Understanding the Impact of Diverse Streaming Workloads on End-User Quality of Service

Mirjana Spasojevic, Nina Bhatti, Leonidas Kontothanassis, Sumit Roy
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2005-33
February 14, 2005*

E-mail: {mirjana.spasojevic, nina.bhatti, leonidas.kontothanassis, sumit.roy}@hp.com

streaming media, quality of service, performance, system measurements

Streaming media has experienced explosive growth over the last few years and will continue to increase in popularity as individual users can easily produce digital images and video. Large scale streaming will require network operators to deploy and manage a larger number of streaming servers within their networks. In order to determine the number of servers needed, and to design the appropriate management policies for a particular workload, administrators will need to understand the performance characteristics of those servers, how to measure performance, and how different workloads can affect performance. In this paper we make the case for a more complete measurement methodology by showing that streaming server performance can vary substantially based on the type of streaming workload, and, as a result, affect the end-user quality of service.

* Internal Accession Date Only Approved for External Publication

Understanding the Impact of Diverse Streaming Workloads on End-User Quality of Service

Mirjana Spasojevic, Nina Bhatti, Leonidas Kontothanassis, Sumit Roy
Hewlett-Packard Laboratories
Palo Alto, CA and Cambridge, MA
{mirjana.spasojevic,nina.bhatti,leonidas.kontothanassis,sumit.roy}@hp.com

ABSTRACT

Streaming media has experienced explosive growth over the last few years and will continue to increase in popularity as individual users can easily produce digital images and video. Large scale streaming will require network operators to deploy and manage a larger number of streaming servers within their networks. In order to determine the number of servers needed, and to design the appropriate management policies for a particular workload, administrators will need to understand the performance characteristics of those servers, how to measure performance, and how different workloads can affect performance. In this paper we make the case for a more complete measurement methodology by showing that streaming server performance can vary substantially based on the type of streaming workload, and, as a result, affect the end-user quality of service.

1. INTRODUCTION

As broadband deployment continues to grow, streaming media is increasingly gaining in importance as a communication and entertainment medium. According to a recent report by AccuStream iMedia Research [4], "... content distribution networks (CDNs) doubled – and in some cases tripled – revenue derived from their streaming media operations in '04 compared to '03, coinciding with an 80% jump in video streams served for the year and a 75% increase in Internet music hours." In addition to the growth of revenue associated with streaming media, the sheer number of streams is also on a steep growth curve. Figure 1 presents growth of the annual number of streams served in the US.

Streaming media companies are required to reconcile the demand for high quality end-user experience with the need to minimize cost and make the best use of their server infrastructure. In the past, servers have been managed simply by examining simple performance metrics like CPU, disk, and network utilization. Unfortunately, such metrics only partially capture an end user's experience and can often mislead a streaming provider either in under- or over-provisioning their infrastructure. We look at end-user experience through measurements of client-side metrics (i.e. failures, startup delays and re-buffering, jitter and thinning/loss of packets). Our goal is to understand if and how deterioration of an end-user's experience is tied to the observable server-side metrics.

WWW2005, May 10–14, 2005, Chiba, Japan.

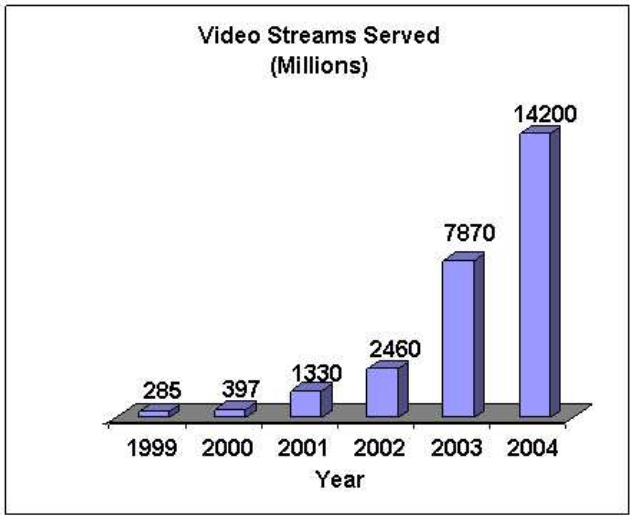

Figure 1: Annual number of streams served in the US for the 1999-2004 period.

In this paper, we propose an exhaustive methodology for evaluating performance of streaming media servers, covering both server and client-side measurements and a wide range of workloads. We illustrate and validate our methodology by extensively measuring the performance of an industry standard server over a comprehensive workload evaluation matrix. While it is intuitive that server behavior is workload dependent, we show exactly *how* the server behaves under different loads, and what parameters lead to server overload. Server performance is one of the critical components that affect end-user quality of service; the other being network effects. In this paper we have chosen to study server performance in isolation since it is independent of network effects (and vice-versa) and we believe this is the best approach to gain a full understanding of the server performance component.

Our evaluation shows wide disparities in server saturation points for the different workloads. We also show that while for some workloads, deterioration of end-user experience can be attributed to the saturation of a particular hardware resource, other workloads exhibit suboptimal end-user experience even when all hardware resources on the server are well below their full utilization. This discovery implies that looking at server-side metrics in isolation will either result

in conservative provisioning approaches that will increase infrastructure costs, or end up causing deterioration of end-user experience in some cases when the server is close to full hardware resource utilization. We have successfully applied this methodology to multiple servers, but due to space constraints in this paper we present just one. In a separate paper we apply the same methodology to another industry standard server, and for both servers demonstrate how the server-side and client-side metrics can be used to produce a performance model that accurately predicts an end-user's quality of service, based on the current server state and the set of incoming client requests [9].

The rest of this paper is organized as follows: Section 2 discusses the performance metrics and workloads with which we characterize a streaming server. Section 3 shows an evaluation of the Darwin streaming server [6] using our methodology. We discuss the general lessons learned from this evaluation in section 4. We present related work in section 5 and we conclude and present ideas for future work in section 6.

2. METRICS AND WORKLOADS

In this section we discuss our choice of metrics and workloads and present arguments as to why such an extended set of metrics and workloads is necessary in order to understand the relationship between streaming server performance and end-user experience.

