The Tabbycat Video Server: From the Protocol to the Implementation[1]

Jehan-François Pâris[2]

Department of Computer Science
University of Houston
Houston, TX 77204-3010

Tel. (1) 713-743-3341 Fax (1) 713-743-3335 Email paris@cs.uh.edu

ABSTRACT

Tabbycat is a video server prototype using a proactive approach to distribute popular videos on demand. Rather than answering individual customer requests, Tabbycat broadcasts the contents of its videos according to a deterministic schedule. As a result, a Tabbycat server consisting of a single workstation with a single ATA disk drive and a Fast Ethernet interface can distribute up to three two-hour videos while achieving a maximum customer waiting time of less than four minutes.

We review the major steps of the development of Tabbycat from the search for the most efficient broadcasting protocol to the actual implementation of the server. Finally, we indicate some direction for future research.

I. INTRODUCTION

One day, we will be able to sit in front of our television set, grab our remote control, order a specific program, and start watching it within a few minutes, if not seconds. The service that will make this dream possible already exists. It is called video-on-demand (VOD) and can already be found in many hotels.

Given the attractiveness of the concept, one might wonder why VOD services are not already more widely available. The reason is simple: it is due to the high costs of providing these services. These high costs result mostly from the high bandwidth requirements of VOD services. Assuming that the videos are in MPEG-2 format, each user request will require the delivery of around 5 Megabits of data per second. Hence a video server allocating a separate stream of data to each request would need an aggregate bandwidth of 5 Gigabits per second to accommodate 1,000 concurrent users. Handling such bandwidths requires costly upgrades of the existing communication infrastructure and equally costly servers to handle all the necessary I/O traffic. As a result, VOD services are still too costly to compete with either video rentals or pay-per-view television.

We built the Tabbycat video server to show that simpler, cheaper alternative was possible. Instead of assigning a separate data stream to each customer request, Tabbycat broadcasts each video according to a deterministic schedule guaranteeing that customers will never have to wait more than few minutes for the video of their choice. Hence, is the ideal solution for services offering ten twenty "hot" videos to a large customer base. All studies of video and movie popularity indicate that these top ten or twenty videos would be the most profitable to distribute [5]. Even videocassette rental stores focus their efforts on having on hand enough videocassettes of the top videos of the day.

Our first design decision was the selection of broadcasting protocol. It had to satisfy two major requirements. First, it had to be efficient and require the minimum amount of bandwidth guarantee a given maximum customer waiting time. Second, it had to be straightforward to implement. We quickly found that none of the existing protocols satisfied both criteria. Juhn and Tseng's *Harmonic*

---

[1] Parts of this paper were previously presented at the 2003 IEEE International Conference on Communications.

[2] Supported in part by the National Science Foundation under grant CCR-9988390.

J. Díaz de León, G. González, J. Figueroa (Eds.): Avances en: Ciencias de la Computación, pp. 20-27, 2003.

Broadcasting [8] protocol was a strong candidate because it required much less bandwidth than any other broadcasting protocol. Unfortunately it suffered from two major flaws. First, it failed to guarantee on-time delivery of video data under all circumstances. Second, it allocated a multitude of very low bandwidth data streams to each video, which made it much harder to implement than its rivals.

The solution we proposed was a new class of broadcasting protocols that allocate to each video a small number of fixed bandwidth video channels and use time-division multiplexing to ensure that all video contents are broadcast at the appropriate bandwidth [12]. As a result, these so-called *Pagoda Broadcasting* protocols achieve bandwidth requirements comparable to those of the Harmonic Broadcasting protocol while being much easier to implement.

The remainder of this paper is organized as follows. Section 2 relates our search for the most efficient broadcasting protocol. Section 3 discusses our Tabbycat server prototype. Section 4 mentions a few possible extensions and Section 5 reviews other video servers. Finally, Section 6 has our conclusions.

II. BROADCASTING PROTOCOLS

Broadcasting protocols offer the best solution for the successful deployment of metropolitan video-on-demand (VOD) services because they provide a way to distribute very popular videos in an efficient fashion and these so-called "hot" videos are expected to account for the majority of customer requests. Rather than reacting to individual viewer requests, broadcasting protocols distribute the video contents according to a fixed schedule guaranteeing that all customers will receive these contents on time. As a result, the number of customers watching a given video does not affect the server workload.

