FAILURE PREDICTION IN VIDEO-STREAMING SERVERS THROUGH PERFORMANCE ANALYSIS OF SERVER AND CLIENT-SERVER INTERACTIONS

Carlos A S Cunha
Centre for Informatics and Systems of University of Coimbra
Portugal
email: ccunha@dei.uc.pt
Luis Moura e Silva
Centre for Informatics and Systems of University of Coimbra
Portugal
email: luis@dei.uc.pt

ABSTRACT

Video-streaming services are nowadays one of the biggest contributors to Internet traffic. Due to the real-time characteristic of video-streaming, very low failure detection and recovery delays are required. Proactive recovery in face of performance degradation is a prominent area to explore. Current performance analysis work in video-streaming focuses mostly on capacity planning and session admission through complex workload and resource modeling. These models have limited application when degradation is not explained by client workload (e.g., dynamic resource reallocation, software faults and misconfiguration).

We explore server-side monitoring of performance degradations in video-streaming servers, based on statistical analysis of server metrics and client-server interaction messages. Statistical analysis of event logs show that analyzed metrics can be used as symptoms of failures to anticipate them and thus enabling proactive recovery. Exception is server overloading caused by streaming of unpopular videos, which are exposed by metrics when QoS degradation is close to accepted quality thresholds.

KEY WORDS
Performance Failures, Video-streaming, Self-healing.

1 Introduction

Streaming services are overwhelming the Internet with TV, VoD and music contents. Studies show a large increase of video-streaming traffic in the Internet in the last years [1]. High definition IPTV and services like Youtube [2] and Joost [3] are high contributors to such exponential increase of traffic [4][5].

Zero downtime is desirable in streaming services. Streaming newcomers are traditional TV consumers, a class of people that soak up decades of TV quality and availability patterns. Due to the importance of these massified utility services in the day by day of people's life, consumers put high expectations on patterns of quality and continuity during watching periods.

The realtime characteristic of streaming demands a monitoring apparatus extremely efficient to avoid service disruption. Proactive techniques fit the requirement of achieving zero downtime, by anticipating session and service failures. Notwithstanding, detection of server instability to trigger repair actions before the occurrence of failures is a desirable achievement. To anticipate hard failures, the failure detector should implement the fail-stutter failure model [6] to address performance failures on top of the traditional fail-stop model.

Prediction of server performance resorts mostly to complex workload and resource modeling. Despite its effectiveness in predicting system overcapacity conditions, it fails in detecting other common causes of performance failures, as software aging, overloading and misconfiguration [7][8][9]. Moreover, it requires frequent model rebuilding in dynamic systems. The growing acceptance of virtualized environments to deploy applications also accelerates model deprecation, as virtual resources can be adjusted dynamically in runtime.

1.1 Approach Overview and Contributions

We propose a performance monitoring approach based on analysis of *server metrics* and *client-server interaction metrics*. Those metrics were selected to circumvent complex modeling, allowing their use in dynamic systems.

The validity of our approach depends on the existence of singularities in interaction and server metrics which enable their use as symptoms of performance failures. Our approach is validated by the following hypotheses:

- **Accuracy**. During performance degradation periods the chosen metrics presents abnormal values.
- **Timeliness**. Performance failures are detected before clients start to experience service quality degradation.

The rational behind the use of *interaction metrics* is that during faulty periods the server struggles to process new messages, leading to large server-side message queues and processing time delays. *Server metrics* cover session traversal monitoring, such as *throughput* and *packet delays* sent by server. These metrics uncover server's states of degradation affecting all or part of the current sessions set.

This paper has the following contributions:

- Application level performance monitoring approach of streaming-servers;
- Analysis of accuracy and anticipation abilities of server and client-server interaction metrics for detection of QoS failures.

1.2 Assumptions

Our work is based on the following assumptions. We address only VoD streaming. Failures are caused by performance degradation and the server should be able to process failure detection routines during resource exhaustion periods.

1.3 Paper Structure

This paper is structured as follows. Section 2 presents related work. Section 3 describes our failure detection approach. Section 4 presents the experimental methodology followed to validate our work. Section 5 presents results and analysis of experimental work. Section 6 presents conclusions.

2 Related Work and Alternative Approaches

Performance analysis work in video servers is prominently focused on admission control, capacity determination and overcapacity failure prediction.

