Calibration and Prediction of Streaming-Server Performance

Michele Covell, Beomjoo Seo[1], Sumit Roy, Mirjana Spasojevic,
Leonidas Kontothanassis, Nina Bhatti, Roger Zimmermann[1]
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2004-206(R.1)
February 25, 2005*

performance evaluation, server modeling, streaming media

Streaming media is gaining in popularity for viewing both, *video-on-demand* content as well as *live* Webcasts. Streaming servers must meet strict data-delivery timing constraints in order to provide acceptable viewing quality. These constraints can be achieved only if the servers are not allowed to exceed their operational saturation point. At the same time, providers of streaming services need to maximize the use of their infrastructure to remain cost-effective. These competing goals motivate development of detailed models that predict server saturation points under extremely diverse workloads.

Due to the intricate effects of distinct usage patterns on low-level measurements, no single server-side or client-side metric can adequately predict saturation for a non-controlled mixture of workloads. Furthermore, the dynamically changing nature of streaming workloads render simple linear statistics inadequate. Instead, we propose a methodology that can build predictive models using a relatively small number of calibration workloads. These models include both server- and client-side metrics and are accurate in predicting server performance, not only for the calibration workloads but also for arbitrary mixtures. We contend that the strength of our approach to modeling streaming-server behavior is its highly data-driven nature. The same calibration regime and modeling method are shown to be applicable to different streaming servers and across the wide variety of workloads seen in today's environments.

* Internal Accession Date Only
[1]Integrated Media Systems Center, University of Southern California, Los Angeles, CA 90089
Approved for External Publication

Calibration and Prediction of Streaming-Server Performance

Michele Covell[1], Beomjoo Seo[2], Sumit Roy[1],
Mirjana Spasojevic[1], Leonidas Kontothanassis[1], Nina Bhatti[1], Roger Zimmermann[2]

ABSTRACT

Streaming media is gaining in popularity for viewing both, *video-on-demand* content as well as *live* Webcasts. Streaming servers must meet strict data-delivery timing constraints in order to provide acceptable viewing quality. These constraints can be achieved only if the servers are not allowed to exceed their operational saturation point. At the same time, providers of streaming services need to maximize the use of their infrastructure to remain cost-effective. These competing goals motivate development of detailed models that predict server saturation points under extremely diverse workloads.

Due to the intricate effects of distinct usage patterns on low-level measurements, no single server-side or client-side metric can adequately predict saturation for a non-controlled mixture of workloads. Furthermore, the dynamically changing nature of streaming workloads render simple linear statistics inadequate. Instead, we propose a methodology that can build predictive models using a relatively small number of calibration workloads. These models include both server- and client-side metrics and are accurate in predicting server performance, not only for the calibration workloads but also for arbitrary mixtures. We contend that the strength of our approach to modeling streaming-server behavior is its highly data-driven nature. The same calibration regime and modeling method are shown to be applicable to different streaming servers and across the wide variety of workloads seen in today's environments.

General Terms

Experimentation, Measurement, Performance

Keywords

Streaming media servers, Calibration, Modeling

[1]Hewlett-Packard Laboratories, Palo Alto CA
[2]University of Southern California, Los Angeles CA

1. STREAMING-SERVER MODELING

The demand both for streaming live events and for streaming file-based Video-on-Demand (VoD) media is increasing as the content base and the available bandwidth increase. The viewing requirements for streaming impose strict timing constraints on the delivery of data. If these timing constraints are not met, the quality of viewing drops due to stuttering audio or frozen video and due to decompression artifacts from lost data. If too many client requests are accepted by a streaming media server, the quality of service degrades across all client sessions or fails completely on a subset of client sessions. The actual saturating load on a streaming server depends on the detailed characteristics of the media being served to the various clients. The load is affected by the specific combination of "live" versus "file-based" streams, their relative popularity, and their bit and packet rates. Since a streaming server will support an unknown, dynamic mixture of these clients, saturation prediction must handle mixtures of these workloads without knowing client counts or request types.

In our experiments, we have found that simple measurements are not sufficient for predicting when the server will fail to maintain high-quality service to the clients. The temporal variance observed in simple server-side measurements, such as *load average*, makes their direct usage ineffective. Yet, the long-term averages needed to reduce this variance are not consistent with the need to quickly predict performance in the dynamically changing loading environment seen by a real-world installation. Similarly, trends in client-side measurements like jitter and rebuffering counts [7] are obscured due to the variance caused either by artificially smoothed transmission (*packet smoothing*) or by bursty transmission (*packet blitting*).

Our approach is distinct from prior work in streaming server characterization in that we focus on streaming-server performance prediction, in contrast to network effects [17], media-access patterns [16, 5] and data-layout effects [8]. We do not assume that the client-imposed workload is purely VoD [4] but instead support mixed workloads. We avoid relying on a list of the client sessions loading the server [3]. This type of information may be hard to get, due to privacy and security concerns, and, once known, may be difficult to translate into media characteristics: the determination of relative content popularity and the gradation between various bit and packet rates may be non-obvious and non-stationary. Finally, we do not rely on expert intuition to define an abstract parameterized model [9, 1].

Instead, we take a black-box approach to server model-

ing: we use no privileged information, neither request types nor client counts. We use low-level measurements from the streaming server machine and from "probe" clients, along with some short-time non-linear statistics derived from these measurements. We successfully predict performance across the diverse usage profiles generated by varying combinations of streaming requests. This includes modeling mixtures of workloads, even though our calibration methodology does not include explicit measurements from these mixtures. The combination of our calibration methodology and prediction modeling allows us to generalize across servers: performance models are mathematically derived from measured data, instead of being based on visual inspection and expert intuition. Our emphasis on data mining to create prediction models is seen in other types of failure modeling [6] but previously has not been applied to predicting streaming-server failures. This is the first model of streaming server performance that has been demonstrated to estimate performance accurately:

- without using categorical client counts
- without restricting the type of client workload
- using only *measured* data.

We support these claims by presenting a summary of our prediction results in the next section. In Section 3, we describe our experimental set-up and the calibration methodology, emphasizing the details critical for achieving reliable measurements on different servers. Section 4 examines some of the single-dimensional measurements that have been proposed in the past for characterizing the load on streaming servers. These single dimensional measurements are shown to be insufficient when faced with uncertainty in the type and number of client sessions. In Section 5, we outline the mathematical approach taken to deriving the predictive models from calibration measurements. We also provide an intuitive interpretation for the different measurement-vector combinations that have been discovered by the mathematical analysis. Section 6 reiterates our measurement/prediction philosophy and argues for its strength and generality.

2. PREDICTING SERVER SATURATION

Conceptually, we characterize a streaming server as using a set of composite resources. We use distinct client workloads during the calibration phase, not only to identify these composite-resource models, but also to derive the corresponding client-to-usage models. Once we have the composite-resource models, we can use server-side and client-side measurements on an active server to estimate the current resource consumption. As a sample application, we can use these models in admission control. When there is a proposal to increase the number of clients, we can compute the new resource consumption as:

$$\begin{aligned}
\mathbf{R}_{target} &= Y_{resource}\mathbf{m} + Y_{client}\Delta\mathbf{c} \\
Y_{resource} &: \text{the measurement-to-resource model} \\
\mathbf{m} &: \text{the vector of current measurements} \\
Y_{client} &: \text{the client-to-usage model} \\
\Delta\mathbf{c} &: \text{a vector of proposed additional clients} \\
\mathbf{R}_{target} &: \text{the resultant resource usage vector}
\end{aligned}$$

Table 1: Summary statistics across all experimental epochs for using an uncertainty region of 5%.

Prediction type	Sample set size	Accuracy (%)			
		mean	sd	min	max
self, unsaturated	54	99.3	1.7	91.2	100.0
self, saturated	28	99.9	0.6	97.2	100.0
cross, unsaturated	102	99.0	2.6	85.0	100.0
cross, saturated	376	100.0	0.01	99.8	100.0

With $Y_{resource}\mathbf{m}$, we estimate the current resource use on the streaming server. With $Y_{client}\Delta\mathbf{c}$, we adjust that estimate according to the expected load from the clients that are being considered for admission.