All recent VOD broadcasting protocols derive in some way from Viswanathan and Imielinski's *Pyramid broadcasting Protocol* [20] and, like it, they assume that most, if not all, customers will watch each video in a strictly sequential fashion. They also require these customers to be connected to the service through a "smart" *set-top box* (STB) capable of (a) receiving data at rates exceeding the video consumption rate and (b) storing locally the video data that arrive out of sequence. In the current state of storage technology, this requires the presence of a disk drive in each STB, a device already present in the so-called digital VCR's offered by TiVo [18], Replay [15] and Ultimate TV [19].

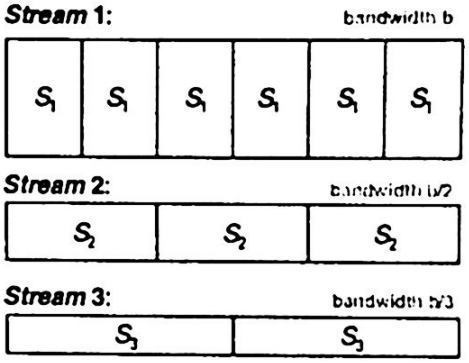

Figure 1. The first three streams for the HB protocol.

A. Harmonic Broadcasting Protocols

When we started our investigation, Juhn and Tseng's *Harmonic Broadcasting* (HB) [8] protocol was the video broadcasting protocol requiring the least bandwidth to guarantee a given maximum customer waiting time. To achieve that goal, HB divides each video into N segments S_i with $i = 1, 2, \ldots, N$. of equal duration d. Each of these N segments is continuously broadcast on a separate data stream whose bandwidth is proportional to the harmonic series. As seen on Figure 1, segment S_1 is broadcast at bandwidth equal to the video consumption rate b, segment S_2 is broadcast at half this bandwidth, segment S_3 is broadcast at bandwidth $b/3$ and so on. More generally, segment S_i with $1 \le i \le N$ is broadcast by stream i at bandwidth b/i.

When customers order a video, their STB waits for the beginning of a new instance of segment S_1 on stream 1. Since the first segment is broadcast at full bandwidth, the customers can start watching it. Meanwhile the STB starts receiving and buffering video data from the remaining $N-1$ streams. Hence the maximum customer waiting time is equal to the duration of a segment $d = D/N$ where D is the duration of the video.

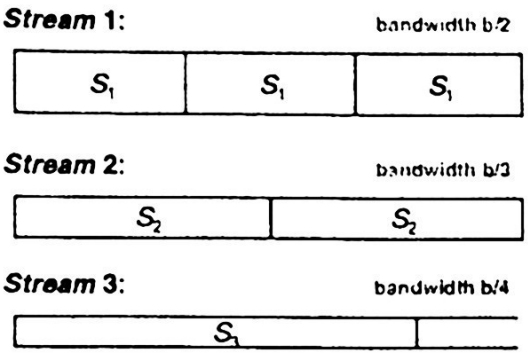

Figure 2. The first three streams for the PHB protocol with $m = 2$.

The total bandwidth used to broadcast the video is given by

$$B_{HB} = \sum_{i=1}^{i=N} \frac{b}{i} = b \sum_{i=1}^{i=N} \frac{1}{i} = bH_N \qquad (1)$$

where b is the video consumption rate and H_N is the harmonic number of N.

The original HB protocol suffers from a fundamental flaw: it cannot always provide in-time delivery of all video data [11]. Two more recent variants of the harmonic broadcasting protocol, the *Cautious Harmonic Broadcasting* (CHB) and the *Quasi-Harmonic Broadcasting* (QHB) protocols remedy this limitation [11].

better harmonic protocol, *Polyharmonic Broadcasting* (PHB) [12] implements a *fixed-delay* policy. Like other harmonic broadcasting protocols, PHB breaks a video into N segments of equal duration. Each of these N segments is allocated a separate stream and segment S_i with $1 \le i \le N$ are broadcast by stream i at bandwidth $b/(i+m-1)$. As a result, any customer having waited m/N times the duration of the video can receive all the video segments by the time they are needed. The bandwidth requirements of the PHB protocol are given by:

$$B_{PHB} = \sum_{i=1}^{N} \frac{b}{i+m-1} = b(H_{N+m-1} - H_{m-1}) \qquad (2)$$

As a result, the PHB protocol requires slightly less than five times the video consumption rate to guarantee a maximum viewing delay of 60 seconds for a two-hour video.

