Adaptive, Workload-Dependent Scheduling for Large-Scale Content Delivery Systems

Kevin C. Almeroth
Department of Computer Science
University of California
Santa Barbara, California 93106-5110
almeroth@cs.ucsb.edu
(805)893-2777(office)
(805)893-8553(FAX)

August 1, 2000
Revised: November 18, 2000

Abstract

Content delivery has become an important application in the Internet. "Content" in this context can be a range of objects from movies to web pages to software distribution. A *streaming* content delivery server should provide nearly immediate and continuous service by provisioning sufficient server and local network resources for the duration of playout. Because of the resource implications of delivering many large files simultaneously, scalability is an important requirement. Good scalability can be achieved by using a single channel to serve multiple users waiting for the same object (referred to as *batching*). Batching is especially useful during high load periods. Typical strategies in use today for allocating channels use a greedy, allocate-as-needed policy with little consideration for anything other than satisfying the current request or maximizing the number of batched requests. Macroscopic system characteristics, like request arrival patterns, have stable long-term averages, but can vary unpredictably across shorter intervals. This variability can cause scheduling algorithms to suffer poor and unpredictable short-term performance. In this paper, we propose a set of *rate-based allocation algorithms* to solve these limitations. We present our work in developing a set of workloads with variable request rates, quantify the drawbacks of traditional greedy channel allocation algorithms, and quantify the advantages of the rate-based algorithms. We also generalize the content delivery model and discuss when rate-based algorithms might be effective other kinds of systems.

Keywords: video server, scheduling, video-on-demand, multicast

1 Introduction

While true video-on-demand (VoD) has not developed into the "killer application" it was once portrayed, video delivery in the Internet has quietly become an important media type. For content providers, video is still sought after as the next key media type that will draw greater numbers of users. At the same time, Internet Service Providers (ISPs) are becoming increasingly wary of video as a data source. Video traffic continues to grow, even as a percentage of the growing volume of total traffic[1]. Of particular concern is that on a per-user basis, video consumes a larger amount of bandwidth than many other applications. Furthermore, a growing amount of video is streamed using UDP/IP. UDP-based video traffic, in the worst case, is bursty, high-bandwidth, long-lived, and sent without congestion control or bandwidth provisioning.

When streamed video delivery and applications like VoD were first investigated, deployment was targeted mainly for non-Internet networks, i.e. cable networks, satellite-based networks, etc.[2]. Increasingly, delivery via the Internet infrastructure is being considered. The goal for many web sites is to develop as large a user population as possible. To this end, these sites must offer the most eye-catching services and applications possible. As a result, more users are beginning to see motion video, albeit low quality, streamed to their desktop. Interest in, and use of streaming video is growing to the point where content providers and ISPs must carefully consider how to provide sufficient capacity.

Researchers have long investigated a number of techniques to reduce the retrieval and delivery burdens of video. Caching[3, 4, 5], pre-fetching[6, 7, 8], mirroring[9, 10], batching[2, 11], etc. are all techniques that have been developed. While these techniques attempt to distribute the workload and move content closer to the user, these techniques *alone* provide minimal effectiveness. There are numerous "corner cases" that make systems based on these techniques perform poorly. With caching, what if there is no similarity or redundancy in the requests made by topologically-close users? With pre-fetching and mirroring, what if no users request the target content? With batching, what if there is not a sufficient number of users with similar requests to batch them together?

In this paper we propose a new technique that combines batching and a novel workload-dependent scheduling algorithm. This algorithm is useful in large-scale systems that stream content to users. While video is a prime example, it is by no means the only content that can be streamed. Information dissementation and software distribution have recently become interesting types of content that can take advantage of similar techniques[12]. In comparing the efficiency and effectiveness of our technique to others, we choose to focus on three issues of particular relevance to the operation of large-scale systems. These are:

- **Service Characteristics**: Content that is *streamed* is likely to require a set of server and delivery resources for a significant period of time. While the rest of the Internet has trended towards shorter and shorter connections, the delivery of streamed content necessitates long-term use of a set of resources. Once resources are committed to satisfying a particular user request, it is unlikely those resources will be freed for the duration of the transmission—a period of time which might last as long as several hours.

- **Workload Patterns**: In today's Internet, which has millions of users and is growing rapidly, one-time external events (important world event) or repeated use patterns (downloading news in the morning) can create significant surges in demand—far above normal or average load. More importantly, because content providers must provide reasonable service for as much of the time as possible, large server farms are needed. However, one consequence is that during normal load periods, a significant portion of capacity is left idle.

- **System Evaluation Metrics**: In analyzing the effectiveness of various techniques for improving performance, there are two subtle, yet important factors to consider. First, the real-world motivation for improving efficiency is not simply to serve more streams. Other factors, particularly the motivation of generating revenue, play an important role. Therefore, a content provider may have an arbitrarily complex economic model that trades efficiency for increased revenue. Second, the real-world contains users who are both fickle and understanding. Users who are expected to wait an indeterminate amount of time will be less patient and wait for a shorter period than those who are told how long their expected wait time is. Furthermore, users who understand that load is high may be more tolerant to waiting longer. Therefore, techniques should not only seek to provide optimal efficiency but should include flexibility and consistency as key objectives.

In this paper, we specifically consider systems which are expected to stream content to one or more users over a lengthy period of time. In these systems, we simplify the capacity of the delivery system into a model of finite, discrete "logical channels" which are capable of delivering a stream to one or more users. These channels are used to satisfy user requests and are reserved for a known duration. In our model, we assume that multiple requests can be serviced together using a technique called *batching*. In this case, multiple requests are satisfied, but only one logical channel is used. Content is sent from one source to many receivers using an efficient multicast delivery mechanism.

Our contribution is the development of a channel scheduling strategy that considers the rate at which channels are being allocated. An ideal rate is calculated based on the number of logical channels and the expected duration that channels will be in use. By ensuring that channels are allocated in a controlled manner, request bursts can be handled without starving requests that come shortly after the burst. Coupled with batching, *rate-based channel allocation* strategies provide excellent scalability and predictable wait times. We analyze the performance of our algorithm and compare it to other baseline algorithms in the context of highly variable workloads and a user

behavior model based on a limited tolerable wait time. We show that rate-based allocation offers a number of advantages with minimal drawbacks.

The remainder of the paper is as follows. In Section 2, we describe the generic architecture of a large-scale content streaming system and our model for variable workloads and user behavior. Section 3 details the range of channel allocation strategies In Section 4, we present simulation results and an analysis of the various channel allocation strategies. Section 5 presents a set of generalizations we have developed based on using rate-based channel allocation strategies. The paper is concluded in Section 6.

2 System Model

The context in which we have developed our channel allocation algorithm is based on a system with three components. First, at the center is the content server architecture, described in the next section. Second, is our model for content server workload. Finally, our user behavior model is the third component of the system.

2.1 Content Server Architecture

Content delivery systems can vary in size and complexity depending on many factors including the number of simultaneous streams the system is capable of providing, and the type of service to be offered. Systems can have servers ranging from one small, centralized machine to a large network of distributed file servers with several control points. The network can also varying significantly; most importantly in what transport layer functions are provided. Any number of protocols over a variety of media and topologies may be available. In order to eliminate superfluous details created by considering any one particular system, we propose the use of a generic architecture.

Figure 1 shows a generic architecture with three main components: content server, network, and users. The typical operation is based on users making program *requests* to the content server. These requests flow over a bi-directional channel established for the purpose of exchanging control messages. A request is received and queued by the control server until the *scheduler* makes available a logical channel to satisfy the request. We assume that the network can provide efficient multicast communication which means that if multiple requests for the same request are in the queue, they can be *batched* and serviced with one logical channel[11].

This paper focuses specifically on the channel allocation process. However, channel allocation is actually part of the larger process of effectively and scalably handling as many requests as possible.