Vin et al [10] proposed an admission control algorithm to guarantee statistical performance levels of streaming services. Prediction of performance degradation resorts to analysis of variation in the access times of media blocks from disk. A new client request is only accepted if the time spent retrieving a media block from disk after accepting the client maintains stable. Simulations show that the proposed algorithm doubles the number of clients serviced simultaneously, when compared with deterministic algorithms constraining the number of clients admitted.

Arias et. al. [11] studied overcapacity prediction in media servers through analysis of the relationship between server metrics (i.e., throughput and resources's usage) and quality of service specifications. Results of experimental work using the Darwin Streaming Server show that the server response time increases with the number of users connected and packet losses increase with the number of users, until reaching a throughput stabilization point. The server drops packets to maintain throughput at a certain level, even if resources's usage is below its respective saturation levels.

Cherkasova et. al. work [12] [13] address capacity determination of media servers for utility-aware streaming media services. Server capacity is specified by its maximum bandwidth and the number of connections when subjected to a workload mix characterized by popularity and encoding bit rate. The cost of each session (representing the resources required by that session) is calculated for each streaming type. Experimental results using several workload mixes demonstrated very low prediction errors. The approach requires session classification by popularity type. Such classification is not straightforward to obtain, introducing practical difficulties.

Covell et. al. [14] propose a modeling approach to predict overloading failures. Client metrics-to-resource and server measurements-to-resource models are proposed for prediction. *Total least squares* is used to model resource utilization and server measurements are represented using *order filters*. Model calibration requires labeled training data representative of pure workloads that lead to server saturation. Models are assessed for self-prediction (i.e., prediction of the current state) and cross-prediction (i.e., prediction of a given workload different from the actual). Results show that the saturation states used for validation were correctly predicted by the approach.

Seo et. al. [15] presented a performance study in two popular video-streaming services. Observations shown that *Startup delay* can be used as a client-side load predictor for Darwin Streaming Server. By contrast, *CPU usage* has low server-side prediction power in both servers.

3 Approach

We propose a monitoring approach for performance failures in video-streaming servers. We focus on performance degradation leading to QoS failures. By detecting performance degradation we expect to anticipate failures, enabling proactive recovery of server.

Our performance analysis concerns identification of singularities in client-server interaction metrics and server metrics, interpreted as symptoms of performance degradation that explain fail-stuttering scenarios. All metrics are collected and analyzed server-side to allow fast server adaption during performance degradation periods.

3.1 Failure Model

We address failure modes defined for the fail-stutter model [6]. Video servers are resource intensive, which means that variations in resource usage may impact considerably the QoS experienced by clients. Thus, resources are key components that should be monitored.

Our failure model encloses *session failures*, *packet losses*, *late packets* and *high interaction delays*. We ad-

dress failures caused by *server overloading* and *resource overloading*. *Server overloading* usually occurs when the server exceeds its normal capacity. It originates in one of the server resources or in the interaction between several resources. *Resource overloading* concerns exhaustion of individual resources (i.e., CPU, memory and I/O) caused by server overcapacity and other fault types - e.g., software faults and misconfiguration.

3.2 Monitoring

We built a Darwin Streaming Server [16] module to log server and interaction data server-side. Table 1 shows metrics considered for logging.

Type	Collected Metrics
Streaming Server	Throughput Number of connections Packets send delay
Processing Delays	Describe Setup Play Teardown
Inter-request delay	Describe - Setup Setup - Setup Setup - Play

Table 1. Performance metrics.

We minimize interference of network and client factors by ensuring their stability through the use of a session generator (Figure 1) located in the server's local network. The generator restricts client configuration to a single client specification with stable behavior and controlled network placement. Interaction metrics are collected server-side only for the sessions established with that generator.

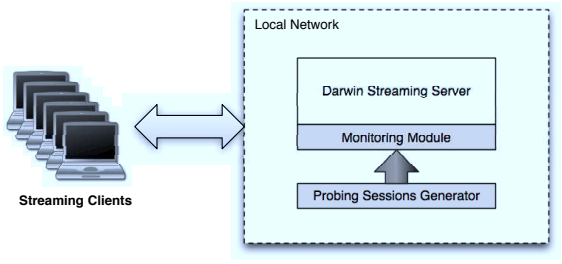


Figure 1. Monitoring Architecture

3.3 Analysis Window

Analysis of monitored data is performed continuously during server operation. We divide such data into time windows for analysis. The time-window size should be chosen carefully, as too large windows comprise detection time and too small windows introduce false positives during non-faulty periods.