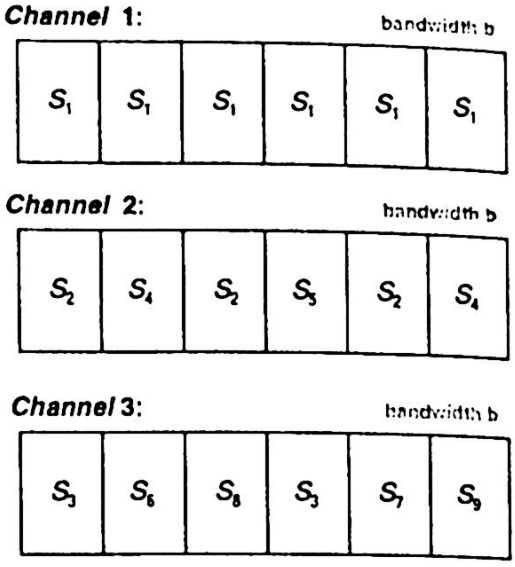

Figure 3. The first three streams for the PB protocol.

Figure 2 shows an example of a PHB protocol with $m = 2$. Observe that the first segment is broadcast at half the video consumption rate, the second segment at one third the video consumption rate and so on.

B. Pagoda Broadcasting Protocols

The excellent performance of the PHB comes at a price. To be able to achieve a maximum waiting time of 60 seconds with slightly less bandwidth than five times the video consumption rate, it needs to partition each video into at least 360 segments and broadcast these 360 segments on 3,600 separate data channels. Hence, a video server broadcasting the top 20 videos would then have to manage 3,600 parallel data streams whose bandwidths would vary between 1/3 and 1/362 times the video consumption rate. Managing such a large number of independent data streams is likely to be a daunting task.

The Pagoda Broadcasting protocol (PB) achieves similar bandwidth efficiencies without requiring as many data streams. Like HB and PHB, PB partitions each video into N segments of duration d. It assigns to each video between four and seven *channels* whose bandwidths are all equal to the video consumption rate and uses time-division multiplexing to ensure that each segment is transmitted as frequently as needed.

Figure 3 displays the first three channels of a Pagoda Broadcasting (PB) protocol. Observe first that each of these three channels is partitioned into *slots* of equal duration, each containing one segment. As in the HB protocol, the first channel continuously repeats the first segment of the video. All odd slots of the second channel transmit segment S_2 while all even slots transmit in turn segments S_4 and S_5. Hence, S_2 is transmitted once every two slots while S_4 and S_5 are both transmitted once every four slots. We will say that channel 2 is portioned into two *subchannels* of equal bandwidth $b/2$, with the first subchannel transmitting segment S_2 and the second subchannel transmitting segments S_4 and S_5. The third channel is similarly partitioned into three subchannels with the first subchannel transmitting segment S_3, the second subchannel transmitting segments S_6 and S_7 and the third subchannel transmitting segments S_8 and S_9.

As Figure 4 shows, the same process can be repeated on all subsequent channels by partitioning each even-numbered channel into two subchannels and each odd numbered channel into three subchannels. The protocol can thus map 49 segments into 5 channels and achieve a waiting time equal to 1/49 of the video duration with a bandwidth equal to 5 times the video consumption rate. This translates into a waiting time of slightly less than two minutes and half for a two-hour video.

The newer *Fixed-Delay Pagoda Broadcasting* (FDPB) protocol [14] achieves even lower customer waiting times by requiring all users to wait for a fixed delay w before watching the video they have selected. As in the PHB protocol, this waiting time is normally a multiple m of the segment duration d. Hence the FDPB protocol can assume that all clients will start downloading data from the moment they order the video rather than from the moment they start receiving the first segment. The FDPB protocol also uses time-division multiplexing to partition each of the k channels allocated to the video in a given number s_i of *subchannels* of equal bandwidth. Unlike the PB protocol, it allocates the N segments of the video to these subchannels in strict sequential order.