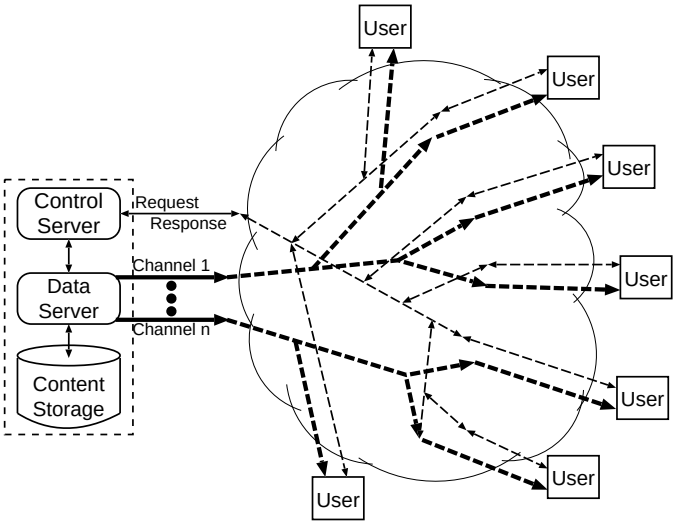

Figure 1: Generic system architecture with multicast capability.

This process is shown in Figure 2. There are three places in this process that various algorithms for improving scalability can be applied. These include:

- **Channel Allocation Control**: The goal of channel allocation is to prevent request bursts from creating large variations in response time or resource starvation. The process responsible for allocating a channel works like a leaky bucket. That is, at periodic intervals, channels are released to the scheduler. A wide variety of policies ranging from simple on-demand strategies to more sophisticated rate-control strategies can be used.
- **Request Scheduling Policies**: Once a channel becomes available, deciding which requests to service is the function of the request scheduling process. The decision can be arbitrarily complex depending on a range of issues like fairness, service classes, priority, etc. For example, one request might receive deluxe or preferred service for a higher fee and thus be served ahead of other queued requests. In this paper, we assume a first come, first served (FCFS) policy because it is fair, effective, and easy to implement[11].
- **Patching Efforts**: Once a channel is in use, additional strategies can be used to increase aggregation. For example, channels showing the same program that are only offset by a short period of time can be combined by slowing playout for one and speeding playout for the other[13].

While request scheduling policies and patching efforts have the ability to improve system performance, these gains are independent of the issue of channel allocation control. Novel techniques for channel allocation, request scheduling, and patching can be implemented together in the same system. Efficiency gains should be fully additive.

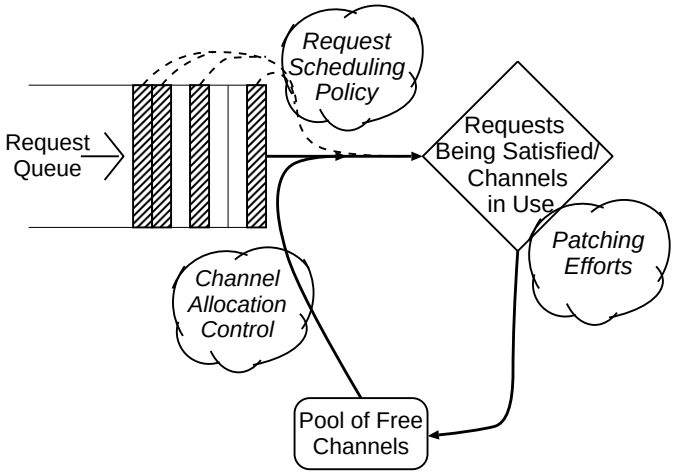

Figure 2: Outline of channel allocation, requesting scheduling and patching functions.

2.2 Workload Description

Server workload in the Internet today can no longer be expected to be based on long-term, well-behaved Poisson distributions. Events external to the Internet conspire to create periods of significant fluctuation. Furthermore, as the Internet-connected population increases, we can expect fluctuations to take on even larger proportions. It is therefore worthwhile to design systems to handle peak loads satisfactorily, and to optimize resource utilization during low load periods. To this end we have developed a model for creating varied workload patterns and then created four workloads to use in our analysis.

Workloads are modeled based on a 24-hour period beginning from 8:00am of one day and running to 8:00am of the next. Using the time of day as a guide, we include in our model a "prime time" period. This simply represents a period half-way through the simulation where demand begins to surge. A real-world example of a prime time period happens in video delivery systems. Surges in demand are created as people arrive home from work. This is equivalent to the most watch time period in traditional broadcast television: the prime time hours between 5:00pm and 11:00pm.

In Figure 3, we propose a baseline steady-state workload and three new non-steady state workloads. The horizontal axis represents time, and the vertical axis shows the number of requests that arrive during a six minute *reporting interval*. The top graph in Figure 3 shows a steady-state request arrival pattern computed using an exponential function. While we believe this is unrealistic, it was useful as a comparison to the other workloads. For the non-steady-state workloads, we used statistics which suggest that request arrival rates during the prime time period can be approximately five times greater than the normal rate[14]. The second and third graphs in Figure

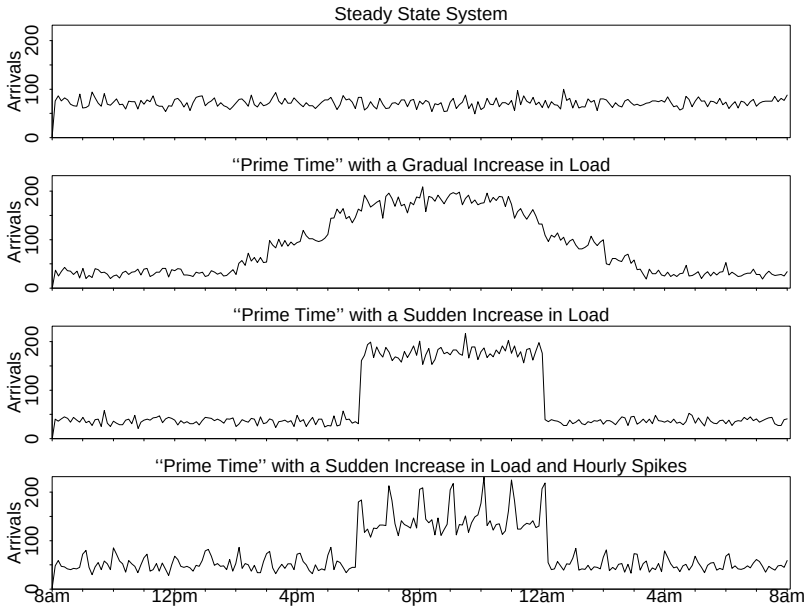

Figure 3: Various request arrival patterns.

3 show two such workloads—one with a gradual increase to high load, and one with a jump to high load. Changes in the request arrival pattern are implemented using an exponential function but with changing averages. The fourth workload is a special case with load surges at hour intervals. This type of workload is based on the belief that the workload for certain kinds of systems might synchronize around an external event like wall clock time, e.g. increased requests at the top and bottom of the hour.

2.3 User Waiting Behavior

Users can exhibit completely arbitrary and random behavior. However, in order to evaluate system performance, we must select some model of behavior. The model used in this study is based on users unwillingness to wait for service more than some finite amount of time[15]. This closely resembles how users behave in the Internet today. As an illustrative example, consider a user who is surfing the web. The user will click on a link and wait. Unless there is some indication of progress, the user will become impatient and very quickly look elsewhere. Because most systems process requests using a first come, first served policy, the user's request would have eventually been satisfied. However, during high-load periods, a user's wait time may be very long—beyond some user-imposed maximum tolerable wait time. Our goal in modeling users is to develop a representation for this behavior. As a result, our model assumes that users agree to wait at least some pre-defined amount of time plus an additional random amount of time randomly chosen with

a given average from an exponential distribution. For example, a user might be willing to wait five minutes to watch a video-on-demand movie plus some random amount of additional time up to 2.5 minutes. If after this period of time a channel has not been allocated to satisfy the user's request, the user is assumed to withdraw the request. Using terminology consistent with related work, we call this process *reneging*. User reneging is analogous to a request that cannot be satisfied and is a key performance metric.

3 Survey of Channel Allocation Policies

We divide the set of channel allocation policies into two groups: (1) the existing set of *greedy* channel allocation policies which do not take workload factors into consideration and simply allocate channels on an as-needed basis, and (2) our proposed *rate-based* policies which attempt to protect system resources from request surges and starvation. Each set of policies is described below.