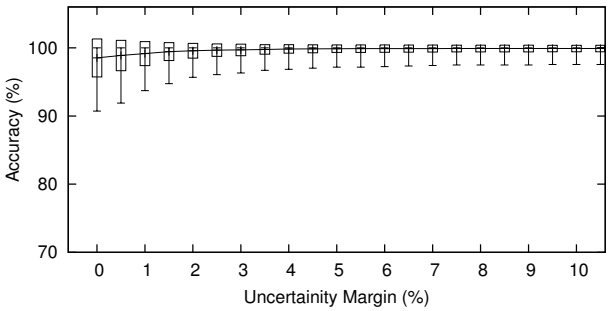
We can admit an additional $\mathbf{\Delta c}$ clients, if the resultant R_{target} does not map into the saturation region for the server. We define saturation as the client workload at which the streaming server can no longer reliably supply high-quality service. If a service failure was seen at any time during an experiment involving some workload mixture and number of clients, then all measurements at that load are labeled as failing: the *reliability* of the service is no longer sufficient to support that specific mixture of clients.[1]

The admission control decision will work well *if* our modeling provides us with accurate matrices $Y_{resource}$ and Y_{client}. The accuracy of $Y_{resource}$ can be evaluated using *self prediction*: We repeatedly estimate the resource usage due to a mixed workload over a local time window. The model estimates how close the server is to saturation. By setting a threshold for maximum resource usage, we can go from the continuous-value estimates to categorical decisions on whether or not the server is operating within a saturated zone. Using the independently determined saturated/unsaturated label for the testing epoch, we then tabulate the percentage of correct predictions for each epoch.

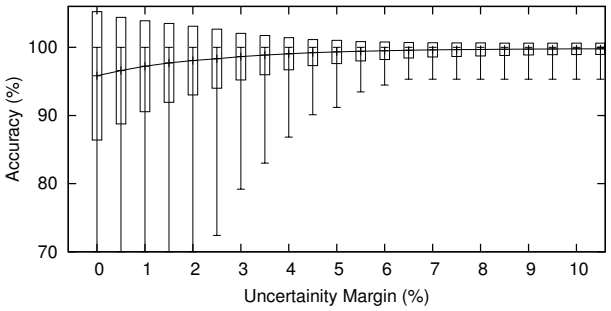
In self prediction, described above, the *target* workload for our prediction is the same as our *initial* workload. We are testing how well we can recognize a saturated or unsaturated state when we are already operating under that workload. The combined accuracy of $Y_{resource}$ and Y_{client} can be tested using *cross prediction*. Here, the target workload for our prediction is heavier than the initial workload. In this case, we are testing how well we can predict a saturated or unsaturated state under a workload that is different than the one we are currently operating under. As with self prediction, we take the sequence of measurement vectors from our initial workload and translate each measurement vector into an estimate of resource usage. We then use our client-to-usage models (i.e., Y_{client}) to determine the resultant resource usage if we were to add the clients needed to create the heavier target workload. These modified estimates of resource usage are then thresholded to give predictions about the saturation of the server under the heavier target workload. We use the ground-truth saturation label given to the target-workload epoch to tabulate the number of correct estimates.

In both self and cross prediction, the threshold that is used can allow a region of uncertainty. If the threshold for predicting saturation on ground-truth-labeled saturated epochs and the threshold for ground-truth-labeled unsaturated epochs are both 100% resource usage, then the region of uncertainty is zero. If the threshold for ground-truth-

[1]We give a more exact definition of service failure is in Section 3.3.

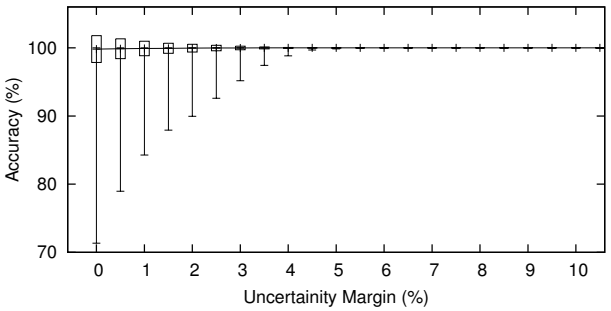


(a) saturated mixed workloads

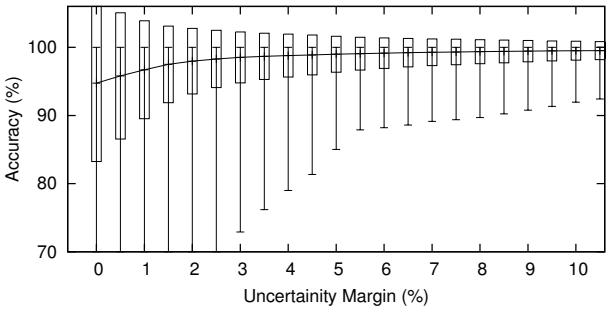


(b) unsaturated mixed workloads

Figure 1: Self-prediction accuracy as a function of uncertainty region. The X axis shows the uncertainty margin, m, such that predicted usage values between $(100 - \frac{1}{2}m)\%$ and $(100 + \frac{1}{2}m)\%$ are left with an *unknown* saturation label and do not contribute to either correct or incorrect prediction percentages.



(a) saturated mixed workloads



(b) unsaturated mixed workloads

Figure 2: Cross-prediction accuracy as a function of uncertainty region on mixed VoD-Live workloads.
.

labeled saturated epochs are reduced to $(100 - \frac{1}{2}m)\%$ resource usage and the threshold for ground-truth-labeled unsaturated is increased to $(100 + \frac{1}{2}m)\%$, then the uncertainty region is $m\%$. The uncertainty region essentially provides us with *slack* in the accuracy of our composite-resource parameterization, the fidelity of the client model, as well as the accuracy of the current measurements.

Our results using an uncertainty region of 5% are tabulated in Table 1. These summary statistics are collected by treating the percentage correct from each experimental epoch as a single data point and computing the resulting mean, maximum, minimum, and standard deviation *across* distinct experimental epochs.

This summary table shows that our prediction performance is highly accurate on both saturated and unsaturated workloads. We are slightly more accurate on self prediction than on cross prediction: it is easier to recognize an existing state than to extrapolate to a new state. We are much more accurate on predicting saturated workloads than on predicting unsaturated workloads. This is a result of our conservative approach to determining saturation: the models generated from conservatively labeled calibration workloads are too conservative when applied to arbitrary mixed workloads.

Figures 1 (for self prediction) and 2 (for cross prediction) show the mean, standard deviation, minimum, and maximum of the prediction results as a function of the uncertainty region. We use *box-and-whiskers* plotting to show these statistics: we draw a box extending one standard deviation above and one standard deviation below the mean and add a single-width line from the minimum to the maximum values within each population. These margin plots show that the *mean* prediction performance is greater than 95%, all the way down to the zero-uncertainty region. However, the standard deviation increases and the worst-case accuracy deteriorates at uncertainty regions less than 5%. As with the 5% uncertainty results in Table 1, we suffer from far fewer *false negatives* than from *false positives*.

3. METHODOLOGY FOR CALIBRATING STREAMING SERVERS

This section describes the types of client workloads used for calibration, lists the details of the experimental set-up, and defines our notion of service failure. We also enumerate the various components of the measurement vector, and finally discuss the actual calibration method.

3.1 Client Workload Categories

As with prior work [3], we found that the per-client load imposed on the server depends on the details of the requests being made by the clients. In our work, we use three axes to describe the content requested by each client session:

Repository: The content can either be served from a locally stored file or it can be relayed from a live stream received from a remote server or media encoder. We refer to the locally stored, file-based content as a *VoD* session and to the live-stream relayed content as a *Live* session.

Popularity: The content can either be viewed once by a single client or viewed by multiple clients. For multiple viewings of live content, each client synchronously views the ongoing transmission of a single source. For multiple viewings of VoD content, each client *asynchronously* views the

Table 2: Acronyms for server workloads

Repository	Popular		Unpopular	
	low-rate	high-rate	low-rate	high-rate
VoD	VPL	VPH	VUL	VUH
Live	LPL	LPH	LUL	LUH
mixed	MPL	MPH	MUL	MUH

source file, with the relative starting offsets determined by the inter-arrival rate of the client requests. Even though, at any given time, the VoD clients are being served different parts of the same content, the popularity of the VoD content affects the server performance since the file-buffer cache can reduce the number of disk accesses. We refer to content that is viewed only once by a single client session as *Unpopular*. *Popular* content is a single source that is viewed (possibly asynchronously) by all client sessions of the same category.