Channel	*Subchannel*	*Segments*
C_1	1	S_1
C_2	1	S_2
	2	S_4 and S_5
C_3	1	S_3
	2	S_6 and S_7
	3	S_8 and S_9
C_4	1	S_{10} to S_{14}
	2	S_{20} to S_{29}
C_5	1	S_{15} to S_{19}
	2	S_{30} to S_{39}
	3	S_{40} to S_{49}
C_6	1	S_{50} to S_{74}
	2	S_{100} to S_{149}
C_7	1	S_{75} to S_{99}
	2	S_{150} to S_{199}
	3	S_{200} to S_{249}

Figure 4. Complete segment to channel mappings of the PB protocol

Figure 5 summarizes the segment-to-channel mappings of a FDPB protocol requiring customers to wait for exactly four times the duration of a segment. The first channel is partitioned into $\sqrt{4}$ subchannels each having one half of the channel slots. This allows the protocol to repeat segments S_1 and S_2 every 4 slots, and segments S_5 to S_7 every 6 slots. The second channel is similarly partitioned into $\sqrt{9}$ subchannels each having one third of the channel slots.

Repeating the same process over all successive channels, the FDPB protocol can map 121 segments into four channels and achieve a deterministic waiting time of 4/121 of the duration of the video, that is, slightly less than four minutes for a two-hour video. Adding a fifth channel would allow the server to broadcast 313 segments and achieve a waiting time of 92 seconds for the same two-hour video. This is 37 percent less than the two minutes and half the PB protocol could achieve. Even lower waiting times can be achieved by increasing m. For instance, an FDPB protocol with $m = 100$ can achieve a waiting time of 58 seconds for a two-hour video broadcast on 5 channels.

Channel	Subchannel	Segments
C_1	1	S_1 and S_2
	2	S_3 to S_5
C_2	1	S_6 to S_8
	2	S_9 to S_{12}
	3	S_{13} to S_{17}
C_3	1	S_{18} to S_{21}
	2	S_{22} to S_{26}
	3	S_{27} to S_{32}
	4	S_{33} to S_{39}
	5	S_{40} to S_{47}
C_4	1	S_{48} to S_{54}
	2	S_{55} to S_{62}
	3	S_{63} to S_{71}
	4	S_{72} to S_{81}
	5	S_{82} to S_{93}
	6	S_{94} to S_{106}
	7	S_{107} to S_{121}

Figure 5. Segment to channel mappings of the first four channels for a FDPB protocol with a delay of four segments ($m = 4$)

Channel	Number of Subchannels	Segments
C_1	3	S_1 to S_{12}
C_2	5	S_{13} to S_{42}
C_3	8	S_{43} to S_{119}
C_4	13	S_{120} to S_{318}
C_5	18	S_{319} to S_{847}

Figure 6. The first five channels for the optimized FDPB protocol with $m = 9$.

There is one major drawback in increasing m. Recall that the protocol ensures that each segment is completely received by the STB before the customers start to watch it. As a result, it provides implicit *forward buffering*. Our measurements of several popular videos in DVD format indicate that this forward buffering can eliminate most of the bandwidth fluctuations inherent to compressed video signal as long as the segment duration remains above one minute.

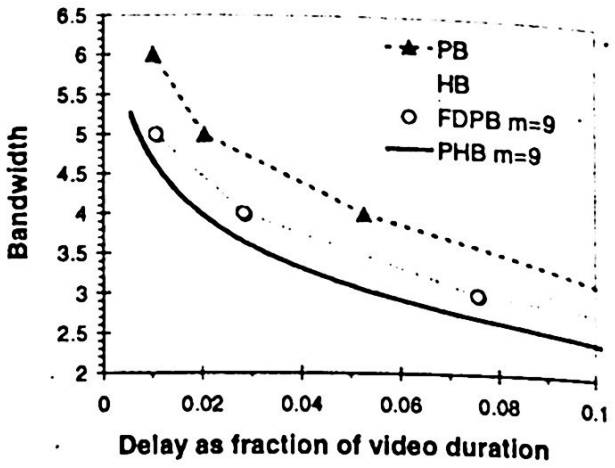

Figure 7. Bandwidth requirements of PB, HB NPB (with $m = 9$) and FDPB (with m = 9) protocols.