3.1 Greedy Allocation Policies

The two most common greedy allocation policies are *on-demand allocation* and *forced wait*. Both are described below.

3.1.1 On-Demand Allocation

Allocation of channels using on-demand scheduling is the most straightforward of all the policies discussed in this paper. The algorithm is as follows:

- When a request is received, the scheduler places it on the queue. If a free channel exists, it is immediately allocated. Otherwise, the request waits.
- When a channel is freed, the scheduler checks to see if there are any requests in the queue. If requests are waiting, the freed channel is immediately allocated to the request at the head of the queue and all others requesting the same content.

Typical operation during low-load periods is that there will be a free channel each time a request arrives. This means no requests will be queued, no batching will occur, and all requests are satisfied immediately. However, during high-load periods, requests arrive faster than channels are freed so request queuing and batching are common. Because of increased queuing, wait times increase. If the load is sufficiently high, requests sit in the queue too long, and the user making the request reneges and the request is cancelled.

3.1.2 Forced Wait

Forced wait is a scheduling policy that requires the request at the head of the queue to wait some minimum amount of time before being serviced. The goal of this forced wait is to make users wait as long as possible—thereby increasing the batching factor and increasing effective server capacity. With a forced wait algorithm, channels are allocated on an on-demand basis but only after the request at the head of the queue has waited the minimum wait time.

Forced wait suffers from several general drawbacks. The first is that there is a *minimum wait time* parameter. This parameter must be set by a person. Finding the optimal value is very difficult for three reasons: (1) user behavior changes over time, (2) the optimal forced wait time is impossible to accurately determine, and (3) system performance is very sensitive to the parameter. The second general drawback is that forced wait is still fundamentally a greedy allocation policy.

3.2 Rate-Based Allocation Policies

The motivation for developing rate-based channel allocation strategies is the poor performance of the greedy algorithms. This section first describes the basic basic rate-based allocation algorithm and then describe several extensions.

3.2.1 Pure Rate Control

The *pure rate control* policy (1) puts a strict upper limit on the rate at which channels can be allocated and (2) limits the total number of channels allocated during a fixed time interval or *reporting interval.* The purpose of the interval is to provide a finite time period over which to accurately calculate and enforce the rate control policy. By limiting the allocation of channels to a set rate, we can ensure that some channels will be available in each reporting interval. This accomplishes several things:

- It ensures that some channel resources will be available in each reporting interval even if there was a large burst of requests in the previous interval.
- It encourages batching, but only when load is high and batching is necessary and useful.

The pure rate control policy attempts to allocate channels at a uniform rate during each reporting interval. In calculating the rate of allocation, we use the following parameters:

C_{max}: the maximum number of channels that can be allocated during each interval
C_{alloc}: the number of channels already allocated
T_{last}: the last time a channel was allocated

T_{next}: the next time a channel can be allocated
T_{left}: the remaining time until the end of the reporting interval

We then calculate:

$$T_{next} = T_{last} + \Delta \tag{1}$$

where Δ is the interval between successive allocations. The formula for Δ is:

$$\Delta = \frac{T_{left}}{C_{max} - C_{alloc}} \tag{2}$$

Next, we need to calculate C_{max}, which is dependent on several additional system parameters. These include:

T_{view}: the average channel holding time, i.e. the time to stream an object
C_{total}: the total system capacity in terms of the number of channels
T_{int}: the length of a reporting interval

C_{max} can now be calculated as:

$$C_{max} = \frac{T_{int} * C_{total}}{T_{view}} \tag{3}$$

As an example of nominal parameters to use in these equations, in this paper we present simulations for a server that is characterized by:

T_{int}: 10 minute reporting interval
T_{view}: 120 minute channel holding time
C_{total}: 1000 channel system capacity
C_{max}: 83.3 channels per reporting interval
$Delta$: 7.143 second interval between channel allocations

These parameters and the allocation rate formula together establish a *maximum allocation rate*. This limit controls the channel allocation rate within a reporting interval. Each time a channel is allocated, Δ is re-calculated. Figure 4 shows an example of a channel allocation scenario. The request arrival rate in the first reporting interval is equal to or greater than the rate at which channels can be allocated. In the second interval, the request arrival rate drops to less than the maximum channel allocation rate. However, in the middle of the interval, an increase in load causes the number of arriving requests to increase dramatically. Pure rate control allows channels to be allocated more quickly, but never to the point where the hard limit for the interval is exceeded.

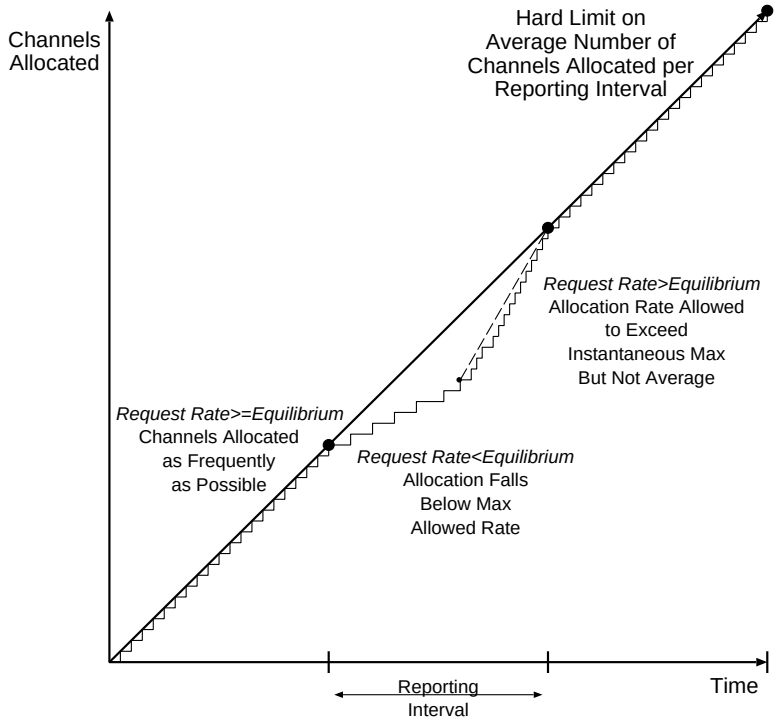

Figure 4: Maximum allocation rate for pure rate control.

Because of the way Δ is computed, the number of channels allocated can never exceed the hard upper limit shown in Figure 4. The relationship between different request arrival patterns and pure rate control operation can be summarized as follows:

- The request arrival pattern during low load periods will not use channels as fast as the allocation rate might allow. In most cases, the request inter-arrival time will exceed the delay imposed between channel allocations. During low load periods, requests will be satisfied immediately upon arrival.
- When the request arrival pattern is high enough, requests will arrive more quickly than channels can be allocated. Requests will be queued and must wait for the scheduler to allocate a channel. When a channel is allocated, batching allows all requests for the same content to be serviced together.