2.1 Server side performance metrics

Server side metrics are by far the easiest ones to identify and understand. CPU, memory, disk, and network identify the critical resources of any modern computer system. Each of those metrics can reach a saturation point independently. CPU utilization indicates whether the server processor can keep up with the tasks associated with serving the streams. A saturated CPU will miss tasks, fail to process requests, and result in a suboptimal user experience. Memory exhaustion in streaming workloads is an unlikely problem in modern systems. However, if the main memory of the server is exhausted, and the system starts paging, performance deteriorates rapidly and CPU utilization spikes. Disk and network utilization indicate how much of the available bandwidth from these two subcomponents is being used by a particular workload. It is possible to saturate either of those components before CPU utilization reaches 100%, and thus they have to be monitored independently rather than be proxied by the CPU utilization of the server.

2.2 Client side performance metrics

We have chosen five client-side metrics to determine whether a streaming server is failing to handle a particular workload: play failures, rebuffering, startup latency, thinning percentage and jitter.

- Play failures experienced by a client indicate either that the server is so overloaded that it can no longer handle requests.[1] Play failures are visible to the end-user and may appear as initial *startup failures* as the request for content is being made, or as *in-play failures*, which may happen at any point during the content playback.
- Rebuffering on the client indicates that the server is experiencing overload and is failing to send packets to the client on time. When a client buffer underflows, play stops until sufficient data is received again. Rebuffering will often precede play failures and tends to be an earlier indicator of server overload. However, it is possible for a server to perform admission control and refuse to accept extra connections rather then fail to serve them adequately. In this case we may end up seeing play failures instead of rebuffering events.
- Startup latency is the elapsed time from the moment a request for content is made to the moment when the first packet arrives. This latency may not be visible to the end-user, as it is often masked by the initial buffering of content. However, increased startup latency experienced by a client indicates that the server is falling behind in processing new requests. Depending on server policy, overload behavior may result in either increased startup latency (when deprioritizing new requests), rebuffering (when deprioritizing existing stream serving in favor of handling new requests), or both.
- A high thinning percentage indicates that the client is receiving the stream at a bitrate that is lower than the one at which the stream was encoded. One would expect a client to rebuffer when it is not receiving adequate data from the server, however, in certain cases a client will continue to play even though it is not getting a full stream from the server. These cases involve streaming formats that can degrade gracefully, either in the time domain or the quality domain. In the first case, the server only sends certain frames of a stream and drops others; for example an overloaded server can send just the keyframes of a stream. This will result in the client displaying something akin to a slideshow, but not in rebuffering. Streaming media can also be encoded into multiple quality layers, where each additional layer improves the quality perceived by the client. Only the first or base layer is required to be decoded by the client for continuous playback. The server may drop additional or enhancement layers based on available resources.
- Jitter captures the delay in packets sent by the server as seen by the client. For a stream encoded at a particular rate, each packet is expected to be sent by the server and arrive at the client with a predetermined deadline. Late packets indicate that something could be going wrong and may be an early sign of more serious problems. Specifically, a small amount of delay in packets will have no effect on the end-user experience due to the buffer that clients build before they start playing. However, as delays increase the client buffer can be depleted and rebuffering events may occur. Therefore, large jitter is a necessary but not sufficient condition for rebuffering and can perhaps aid us in identifying safe operating regimes more reliably.

Even though there is no consensus on what constitutes an acceptable operating regime from a client metrics perspective, we have chosen conservative thresholds for our metrics

[1] It may also indicate a problem with the network link between the client and server but this is beyond the scope of defining a performance model for the server and in a controlled experimental setting, we can ensure problem-free network links.

to ensure that we catch problems earlier rather than later. Our choices are outlined and explained in section 3.

2.3 Workloads

In selecting our workloads we have identified three dimensions that have an important effect on how resource utilization interacts with end-user experience. The three dimensions are: streaming type, content popularity and encoding rate.

2.3.1 Stream Type

The first dimension (henceforth referred to as the *Stream Type Dimension*) concerns the type of streaming requested. Live streaming has very different properties than on-demand streaming for a number of reasons [13]. First and foremost, live streaming imposes no demand on the disk subsystem of a server since all data is received via a network connection. Second, every client receives the same data at the same time, while this is not the case for on-demand streaming, even if clients request the same stream. Finally, servers have to prioritize the receipt of streams from live sources over the sending of streams to clients. Losing packets from a source will affect all clients and thus data receipt is critical for live streaming.

In contrast, for on-demand streaming, reading data from a disk does not have the same real time constraints and a loaded server can choose on whether to prioritize disk accessing or client serving. At the same time, the offset in time for different requests requires the server to manage each request independently.

It is also important to consider situations where a server provides a mixture of Video-on-Demand and live streams, thus having to make complex tradeoffs.

2.3.2 Popularity

The second dimension (termed the *Popularity Dimension*) covers the popularity of the streams being served. In the case of on-demand streaming, high popularity streams impose less demand on server resources, since they can be served out of a buffer cache, rather than requiring disk accesses. For live streaming, high popularity streams imply less buffer management overhead and less memory pressure on the server. A server that has to support a large number of different on-demand clips is expected to experience much higher load. Unpopular live content could be the result of video instant messaging, or personal 'web-casts' (e.g. birthday parties). The workloads covered in this paper belong to the two extremes in terms of stream popularity: a case where all requests are for the same stream and a case where all requests are for different streams.

2.3.3 Encoding Rate

The final dimension (henceforth termed the *Encoding Rate Dimension*) covers the encoded bitrate for the streams being served. It is important to look at the encoded bitrate for the streams being served rather than just the aggregate bit demand by the clients. The reason is that a server experiences overhead for each additional stream that it has to manage and also experiences overhead for every packet it has to send to clients regardless of the actual packet payload. In order to achieve the same aggregate bit rate with a stream encoded at a lower rate, a server would have to manage additional connections, and often send more packets since packets for streams encoded at lower rates tend to be of smaller size. Hence, one would expect that, for a fixed CPU budget, a server handling low bitrate streams will achieve a smaller aggregate bitrate than a server handling high bitrate streams.