3.4 Server Metrics

We consider high-level server metrics for performance analysis (Table 1). *Number of connections* and *throughput* are two important server performance metrics. We define performance degradation as the state where *throughput* (measured in kbps) and *number of connections* variation change unproportionally. We correlate both metrics to determine degradation levels.

Packet send delay metric measures the timespan between the specified packet send timestamp and the effective send time. Each packet has an assigned *send timestamp* that is used by the server to control the time each packet should be send. During faulty states, two scenarios can occur:

- **Positive delays**. The server struggles to send packets, leading to effective sent times later than the ones appearing in the timestamp;
- **Negative delays**. The server detects the gap between the packet sent timestamp and the effective time and start sending packets in advance to avoid packet delays.

The scenarios presented before are opposite between each other. Although, they can both help detect a faulty scenario, as the gap between the recorded and effective sent time grows in one direction (i.e., negative or positive) during faulty periods.

3.5 Interaction Metrics

The RTSP state machine is presented in Figure 2. It shows the sequence of states and corresponding RTSP commands that triggers state transition.

Client-server interaction metrics captures the time spent processing RTSP commands and inter-command delays (e.g., SETUP-PLAY). As RTSP commands are issued sequently by clients, inter-command delays capture queuing delays caused by the server inability in processing requests.

RTSP command processing times are calculated as the difference between command pre-processing times (i.e., the time server receives a given request) and post-processing times (i.e., the time server sends the response to client). Our monitoring module intercepts both events and logs them for further analysis. *Inter-command time* metrics capture the time window between a RTSP response and the server's reception of the next RTSP command. These metrics provide information about the time that a given request is stored in TCP and streaming server queues, assuming stable client-side and network delays.

3.5.1 Baseline

Interaction performance analysis demands a reference baseline of request processing and inter-request times.

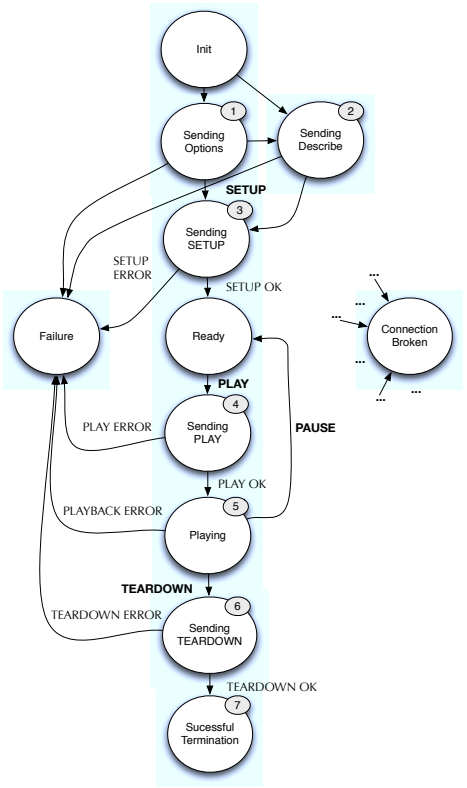


Figure 2. Client RTSP State Machine

Anomalies are determined as deviations from values of reference.

We classify performance degradation periods as the ones where interaction times exceeds normal values. Thresholds separating outliers from normal values are calculated according to Equation 1, a typical method for outlier detection in statistics [17][18].

$$max(3Q^{+}_{-}1.5IQR, 3Q(1+1/3)) \quad (1)$$

The *IQR* is calculated as the difference between the first and the third quartile. The second parameter of *max* is used when variance is close to zero.

4 Experimental Methodology

This section describes the experimental methodology followed to validate our approach. Further, we present the architecture and workload used for validation.

Our experimental methodology is described as follows. We launch short term sessions using a probing client located in the server's network, stress the server and individual resources and finally, perform statistical analysis to evaluate metrics's ability in predicting QoS failures observed by the client.

Validation comprises assessing the accuracy and anticipation ability of our failure prediction approach. Basically, we want to answer the following questions:

- Do predicted failure scenarios explain client experienced failures?
- Can QoS failures be predicted in advance?