Tabbycat uses an FDPB protocol with $m = 9$ as it was found to be a good compromise. We were able to improve upon the performance of the original FDPB protocol by slightly increasing the number of subchannels for channels C_3, C_4, and C_5. As seen on Figure 6, this optimization allowed us to map 847 segments into 5 channels, which corresponds to a waiting time of 77 seconds for a two-hour video.

Figure 7 compares the bandwidth requirements of our optimized FDPB protocol with those of the original HB, PHB with $m = 9$ and PB protocols. Customer delays are expressed as fractions of the video duration and bandwidths in multiples of the video consumption rate. As one can see, the Tabbycat protocol performs much better than PB, slightly better than the original HB but not as well as PHB with $m = 9$.

We further modified the protocol to provide the system users with equivalents of the VCR *pause* and *rewind* commands. Tabbycat clients keep in their buffer all the previously watched segments of each video until the end of that video. As a result, Tabbycat customers can either temporarily suspend the viewing process or return to any video scene they have already watched. Since these two features are provided by the STB without any server intervention, their sole cost is a few extra gigabytes of additional temporary data on the STB hard drive.

TABLE I. OUR PROTOTYPE CONFIGURATION

Server	Intel Pentium 4 1.7 GHz 512 MB Rambus RAM 40 GB ATA-100 (7200 RPM) HDD 100 Mb/s Ethernet Interface Linux Kernel 2.4.x with ext2fs
Network	Fast Ethernet
Clients	Intel Pentium III 600 MHz 256 MB RAM 10 GB ATA-66 HDD 100 Mb/s Ethernet Interface Linux Kernel 2.4.x with ext2fs
Videos	Full-length videos in DVD format (MPEG-2)

III. THE TABBYCAT SERVER

A Tabbycat server consists of one or more workstations each distributing a few of the most popular videos. These workstations act autonomously under normal circumstances. Our prototype consisted of a single Pentium 4 system whose characteristics are summarized in Table 1.

We first benchmarked the transfer rate of the ATA drive and found it was about 40 MB/s. Having measured the instantaneous bandwidths of several MPEG-2 videos [10], we found that a typical MPEG-2 would require an average channel bandwidth of 750 kB/s with cartoons having a slightly higher bandwidth than other videos. We also found that the occasional peaks in bandwidth would average out because of *forward buffering* of the FDPB. Hence a Tabbycat with a single ATA drive should be able to broadcast 50 channels. With these 50 channels, we could broadcast 10 videos using 5 channels per video.

We decided to use UDP instead of TCP because of its low overhead. Though UDP is an unreliable protocol, we found that in a LAN there was almost no packet loss as long as the client and the server were connected to the same Ethernet switch. Moreover, using FDPB in a cable TV environment would mean that the network would have the majority of its traffic coming from the server and given the improvements in network reliability, this seems to be a reasonable choice.

TABLE II. RELEVANT BANDWIDTHS

VOD Channel	720 kB/s
ATA-100 Drive	50 channels (around 40 MB/s)
Fast Ethernet	13 channels (around 10 MB/s)
Gigabit Ethernet	100 channels (around 80 MB/s)

We found that we could achieve speeds of about 10MB/s on a 100Mb/s Ethernet. As a result, we could have about 13 channels per server.

As shown on Table I, the clients were older workstations with 600 MHz processors and 256 MB of memory. They rely on the freely available *xine* video player for decoding and playing videos. We increased their kernel network buffer sizes from 65kB to about 70MB to avoid packet losses due to congestion on the client kernel buffers.

We measured the performance of our server when it was broadcasting three videos on twelve 720 kB/s channels.

All three videos were broadcast using the modified version of the FDPB protocol with $m = 9$. As shown on Figure 6, this allowed us to partition each video in exactly 318 segments, which should allow us to achieve a customer waiting time equal to 9/318 of the duration of each video. Note that this value assumes that the client can start downloading data from segments that have already started. Since our clients could not do that, our customer waiting times will be closer to 10/318 of the video duration

Our first video is a full-feature movie in MPEG-2 format lasting 140 minutes. The observed customer waiting time on a client machine was 273 seconds, that is, 9 seconds more than expected. Our second video is another full-feature movie lasting 130 minutes. Unlike the first video it was encoded at a lower bandwidth (slightly below 360 kB/s). As a result, our segment transmission time is roughly equal to half its viewing time. This resulted in an observed customer waiting time of 156 seconds. Our third video shows highlights of professional hockey games in MPEG-2 format. The video lasts 25 minutes and the observed customer waiting time is 52 seconds, that is, 5 seconds more than expected.