	Popular Streams		Unpopular Streams	
	High Rate	Low Rate	High Rate	Low Rate
VoD	VPH	VPL	VUH	VUL
Live	LPH	LPL	LUH	LUL
Mixed	MPH	MPL	MUH	MUL

Table 1: Streaming workload matrix

3. PERFORMANCE EVALUATION

In this section we look at the different workloads we proposed in section 2 and show how they affect server and client-side metrics as the server reaches saturation. On the server side we measure the CPU, memory, network and disk utilization. On the client side we measure errors (startup and play failures), rebuffering events, startup delay, thinning and loss statistics and packet arrival jitter. We consider the client side metrics to be in the acceptable range if there are no failures in play. In addition, we consider a streaming session as having acceptable performance when total waiting time (spent in startup, initial buffering, and rebuffering events) represents less than 3% of total play time and thinning/packet loss is no more than 3% of the encoded content. This definition is based on what is expected in order to receive the highest score in certain professional streaming measurement services [12]. We demonstrate that, depending on the workload, a different server-side resource may become a bottleneck.

In order to make it easier to refer to each workload we use the following naming scheme: The first letter describes whether the workload is live, video-on-demand, or mixed (L,V, or M respectively), the second letter describes whether the workload is for popular or unpopular streams (P or U), and the final letter captures whether the workload is based on high or low bitrate streams (H and L).

Since our workloads have three dimensions, with two possible choices for two of the dimensions and three choices for the third dimension, we end up with a workload matrix with twelve entries. This is summarized in table 1. We have organized the rest of the section across the *Stream Type Dimension* for the sake of clarity.

3.1 Experimental Setup

We have developed a client application that can request an RTP [14] stream (we only looked at true streaming as opposed to streaming over HTTP) from a media server, accept the packets and report statistics about play failures, startup delay, packet delays, and lost packets. Our client application does not render the received packets and thus is significantly less CPU intensive than typical streaming players. We can therefore run a large number of clients on a single machine and put substantial demand on a server with only a modest number of client machines. We configure the test clients to wait 500 milliseconds between initial requests during the "ramp up" period when starting the tests. We ignore data until the full concurrency level is achieved.

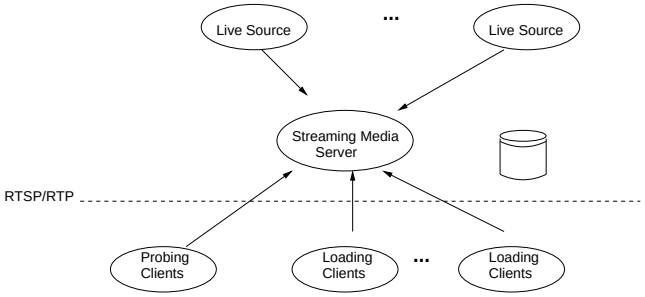

Figure 2: Experimental setup, Streaming Media Server with Live Source generators, probing and loading clients.

The physical hardware for our experiments consists of a number of *hp Netserver lp1000r* machines, with dual 1.4 Ghz Pentium III processors, 1 Gbyte of memory, running unmodified SuSE Linux 8.2. The server machine runs the *Darwin Streaming Server* [6]. The content is on a single SCSI 36 GB hard drive with a throughput of 50Mbytes/sec as reported by `hdparm(8)`. Logging is done on a separate 18GB SCSI hard drive. The server is connected to an *hp Procurve 5304XL* switch via Gigabit Ethernet connection. The client machines are connected to the same switch using 100 MBit connections.

We ensure that resource utilization on the clients stays comfortably low (CPU, network and memory utilization below 50%; there is no appreciable disk activity on the clients). This way any errors reported are guaranteed to be due to server overload rather than client artifacts.

For our experiments, servers are subjected to different workloads and levels of concurrent requests from our client machines. The client machines maintain logs for each request including completion times, errors, thinning, missed deadlines, and startup times. We use statistics from a "probe" client after an experiment has reached steady state. The probing client collects the same set of statistics as the clients used to induce load and the results from those probes are the ones reported in this paper. Server performance metrics for the duration of each experiment are collected on the server itself using statistics collected by the Linux kernel and reported under the */proc* filesystem. The clips used during experimentation were encoded at 300Kbps for the high bitrate, and at 78Kbps for the low bitrate. Each experiment lasted 20 minutes, during which client side statistics were collected from 20 sequential probes, lasting 60 seconds each. The numbers presented in the subsequent sections for startup latency and jitter are averages of those values.

The particular choice of bitrates is motivated by the assumption that, in the future, video content will be more and more viewed and created on thin mobile clients such as cellphones and PDAs. Content produced by traditional sources like television broadcasts or training videos would be adapted to the downstream bandwidth and screen size of the device. Content originating from camera phones would be even more constrained, since transmitting data requires more power than receiving it. In addition, pure audio streams also tend to have a lower bit-rate. Presently, the lower bitrate content is also very much present in the content served by various content providers. The deployment of broadband access to homes will help migrate content to higher bitrates, but that has not happened yet.

For our video-on-demand experiments the clips reside on the local disk. For our live experiments the clips reside on a separate machine, where a pseudo-encoder reads them and sends them to the server over the network, emulating the behavior of a live streaming setup. Figure 2 show the logical experimental setup.

In the next subsections we present results of experiments from video-on-demand, live and mixed workloads. While we collect data for four server side metrics and five client side metrics, we chose to present only plots of CPU utilization, startup delay and jitter. We found those metrics to be the most interesting to monitor as the concurrency level of the workload changes.

3.2 Video-on-Demand Workloads

3.2.1 Popular Streams

We measured the performance of popular streams by configuring our loading clients to all request the same video clip. For the popular high bitrate tests (VPH) shown in Figure 3 we see no client errors for up to 438 concurrent requests. Low bitrate popular clips (VPL) have no errors until reaching 780 requests (Figure 4). These results affirm our earlier hypothesis that low bit rate clips put a higher demand on the server than high bit-rate clips, for the same aggregate network throughput. This is particularly important with wide scale adaptation of mobile wireless clients, that would typically request low bit-rate clips.

