Session Based Admission Control: A Mechanism for Improving the Performance of an Overloaded Web Server

Ludmila Cherkasova, Peter Phaal
Computer Systems Laboratory
HPL-98-119
June, 1998

E-mail: [cherkasova,phaal]@hpl.hp.com

web servers, session-based workload, performance, simulation, overloaded server, admission control mechanism, web quality of service

In this paper, we introduce a new, session-based workload for measuring a web server performance. We define a session as a sequence of client's individual requests. We then measure server throughput as a number of successfully completed sessions. Using a simulation model, we show that an overloaded web server can experience a severe loss of throughput measured as a number of completed sessions comparing against the server throughput measured in requests per second. Moreover, statistical analysis of completed sessions reveals that the overloaded web server discriminates against longer sessions. For e-commerce retail sites, longer sessions are typically the ones that would result in purchases, so they are precisely the ones for which the companies want to guarantee the completion.

We introduce a *session based admission control (SBAC)* to prevent a web server from becoming overloaded and to ensure that longer sessions can be completed. If a server is functioning near its capacity a new session will be rejected (or redirected to another server if one is available). If there is enough capacity, the admission control mechanism will admit a new session and process all future requests related to it. We introduce a simple implementation of session based admission control based on server CPU utilization and analyze its properties and performance characteristics.

We show that a web server augmented with the admission control mechanism is able to provide a fair guarantee of completion, for any accepted session, independent of a session length. This provides a predictable and controllable platform for web applications, and is a critical requirement for any e-business.

Contents

1 Introduction

As the Internet matures, companies are implementing mission critical Internet applications. These applications provide dynamic content, integrate with databases and offer secure commercial transactions.

Customers are becoming increasingly reliant on these complex business applications for services such as banking, product purchases and stock trading. These new services make greater demands on web servers at a time when traffic is increasing rapidly, making it difficult to ensure adequate level of service.

Evaluation of web server performance generally focuses on achievable throughput and latency for request-based type of workload as a function of traffic load. SpecWeb96 benchmark [SpecWeb96] is an industry standard for measuring Web Servers performance. It is based on generating HTTP requests to retrieve different length files accordingly to a particular distribution. The server performance (throughput) is characterized as a maximum achievable number of connection per second while maintaining the required file mix.

However, commercial applications impose a set of additional, service level expectations. Typically, access to a web service occurs in the form of a *session* consisting of many individual requests. Placing an order through the web site involves further requests relating to selecting a product, providing shipping information, arranging payment agreement and finally receiving a confirmation. So, for a customer trying to place an order, or a retailer trying to make a sale, the real measure of a web server performance is its ability to process the entire sequence of requests needed to complete a transaction.

In fact, a session is present even in the case when a client requests only a single web page. For example, accessing a company home page involves requesting the HTML page, and then making further requests for all the images embedded in this document. Thus even simplest client action may require a web server to provide up to 6-12 separate files.

In this paper, we introduce a new model of workload based on sessions. We discuss and extract the essential server and client parameters necessary to build a simplified simulation model. Session-based workload gives a new interesting angle to revisit and re-valuate the definition of web server performance. It naturally proposes to measure a server throughput as a number of successfully completed sessions.

Let us consider the situation when a server is processing a load that exceeds its capacity.

If a load consists of single, unrelated requests then the server throughput is defined by its maximum capacity, i.e. a maximum number of connections the server can support. Any extra connections will be refused and extra load-requests will be dropped. Thus, once a server has reached its maximum throughput, it will stay there, at a server maximum capacity.

However, if the server runs a session-based workload then a dropped request could occur anywhere in the session. That leads to aborted, incomplete sessions. Using a simulation model, we show that an overloaded web server can experience a severe loss of throughput when measured in completed sessions while still maintaining its throughput measured in requests per second. As an extreme, a web server which seems to be busily satisfying clients requests and working at the edge of its capacity could have wasted its resources on failed sessions and, in fact, not accomplishing any useful work. Statistical analysis of completed sessions reveals that an overloaded web server discriminates against the longer sessions. Our analysis of a retail web site showed that sessions resulting in sales are typically 2-3 times longer than non-sale sessions. Hence discriminating against the longer sessions could significantly impact sales and profitability of the commercial web sites.