Bitrate: The encoding bitrate of the content is nominally a continously varying dimension. However, in practice, there are a few standard bit-rates, depending on the media codecs and target networks. For our current calibration purposes, we use 300 kbps source material as *High-rate* and 78 kbps source material as *Low-rate* content. These rates were expected to be seen in mobile wireless streaming applications. The specific rates used can be changed as appropriate for the server-access pattern, without invalidating the measurement or modeling methodology.

These axes provide a description of each client session. If all of the client sessions that are requested from the server fall in the same three-axis category, we refer to the workload as being *Pure*. If the sessions fall into distinct categories along one or more of the axes, we refer to the workload as *Mixed*. For brevity, we only discuss pure workloads and workloads that are mixed repository. This type of mixture of VoD and Live streaming requests has not, to our knowledge, been modeled in prior work.

We use the following abbreviated naming scheme: the first letter defines the repository (**L**ive, **V**oD, or **M**ixed), the second letter defines the popularity (**P**opular or **U**npopular), and the third letter defines the bit-rate (**H**igh-rate or **L**ow-rate). For brevity, we replace the third letter with x to signify both high and low bit rates when appropriate. Table 2 summarizes these conventions.

3.2 Experimental set-up

Our experiments run on three distinct sets of machines: the streaming-server machine that is being calibrated or tested; up to four live-source machines; and up to six client machines. The server machine is a dual 1.4 GHz Pentium III PC with 1 GB memory, running SuSE 8.2 (kernel version 2.4.20). The other machines are selected to have sufficient compute and I/O capacity, so that they do not influence the experimental results.[2] The streaming-server and the live-source machines used SCSI disk drives with 50 Mbytes/sec throughput for the VoD and Live content, respectively. All the machines were connected to a switched Gigabit network, isolated to avoid uncontrolled network interference. The streaming-server software suites used for this paper are the *Darwin Streaming Server, v4.1.3* [2], and *Helix Universal Server, v9.0.3.916* [13].

To avoid performance variation due to differences in the source content, we use multiple copies of the same material for Live and VoD tests. The content consists of MP4 files, optimized for streaming using hint tracks [11]. For our Live tests, the material is stored on the live-source machines and is relayed through the streaming server under test using the *Darwin PlaylistBroadcaster* [2]. We chose *PlaylistBroadcaster* due to its inter-operability with both the *Helix* and *Darwin* servers and its lightweight CPU utilization: it allows us to reliably transmit hundreds of independent Live streams from a single live-source machine. For each live stream received, the streaming server suffers loading overhead, whether or not that stream is used. To avoid this additional penalty, we exactly match the number of independent live-source streams to the streams relayed by streaming server for each experiment.

To avoid performance variation due to interfering disk accesses, measurement logging on the streaming server is done to a separate disk. Similarly, to minimize the performance variation due to differences in content layout, the VoD material is stored on the streaming server disk just after reformatting, thereby avoiding file fragmentation. To insure that each distinct VUx request retrieves the content from disk (instead of the file-buffer cache), we create 300 distinct copies of the VoD content on the streaming-server disk.

In our experiments, reaching server saturation required nearly 3000 clients in some situations. This large number made it necessary to develop an application that could support a large number of simultaneous streaming sessions without overloading the client machine. Our client application creates RTP [14] (over UDP) streaming sessions, using RTSP [15] as the control protocol. For each session, we can record session-level statistics, like play failure, startup delay, total duration of data delivery and number of bytes delivered. The client can also record a trace of RTP/RTCP packets. This provides the packet arrival time, size, and sequence number, as well as the media decode time.

Each experimental period has three distinct phases: ramp-up, steady-state, and termination. During the first phase, loading clients are added at 500 ms intervals, which avoids start-up failure purely due to transient effects. The loading clients are used to *induce* a particular type of workload on the server. After reaching the steady-state period, we collect measurements from the streaming server machine.We also sequentially launch 20 probing-session clients, which run for non-overlapping 1 minute periods. These probing clients collect statistics and traces, both within our calibration experiments and, ultimately, on an in-service streaming server. Since probing-client statistics will also be collected on in-service streaming servers, we want the probing client to be both informative and low-overhead. For this reason, we use a VUL request for each probing session. We chose a VoD request since we could always be assured that the chosen file-based content would be available and unchanging from one probe to the next. We chose Unpopular since we do not always know what content is currently Popular (that is, likely to be partially in the file-buffer cache) but we can request distinct probe-only content and be assured that it is unpopular. We chose Low-rate to minimize the overhead induced by the probe.

3.3 Saturation Criteria

We use the client-side requirements published by Keynote Inc. [12] for a highest quality-of-service grade to *guide* us in

[2]In our testbed, the live-source and client machines have 1.0 - 2.4 GHz Pentium III processors with 256 MB - 1 GB memory.

our definition of streaming-server failure. However, the parallel with Keynote is not complete: unlike Keynote, we do not set up actual playback clients to evaluate these criteria. Instead, in calibration and validation, we use the session-level data from all of the clients and the packet-level data from our probing clients.

Our criteria for saturation are:

Play-Request Failure: If any loading- or probing-client session fails to establish a streaming session, the server is considered saturated.

Duration Violation: If the actual duration of any client session is less than 97% or greater 103% of the requested duration, there has been a failure at the server to provide the data-delivery timing to support smooth, uninterrupted streaming without risking client-buffer over- or under-flow.

Size Violation: If the number of bytes received by any loading- or probing-client session is less than 97% of the expected data from the requested source material, there has been a failure at the server.

Rebuffering Violation: If the amount of time that the probing-client sessions spend waiting for start-up delays and mid-stream data rebuffering delays, plus a *rebuffering-event penalty,* exceeds 3% of the total play time for the probe-client sessions, there has been a failure at the server to provide the required streaming quality. We use a rebuffering-event penalty of 2 seconds for each distinct mid-stream buffer violation. This requirement is from a Keynote requirement to avoid excessively long startup delays and to also avoid *frequent* mid-stream rebuffering events.

For calibration experiments, we refine our definition of rebuffering events. In the experiments on the *Helix* server [13], we saw increasingly bursty packet transmission as the server workload increased. The timing of these bursts was such that, on occasion one or two packets would be delayed beyond their delivery deadline. This small amount of over-delayed data resulted in rebuffering violations on those experiments, even when the server was otherwise not saturated. We found that, by re-categorizing these few packets as being lost data (instead of late data), we could avoid a rebuffering violation without inducing a size violation. This greatly improved the reliability and reproducibility of our decision surface. Therefore, in our saturation decision criteria, if the amount of sequential late-arriving data is less than 3% of the previously received data, that sequential late-arriving data is relabeled as missing and the rebuffering-event penalty is not imposed. Otherwise, that whole sequence of the late-arriving data is marked as a rebuffering event and incurs the rebuffering-event penalty.

If any of these violations are seen at any time during the experimental epoch, the streaming server is labeled as being saturated for the full experimental epoch. For our calibration data, each experimental epoch used to determine the saturation point consists of five 20-minute measurement sets at the possibly saturating workload. This repetition insures a reproducible, internally consistent categorization of the server state.

3.4 Measurement and Labeling Data

We derive our predictive models under a classic *labeled training data* approach. This requires that our calibration measurements include both the measurement data, which will be available from an in-service media-streaming server, and the label data, which will not be available.

Table 3: Saturation points for pure workloads on the Apple *Darwin* and the RealNetworks *Helix* servers.