The behavior of other client side metrics is more nuanced. Startup latency increases with the level of concurrency and jumps by more than a factor of two as the server approaches its saturation point. Jitter appears to be bimodaly correlated with load as it starts high on a lightly loaded server, is reduced for medium loads, and then increases again as the server approaches saturation. Our analysis of the client logs indicates that the higher jitter at low loads is due to packets that arrived too *early*. Since jitter is defined as the standard deviation of the difference between actual and expected packet arrival times, it is affected just as much by early as well as late packet arrival. Given that early packet arrival is unlikely to have any adverse user effects on most computer platforms, we would encourage the community to re-evaluate the definition of jitter and only take into account late arriving packets.

Rebuffering does not appear to be of particular concern as few occurrences are observed when the number of client increases. This is in contrast with a previous study [8] that showed rebuffering as the client side metric to be first affected by an overburdened server. Since different media servers prioritize different tasks differently, it is possible (even likely) that a different part of the client experience gets affected when the server reaches saturation.

Observing the server side metrics for these workloads shows that the CPU and memory utilization of the server increases as the number of clients increases. However, at least for the VPH workload, client experience deteriorates while CPU utilization is still around 70% and no other server side resource shows saturation either. This lends credence to our hypothesis that monitoring four basic server-side resource alone is insufficient in guaranteeing good end-user experience. One can set very conservative thresholds for hardware resource utilization that would guarantee quality of service

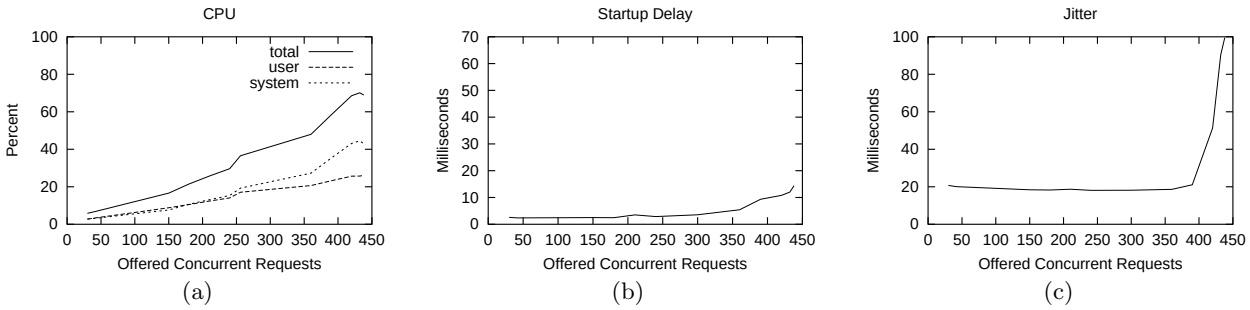

Figure 3: VPH, Video-on-Demand, popular, high bitrate (300Kbps).

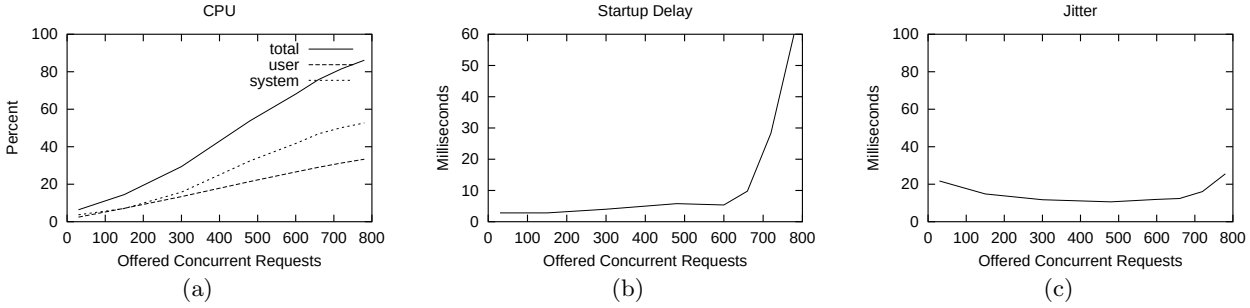

Figure 4: VPL, Video-on-Demand, popular, low bitrate (78Kbps).

but that is undesirable for obvious economic reasons.

Another interesting observation is the fact that the VPH workload achieves a much higher bit throughput than the VPL workload (131.4 Mbits/sec vs. 60.8 Mbits/sec), despite the lower client count saturation point (438 clients vs. 780 clients) and while serving approximately an equal number of packets (20,361 packets/sec vs. 20,438 packets/sec). Two effects account for the difference. The first is the increased overhead that the server incurs when dealing with more connections as is the case in the low bitrate experiment. The second is that lower bitrate connections tend to send packets of smaller payloads, consistent with design of many codecs which tend to put a frame into each packet. Therefore, a larger number of packets needs to be sent by the server in order to send the same number of bits. The per packet overhead consumes a significant fraction of CPU resources resulting in reduced bit throughput.

3.2.2 *Unpopular Streams*

To measure the performance of unpopular streams, we configured the clients to request a unique clip for each request. As expected, the VUH (Figure 5) and VUL (Figure 6) workloads start experiencing failures much earlier than their popular counterparts, because the content cannot be cached and must be retrieved from disk.

In particular, the VUH workload encounters errors at 37 concurrent requests with a very abrupt cliff behavior; i.e. an experiment with 36 concurrent requests experiences no errors, but one with 37 concurrent requests results in all requests failing. Such reduction in serving capacity is much more than would be explained by disk accesses alone. By using the `oprofile` [1] performance analysis tool we were able to determine that most of the CPU was spent in copying data between kernel and user space. We traced this behavior to thrashing in the server management of user level stream buffers. The Darwin server performs aggressive prefetching of clips it serves from the disk. However, the default amount of user level buffer space allocated to clips is not sufficient to accommodate the prefetching necessary for high bitrate clips. Therefore, data from the user level buffers is replaced by the newly prefetched data before it has a chance to be used, forcing the server to re-read it multiple times. These re-read requests do not actually access the disk system as they can be satisfied by the kernel buffer cache, but result in excessive data copying and CPU saturation at a relatively small number of requests. Of our other client metrics, the startup time and jitter are significantly affected by the increase in the number of request (Figure 5). However we saw no thinning or rebuffering for this workload.