To answer the previous questions, we compare server performance metrics with streaming QoS metrics observed by the probing client.

4.1 Workload

Validation of our approach for server overloading requires covering all workload type space. Streaming workload type in VoD is often characterized by its *popularity*, *bit rate* and *number of connections* [15][12][14]. We add *inter-session request time* to enclose the impact of transient states of bursty session arrivals. Table 2 presents the list of workload parameters and respective levels.

Popularity is an ambiguous metric, as location of a given object in the memory hierarchy is difficult to determine. Object access sequences and caching policies exhibit its location in a given moment. We generate a pure workload for each test to guarantee that we are streaming a specific workload type. *Popular* workloads are generated by streaming the same file on each session. By contrast, *unpopular* workloads are generated by allocating a different file for each stream.

Parameter	**Levels**
Number of connections	Up to its maximum capacity.
Encoding bit rate	300 Kbps 1000 Kbps 2000 Kbps
Popularity	Popular Unpopular
Inter-session request delay	Fast (200ms delay) Medium (500ms delay) Slow (2 seconds)

Table 2. Workload Parameters

4.2 Fault Injection

To evaluate prediction capabilities of our metrics we induce server overloading and exhaustion of individual resources.

Server overloading is induced by bringing server to its limit for each workload type. *Number of connections* vary from zero (idle) to 120% (overloading) of server capacity, for each *bit rate* and *popularity* level. Server workload limit is determined by observing the maximum number of connections admitted by the server without impacting QoS at clients.

Stress tool [19] forces exhaustion of individual resources by creating several processes consuming memory, CPU or I/O. Those processes are launched some time after server starts streaming a predefined workload.

4.3 QoS Analysis

QoS analysis aims to evaluate the ability of server-side captured metrics in predicting QoS degradation. We relate QoS collect at clients with server-metrics for several server states, during *normal*, *overcapacity* and *resource overloading* induced periods.

We launch one streaming session each 5 seconds, after which a TEARDOWN is issued. Sample frequency is multiplied using several probing clients. The client collects for each streaming session, the *startup delay*, *packet losses*, *late packets*, *session length*, and *session size*.

We define the following QoS thresholds to separate normal QoS from QoS failures:

- *Startup Delay:* 200% of normal value.
- *Session time and session size:* 120% of normal value.
- *Packet losses/Late packets:* 3%.

Association between QoS parameter values and client experienced failures is extremely complex due to inherent subjectivity of video content [20][21]. We establish reasonable values for QoS thresholds according to experimental work observations and previous QoS studies [20][21].

5 Experimental Work

This section presents the results and respective analysis of experimental work. To ensure statistical validity of results when evaluating prediction capabilities of metrics we repeat all tests 10 times for each configuration of parameters. Results obtained are coherent for all tests and thus, we present one representative graph for each configuration.

5.1 Testbed Configuration

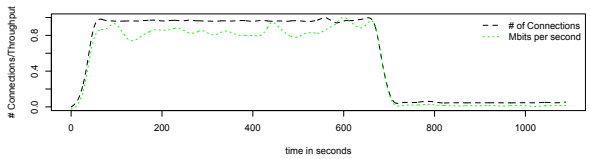
Our server is configured with a Intel(R) Pentium(R) D CPU 3.00GHz, 1Gb RAM, running a Linux 2.6.18-92.1.22.el5 Kernel. The other machines have similar characteristics. All computers are connected by a Gigabit Ethernet Network.

To emulate a wide area network connecting players and servers, we use the Netem [22] for traffic shaping. We chose reasonable parameters for WAN emulation [23].

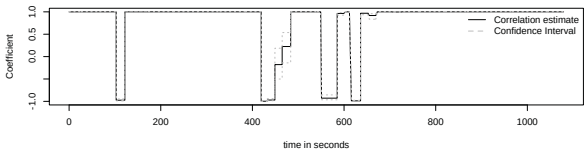
5.2 Selection of Representative Metrics

A pre-selection of server metrics through visual inspection of graphs discards processing times as metrics, due to their low prediction accuracy for several workload types.

We assessed the use of correlation between *throughput* and *number of connections* as performance metric. It showed low prediction precision due to fluctuations on *throughput* during normal server operation (Figure 3).