3.2.2 Deviated Rate Control

Deviated rate control is similar to pure rate control, except that under certain conditions deviated rate control allows the channel allocation rate to temporarily rise above the hard limit. Figure 5 shows the ideal allocation rate and an *allowable deviation*. An example load surge is shown that drives the actual allocation rate above the ideal rate temporarily. However, the maximum number of channels that can be allocated during a reporting interval is still strictly enforced. Deviated rate

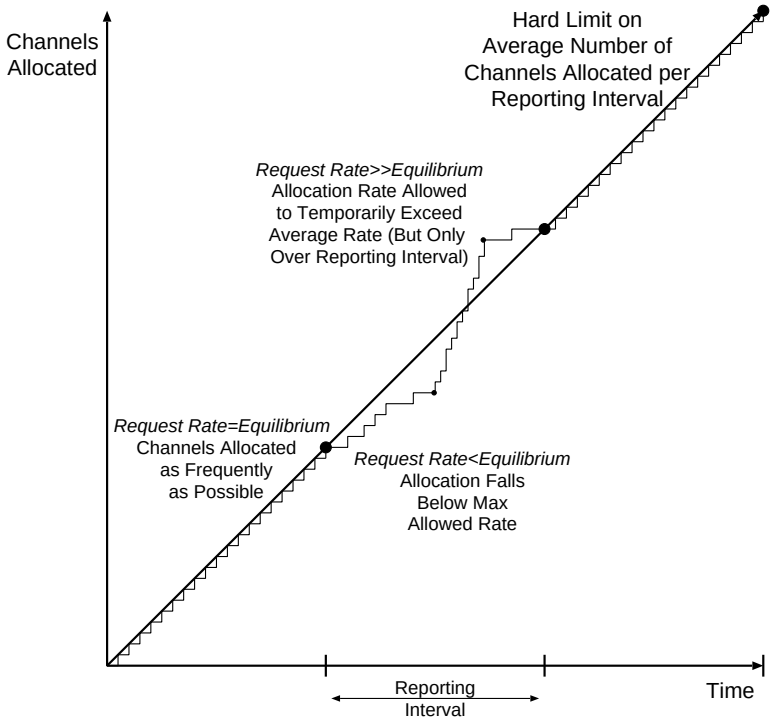

Figure 5: Maximum allocation rate and deviation for deviated rate control.

control simply provides more flexibility in allocating channels within an interval. This additional flexibility is an important feature that can further reduce reneging during short-term load surges.

Deviated rate control requires two parameters to be defined, (1) the *maximum allowed deviation*, and (2) a *maximum tolerable wait* threshold. The maximum allowed deviation, which is defined between one and zero, shortens the delay between one channel allocation and the next. The time of the next channel allocation is still the same as Equation 1, but now Δ becomes Δ':

$$\Delta' = \Delta * \alpha, \qquad \text{where } \alpha \text{ is the maximum allowed deviation} \tag{4}$$

Deviation from the ideal allocation rate should only occur when load conditions are such that either a channel be allocated quickly or users will start to renege. Deviated rate control should only be used when the request rate increases for a short period of time[1]. If the request rate increase will continue for a period greater than the average service time, pure rate control should be used. A sustained rate means channels should be allocated consistently but a brief surge could be better handled with additional channels and without adversely affecting future requests. This goal is

[1]Knowing whether users are about to renege and if a load surge is short-lived is not possible in reality. Therefore, while deviated rate control achieves better performance, it has disadvantages in terms of a service provider's ability to implement it.

achieved by allowing the allocation rate to exceed the ideal rate only when the user at the head of the queue has exceeded the minimum wait time threshold and is believed to be close to reneging. The maximum allowed deviation parameter can be used to implement a rate-based policy with any level of control. A value of one implements the pure rate control policy while a value of zero is the same as forced wait but with a limit on the total number of channels that can be allocated per reporting interval.

3.2.3 Multiple Service Classes

A channel allocation policy using *multiple service classes* gives a service provider the ability to offer multiple service levels. One example is to give a higher scheduling priority to "hot" objects. For these objects, the likelihood of batching multiple requests is greater, thereby increasing profit potential. The determination of which objects are popular and which are not can be made many ways. The method used in this paper is to use the last reporting interval's request count as a heuristic. Any object with sufficient requests in the last interval is considered hot for the next interval. Once objects are identified as hot or cold, one of two channel allocation policies is used:

1. **Hot**: The first request for a hot object must wait a minimum wait time (similar to the forced wait policy). As soon as this condition is met, a channel is immediately allocated and all requests for the content are satisfied. Hot objects have scheduling priority over cold objects, and can be guaranteed to start within the minimum wait time.
2. **Cold**: Scheduling for cold objects is done using pure rate control. However, any channels allocated for hot objects are still counted in the total number of channels that can be allocated in a reporting interval. If no limit per reporting interval is used, channels can be allocated too quickly and cycles may form. Starvation of requests for cold objects is possible but unlikely since only a few objects are typically hot.

4 Simulation Results and Analysis

In this section we use our proposed workloads, the set of channel allocation policies, and a simulator to compare and understand system performance. In the next section we describe the simulation environment. Then, we show results for the different channel allocation policies. We also investigate the affect of secondary system parameters (system size, object characteristics, etc.) and perform capacity planning simulations.

4.1 Simulation Environment

The following is a list of simulation parameters considered in analyzing the various channel allocation policies.

1. **System Capacity**: Capacity measures the number of simultaneous streams that can be supported. Most of the results presented in this paper are for systems with 1000 logical channels. In Section 4.4 we show the effects of varying the number of channels.
2. **Object Library**: The library is the number of objects available to users. Most of the results presented are for systems offering 100 objects. In general, larger numbers of objects reduces the probability of request aggregation and therefore reduces the effectiveness of batching. In Section 4.4 we show results for systems offering different library sizes.
3. **Object Selection**: For each request made, an object is selected using a Zipf distribution[16]. The probability that object i is chosen equals $\frac{c}{(i^{1+\theta})}$, where c is a normalizing constant. Empirical evidence suggests that $\theta = 0.271$ is a reasonable value to use[11].
4. **Object Playout Duration**: The playout duration is the amount of time it takes to stream an object to a user. For the results presented we assume a duration chosen from a uniform distribution between 105 and 135 minutes.
5. **Channel Allocation Policy**: The policies discussed in Section 3 are each simulated.
6. **Request Arrival Pattern**: Each of the four workloads shown in Figure 3 have been thoroughly simulated.
7. **Reneging Behavior**: User behavior is based on the model presented in Section 2.3. The specific model is that each user is willing to wait at least 5 minutes plus a random amount of additional time chosen from an exponential distribution with an average of 2.5 minutes.

The performance of the various channel allocation algorithms are compared using the following set of metrics:

1. **Reneging Probability**: The probability of reneging is measured by taking the number of users who renege divided by the total number of requests made. Reneging probability is measured over both the short- and long-term.
   - **Long-Term Reneging**: This is a measure of the total number of users who reneged during the 24-hour simulation period. As a guideline, we use a rough target for long-term reneging of 5% or less.
   - **Short-Term Reneging**: This measures average reneging over a 6 minute reporting interval. By controlling short-term reneging, a system can provide a consistent level of service and meet user performance expectations. Generally, a rough target for short-term reneging is 15% or less.
2. **Average Wait Time**: Wait time is measured from when a user makes a request to when the user begins to receive the object. Average wait time is only measured for requests that are

satisfied. In addition to a straightforward quantitative comparison, we can also compare the channel allocation policies based on their ability to provide predictable response time. We do not require that the average be the same for all periods of the day, but wait times should be reflective of system load. Generally, a rough target for average wait time during high load should be 5 minutes or less.

3. **Ease of Implementation**: As another qualitative measure, we believe that channel allocation policies should not require any indeterminable information about the system. For example, a policy should not require knowing how long users are actually willing to wait.

In addition to these metrics, there are a number of additional factors that give additional insight into how and why the simulated systems perform as they do. A majority of our results present these additional factors. When reporting simulation results, each graph has the following parts:

1. **Arrivals**: The number of requests received by the scheduler in the (6 minute) reporting interval.
2. **% Renege**: The percentage of reneged requests. In these sub-graphs, there is also a horizontal dashed line representing the short-term reneging objective of 15%. Finally, there is a dotted line which represents the *cumulative average*. This average is computed from the beginning of the simulation period up to the current reporting interval.
3. **Channels Alloc**: The absolute number of channels allocated during each reporting interval. Again, the dotted line represents the cumulative average.
4. **Wait(secs)**: The average wait time of all successfully satisfied requests. Again, the dotted line represents the cumulative average.
5. **In Queue**: The average number of requests waiting in the queue. Again, the dotted line represents the cumulative average.

4.2 Performance of Greedy Allocation Policies

In this section, we describe the simulated performance of systems that implement greedy channel allocation policies.