Low bit rate streams do not suffer from the read ahead anomaly and as such do exhibit higher concurrency capacity. More specifically the server appears to reach saturation at 175 concurrent requests with 4% of all requests failing at that level. Startup latency increases drastically once we hit the 160 concurrent requests level, more than doubling from 12 milliseconds to 33 milliseconds and continues to increase further as offered load increases. Jitter, rebuffering, and thinning show little change as the load on the server increases even as it reaches overload levels. With respect to server-side metrics, this particular workload shows very light CPU utilization (21%). Disk bandwidth utilization is also far from peak disk bandwidth, even though it is clear that disk is the bottleneck resource. We have discovered that the explanation for this behavior lies in the nature of

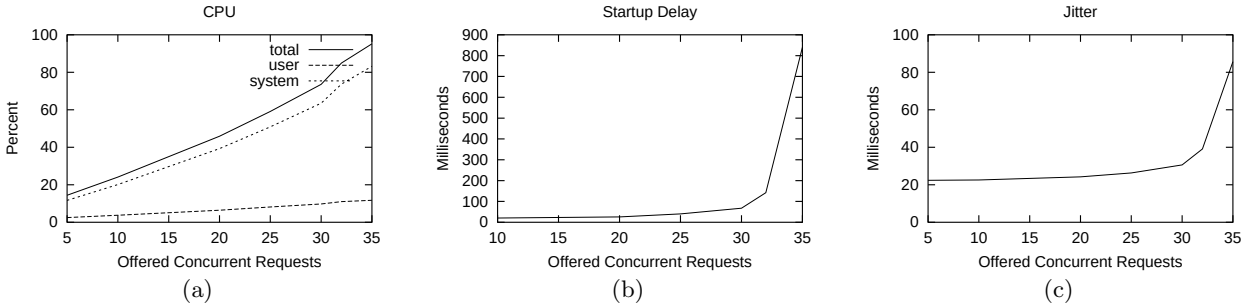

Figure 5: VUH, Video-on-Demand, unpopular, high bitrate (300Kbps).

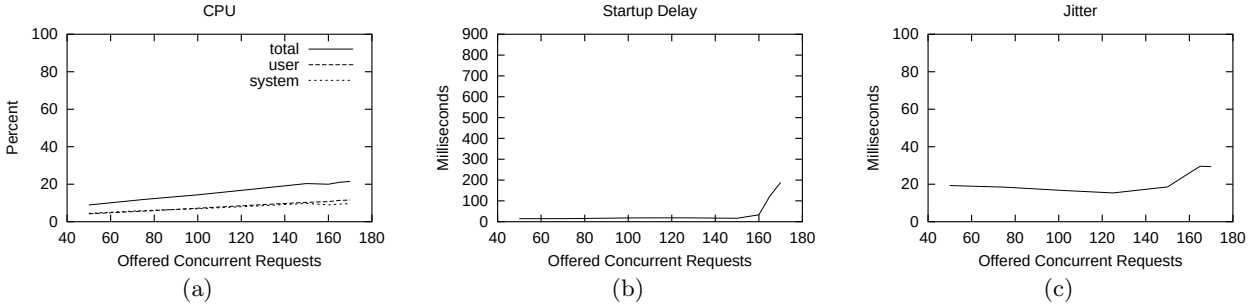

Figure 6: VUL, Video-on-Demand, unpopular, low bitrate (78Kbps).

disk reads which include substantial amounts of seek time. This significantly reduces the disk utilization and limits the throughput of the system. Obviously, this problem can be mitigated by using multiple disks and better file layout, but our goal was in studying an off-the-shelf server rather than designing a hardware architecture optimized for streaming tasks.

3.3 Live Workloads

3.3.1 Popular Streams

To measure the performance of popular live streams we set up an encoding station which relayed the same clips that was used in the VPH and VPL experiments to the streaming server. We then configured our clients to request this clip from the server. Upon receiving a request the server accepts any incoming packets from the encoder and relays them to the requesting clients.

The LPH workload achieves much higher throughput than any of the Video-On-Demand workloads with the server being able to handle 996 requests before encountering any errors (see Figure 7). Startup latency again grows with the number of clients, and the rate of increase becoming more pronounced as the server approaches overload conditions. Live workloads show a linear increase in the amount of jitter as the number of requests increases. Since increased jitter implies packets that could be arriving late we would expect rebuffering and thinning to also be present. To our surprise this was not the case with both metrics showing no adverse effect. The reason for this is that the server starts each streaming session with a burst of packets sent at a much higher rate than the encoded rate of the stream. This allows clients to build enough of a buffer to cushion the effects of late arriving packets down the road.

As expected, CPU is the limiting resource on the server-side. However, network utilization approaches 300Mbits/sec and would easily overwhelm a Fast Ethernet interface card, indicating that many typically configured servers would run out of network bandwidth before saturating other resources.

The LPL workload scaled beyond our testing capacity. At 1000 clients CPU utilization was only around 60%. It is therefore difficult to establish the scalability limitations for this workload. However, we see that the server is less efficient in moving bits through the network for reasons similar to those encountered with the VPL workload. Additional connection and per packet management overhead results in a reduction of the server bit throughput. For equivalent points of CPU utilization, the server can push more than three times as many bits under the LPH workload than it can under the LPL workload.

3.3.2 Unpopular Streams

For this benchmark we had to setup a number of encoding stations each producing a separate live stream that was then sent to the server. Since sending a live stream to the server incurs some overhead for the server even if no client is requesting this stream we ensured that the number of encoded clips matched exactly the number of requested clips for every experiment we run.