Using a 100Mb Ethernet causes the network bandwidth to be a bottleneck as we can only use 25 percent of the available disk bandwidth.

A better solution would be to use a gigabit Ethernet interface. We estimate this would allow transfer rates of up to 80 MB/s. Unfortunately, the disk bandwidth limits us to 40 MB/s, that is, half of that bandwidth.

There would be several ways to eliminate that disk bottleneck. First we could attach to each workstation two SCSI disk drives and divide the disk workload among these two drives. The main disadvantage of this solution is the higher cost of SCSI drives.

A second option would be to store in the server's main memory the most frequently transmitted segments of each video. Since we are using a deterministic broadcasting protocol, we can predict ahead of time the I/O bandwidth savings that could thus be achieved [17].

IV. POSSIBLE EXTENSIONS

Three possible extensions could enhance the current implementation of our Tabbycat prototype.

First, our prototype relies on UDP for distributing the videos. As a result, its applicability is limited to either cable TV environments or well-controlled LANs, where packet losses are small enough to be tolerated by the video-encoding scheme. Deploying Tabbycat over a shared WAN would require implementing a reliable multicast protocol. We are currently investigating several possible solutions.

Second, Tabbycat now requires each STB to receive at the same time data from all the k channels allocated to the video being currently watched. This requirement complicates the design of the client and increases its cost.

One possible solution to this problem is to use an FDPB protocol limiting the STB receiving bandwidth to two channels. We found out that such a protocol with the same value of $m = 9$ could map up to 220 segments into four channels [17]. Hence it would achieve a waiting time equal to $9/220^{th}$ of the video duration, that is, a little less than 5 minutes for a two-hour video. Adding a fifth channel would allow us to partition the video into 474 segments and achieve a waiting time of 137 seconds for the same two-hour video. This is almost twice as much as the customer waiting time of an FDPB protocol requiring the customer STB to receive at the same time data from all the k channels. As a result, we would have to add one extra channel per video to maintain the same waiting time.

Finally, we can see from Figure 5 that our FDPB protocol requires as much bandwidth to broadcast the first 12 segments of a video as for broadcasting S_{319} to S_{847}, that is, a total of 529 segments. Significant bandwidth savings could be achieved by requiring each customer STB to preload in advance the first few minutes of each video. This would also enhance the user experience by allowing us to provide instant access to these videos [13, 16].

V. OTHER VIDEO SERVERS

The Berkeley Distributed Video-on-Demand System [3] allowed clients across the Internet to submit requests to view audio, video and graphical streams. Playback was accomplished by streaming data from a media file server through the network to the client's computer. No effort was made to share data among overlapping requests.

The Tiger system was a scalable, fault-tolerant multimedia file system using commodity hardware [1]. Unlike Tabbycat, it dedicated a separate data stream to each customer request and made no attempts to share data among overlapping requests. Hot spots were avoided by striping all videos across all workstations and disks in a Tiger system. Tiger prevented conflicts among requests by scheduling incoming requests in a way that ensures that two requests will never access the same resource at the same time. This task was distributed among all of the workstations in the system, each of which having an incomplete view of the global schedule. The main disadvantage of the approach was its poor scalability: Tiger designers found that a system with ten workstations could only handle one hundred concurrent user requests.

More recently, Bradshaw *et al.* have presented an Internet streaming video testbed [2] using both periodic broadcast and patching/stream tapping [4, 7]. This allowed the server to select the best distribution protocol for each video, namely, broadcasting videos in very high demand while distributing less popular videos through stream tapping/patching.

This feature resulted in a much more complex system than our Tabbycat server. In addition, the *greedy disk-conserving broadcasting* protocol [6] used by their system is less bandwidth-efficient than the optimized FDPB protocol used by Tabbycat. While the optimized FDPB protocol only requires five channels to achieve a waiting time of 77 seconds for a two-hour video, the greedy disk-conserving broadcasting protocol requires 6 channels to achieve a waiting time of 114 seconds for the same video.

VI. CONCLUSION

Current wisdom is that distributing video on demand to large audiences requires complex expensive farms of video servers. We built the Tabbycat video server to show that a metropolitan video server distributing the top ten to twenty videos could consist of a few powerful workstations running unmodified versions of a standard operating system.