Darwin/ *Helix*	Popular		Unpopular	
	low-rate	high-rate	low-rate	high-rate
VoD	726/1220	425/590	259/228	33/91
Live	1976/2850	1158/1460	405/492	405/396

For our measurement vector, we use order filters on instantaneous measurement values. Order filters provide local percentile measures. For example, if we select a 60-sample, 5^{th}-percentile filter, then, for each instant, the order filter would collect the 60 previous input values, sort them from largest (100^{th} percentile) to smallest (0^{th} percentile), and then output the 12^{th} smallest (5^{th} percentile). Thus, a running median over a local-time window is a 50^{th}-percentile order filter. We use multiple order filters instead of local means and local standard deviations. This choice provides statistical measures on small-sample populations that are less prone to distortion by the outliers within the population. This is important when trend and range estimates are needed with low-latency. Actual in-service streaming server management will be latency sensitive, due to the dynamic nature of their workloads.

We derive a measurement vector once per second that includes sample values that were measured during that one-second interval and outputs from 60-second order filters for the 0^{th}, 5^{th}, 25^{th}, 50^{th}, 75^{th}, 95^{th}, and 100^{th} percentiles. For our measurement vector, we collect measurements both from the streaming-server machine and from the probe sessions:

Instantaneous measurements from the streaming-server machine: Once per second, we record: interrupt rate, context-switching rate, time running non-kernel code, time running kernel code, idle time (including IO-wait time), one-minute load average, incoming UDP-packet rate, outgoing UDP-packets rate, disk-read-access rate, disk-sector-read rate, disk-write-access rate, disk-sector-write rate. In addition, we create some combined statistics from the server reports: the summed incoming and outgoing UDP-packet rates, the summed disk-read-access and -write-access rates, the summed disk-sector-read and -write rates.

Startup metrics derived from the probe-session statistics: The probe clients use the RTSP protocol to initiate each streaming session: They obtain a description of the media using a DESCRIBE command, indicate their desire to receive audio and video using two SETUP commands, and then start these media streams with a PLAY command. We compute two delay metrics: one starting from the first SETUP request, the other starting from the PLAY request, and both ending at the first RTP packet arrival. The DESCRIBE request is not used as a starting point, since a server might have cached an earlier response. Using SETUP and PLAY protects us from different server designs (i.e., eager vs lazy). We use replication to translate these once-per-probe-launch delay values into once-per-second values for our measurement vector.

Metrics derived from the probe-session traces: Once per second, we obtain simple metrics from the probe-session trace, e.g. bytes received per second, packet-loss rate. We further compute some statistics from the packet trace: the packet-arrival offset and the fine-grain variation in bandwidth.

The packet-arrival offset is the difference between each packet-delivery time and its deadline. This offset value is

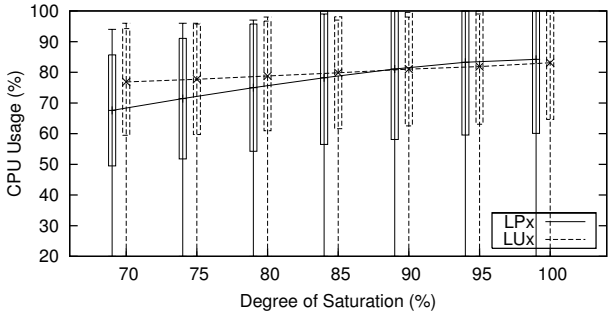
greater than zero when the packet is late and less than zero when it is early. To obtain half-rectified packet-arrival offsets, the negative offsets are replaced by zeroes, thereby discarding everything about early packets, other than their count. We include this metric since late packets are often more problematic, resulting in rebuffering events or in increased lost-data counts. For fully-rectified packet-arrival offsets, the negative offsets are replaced by their magnitude, so that early and late packets are treated equally. We include this metric to help determine when a server is resorting to packet blitting. Packet blitting is typically seen on best-effort servers that are heavily loaded. We use one-second-window averages, medians, maxima, and minima to translate these once-per-packet arrival-offset values into once-per-second values for our measurement vector.

For fine-grain variations in the received bandwidth, we use one-second-window median, minimum, and maximum from the ratio between the bandwidths received by the probe client during a sliding 100-ms period and the bandwidth received during the full one-second interval. This statistic was included to try to distinguish between packet smoothing and packet blitting by measuring uniformity of the sub-second bandwidth usage. Packet smoothing is the artificial smoothing of local variations in bandwidth by spreading out the delivery of packets that have the same media decode time stamp, such as would occur for multi-packet video frames. When lightly loaded, a well-written streaming server will smooth the packet-delivery times, to avoid overloading the client's network-input buffer. In terms of the packet-arrival offset measurements defined above, packet smoothing looks similar to packet blitting: it will have artificially early and late packets. Since packet smoothing is a desirable behavior seen on lightly loaded servers and packet blitting is an undesirable behavior seen on heavily loaded servers, including this disambiguating statistic is helpful.

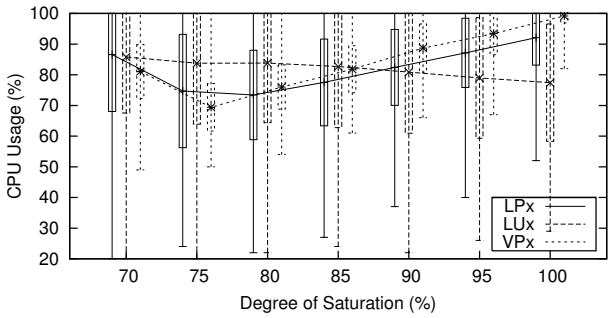
Ground-truth data for calibration As mentioned above we create our predictive models using labeled training data. For label data, we use our normalized client load, $\mathbf{x} = [\cdots c_{ijk}/S_{ijk} \cdots]$ where $\mathbf{c} = [\cdots c_{ijk} \cdots]$ is the vector of loading-clients counts for each pure workload type ijk and S_{ijk} is the saturating client count for that pure workload type. These saturating client counts, S_{ijk}, are determined in our calibration process and are listed in Table 3 for *Darwin* and *Helix*. We do not need this data for anything other than calibration or for anything other than pure workloads.

3.5 Calibration Process

Our calibration process consists of two phases. We first determine the saturation point, S_{ijk}, for each of the pure-workload types. As mentioned in Subsection 3.3, the saturation point is the minimum workload at which one or more QoS violations occur during any of five 20-minute measurement periods under that workload. Once we have found the saturation workload for a given pure workload type, we collect measurement data at pure workloads below that saturation point. Since we are primarily interested in recognizing server saturation and in predicting the transition from unsaturated to saturated states under additional workload, we collect this measurement data in the range from 70% to 100% saturated. For the specific models reviewed in this paper, we collected one 20-minute measurement period on 70%, 75%, 80%, 85%, 90%, 95%, and 100% of the saturating loads for each of the pure workloads. This is the data that we used to create the models described in Section 5.



(a) *Darwin*



(b) *Helix*

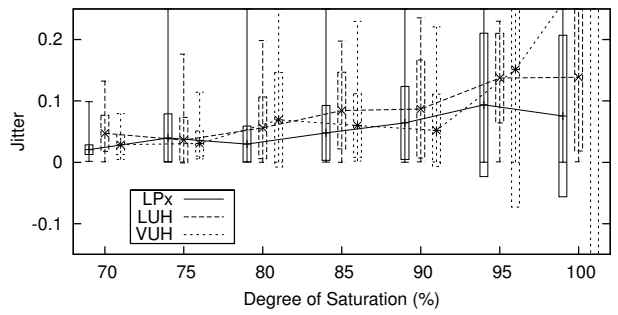
Figure 3: CPU usage for LPx, LUx, and VPx

We validate these prediction models on *mixed* workloads only. Using only pure-workload calibration data allows us to minimize the required number of experiments: the server administrator who uses our approach does not need to recreate all of the different workloads mixtures under which the server will operate. Using only mixed-workload data to test our model minimizes the chance that our validation results are overly optimistic. There will always be some degree of mismatch between training and testing configurations. Training and testing on the *same* experimental configuration does not provide a realistic measure of the robustness of the derived models. We believe that we avoid that unrealistic optimism in this paper by training and testing on distinct configurations of workloads.