Live streaming of unpopular high bitrate streams (LUH) saturated our server at approximately 205 concurrent streams (Figure 9) at which point we started encountering errors. This number is almost a factor of five smaller than the equivalent number achieved under the LPH workload. The per-

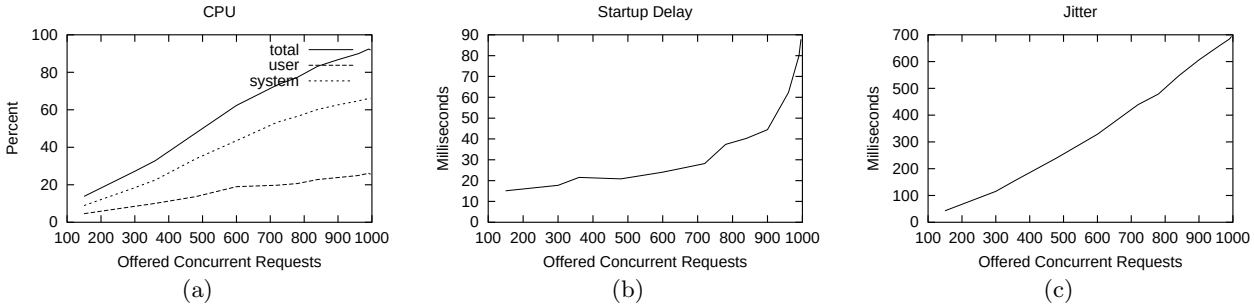

Figure 7: LPH, live stream, popular, high bitrate (300Kbps).

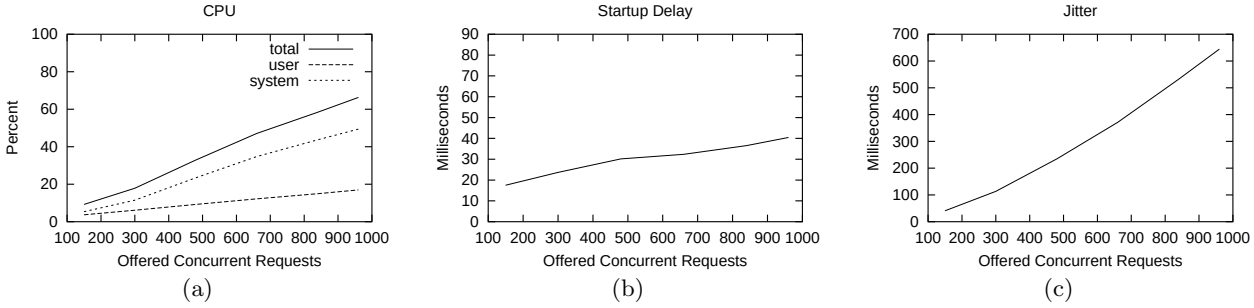

Figure 8: LPL, live stream, popular, low bitrate (78Kbps).

formance discrepancy can be explained by two causes. By far the most important is the fact that the server has to read a significantly larger amount of data from the network since there is a separate encoded clip for every client making a request. Second, the server has to read this data over a larger number of sockets (one per clip) which is known to cause some performance problems with the Linux `select(2)` implementation. None of the client side metrics show any marked deterioration during the ramp-up to server overload.

Unpopular low bitrate streams (LUL) exhibit behavior quite similar to that of their higher bitrate counterparts. Since the graphs for this workload are quite similar to those of the LUH workload we have omitted them for brevity. Saturation is reached at 405 concurrent requests with peak CPU utilization reaching 96% before errors occur.

3.4 Mixed Workloads

Mixed workloads demonstrate significantly different behavior than pure live or pure Video-on-Demand workloads. For an equal mix of Video-on-Demand and live popular high bitrate requests (MPH) the server reaches a saturation point at 552 concurrent streams (Figure 10). However, instead of seeing outright failures as was the case in the pure workloads, we see a large reduction in the number of bytes received by live streams due to lost packets. Since network conditions are carefully controlled we can be certain that the packet loss is due to server overload rather than networking artifacts. This implies an internal server scheduling problem that penalizes live stream requests relative to the on-demand ones when the server is under stress. Similar to the other workloads however, this significant reduction in received bytes is presaged by a sharp increase in the startup latency of the streams. We also see a significant increase in the amount of jitter experienced by the live streams, but much less so for the on-demand ones. Finally, we observed rebuffering events for majority of the live probe requests as the server reaches the saturation point. While we had evidence that server side metrics are an insufficient indicator of client side experience from our pure workloads, mixed workloads provide additional proof that this is indeed the case. Overall, out of a total of nine workloads presented in this paper at least two had pre-overload ranges just prior to stream failures becoming visible and end-user quality degradation, even though no server side resource was clearly saturated. That number would rise higher if we were to focus mostly on mixed workloads but it is sufficient in making the case that probes used to obtain client side observations are a necessary tool for establishing a streaming server performance model.

4. DISCUSSION

We summarize the saturation points we encountered for each of our workloads in table 2. To-date our experiments are characterized by the emergence of the following lessons:

- Different workloads affect server capacity differently. In particular we find that all dimensions of our workload matrix –type of streaming, popularity of streams, and encoding rate– are significant in trying to assess media server capacity. As a general rule however, live streams produce higher network throughput than Video-on-Demand streams, popular streams produce higher network throughput than unpopular ones, and high bitrate streams produce higher network through-

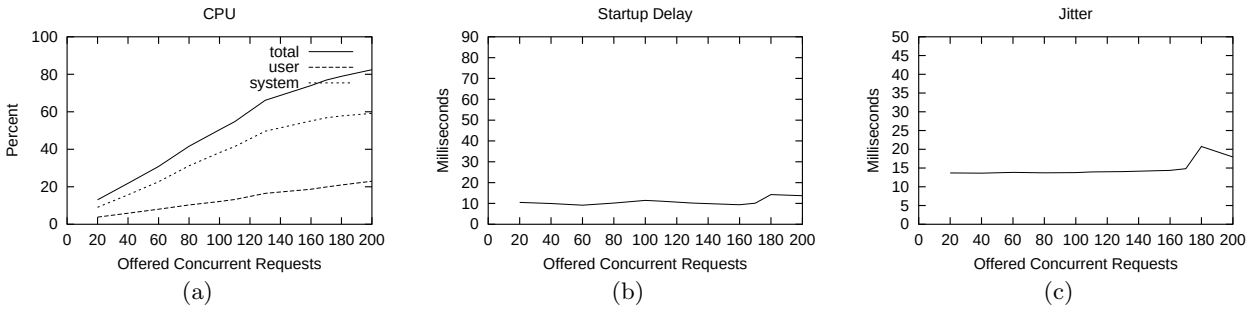

Figure 9: LUH, live stream, unpopular, high bitrate (300Kbps).