The secret of Tabbycat's good performance is the broadcasting protocol it uses to distribute the videos. We knew ahead of time that the FDPB protocol would provide smaller waiting times than any other protocol broadcasting fixed-size segments over channels of equal bandwidth [14]. We found it easy to implement and easy to tune thanks to its regularity.

As it stands now, Tabbycat is a mere proof-of-concept prototype. More work is still needed to allow its deployment in environments where packet losses are likely to happen.

ACKNOWLEDGEMENTS

We want to thank especially Mr. Saurab Mohan for his careful analysis of the bandwidth variations of actual videos in MPEG-2 format and Mr. Karthik Thirumalai for his implementation of the Tabbycat prototype. Most of the work on protocol development was done in close cooperation with Prof. Darrell D. E. Long from the University of California, Santa Cruz.

The Tabbycat project was supported in part by the USENIX Foundation, the Texas Advanced Research Program under grant 003652-0124-1999 and the National Science Foundation under grant CCR-9988390.

REFERENCES

[1] W. J. Bolosky, R. P. Fitzgerald and J. R. Douceur, Distributed schedule management in the Tiger video fileserver. *Proc. 16th ACM Symposium on Operating Systems Principles*, pp. 212–223, October 1997.

[2] M. K. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley, Periodic broadcast and patching services—Implementation, measurement, and analysis in an Internet streaming video testbed. *Proc. 9th ACM Multimedia Conference*, Oct. 2001.

[3] D. W. Brubeck and L. A. Rowe, Hierarchical storage management in a distributed video-on-demand system." *IEEE Multimedia*, 3(3):37–47, 1996.

[4] S. W. Carter and D. D. E. Long, Improving video-on-demand server efficiency through stream tapping. *Proc. 5th International Conference on Computer Communications and Networks*, pp. 200–207, Sept. 1997.

[5] A. Dan, D. Sitaram, and P. Shahabuddin, Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4(3):112–121, June 1996.

[6] L. Gao, J. Kurose, and D. Towsley, Efficient schemes for broadcasting popular videos. *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.

[7] K. A. Hua, Y. Cai, and S. Sheu, Patching: a multicast technique for true video-on-demand services. *Proc. 6th ACM Multimedia Conference*, pp. 191–200, Sep. 1998.

[8] L. Juhn and L. Tseng, Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, Sept. 1997.

[10] S. Mohan, Characterizing the Bandwidth Requirements of Compressed Videos. MS Thesis, Department of Computer Science, University of Houston, May 2001."

[11] J.-F. Pâris, S. W. Carter, and D. D. E. Long, Efficient broadcasting protocols for video on demand. *Proc. 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 127–132, July 1998.

[12] J.-F. Pâris, S. W. Carter and D. D. E. Long, A low bandwidth broadcasting protocol for video on demand. *Proc. 7th International Conference on Computer Communications and Networks*, pp. 690–697, Oct. 1998.

[13] J.-F. Pâris, D. D. E. Long and P. E. Mantey, A zero-delay broadcasting protocol for video on demand. *Proc. 1999 ACM Multimedia Conference*, pp. 189–197, Nov. 1999.

[14] J.-F. Pâris, A fixed-delay broadcasting protocol for video-on-demand. *Proc. 10th International Conference on Computer Communications and Networks*, pp. 418–423, Oct. 2001.

[15] ReplayTV, http://www.replay.com/.

[16] H.-K. Sul, H.-C. Kim and K. Chon, A Hybrid Pagoda Broadcasting protocol: Fixed-Delay Pagoda Broadcasting protocol with partial preloading. *Proc. 2003 IEEE International Conference on Multimedia and Expo*, July 2003.

[17] K. Thirumalai, J.-F. Pâris and D. D. E. Long, Tabbycat: an inexpensive scalable server for video-on-demand, *Proc. IEEE International Conference on Communications*, pp. 896–900, May 2003.

[18] TiVo Technologies, http://www.tivo.com/.

[19] UltimateTV, http://www.ultimatetv.com/.

[20] S. Viswanathan and T. Imielinski, Metropolitan area video-on-demand service using Pyramid Broadcasting. *Multimedia Systems*, 4(4):197–208, 1996.