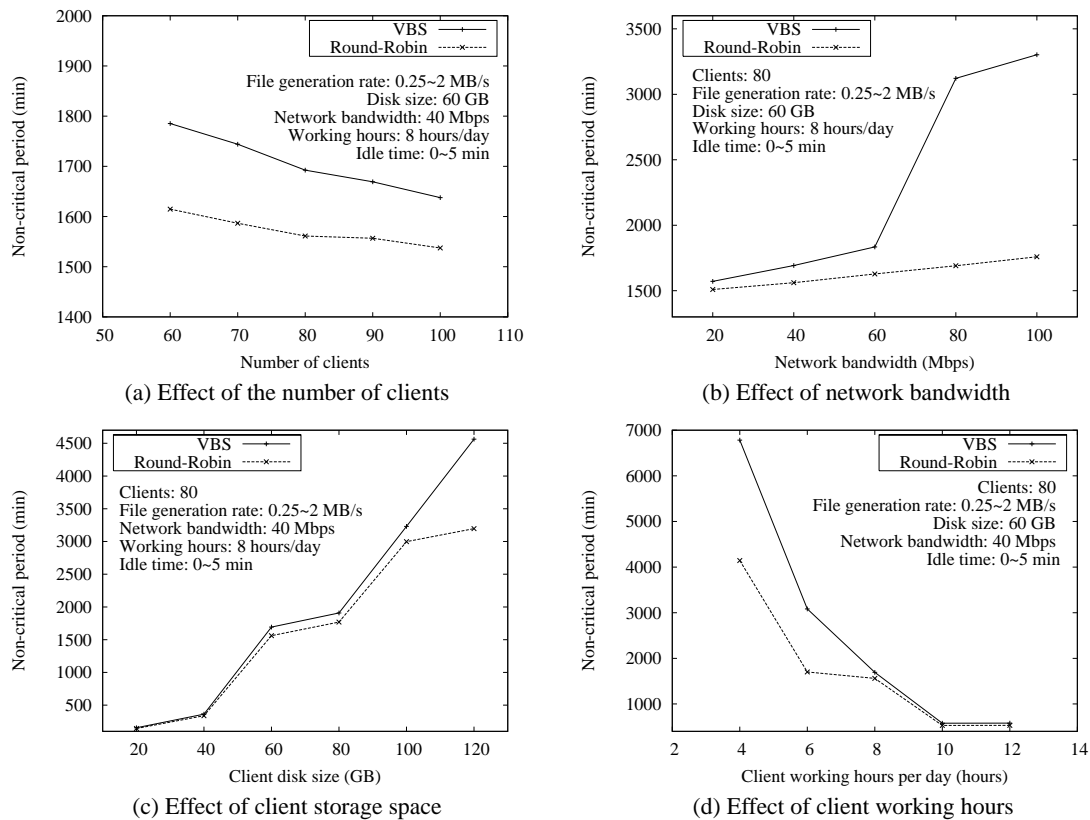


Figure 5. Round Robin vs VBS

straints. Such a system has not been previously studied. We introduce the vulnerability-based scheduling that uploads a file from the client who is most likely to enter its critical stage first. Our simulation results show that VBS consistently outperforms Round Robin. The improvement is as much as 88% in some cases. We provide the system sizing tool that can be used to determine the number of clients the uploading system can support such that no clients enter its critical stage. Our future work includes solutions that provide security and privacy to protect multimedia data. We plan to extend the uploading system for applications in other network environments such as uploading surveillance videos from police vehicles in wireless ad hoc networks where the vehicle may move out of the server transmission range.

References

- [1] IBM Bolsters Police Fleet with "In Car" Digital Video Technology. In <http://www-1.ibm.com/solutions/digitalmedia/doc/content/news/pressrelease/942113122.html>.
- [2] Napster Inc. The napster homepage. In <http://www.napster.com/>, 2001.
- [3] Open Source Community. Gnutella. In <http://gnutella.wego.com/>, 2001.
- [4] KaZaA file sharing network. In <http://www.kazaa.com/>, 2002.
- [5] B. Bhattacharjee, W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. Bistro: a Platform for Building Scalable Wide-Area Upload Applications. *Performance Evaluation Review*, 28(2):29–35, September 2001.
- [6] J. da Silva and O. Guomundsson. The amanda network backup manager. In *Proceedings of USENIX Systems Administration (LISA VII) Conference*, pages 171–182, November 1993.
- [7] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.
- [8] K. A. Hua, M. A. Tantaoui, and W. Tavanapong. Video delivery technologies for large-scale deployment of multimedia applications. *To appear in Proceedings of the IEEE*, 2004.
- [9] S. M. Inc. NFS: network file system protocol specification. rfc-1094. March 1989.
- [10] C. L. Liu and J. W. Layland. Scheduling algorithm for multiprogramming in a hard real time environment. *Journal of ACM*, 20:46–61, January 1973.
- [11] U.S. National Science Foundation. Fastlane. In <https://www.fastlane.nfs.gov>.
- [12] M. Zhang. Media uploading systems. In *M.S. Thesis Department of Computer Science Iowa State University*, May 2004.