4.2.1 On-Demand Allocation

Figure 6 shows the performance of a content delivery system that implements on-demand channel allocation. This system has poor long-term reneging (almost 20%) and very pronounced oscillations in short-term reneging (see second graph in Figure 6). Oscillations in short-term reneging occur because channels are allocated more quickly than they are being freed. More precisely, when the load jumps at 6pm (see top graph in Figure 6), there are only enough channels to satisfy requests on an on-demand basis for a half hour (see third graph in Figure 6). After this time, all free channels

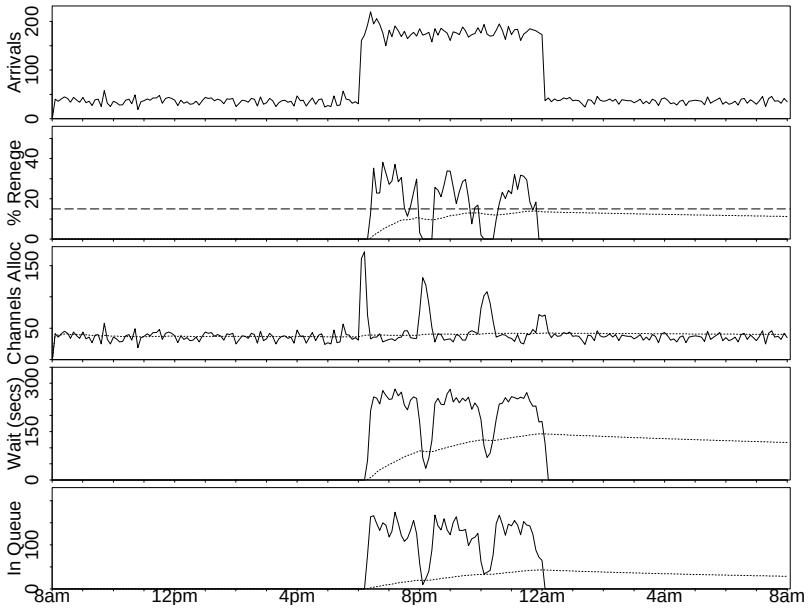

Figure 6: The on-demand channel allocation policy.

have been allocated. Furthermore, since most of the channel pool was just allocated, these channels will not be available for the remainder of their service time (on average two hours). Until these channels are eventually freed a majority of new requests will not be satisfied. When the group of channels does eventually becomes free they will again be completely allocated in a short period of time. These cycles will eventually go away, but it either takes many hours or a return to low load.

4.2.2 Forced Wait

Figure 7 shows the reneging levels for a content delivery system using a forced wait policy. Three different minimum wait times are used. For a system with 4, 5, and 6 minute minimum waiting time, the long-term average reneging is 8.9%, 6.2%, and 17.6% respectively. Recall that the behavior model says users will be satisfied if their request is satisfied within 5 minutes plus additional time chosen from an exponential distribution with an average of 2.5 minutes. The optimal minimum wait time is slightly more than 5 minutes. This is known only because there is a known, static behavior model. Other behavior models that are more dynamic will have a significant adverse effect on performance. Forced wait performs especially poorly when the user's maximum tolerable wait time is over-estimated. Channels might be available for use but a user is unwilling to wait the minimum wait time and reneges. This result is shown in the bottom graph of Figure 7. The challenge of choosing an appropriate minimum wait time aside, forced wait does perform better than on-demand channel allocation.

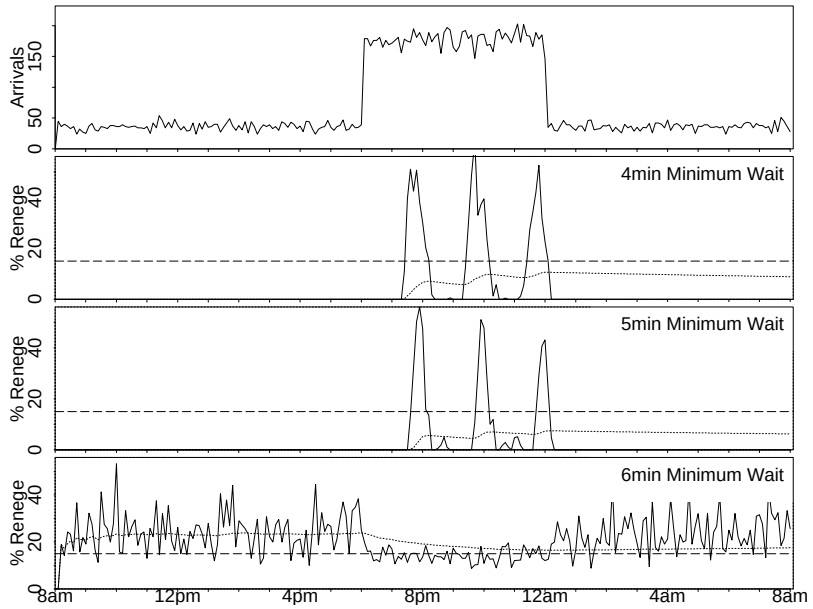

Figure 7: The forced wait policy with different minimum wait times.

Forced wait has another disadvantages. It does not fix the problem of cyclical behavior even using the optimal minimum wait time. Figure 8 shows results for forced wait with an optimal wait time and in a format similar to Figure 6. The results show similar cyclic behavior. When the load increases, all free channels are allocated in a short period of time, starting the cyclical behavior effect. One difference is that the duration of high reneging periods in forced wait is not as long as with the on-demand policy. This is because of the ability of forced wait to batch requests. During the high workload period, an average of between 2.5 and 3.5 users are satisfied with each channel. Still, short-term reneging is above our target of 15% for almost half of each two hour cycle.

A final drawback of forced wait is the high waiting time experienced by each user (see the fourth graph in Figure 8). Even during low loads, users are forced to wait the minimum waiting time. Forced wait is therefore not appropriate for systems that do not experience high load all of the time. During low load periods users should be serviced more quickly. Furthermore, for content infrequently requested, the likelihood of having multiple requests in the queue at the same time is very low. Therefore, using a forced wait policy for these requests does nothing to increase effective capacity and only increases the average wait time. Using a hybrid of on-demand for low workload periods and forced wait with optimal wait time might be successful but it requires accurate estimates of user behavior and workload patterns.

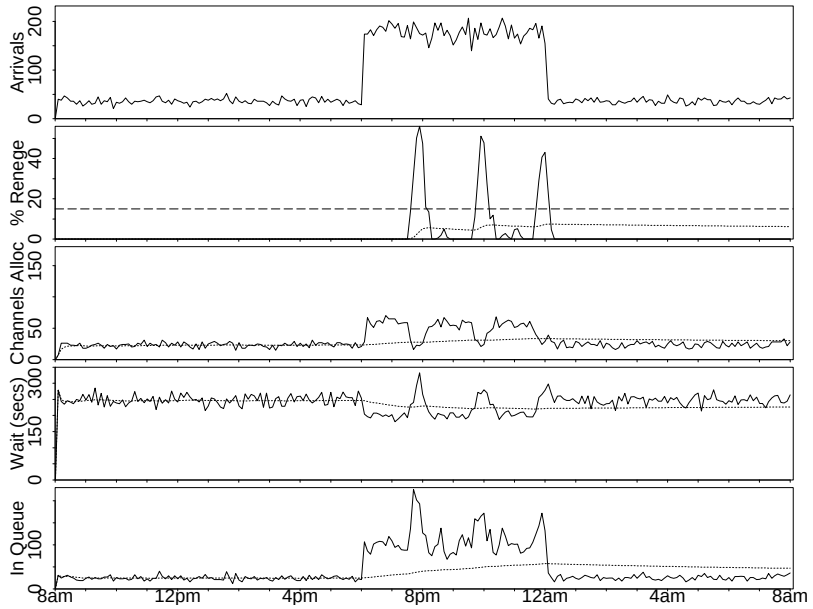

Figure 8: The forced wait policy with optimal minimum wait time.

4.3 Rate-Based Allocation Policies

Greedy allocation policies suffer from poor performance in terms of short- and long-term reneging and average wait time. In this section, we simulate our rate-based channel allocation policies and show how these new policies solve the problems of the greedy allocation policies.