Quality of service is a way of describing the end to end performance requirements and conditions that a particular application imposes to be successfully executed. For a web server running a commercial application the following *web quality of service* requirement is crucial:

- a fair chance of completion for any accepted session, independent of session length.

We introduce *session based admission control* as a way to provide a web quality of service guaranties for a server running a session-based workload.

The main goal of a session based admission control is the prevention of web server from overload. An admission control mechanism will accept a new session only when a server has the capacity to process all future requests related to the session, i.e. a server can guarantee the successful session completion. If a server is functioning near its capacity, a new session will be rejected (or redirected to another server if one is available).

We introduce a simple implementation of session based admission control based on server CPU utilization.

Deferring a client at the very beginning of their transaction (session) rather than in a middle - is another desirable web quality of service property for an overloaded server. It will minimize an amount of wasted server work.

We believe that sending a clear message of rejection to a client is very important. It will stop clients from unnecessary retries which could only worsen the situation and increase the load on the server. However, issuing an explicit rejection message imposes an additional load on a web server. We derive a worst case bound to estimate a rejection overhead as a function of the applied load and an average session length. For the most load values and workloads of interest – the overhead is less than 5-10% of total server work.

We examine trade off between two desirable properties for an admission control mechanism: *responsiveness* and *stability* and introduce a family of admission control policies which cover

the space between stable and responsive policies. To compare different admission control policies, we design special metrics, taking into account both: policy performance and policy ability to guarantee a specified web-quality of service. We conclude with a few recipes to tune the admission control mechanism for better performance.

We show that a web server augmented with session based admission control is able to provide a fair guarantee of completion, for any accepted session, independent of a session length. This provides a predictable and controllable platform for web applications, and is a critical requirement for any e-business.

On May 11, 1998, Hewlett-Packard, as a part of its "How to succeed in E-Business" [HP-98] press event, announced the intrduction of the HP Service Control product [HPSD-98]. This product deploys the session based admission control mechanism, described in this paper, in order to ensure the high levels of service required to successfully complete commerce transactions on the web.

2 Workload Model: Requests versus Sessions

WebStone [WebStone] and SpecWeb96 [SpecWeb96] are the industry standard benchmarks for measuring web server performance. Using a finite number of clients to generate HTTP requests they retrieve different length files according to a particular file size distribution.

For example, SpecWeb96 file mix is defined by the files (requests) distribution from the following four classes:

- 0 Class: 100bytes - 900bytes (35%)
- 1 Class: 1Kbytes - 9Kbytes (50%)
- 2 Class: 10Kbytes - 90Kbytes (14%)
- 3 Class: 100Kbytes - 900Kbytes (1%)

The web server performance is measured as a maximum achievable number of connection per second supported by a server when retrieving files in the required file mix. Current typical web servers running SpecWeb96 achieve 1000 - 4000 connections per second per processor.

Commercial applications exhibit very different behavior: a typical access to a web service consists of a sequence of steps (a sequence of individual requests). A transaction is successful only when the whole sequence of requests is completed. The real measure of server performance is the server's ability to process the entire sequence of requests needed to complete a transaction.

We introduce a notion of a session as a unit of session workload. *Session* is a sequence of clients individual requests.

In our simulation, the session structure is defined by the following parameters:

- client (sender) address;
- original session length;
- current session length;
- time stamp when the session was initiated.

For a new session, the original and current session lengths coincide. For a session in progress, the current session length reflects the number of requests left to complete.

We define a request as a structure specified by the following parameters:

- the session that originated the request;
- requested file size;
- time stamp when the request was issued.

Throughout this paper, we consider a file mix as defined by a SpecWeb96. So, the individual requests retrieve the files defined by a SpecWeb96 distribution.