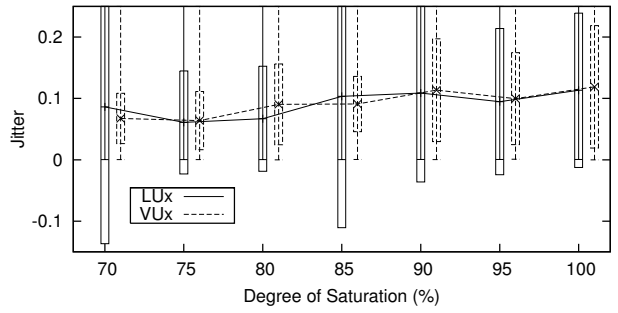
4. ISOLATED MEASUREMENTS ACROSS WORKLOADS

In this section, we examine measurements from pure workloads that have previously been proposed as reliable indicators of server performance: specifically, we consider CPU usage, disk-sector-read rates, UDP network traffic, and client packet jitter. Since our goal is to enable a black-box evaluation of in-service streaming servers, we need to find an evaluation metric or set of metrics that can predict saturation without knowing the details of the client workload. If we decide to use a measurement to estimate the resource stress that caused saturation for a client type, we need to satisfy several criteria with regard to that measurement.

Reliable: The measurements should be predictive of the workload level using local values. In particular, the statistical variance in the measurement at a fixed workload level should not exceed the change in the mean measurement value across distinct loads of the same client type. Oth-

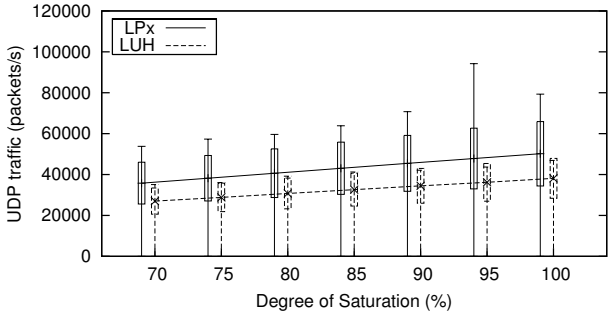


(a) *Darwin*

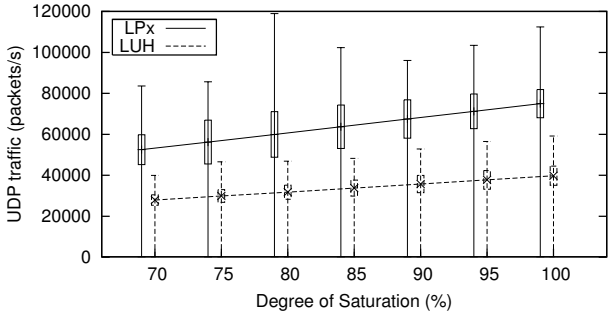


(b) *Helix*

Figure 4: Probe-client packet-arrival jitter for LPx, LUx, and VUx. Only unambiguous and nearly-unambiguous measurement curves are shown in these plots. Also, low- and high-bitrate data were combined before plotting whenever the two showed no statistically significant differences.



(a) *Darwin*



(b) *Helix*

Figure 5: Combined UDP incoming- and outgoing-packet rates for LPx and LUH

erwise, one cannot distinguish a change in workload level from measurement noise.

Unambiguous: A measurement may reflect the load due to different workload types. In that case, the measurement should be used for predicting only the workload types with the largest effect. Otherwise, when operating under mixed workloads, this measurement will over-estimate the saturation level of the server, due to the contributions from more dominant workload types.

Complete: Each client type must have at least one way to predict its saturation. If we cannot provide this level of prediction on pure workloads, we cannot hope to do so on an in-service streaming server that will have an unknown mixture of client types.

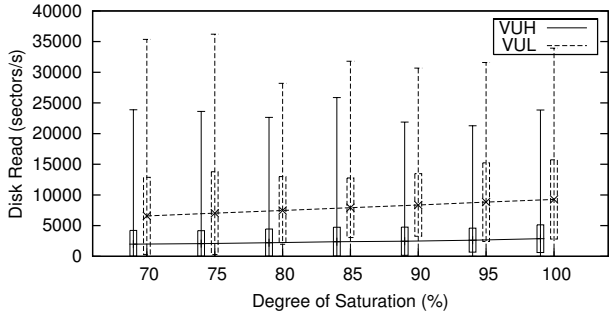
In the plots that follow, and the related plots in Section 5, we show measurement curves only for those pure-workload client types on which the measurements are unambiguous or close to unambiguous. This removes measurement curves that would give incorrect saturation predictions on the unlabeled mixed workload seen on an in-service streaming server. For multi-curve plots, the curves are staggered along the X-axis for clarity.

Throughout this section, we discuss the results that we observed when calibrating the Apple *Darwin* and the RealNetworks *Helix* servers. These two servers represent state-of-the-art streaming servers, and have a similar core architecture: state-machine-based packet dispatchers. However, they have very different internal policies, leading to differing performance on the same hardware (Table 3). While a single specific performance model does not need to handle both servers, the methodology that is used to create the performance model should be independent of the particular server software. If the prediction model depends on expert intuition to select the measurement dimensions, the measurement dimensions must be reliable, unambiguous, and complete, at least for these two example servers.

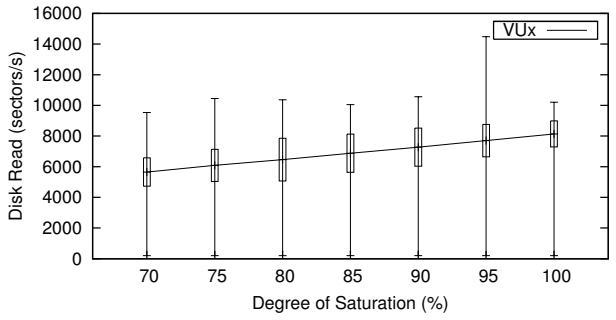
We show the unambiguous (or nearly unambiguous) curves for percent CPU usage in Figure 3, for probe-client jitter in Figure 4, for UDP traffic in Figure 5, and for content-disk reads in Figure 6. We use the same box-and-whiskers plots as in Section 2: the horizontal line is the mean value, the box extends a standard deviation above and below the mean, and the vertical lines extend from the minimum to the maximum values.

The CPU measurements (Figure 3) and jitter measurements (Figure 4) for both *Helix* and *Darwin* are not reliable: they do not show statistically significant trends over the workload ranges of interest. In fact, on the *Helix* server, the mean CPU measurements for LUx workloads (Figure 3-b) show a slight negative trend with increasing load but this negative trend is not statistically significant. Also on the *Helix* server, the CPU usage for LPx and VPx is non-monotonic with changing load. We believe this represents an implicit change in the server policy as it moves from packet smoothing at low workloads towards packet blitting at high workloads.

The UDP-traffic measurement (Figure 5) on *Helix* is reliable and unambiguous for LPx workloads, with prediction-error levels of 9%, but it is not reliable on *Darwin*. Similarly, the disk-read measurement (Figure 6) on *Helix* is reliable for VUx workloads, with prediction-error levels of 5%, but it is not reliable on *Darwin*. Thus, this set of low-level metrics does not meet our criteria of being reliable on different



(a) *Darwin*



(b) *Helix*

Figure 6: Rate of content-disk sector reads for VUx

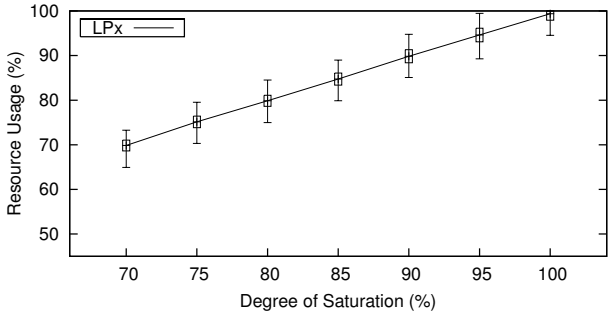


Figure 7: Resource R1 for LPx. This resource is largely a measure of UDP-traffic rates and should be contrasted with Figure 5.