4.3.1 Pure Rate Control

Figure 9 shows the performance of a system using pure rate control. The results show that pure rate control solves the problem of oscillations in short-term reneging and it has the best long-term reneging of any of the policies discussed so far at 4.1%. Cycles are eliminated because the number of channels allocated during the high load period does not spike, but only increases slightly. When the load is low the average wait time is almost zero, and there are few requests in the queue. During high load periods, channels are allocated at the maximum rate and some requests are queued. Because multiple requests are in the queue, the likelihood that the system can batch requests increases. This increases effective throughput. This is similar to the case of forced batching but pure rate control does not require a minimum wait time parameter and handles variations in workload automatically. Rate control ensures reasonable channel availability and wait times even during high load periods.

To show the robustness of pure rate control, we also show results for the workload with a gradual increase in load. The results, shown in Figure 10, are almost identical to those in Figure 9. In these results the long-term average reneging improves slightly to 3.9%.

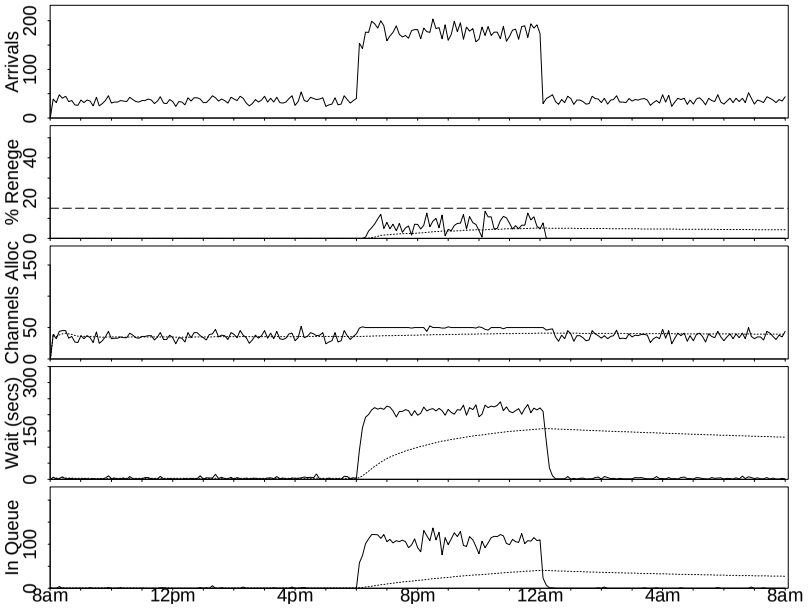

Figure 9: The pure rate control policy.

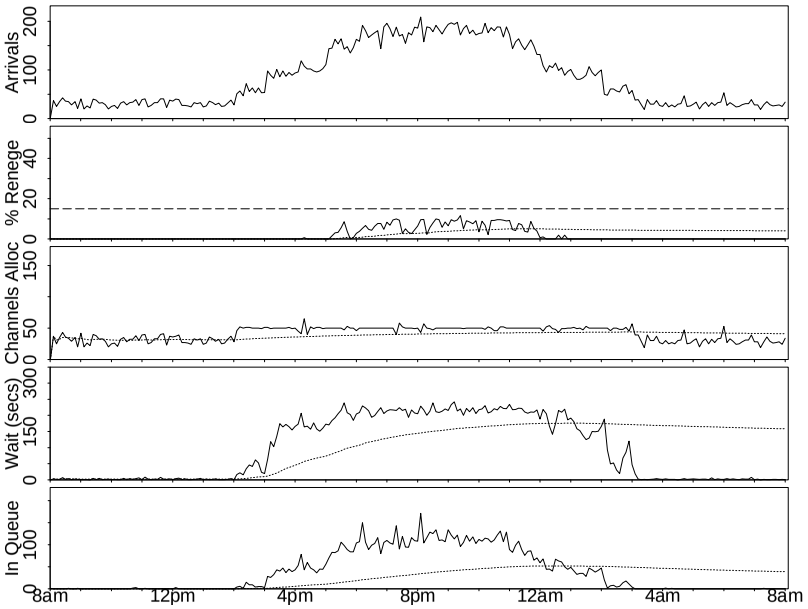

Figure 10: The pure rate control policy with gradually increasing workload.

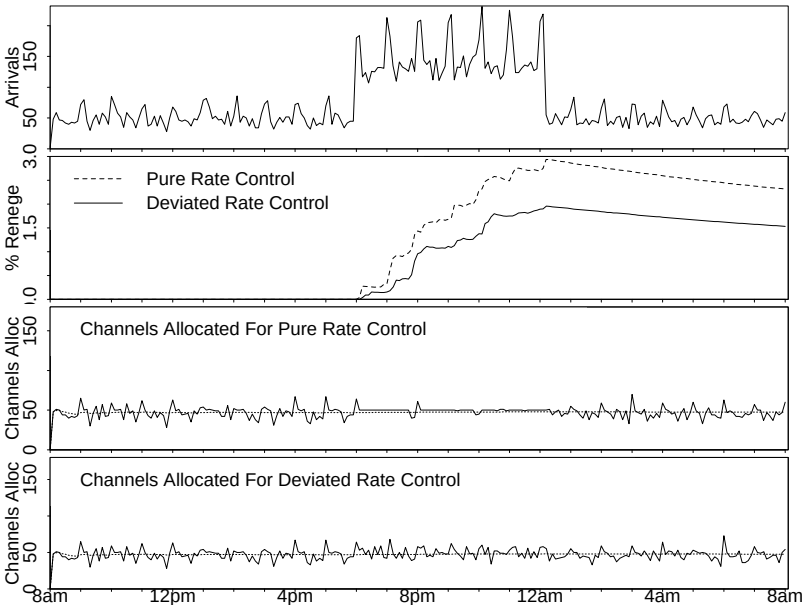

Figure 11: The deviated rate control policy.

4.3.2 Deviated Rate Control

Deviated rate control is designed to perform better when there is a need to be more flexible than what pure rate control allows. This is the case in the workload with hourly spikes. Figure 11 compares the performance of the two rate-based control schemes for this workload. As Figure 11 shows, the long-term averaging reneging is 2.3% for pure rate control and 1.5% for deviated rate control. For the results shown, the maximum allowed deviation parameter is set to 0.45 (determined to be optimal through simulation) and the maximum tolerable wait threshold is 5 minutes. Allowing the allocation rate to deviate gives the scheduler greater ability to handle short-term surges in demand. However, the tradeoff with deviated rate control is that it requires an operator to set additional parameters. In practice, it is unlikely that the small comparative advantage will justify the extra complexity.

4.3.3 Multiple Service Classes

Figure 12 shows the performance of the multiple service class allocation policy. Only during the high load period were there enough requests to make any of the objects hot. During the entire simulation no request for a hot object ever went unsatisfied (see the second graph in Figure 12). Because only a few objects were hot, fewer channels needed to be allocated for these requests (see the third graph in Figure 12). Wait time for hot objects was slightly less than that of cold objects (see the fourth graph in Figure 12). Finally, only a small percentage of the queue was actually for

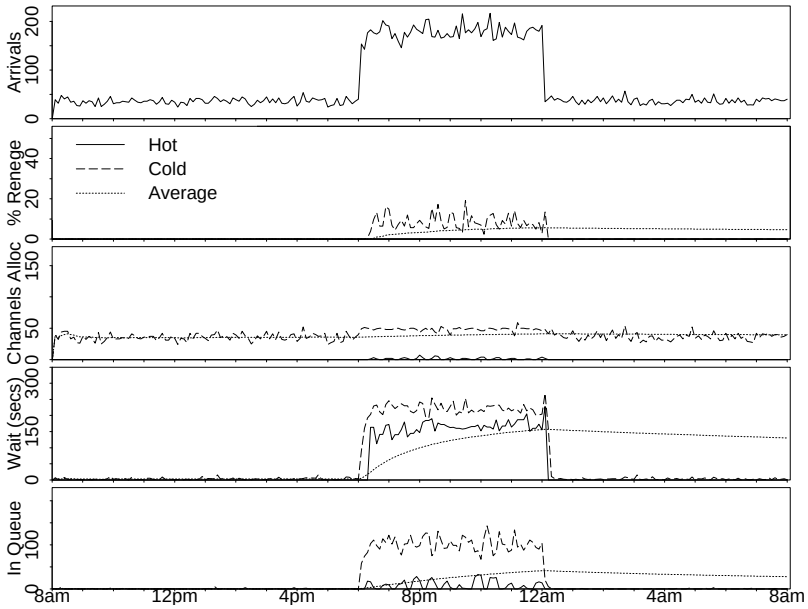

Figure 12: The multiple service class allocation policy.

hot requests (see the fifth graph in Figure 12). While in these results we used hot versus cold as our class criteria, other, equally arbitrary classes could have been used.