(a) Number of Connections and Throughput



(b) Pearson's Correlation Coefficient

Figure 3. Normalized *Number of Connections* and *Throughput* during normal and disk I/O stressed periods.

Command's processing times are also discarded in our analysis. In our experiments, inter-command metrics show higher accuracy over processing time metrics. Low prediction accuracy of processing times has origin in packet dropping policy implemented by server to maintain low packet delays during overloading [11]. Furthermore, they are redundant in cases where prediction accuracy of inter-command and processing times are similar.

5.3 Overcapacity Analysis

We evaluate the ability of interaction and server metrics in capturing anomalies caused by server overloading for different workload types.

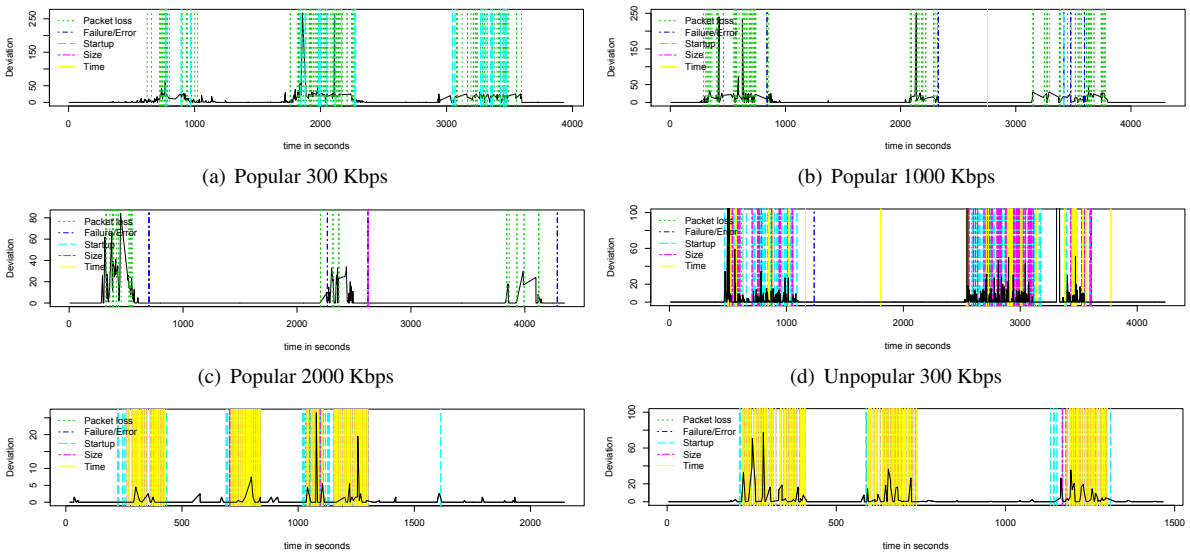
5.3.1 Interaction Metrics

Figure 4 shows deviations from the normality threshold for the more accurate interaction metric. Performance anomalies appear as vertical lines in the graphs. High amplitudes of interaction times appear before performance anomalies, which enable prediction of failures before their occurrence.

Unpopular videos are less CPU demanding than popular videos. Workload is disk-intensive for this streaming type, freeing CPU to process new command requests. Prediction timeliness for unpopular videos is reduced when compared with the popular counterpart. Failures precede abnormal inter-command times for high encoding bit rates, avoiding failure prediction in advance. Such phenomena is explained by the low number of client-server interactions due to the small number of sessions allowed by the server for this workload type.

5.3.2 Server Metrics

Figure 5 presents the cumulative difference between the *send time timestamp* established by the server and the *effective send time* for popular videos (see Section 3.4). During overloading periods we can observe a negative slope for the



(a) Popular 300 Kbps

(b) Popular 1000 Kbps

(c) Popular 2000 Kbps

(d) Unpopular 300 Kbps

(e) Unpopular 1000 Kbps

(f) Unpopular 2000 Kbps

Figure 4. Inter-command time deviations above the outlier threshold for popular and unpopular videos. Each graph shows responses for loads with inter-command times of 2, 1 and 0.2 seconds, in such order.

cumulative values, as the *effective send time* is higher than the *packet send timestamp*. As soon as the server detects overloading, it starts to send packets in advance to reduce the impact of overloading, causing a positive slope. Both positive and negative slopes are symptoms of performance degradation.