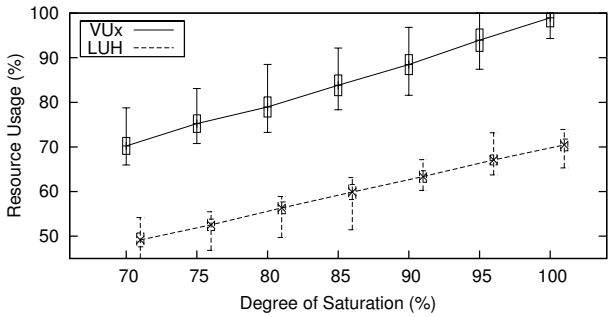


Figure 8: Resource R2 for VUx and LUH. This resource is largely a measure of content-disk usage and should be contrasted with Figure 6.

server types or of being complete in predicting saturation across workloads.

5. DERIVED MODELS FOR PREDICTING SERVER SATURATION

Using the calibration measurements described in Section 3, we created *composite-resource* usage models on the streaming-server machine that were reliable, unambiguous, and complete. We also created models of the additional usage that we expected to be induced on the server by each additional client of the different workload types. We describe the technique that we used to derive the model and then illustrate the details using the derived model for the *Helix* server.

For each calibration experiment that we run on a pure workload below saturation, we collect measurements once per second for each 20 minute run and annotate it with the normalized client workload vector. We start with eight nominally distinct saturating resource directions, one for each of the eight workload types. At saturation, each pure client workload must use 100% of the resource direction on which it saturates. It can also use between 0% and 100% of the other resource directions at saturation. In all cases, the usage of a resource is separately constrained both to be an affine function of the measurement vector and to be an affine function of the client workload level. We find the solution to this problem using projection-onto-convex sets. In the measurement-to-resource domain, we solve the problem using robust, total least squares under inequality and vector-norm constraints [10]. The inequality constraints on the robust total-least squares include the constraint for non-negative, non-oversaturating resource usage at the same time as finding the measurement-to-resource models. The model for the client-to-resource usage is then refined in the alternate projection step, using the resource usage estimates derived from the measurements and the most recent measurement-to-resource models along with the actual client workloads. Before this model generation, we adjust our resource usage estimates to the correct range (0 to c_{ijk}/S_{ijk}) and then use total least squares. After completing this process for a fixed number of resource directions, we consider lowering the number of resource directions by merging directions that are similar. We measure this similarity in direction using the correlation coefficients on the resource usage across all client types. If the correlation coefficient in client usage across two resource dimensions is greater than 90%, we merge the two resource dimensions together, in terms of which clients saturate on each dimension, and repeat the whole model estimation process with this smaller number of resource dimensions.

For this paper, we applied this approach to the measurements taken from the *Helix* server, since this server showed more complex behavior than the *Darwin* server and therefore should be the more challenging to model. The result of that modeling was four measurement-to-resource models and a matrix description of client-resource usage. In Section 4, we examined the possibility of using single measurement models for predicting saturation. We found the UDP-traffic and disk-read measurements were reliable and unambiguous for LPx and VUx workloads but had fairly large usage-prediction errors (standard deviation=9% and 5%, respectively). We did not find reliable, unambiguous measure of saturation for LUx or VPx using single-measurement models. In contrast, using our modeling approach, we found four resource dimensions that were reliable and unambiguous for

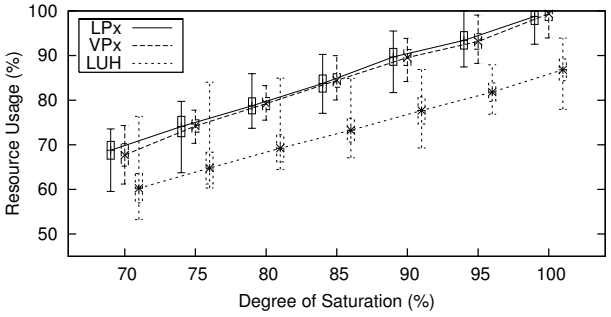


Figure 9: Resource R3 for VPx, LPx, and LUH. This resource is largely a measure of CPU usage and should be contrasted with Figure 3.

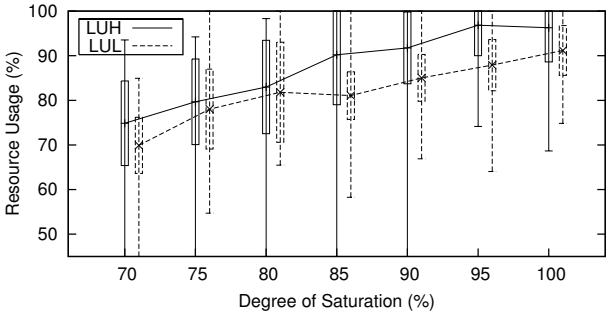


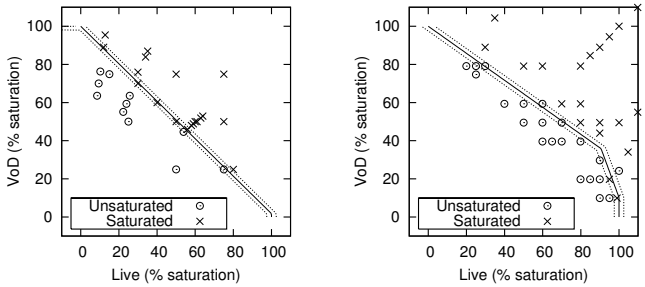
Figure 10: Resource R4 for LUx. This resource is largely a measure of network-usage and packet-arrival variability. The packet-arrival variability portion could be compared to the packet-jitter shown in Figure 4.

predicting saturation across the eight client types. Figure 7 shows the unambiguous curves for resource R1. Using this resource model, the usage-prediction error for LPx is 0.8%, in contrast with the 9% error given by direct UDP-traffic measurements (Figure 5). Figure 8 shows the unambiguous curves for resource R2. Using this resource model, the usage-prediction error for VUx is 1.5%, in contrast with the 5% error given by direct disk-read measurements (Figure 6). Finally, Figures 9 and 10 show the unambiguous curves for resource R3 and R4, on which VPx and LUx, respectively, saturate. The usage-prediction error for VPx using resource R3 is 2.1% and the usage-prediction error for LUx using resource R4 is 5.6%.

Section 2 summarized the results of applying these models, derived using only *pure* workload calibration data, to *mixed* workloads. To determine these results, we first did a single 20-minute experiment at the various mixed-workload combinations that we wished to test and made a binary determination (saturated/unsaturated) on each of these mixed-workload tests. This is in contrast with the five 20-minute experiments we did on each of the pure workloads to determine saturation. Sixteen of the 98 mixed-workload experiments that initially tested as unsaturated consistently gave contrary self predictions: they self labeled as saturated using our measurement-based models. When we re-tested those sixteen outliers, all failed to meet some QoS criteria and, using our conservative labeling, actually *were* saturated. We contend that the detection by our prediction approach of these incorrectly labeled test points is a strong

Table 4: Client resource usage. The percentages listed here are the resource amounts that each pure client workload uses when loaded to saturation.

	Resource usage (percent)			
	R1	R2	R3	R4
LPH	100	1	98	30
LPL	100	2	99	50
VUH	0	100	1	39
VUL	2	100	13	4
VPH	29	28	100	17
VPL	36	24	100	5
LUH	8	71	87	100
LUL	6	43	51	100



(a) MPH streams (b) MUH streams

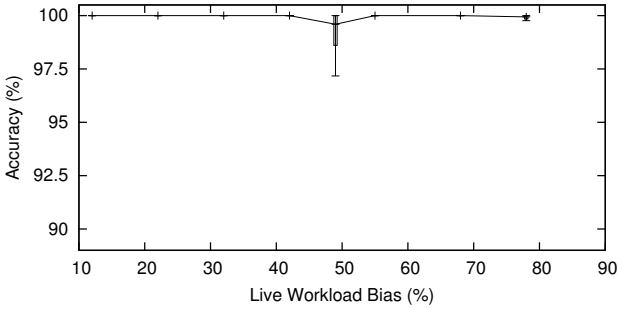
Figure 11: Decision surface dictated by the client-resource model for mixed VoD-Live. The two dotted lines represent the two decision surfaces for a 5% uncertainty margin, as defined in Section 2.

indication of the predictive power of our models.[3]

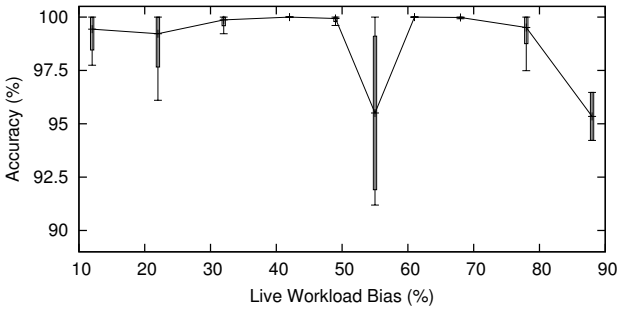
The exact client-to-resource-usage matrix is given in Table 4. If the normalized client counts were available, we could simply use this client-usage matrix to define a piecewise-linear decision surface in the eight-dimensional client-count space, according to the four constraint equations $Y_{client}\mathbf{x} \leq \mathbf{100}$ where Y_{client} is the client-to-resource usage model (such as given in Table 4) and $\mathbf{x}$ is the eight-element *normalized* client count vector, defined in Subsection 3.4. Figure 11 illustrates what these decision surfaces would look like for high-rate, mixed VoD-Live streams, using a 5% uncertainty margin, as defined in Section 2.