4.4 Effects of Other System Parameters

Having analyzed the performance of the greedy allocation algorithms and pure rate control in some detail, we now briefly look at the impact of varying the number of channels in a system and the number of objects available to the user. In Figure 13 results are presented only for (1) the final long-term average reneging and (2) the average wait time computed over an entire 24-hour simulation period. The results show that as the number of channels increases, performance increases. The rate at which reneging and wait time decrease is roughly the same except for the case of reneging for the on-demand policy which initially decreases more quickly and then more slowly. This result occurs because of the additional request aggregation gains that can be made with more channels and more frequent batching.

Figure 14 is similar in nature to Figure 13 except that instead of varying the number of channels, the number of objects is varied. Results are again presented for the long-term average reneging and the wait time. The graphs show similar results to those found in the more detailed system simulations. As the number of objects increases, request aggregation becomes less probable for all three systems and so reneging increases. Reneging increases at a slightly slower rate in the

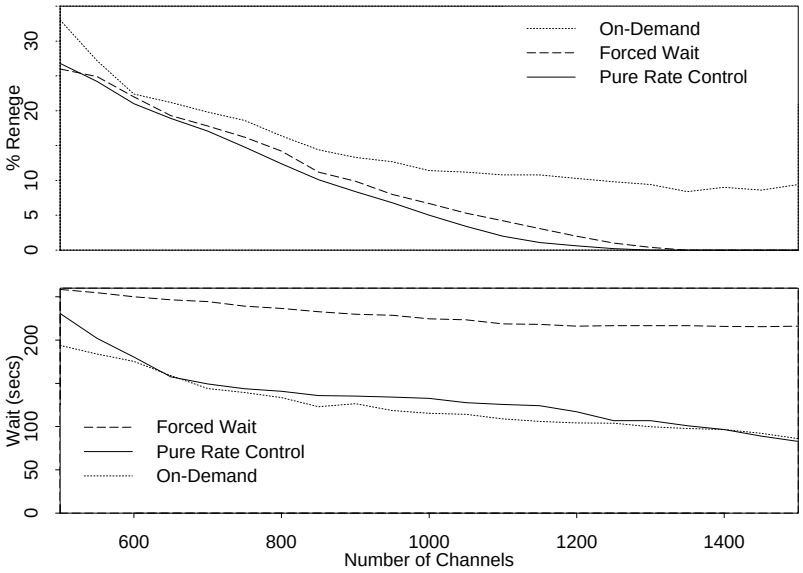

Figure 13: Effect of system size on channel allocation policies.

on-demand system because batching was already occurring less frequently. Average wait time for all three policies increases at first and then becomes almost flat.

4.5 Capacity Planning

An important aspect in the design of a content delivery system is determining the number of channels needed to provide a certain level of service. Instead of defining a system and then simulating the performance, we use a simulator to determine the minimum number of channels needed to meet a long-term average reneging objective of 5% (as measured over the entire 24-hour simulation period), and a short-term reneging objective of 15% (as measured over each 6 minute reporting interval). Results presented in this section are generated by iteratively simulating larger and larger systems until the short- and/or long-term reneging objective are met.

4.5.1 Long Term Reneging Objective

Figure 15 shows the number of channels needed to meet only the 5% long-term reneging objective. The workload model used is the same as the one used for the other results presented in this paper. The x-axis represents the average number of requests arriving per minute for the non-prime time hours (during prime time, the average request arrival rate is 5 times greater).

The results in Figure 15 show that as the request arrival rate increases, the number of channels needed increases quickly for a system that cannot batch requests and more slowly for batching-

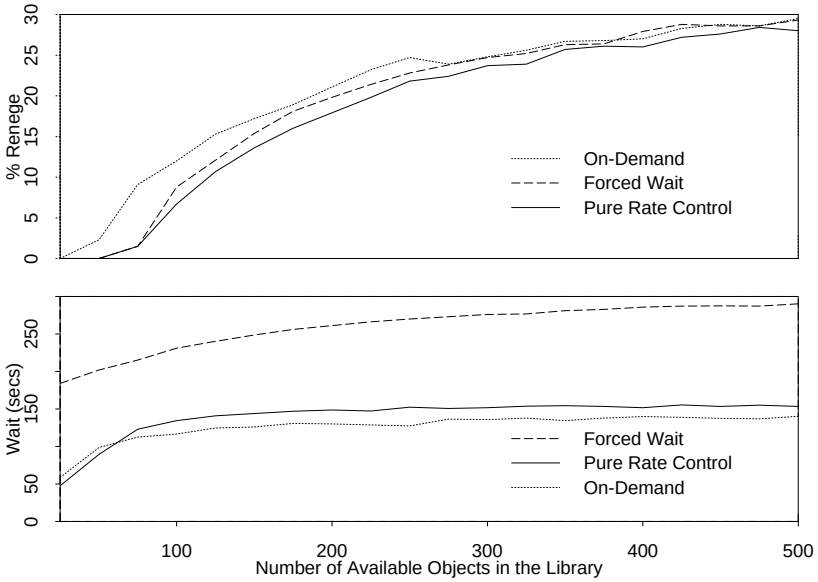

Figure 14: Effect of the size of the object library on channel allocation policies.

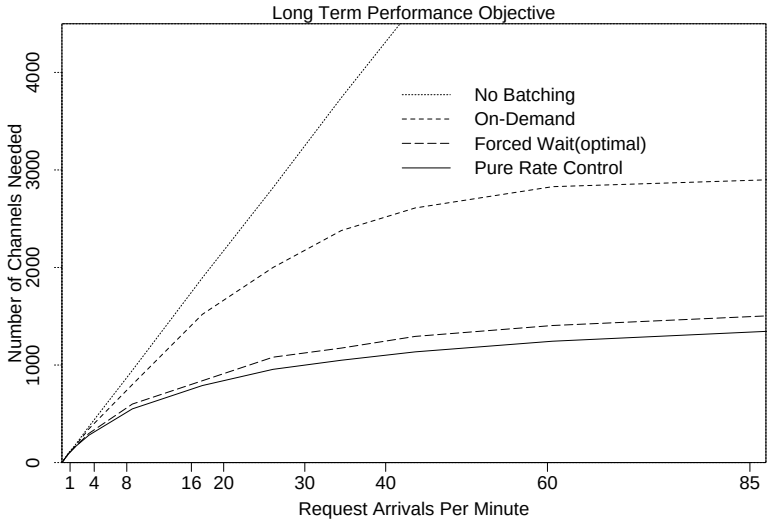

Figure 15: Required server capacity using a long term reneging objective only.