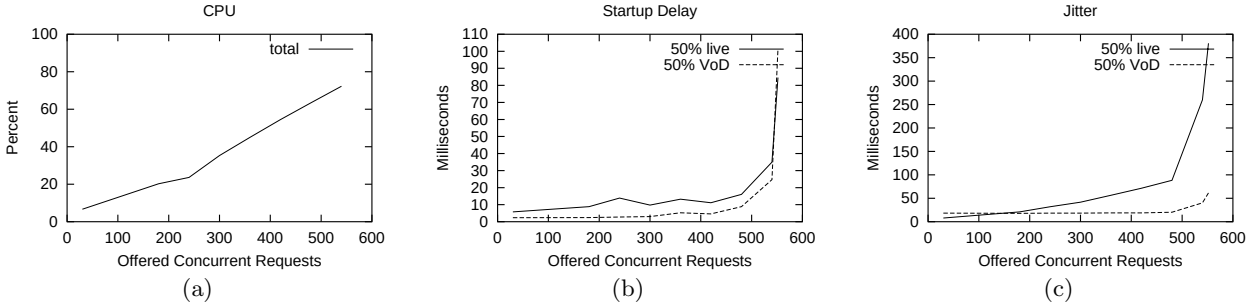

Figure 10: MPH, Video-on-Demand (50%) and live (50%) streams, popular, high bitrate (300Kbps).

put than low bitrate ones.

- Of our client side metrics, startup delay is most often affected by changes in the level of concurrency. This metric can easily be obtained through self-probing at the server side. However, it is important to look not at the absolute value of startup delays since that varies quite a bit between different workloads, but at the change in value as the load on the server increases. For example, absolute values for Video-on-Demand clips tend to be much lower than for live clips, but all workloads show marked increases in startup latency as the load gets higher. We do see increased jitter and rebuffering for some of our workloads, especially as the server reaches saturation. To our surprise, none of our experiments demonstrated the existence of thinning. As it turns out thinning can occur only when the client indicates that it is capable and willing to accept thinned content. We verified that thinning can indeed occur by starting a Quicktime player against a loaded server and observed the slideshow effect.

- Of the server resources we monitored, three – CPU, disk, and network – have shown the ability to be bottlenecks for the workloads we tested (although network bandwidth would only be a problem with a 100 Mbit Ethernet card and not with a Gigabit card). Memory in general never appeared to be much of an issue with utilization staying comfortably below the 1 Gbyte of memory available on our server. Furthermore, depending on the workload it could be a different resource that becomes the bottleneck. More interestingly, we discovered that for certain workloads we encountered errors before any resource was fully saturated. This implies some inefficiencies in the server scheduler and showcases the need to have access to client side metrics when designing server management and scheduling policies.

- Disk resources may appear severely underutilized for certain workloads even though in reality there are at a saturation point. The reason is that streaming workloads that are disk intensive have a heavy component of disk seeks, rather than being pure disk reads. This limits disk utilization by almost an order of magnitude.

- Logical resources can be just as much a bottleneck as physical resources (i.e. the amount of user level buffers available for the VUH workload). While those can often be resolved through appropriate server configuration, it is unclear that a single server configuration can accommodate all types of workloads. It may therefore become important for a cluster or CDN environment to have multiple servers configured specially for each workload and funnel requests to the appropriate server based on the request characteristics.

Using the lessons learned we can come up with a few simple rules and insights for managing clusters of streaming servers. Our first rule is that monitoring server-side metrics is not a sufficient indicator of end user quality, and that periodic streaming probes are a valuable addition to performance monitoring. A second rule is that maximizing stream popularity by carefully assigning requests for a stream to servers already servicing that stream can greatly enhance server efficiency. We have also found that server efficiency

	Popular Streams		Unpopular Streams	
	High Rate	Low Rate	High Rate	Low Rate
VoD	438 or 131.4Mbps (VPH)	780 or 60.8Mbps (VPL)	36 or 10.8Mbps (VUH)	170 or 13.3 Mbps (VUL)
Live	996 or 298.8Mbps (LPH)	996++ or 77.7++Mbps(LPL)	200 or 60Mbps (LUH)	405 or 31.6Mbps (LUL)
Mixed	552 or 165.6Mbps (50% VPH, 50% LPH)			

Table 2: Saturation points for workload matrix, maximum concurrent streams and maximum bandwidth throughput without errors.

depends more on the stream packet rate than the stream bit rate. Encoding low bit rate streams to also exhibit low packet rates can increase server capacity. Finally, additional performance benefits can be had by separating requests for on-demand and live streams to different servers and by converting on-demand requests to live streaming whenever possible (e.g. by regular scheduling of live streaming for the very popular content).

5. RELATED WORK

Cherkasova et al. [7, 8] provide one of the first analysis of media workloads and performance under video-on-demand workloads of popular and unpopular content. They identify important client side performance metrics, namely jitter and rebuffering. The paper also recognizes the need to measure the basic capacity of the server under different workloads. Our work extends both the workload space by examining live streams in addition to video on demand, as well as considering their mix, and the client metrics space by looking into failures, startup latency, and thinning. Our premise is that one needs to look at all important client side metrics since it may be a different one that gets affected under a different workload or even a different streaming server.

Another study that deals with application level quality measured the effect of network loss and bandwidth variation on client metrics of rebuffering rates, rebuffering duration, streaming data rate, and packets arriving after their deadline [16]. In this study the network conditions are manipulated to go through a variety of loss and bandwidth rates. Client statistics are gathered to measure how well the Windows Media Player (WMP) adapts to the changing network conditions. The client communicates with the WMP about the network conditions which the server then uses to adapt sending behavior. This study does not address the issue of server load and performance, but rather network effects on performance. But the authors did note that in general the server could adapt play rates for a variety of network conditions but occasionally noticed high server loads which negatively impacted the ability to adapt to network conditions.