The client issues the next request only when it receives a reply for the previous request. The client issues its next request with some time delay, called *think time*. Think time is a part of the client definition rather than a session structure. The client waits for a reply for a certain time, called *timeout*. After a timeout, the client may decide to repeat its request – this is termed a retry. A limit is placed on retries – if this limit is reached and the reply is not received in time, both the request and the whole session is aborted.

Thus, a client model is defined by the following parameters:

- client address;
- think time between the requests of the same session;
- timeout - a time interval where the client waits for a server reply before reissuing the request;
- the number of retries before the session is aborted.

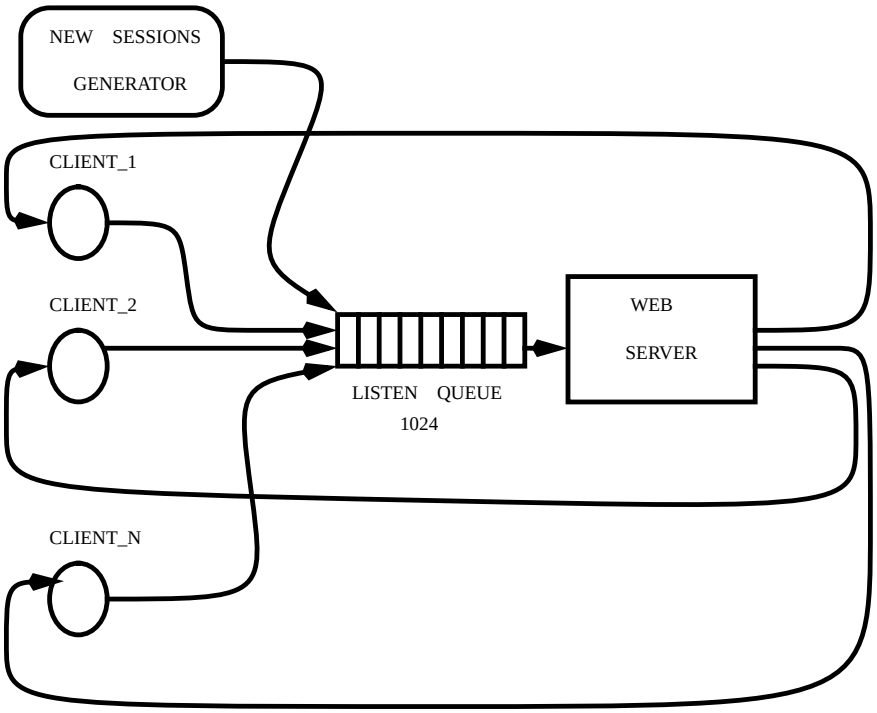

Figure 1: Basic Structure of Simulation Model.

A session is successfully completed when all its requests are successfully completed.

We will evaluate web server performance in terms of successfully completed sessions.

3 Server Model: Functionality and Basic Parameters

To understand the difference in web server behavior while it runs request-based or session-based workloads we built a simulation model using C++Sim [Schwetman95]. Basic structure of the model is outlined in Figure 1.

It consists of the following components:

- a session workload generator;
- N clients;
- a web server.

A session workload generator produces a new session request accordingly to specified input model parameters:

- session load and

- sessions length distribution.

A session request (i.e first request of a session) is sent to a web server and is stored in the server *listen queue*. We limit the size of the listen queue to 1024 entries which is a typical default value.

In this way, we are able to use an open model for sessions generation. Each consequent request from a session is issued and handled by a specified client. Client behavior is defined by a closed (feed back) loop model: the client issues the next session request only when it receives a reply from the previous request.

Two reasons could cause a request, and a session it belongs to, to be aborted:

- if a listen queue is full then the connection to a server is refused, and both the request and the whole session is aborted.
- after issuing the request, the client waits for a server reply for a certain time. After timeout, the client resends the request. There is limited number of *retries*. If the reply still has not been received in time, both the request and a whole session is aborted.