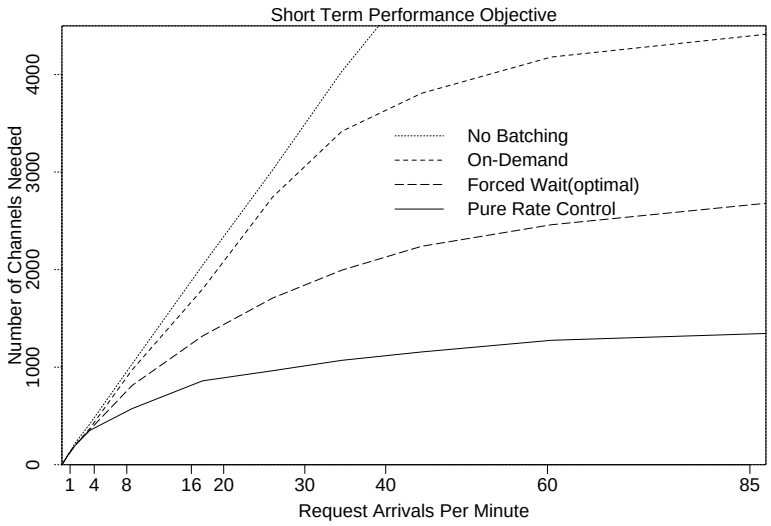

Figure 16: Required server capacity with short- and long-term reneging objectives.

capable systems. In the system allowing batching but implementing only on-demand allocation, the number of channels needed increases rapidly but then levels off at approximately 3000. For the optimal forced wait and pure rate control policies the number of channels needed levels off about 1000, with pure rate control performing slightly better. Because the only requirement is long-term average reneging, forced wait performs nearly as well as pure rate control, but as Figure 7 shows, forced wait has occasional very high short-term reneging. Generally, in systems that allow batching, the number of channels needed levels off as load increases since the effective capacity grows with increased request aggregation.

4.5.2 Short- and Long-Term Reneging Objective

Because predictable and consistent performance is as important as overall performance, both short- and long-term reneging objectives need to be considered. The results in Figure 16 show that for the no batching case, the slope of the line is slightly greater than in Figure 15 indicating that some additional channels are needed to eliminate cycles. For the on-demand policy, cycles are a significant problem and 25% more channels are needed to achieve both reneging objectives. The same holds true for the forced wait policy which requires more than twice as many channels. Finally, because pure rate control was designed to smooth peaks, additional channel requirements are negligible.

5 Characterization of Content Delivery Systems

Rate-based channel allocation algorithms have been shown to improve both short- and long-term performance for a content delivery system. We have developed a set of general characteristics which we believe are the basis for the performance gains. These characteristics are:

- **Finite resource pool**: Obviously there must be a finite number of resources, but the resource pool must be in relative equilibrium with the average user request rate. Systems that have significant excess capacity can handle load surges. Likewise, systems that are always heavily loaded will consistently see poor performance.
- **Long periods of resource use**: As an example, streaming VoD systems require resources to be reserved for long periods. In the general case, holding resources means they will be unavailable again for re-use for significant periods of time. Care must be taken not to allocate too many channels too quickly.
- **Short expected waiting time**: If response time is expected to be short, especially with respect to the duration of resource use, there is little a scheduling algorithm can do to smooth the effects of a varying user request pattern. Some amount of time is needed to make batching (and scalability) effective.
- **Small variations in average service time**: When there is little variation in service time, the allocation and freeing of resources become synchronized, and any oscillations that develop will cause long term cyclic behavior to develop. Even when a large percentage of resources are allocated, reasonable variations in service time mean resources will be freed at more even intervals, and cycles will disappear quickly.
- **Variability in the user request pattern**: Sudden surges in user requests is the catalyst that starts the cyclic behavior pattern. The optimal condition for the formation of cycles occurs when a system is operating at less than equilibrium and then has a surge to above-capacity. This first surge in requests typically causes the allocation of a large percentage of a system's resources. Cycles will then be perpetuated over a long period of time as influenced by the other characteristics listed above.

Object delivery, especially in the case of streaming video, is the quintessential example of an application that requires rate-based channel allocation. This includes content delivery in several different forms, and even over different infrastructures, including VoD over cable TV networks, and VoD over the Internet via web-driven interfaces. However, in addition to video delivery, there are a number of other applications that could benefit from rate-based resource allocation. One particular example is a medium access control protocol designed for cable modems[17]. The central problem is how to allocate "slots" for data transmission based on host requests. Variability in the user request pattern occurs when users desire to send bursts of data. Obviously there is a need for quick response, but the request mechanism is such that slots are allocated based on the maximum round trip delay, which can be quite long relatively. In these systems, all the conditions seem to exist leading to cycles and the need for rate based resource allocation.

Another example is the dynamic allocation of multicast addresses, especially for applications that use many multicast addresses for long periods of time. A specific example, is the delivery of web pages using multicast and the cyclic, redundant transmission of pages[12]. Once a multicast address is allocated, it may be in use for a long period. If addresses for a particular application are allocated from a small, fixed range, large-scale and long-term cycles may develop.

6 Summary

In this paper, we identify through simulation the limitations of greedy channel allocation policies, especially in the context of video delivery systems with multicast/batching capability. These algorithms are poor solutions because: (1) they do not minimize user waiting time, (2) they perform poorly when load conditions vary, and (3) they have wild oscillations in short-term performance. To solve these limitations, we propose a set of rate-based policies that work by ensuring that channels are available for allocation on a consistent basis. Not only do these policies achieve a consistent and predictable level of service in terms of user reneging, but they also improve effective capacity and resource utilization.

In this paper, we have developed four contributions. First, we develop workload models that are not steady state Poisson models and more accurately reflect user behavior over a 24-hour period. Second, we use a content delivery system model to simulate the performance and show the limitations of several greedy channel allocation algorithms. Third, we propose rate-based channel allocation algorithms that solve the problems of the greedy algorithms. Fourth, we generalize the content delivery model into a set of characteristics which make the performance gains of rate-based channel allocation applicable to other similar systems.

References

[1] S. McCreary and K. Claffy, "Trends in wide area IP traffic patterns, a view from ames internet exchange." http://www.caida.org/outreach/papers/AIX0005.

[2] K. Almeroth and M. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1110–1122, August 1996.

[3] S.-H. Chan and F. Tobagi, "Caching schemes for distributed video services," in *IEEE International Conference on Communications*, (Vancouver, British Columbia, Canada), June 1999.

[4] D. Eager, M. Ferris, and M. Vernon, "Optimized regional caching for on-demand data delivery," in *Proceedings of the SPIE - The International Society for Optical Engineering*, (San Jose, California, USA), pp. 301–316, January 1999.

[5] G. Dammicco and U. Mocci, "Program caching and multicasting techniques in VoD networks," in *Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS)*, (Darmstadt, GERMANY), pp. 65–76, September 1997.

[6] T. Choi, K. Young-Ju, and C. Ki-Dong, "A prefetching scheme based on the analysis of user access patterns in news-on-demand system," in *ACM Multimedia*, (Orlando, Florida, USA), November 1999.

[7] M. Reisslein and K. Ross, "High-performance prefetching protocols for VBR prerecorded video," *IEEE Network*, vol. 12, pp. 46–55, November/December 1998.

[8] S. Bakiras and V. Li, "Smoothing and prefetching video from distributed servers," in *International Conference on Network Protocols (ICNP)*, (Toronto, CANADA), November 1999.

[9] J. Gafsi and E. Biersack, "Performance and relability study for distributed video servers: mirroring or parity?," in *IEEE International Conference on Multimedia Computing and Systems*, (Florence, ITALY), June 1999.

[10] S. Stoller and J. DeTreville, "Storage replication and layout in video-on-demand servers," in *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Durham, New Hampshire, USA), pp. 351–362, April 1995.

[11] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *ACM Multimedia*, (San Francisco, California, USA), October 1994.

[12] K. Almeroth, M. Ammar, and Z. Fei, "Scalable delivery of web pages using cyclic best-effort (UDP) multicast," in *IEEE Infocom*, (San Francisco, California, USA), March 1998.

[13] L. Golubchik, J. Lui, and R. Muntz, "Reducing I/O demand in video-on-demand storage servers," in *ACM Sigcomm*, (Boston, Massachusetts, USA), August 1995.

[14] T. Little and D. Venkatesh, "Prospects for interactive video-on-demand," *IEEE Multimedia*, pp. 14–23, Fall 1994.

[15] Z. Zhao, S. Panwar, and D. Towsley, "Queueing performance with impatient customers," in *IEEE Infocom*, (Bal Harbor, Florida, USA), pp. 400–409, April 1991.

[16] G. Zipf, *Human Behavior and the Principle of Least Effort.* Reading, MA: Addison-Wesley, 1949.

[17] J. Limb and D. Sala, "A protocol for efficient transfer of data over fiber/cable systems," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 872–881, December 1997.