The client-count based approach was used by Cherkasova et al. [3]. In our work, we have chosen to use real-time measurements from the streaming-server machine and from a probe client to determine the likely saturation behavior. Our choice is based on two concerns. First, we do not believe that accurate client counts, categorized according to popularity, bit-rate, and repository, will be available from in-service streaming servers, due to the overhead of accurately creating and maintaining these counts and due to concerns around possible misuse of this information. Second, we believe that taking a real-time measurement-based approach will ultimately be more accurate, since these measurements will include information about transient overload situations that could not be captured using non-measurement-based

[3]In all of the plots shown in Section 2, we used the *correct* final label for these outlier points: that is, saturated.

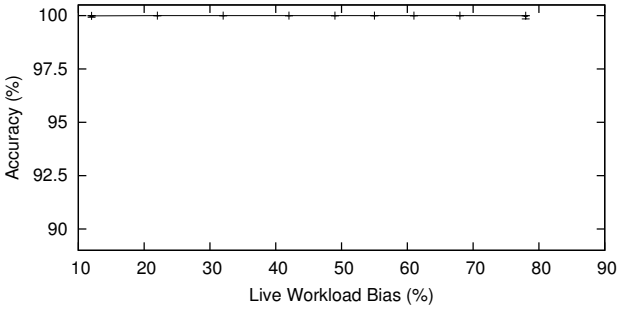


(a) saturated mixed workloads

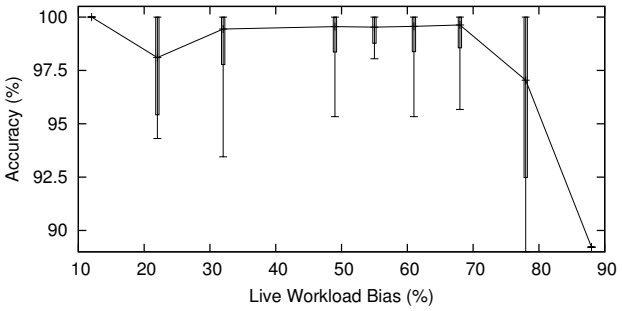


(b) unsaturated mixed workloads

Figure 12: Self-prediction accuracy for mixed workloads. These results were achieved with a 5% uncertainty region allowed for labeling. The X axis shows $\sum_{j,k} c_{\mathrm{L}jk}/S_{\mathrm{L}jk}/\sum_{i,j,k} c_{ijk}/S_{ijk}$ where $c_{\mathrm{L}jk}$ are the client counts for pure live workloads, according to popularity (j) and bitrate (k).



(a) saturated mixed workloads



(b) unsaturated mixed workloads

Figure 13: Cross-prediction accuracy for mixed VoD-Live. These results were achieved with a 5% uncertainty region allowed for labeling.

methods. In lieu of using client counts, we take the measurements described in Subsection 3.4 whenever we need to estimate the status of a running streaming server and use our measurement-to-resource-usage models.

We show validation results, derived from short-time *measurements* for self prediction in Figure 12, and for cross prediction in Figure 13. Both of these figures show percent--correct prediction as a function of the workload *bias* towards VoD or towards Live. For these plots, we define this bias to be the ratio of the normalized Live-client count to the sum of the normalized Live- and VoD-client counts, where the normalization equalizes their saturated-workload counts. Thus, all pure workloads will map to 0 (for pure VoD) or 100 (for pure Live), independent of the number of clients. The mapping from the two axes in Figure 11 to the X axis in Figures 12 and 13 can be completed by starting with a vertical line through the origin, sweeping the slope of that line down until it is horizontal, and placing the measurement epochs according to when they are encountered in this data sweep.

We can compare the prediction performance that would be achieved using a perfect client-load oracle and the decision surfaces shown in Figure 11 to the prediction performance that we achieve using dynamic measurements and our measurement-to-resource-usage models. The performance using measurements is what is shown Figure 12. The performance using an oracle can be determined by visual inspection of the categorical errors on the marked mixed-workload experiments. Surprisingly, our average prediction performance does not suffer significantly when we use measurements in place of an oracle and the worst-case performance is significantly better, since errors using the oracle-based approach will result in 0% correct.

Table 5: Most-significant component measurement weights used in resource R1. Resource R1, on which LPx saturates, generally corresponds to UDP usage. The measurement type is given in the first column. The remaining columns indicate the normalized weights on the different percentile filters.

Metric	%-filter					
	0^{th}	5^{th}	25^{th}	50^{th}	75^{th}	95^{th}
UDP out	0.005	0.006	0.024	0.014	0.020	0.009
UDP in+out	-	0.004	0.005	0.013	0.013	-
UDP in	-	-	0.005	-	-	-

The measurement-to-resource-usage models were derived mathematically, instead of relying on expert intuition. As such, they are mathematical constructs and do not directly correspond to the physical resources of a computer system. Even so, we can examine the weighting coefficients for these models and extract some intuitive explanations for the types of information they represent.

Tables 5 through 8 give the most significant coefficients for the various measurement-to-resource-usage models. In all cases, the reported coefficients have been normalized so that the corresponding measurement dimension would have a standard deviation of one.

***R1* Saturating resource for LPx: UDP usage.** As can be seen from Table 5, the resource on which LPx saturates is basically a measure of the UDP usage. The top 10 coefficients and their cohorts, accounting for 97% of the model's vector norm, combine different measures of UDP packets sent and, to a lesser extent, received.

This mathematically derived resource measure corresponds well to our intuition about the limiting resource for LPx. For

Table 6: Most-significant component measurement weights used in resource R2. Resource R2, on which VUx saturates, generally corresponds to disk usage.

Metric	%-filter				
	5^{th}	25^{th}	50^{th}	75^{th}	95^{th}
disk sector read	0.011	0.032	0.050	0.029	0.014
disk block read	0.008	-	-0.021	-	-
half-rectified receive-time offset					
local median	-	-	-	-	-0.024
rectified receive-time offset					
local median	-	-	-	-	0.011
disk sector read+write					
	-	-	-	-0.010	-

Table 7: Most-significant component measurement weights used in resource R3. Resource R3, on which VPx saturates (and LPx nearly saturates), generally corresponds to CPU usage and variability.

Metric	%-filter				
	5^{th}	25^{th}	50^{th}	75^{th}	95^{th}
user+nice time	0.013	0.024	0.026	0.009	-
idle+IO-wait time					
	-	-0.012	0.021	-	-0.020
interrupts	-0.013	-	-0.009	0.014	-
half-rectified receive-time offset					
local median	-	-	-	-	-0.019
local mean	-	-	-	0.012	-0.015
local max	-	-	-	-	0.012
UDP in+out	-	-0.019	0.051	-0.019	-
UDP out	-	0.028	-0.043	0.024	-

live popular streams, the small number of incoming sources and the synchronized retransmission of that data onto the client streams reduces the per-client workload usage of the CPU and file-buffer cache. In this case, the inability to quickly transfer packets to the network interface becomes the constraining resource.