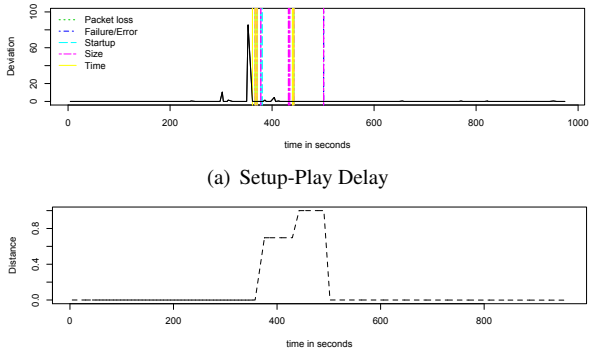
Packet send delay metric is less sensible to overloading caused by streaming of unpopular videos, compromising prediction accuracy of this metric.

5.4 CPU overloading

We generate a constant workload of 80 sessions of unpopular videos encoded at 300 Kbps and stress the server using the stress tool after 200 seconds. All metrics are sensible to CPU exhaustion as expected (Figure 6) caused by inability of CPU in processing new command requests and assuring timely delivery of packets.

5.5 I/O overloading

Disk I/O overloading is generated by the stress tool using several processes spinning on *sync()*. Interaction parameters are less sensitive to disk I/O overloading than to CPU overloading, as observable in Figure 7. Although, both interaction and server metrics capture failures before their occurrence.



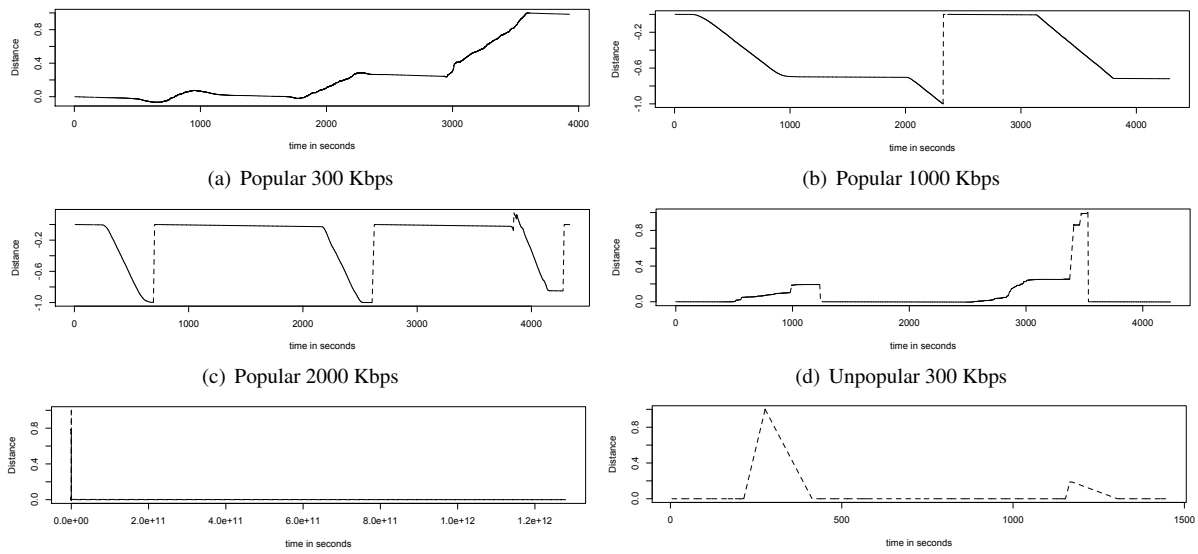
(a) Setup-Play Delay

(b) Packet Send Delay at Server

Figure 6. Interaction metrics and server metrics during CPU overloading

5.6 Memory overloading

The stress tool spawns several processes allocating and dirtying memory to generate a memory overloading condition. As observed in Figure 8, interaction metrics are sensible to performance degradation caused by memory overloading. By contrast, the *packet send delay* is less sensitive, uncovering failures several seconds after QoS degradation. Carefully analysis show that the metric is unable to capture failures when QoS degradation is close to its threshold, which separates normality from failure states. Although, the metric start to expose failures as degradation distantiates from respective QoS thresholds.