Dalal and Perry [10] also focus on application level measures by collecting self reported performance data from the Windows Media Player. These measures are collected by either passive observation of these statistics when users initiate streams or by deploying active probes that periodically cause the clients to request streams. Received bandwidth, packet losses (arrival too late or missing), packet retransmission successes, received packets and rebuffering events are captured. These distributed performance probes are collected for report generation by the measurement infrastructure. These probes were analyzed looking for early predictors of rebuffering events. They found that small packet loss intervals are not good predictors for rebuffering events, but that very large packet loss intervals are.

Many efforts have put forth analyses of media traffic and tools that can generate synthetic workloads that mimic real traffic. Using these workload generators, a CDN or media server could be evaluated under "real" workloads. From these tests we could generalize the behavior under real operational use. GISMO [11] generates streaming access workloads which can be configured to generate requests that follow distributions for object popularity, temporal correlations of requests, seasonal access patterns, user session durations, user inter-activity, and VBR long-range dependence and marginal distribution. MediSyn [15] is another workload generator for streaming media which focuses on file duration, encoding rates, popularity, and file access prefix (length of playout). They also cover temporal properties such as new file introduction, life span, seasonal and daily access patterns.

There are several commercial efforts to continuously measure CDN and streaming media performance. Akamai [5] monitors its own proprietary streaming media CDN through the continuous collection of performance statistics by constantly probing from agents across the globe. Availability, startup time, thinning, loss, and rebuffering rates are measured. This continuous collection of data allows Akamai to audit their performance for their CDN management and reporting.

Independent monitoring and verification of performance is also provided by several commercial services such as Keynote [12], Streamcheck [3] and Broadstream [2]. Keynote is a leader in the creation of world wide monitoring of web performance, monitors streaming media performance measuring startup times, audio and video bandwidth, and packet counts (delivered, recovered, late). They also provide a weighted scoring that gives a single number summary of overall performance derived from low level metrics. Streamcheck does similar probing for performance while Broadstream focuses on their IP TV network and measures performance at each receiver.

6. CONCLUSIONS

In this paper we have presented a methodology for evaluating media server performance and capacity limitations and conducted a case study of a commercial streaming server using this methodology. We have shown that in addition to well understood server side metrics (i.e. CPU, disk, network, and memory utilization), one needs to look at client side metrics as well, in order to understand the real limitations of such applications. We have identified five client-side metrics of interest: startup and play failures, startup delay, rebuffering, thinning, and packet arrival jitter. The presence of failures provides the absolute breaking point for a server,

while the other four metrics serve as indicators of "softer" failures associated with reduced quality of service.

We have also shown that failures can occur well before server side metrics reach saturation, due to internal server scheduling inefficiencies. Finally, we have demonstrated that both server side and client side metrics are affected by the workload imposed on the server and that server capacity is greatly affected by the workload type. In particular we have identified three important dimensions in the workload space that affect performance, stream type (live, Video-on-Demand, or mixed), stream popularity, and stream encoding rate.

The lessons we learned through these extensive measurements of a single streaming server are directly applicable to the management of multiple servers (e.g. in a cluster configuration). Some of the lessons we have learned is that better throughput can be achieved by assigning requests and content so that the popularity of clips is maximized, by separating requests for on-demand and live streams to different servers and by converting on-demand requests to live streaming whenever possible (e.g. by regular scheduling of live streaming for the very popular content).

We expect to continue our work by understanding how mixed workloads perform for different ratios of live and video-on-demand streaming and hope to create a server performance model for arbitrary mixtures of live, video-on-demand, popular, unpopular, high, and low bitrate streams. Our goal then is to use those models to build cluster server management policies that optimize performance by carefully assigning stream requests to the appropriate server based on the request characteristics and the current server state.

7. REFERENCES

[1] http://oprofile.sourceforge.net/news/.

[2] http://www.broadstream.com.

[3] http://www.streamcheck.com.

[4] Accustream iMedia Research. http://www.accustreamreseach.com.

[5] Akamai Technologies. *Akamai Streaming: When Performance Matters*, 2002. http://www.akamai.com-/en/resources/pdf/Streaming_Akamai.pdf.

[6] Apple. Darwin Streaming Server 4.1.3. http://developer.apple.com/darwin/projects/streaming/, 2003.

[7] L. Cherkasova and L. Staley. Building a performance model of streaming media application in utility data center environment. In *Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, Tokoyo, Japan, May 2003.

[8] L. Cherkasova and L. Staley. Measuring the capacity of a streaming media server in a utility data center environment. In *ACM Multimedia Conference*, Juan Les Pins, France, Dec 2003.

[9] M. Covell, S. Beomjoo, S. Roy, M. Spasojevic, L. Kontothanassis, N. Bhatti, and R. Zimmermann. Calibration and prediction of streaming-server performance. HP Labs Tecnical Report HPL-2004-206, 2004.

[10] A. C. Dalal and E. Perry. A new architecture for measuring and assessing streaming media quality. In *Passive and Active Measurement Workshop*, La Jolla, California, April 2003.

[11] S. Jin and A. Bestavros. A genertator of internet streaming media objects and workloads. Technical Report 2001-020, Boston University, 2002.

[12] Keynote Inc. *Measurement and Monitoring: Streaming Perspective*, 2003. http://www.keynote.com/downloads/datasheets/streaming_0104.pdf.

[13] D. Luparello, S. Mukherjee, and S. Paul. Streaming Media Traffic: an Empirical Study. In *6th International Workshop on Web Caching and Content Distribution*, June 2001.

[14] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, January 1996.

[15] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat. Medisyn: A synthetic streaming media service workload generator. In *NOSSDAV'03*, Monterey, California, June 2003.

[16] Z. Wang, S. Banerjee, and S. Jamin. Studying streaming video quality: From an application point of view. In *ACM Multimedia Conference*, Berkeley, California, November 2003.