***R2* Saturating resource for VUx: Disk-interface load** As can be seen from Table 6, the resource on which VUx saturates is largely a measure of the disk-interface load. The top 10 coefficients and their cohorts account for 97% of the model's vector norm. These top coefficients combine different measures of disk usage with variance measures on the server interrupt rate and the packet-delivery timing, similar to resource R3 and R4.

Again, our intuition supports disk usage as the limiting resource for VUx, since each client request results in additional access requests to the hard disk, none of which can be avoided through file-buffer-cache usage. The only non-obvious component of this resource (and of the R3 resource) is its dependence on the packet-delivery variations seen by the probe client. As with R3 and R4, this resource folds in client-side timing information to determine the variability of the server behavior and presumably the resulting likelihood that the disk interface will become overloaded due to a local peak in the accessing requirements. We conclude that using client-side statistics increases the reliability of this type of variability estimation in all three of the resource dimensions that it is used: otherwise, the norm-constrained least squares solution would not have devoted that much of its total weight budget to these measures.

***R3* Saturating resource for VPx: CPU load** As can be seen from Table 7, the resource on which VPx saturates (and LPx nearly saturates) is basically a measure of the CPU usage and variability. The top 10 coefficients and their cohorts account for 93% of the model's vector norm. These

Table 8: Most-significant component measurement weights used in resource R4. Resource R4, on which LPx saturates, generally corresponds to network-usage and packet-delivery variability.

Metric	%-filter				
	5^{th}	25^{th}	50^{th}	75^{th}	95^{th}
interrupts	0.397	0.181	0.077	-0.745	-
UDP in	-	0.614	-0.309	-0.119	-
UDP in+out	-	0.045	-0.288	-0.125	0.057
UDP out	-	-	0.144	0.072	-
rectified receive-time offset:					
local mean	-	-	0.068	0.181	-0.235
local median	-	-	-0.085	-0.124	0.077
local min	-	-	-	-	0.124
half-rectified receive-time offset					
local mean	-	-	-	-	0.141
local max	-	-	-	-	-0.106
local median	-	-	-	-	-0.132
fine-grain variation in probe bandwidth					
local min	-	-	-	-	-0.134
local median	-	-	-	-0.060	0.097
local max	-	-	-	-	0.095

top coefficients combine different measures of CPU usage and idle time with variance measures on the server interrupt rate and the packet-delivery timing, similar to resource R2 and R4.

Again, this mathematically derived resource measure corresponds well to our intuition and to past results [3]. The VPx workload will not tend to overload the disk interface, since even its asynchronous transmission will be mostly served from the file-buffer cache. Each individual VPx client induces a larger workload on the CPU than the corresponding individual LPx, resulting in a smaller number of supported clients at saturation. Again, this is intuitive, since the VPx workload requires the additional overhead of non-synchronous transmission of the cached data. When normalized by their respective saturation points, VPx and LPx both use the CPU at nearly the same level. However, by referring to resource $R1$, it is clear that LPx saturates on the inability of the CPU to transfer packets through the network stack to the network interface. On the other hand, VPx has a much lower resource consumption for $R1$, in this case, the CPU is busy managing file-cache buffers. For server machines with a different combination of CPU and network-interface capacity, these relative usage numbers will change. This highlights the need for a well-defined, formulaic calibration methodology, so that streaming servers can be quickly characterized on the hardware that they will ultimately use when they are in service, without expert intervention.

***R4* Saturating resource for LUx: Variability in network load and packet delivery** Based on the more than 5% prediction-error standard deviation, this resource direction is the most difficult to capture from the raw measurements available. As shown in Table 8, the modeling coefficients are comparatively diffuse, with 80 coefficients required to collect 96% of the model vector norm. Looking at the detailed coefficients suggests that this resource is modeling the variability in the network load and the packet-delivery times. We base this statement on the large number of differences taken between different order statistics on the measurements that would capture this information. The differences taken between the order filters on the interrupts per second would capture variations in packet arrivals, since packet arrivals will generate interrupts. The

differences taken between the order filters on UDP activity will give a similar measure of network-usage variation. The differences in the probe-client receive-time offsets will tend to capture a measure of packet smoothing and of packet blitting. Combining that with the metrics describing fine-grain (sub-second) variations in the probe-received bandwidth can help to disambiguate these two packet-delivery behaviors.

This resource dimension is not as intuitive as the others. However, on consideration, it does make sense. For live unpopular workloads, the server must handle numerous interrupts, caused by incoming packets from the large number of live input streams. If the server does not give these inputs high priority, it could lose data and fail due to a size violation. If it does give these inputs high priority, it will result in less uniform transmission of packets to its output clients, increasing the chances of a rebuffering failure. These failures are most likely when the delivery timing on the live input streams is such that their relative packet delivery timing results in variable loading of the server. This type of failure will also be more difficult to predict, since it depends on timing coincidences that may not be local to the available measurement vector.

6. CONCLUSIONS

In this paper, we have described an approach to streaming-server calibration, monitoring, and saturation prediction. Our methodology generalizes across streaming servers and across streaming-server hardware configurations. Since our models are based on calibration data to select the most salient measures of saturation, new software-hardware configurations can be modeled without expert intervention.

Our saturation prediction does not assume that categorized client counts are available from the server. Instead we actively monitor the status of the server machine and, using a probe client, we actively sample the client-side statistics. We carefully structure our measurements so as to be low latency, with no measurement memory beyond the local one-minute period. Our use of time-localized models allows us to handle the dynamics of in-service streaming workloads. Our use of data-driven models allows us to detect and respond to transients in resource usage in a way that client counts would not allow.

To allow our work to be compared with previous models [3], we examined the saturated/unsaturated decision surfaces that would be generated by our client-to-usage models in combination with an explicit categorized client count. Our measurement-based prediction achieves the same mean performance as this oracle-based approach, with a lower variance in the per-experiment prediction performance.

Our measurement-based self-prediction results allowed us to correctly detect several mislabeled test runs.

We plan on examining the effects of model-adaptive reduction in the measurement vectors that are collected from the streaming server machine and from the probe clients. As pointed out in Section 5, the coefficients in the measurement-to-resource-usage models tend to be highly concentrated on a small number of measurement dimensions. By recursively omitting measurements during our calibration stage and retraining, we should be able to reduce the size of the measurement vector while minimizing the impact of this reduction on the prediction performance. Folding this process into the calibration stage will allow the data reduction to be responsive to the specific streaming server software and hardware.

7. REFERENCES

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] Apple Computer Inc. Darwin Streaming Server. http://developer.apple.com/darwin/projects/-streaming.

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

[4] L. Cherkasova, W. Tang, and A. Vahdat. Mediaguard: a model-based framework for building qos-aware streaming media services. In *SPIE Conference on Multi-Media Computing and Networking (MMCN)*, San Jose CA, 2005.

[5] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and analysis of a streaming-media workload. In *USENIX Symposium on Internet Technologies and Systems*, San Francisco CA, 2001.

[6] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, San Francisco CA, 2004.

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

[8] A. E. Dashti, S. H. Kim, C. Shahabi, and R. Zimmermann. *Streaming Media Server Design*. IMSC Press and Prentice Hall, 2003.

[9] R. Doyle, O. Asad, W. Jin, J. Chase, and A. Vahdat. Model-based resource provisioning in a Web service utility. In *USENIX Symposium on Internet Technologies and Systems*, Seattle WA, 2003.

[10] G. Golub and C. Van Loan. *Matrix Computations*. John Hopkins University Press, 1989.

[11] International Standard. *Information technology - Coding of audio-visual objects - Part 14: MP4 file format*. ISO/IEC 14496-14, first edition, 2003.

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

[13] RealNetworks Inc. Helix Universal Server. http://www.realnetworks.com/products/-media_delivery.html.

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

[15] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, 1998.

[16] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin. A hierarchical characterization of a live streaming media workload. In *ACM SIGCOMM Internet Measurement Workshop*, 2002.

[17] Y. Wang, M. Claypool, and Z. Zuo. An empirical study of real video performance across the internet. In *ACM SIGCOMM Internet Measurement Workshop*, San Francisco CA, 2001.