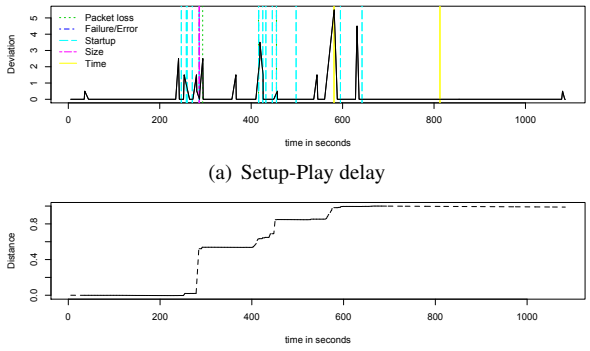


(a) Popular 300 Kbps (b) Popular 1000 Kbps

(c) Popular 2000 Kbps (d) Unpopular 300 Kbps

(e) Unpopular 1000 Kbps (f) Unpopular 2000 Kbps

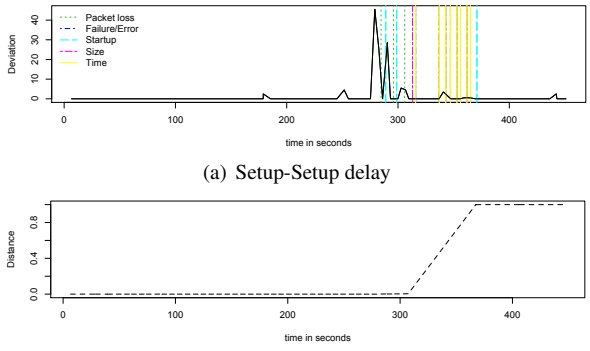
Figure 5. Cumulated difference between *packets send timestamp* and their *effective send time* for popular and unpopular videos.



(a) Setup-Play delay

(b) Packet Send Delay at Server

Figure 7. Interaction metrics and server metrics during I/O overloading



(a) Setup-Setup delay

(b) Packet Send Delay at Server

Figure 8. Interaction metrics and server metrics during memory overloading

5.7 Analysis of Results

We evaluated interaction and server metrics in the context of prediction of QoS performance degradation. We executed all tests 10 times, for each workload and fault type, to ensure statistical representativeness of results.

Table 3 summarizes the ability of each metric in capturing failure scenarios and its timeliness. Observed results show that interaction and server metrics predict failures for most workload parameter configurations during server overloading and resource overloading conditions. When interaction and server metrics are used together, failures are predicted for all workload configurations.

All metrics have timeliness problems capturing overloading conditions for high bit-rate unpopular videos. Such observation can be explained by lower CPU usage due to the reduced number of connections supported by server, all of them streamed from disk.

6 Conclusion

We propose a performance degradation monitoring approach for video-streaming servers. Monitoring resorts to outlier detection of client-server interaction times and server parameters representative of QoS failures.

Our approach accurately and timely predicts performance failures caused by excess of workload and other faults compromising resources' availability. Exception is server overloading caused by streaming unpopular videos, which have limited anticipation of QoS failures. Thoroughly analysis show that only server states causing QoS degradation close to thresholds are ignored by our metrics. As QoS thresholds are defined below the ideal limit (which

Resource	Bit Rate	Send Delay		Setup-Setup		Setup-Play	
		Timeliness	Prediction	Timeliness	Prediction	Timeliness	Prediction
CPU		X	X	X	X	X	X
IO		X	X		X	X	X
Memory		*	X	X	X	X	X
Server Overloading (Popular)	300k	X	X	X	X	X	X
	1000k	X	X	X	X	X	X
	HD	X	X	X	X	X	X
Server Overloading (Unpopular)	300k	X	X	X	X	X	X
	1000k	**	**	**	X	**	X
	HD	**	**	**	X	**	X

* Unable to capture failures close to threshold ** Low accuracy

Table 3. Metrics' ability to capture failure states and metrics' timeliness

depend on video characteristics - e.g., format and encoding bit rate) we believe that we could attain timeliness in real-world systems.

Online server adaption is allowed by low computational demanding detection and abstinence of communication due to server-side collected metrics.

We believe that our metrics are generic enough to be used in other streaming servers through a similar analysis of *server collected metrics* vs *QoS metrics*. The dependence of metrics undertaken with respect to server implementation makes conclusions taken in performance studies hard to extrapolate between servers. Although, it is expected deviations on packet send delays and interaction times during performance degradation periods in other servers as well. These deviations are used as symptoms for QoS failures.

As future work, we will perform a thorough study to uncover timeliness values for prediction. Due to the number of workload combinations, the number of tests required to attain statistical significant values is considerably high.

Acknowledgment

This work was partially supported by FCT-Portugal under grant SFRH/BD/35784/2007 and CISUC (Centre for Informatics and Systems of University of Coimbra).

References

[1] "Igi forecast 2010." [Online]. Available: http://www.highbeam.com/doc/1G1-192407604.html

[2] "Youtube." [Online]. Available: http://www.youtube.com/

[3] J. Lei, L. Shi, and X. Fu, "An experimental analysis of joost peer-to-peer vod service," *Peer-to-Peer Networking and Applications*. [Online]. Available: http://dx.doi.org/10.1007/s12083-009-0063-5

[4] "Ipoque press release." [Online]. Available: http://www.ipoque.com/userfiles/file/press-release-20082009-study-english-final.pdf

[5] Allot, "Global mobile broadband traffic report." [Online]. Available: http://www.allot.com/mobiletrends.html

[6] R. Arpaci-Dusseau and A. Arpaci-Dusseau, "Fail-stutter fault tolerance," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, May 2001, pp. 33–38.

[7] D. Oppenheimer and D. A. Patterson, "Studying and using failure data from large-scale internet services," in *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop*. New York, NY, USA: ACM, 2002, pp. 255–258.

[8] S. M. Pertet and P. Narasimhan, "Causes of failure in web applications," Carnegie Mellon University, PDL Technical Report PDL-CMU-05-109, December 2005.

[9] J. Gray, "Why do computers stop and what can be done about it?" in *Fifth IEEE Symposium on Reliability in Distributed Software and Database Systems*, C. S. Press, Ed., 1996, pp. 3–12.

[10] H. Vin, P. Goyal, and A. Goyal, "A statistical admission control algorithm for multimedia servers," in *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*. New York, NY, USA: ACM, 1994, pp. 33–40.

[11] J. Arias, F. Suárez, D. García, X. García, and V. García, "Evaluation of video server capacity with regard to quality of the service in interactive news-on-demand systems," *Protocols and Systems for Interactive Distributed Multimedia*, pp. 15–25, 2002.

[12] L. Cherkasova and L. Staley, "Building a performance model of streaming media applications in utility data center environment," in *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2003, p. 52.

[13] ——, "Measuring the capacity of a streaming media server in a utility data center environment," in *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*. New York, NY, USA: ACM, 2002, pp. 299–302.

[14] M. Covell, B. Seo, 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.

[15] B. Seo, M. Covell, M. Spasojevic, S. Roy, R. Zimmermann, L. I. Kontothanassis, and N. Bhatti, "Reliable performance data collection for streaming media services," in *ICEIS (1)*, 2006, pp. 124–129.

[16] Darwin streaming server. [Online]. Available: http://dss.macosforge.org/

[17] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 10 2004. [Online]. Available: http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9

[18] D. H. Nash, "Outliers in statistical data : Vic barnett and toby lewis, 2nd ed. (wiley, new york, 1984), pp. 463," *International Journal of Forecasting*, vol. 2, no. 3, pp. 390–391, 1986. [Online]. Available: http://ideas.repec.org/a/eee/intfor/v2y1986i3p390-391.html

[19] "Stress tool." [Online]. Available: http://weather.ou.edu/ apw/projects/stress/

[20] D. Wijesekera, J. Srivastava, A. Nerode, and M. Forrsti, "Experimental evaluation of loss perception in continuous media," *Multimedia Syst.*, vol. 7, no. 6, pp. 486–499, 1999.

[21] R. Steinmetz, "Human perception of jitter and media synchronization," *Selected Areas in Communications, IEEE Journal on*, vol. 14, no. 1, pp. 61–72, Jan 1996.

[22] "http://www.linuxfoundation.org/en/net:netem." [Online]. Available: http://www.linuxfoundation.org/en/Net:Netem

[23] S. Hemminger, "Network emulation with netem," in *Proceedings of the 6th Australia's National Linux Conference (LCA2005)*, Canberra, Australia, April 2005.