REMARK: When the client receives "connection refused" message due to a full listen queue, he/she can try to resend the request again. In case of overloaded server, it only can worsen the situation. We decided to simplify the model by aborting the request and the whole session, when a listen queue is full, without an additional client retry.

In this paper, we assume that a server capacity is 1000 connections per second for SpecWeb96. This assumption does not influence the results validity. In those rare cases, when assumed server speed can influence the results of the study, we will have special remarks and discussion related to the matter.

4 Overloaded Web Server: Behavior and Characteristics

4.1 Simplified Model: Fixed Length Sessions. Analysis of Client Parameters: Think Time, Timeout, Number of Retries

Let us start with a simplified session model: we will assume that all the sessions have the same length.

Using this simplified model we will investigate the importance of the server and client parameters, such as think time, timeout, and the number of retries, in order to narrow the simulation space.

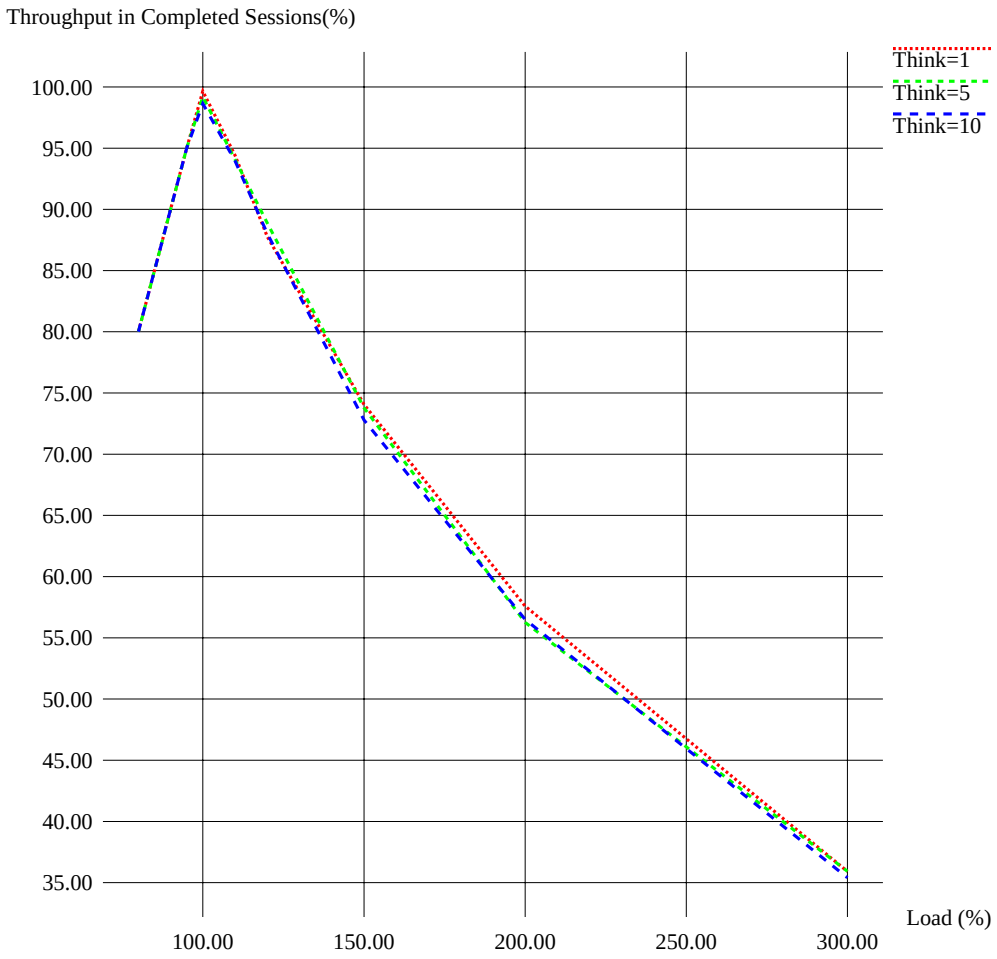

Figure 2: Varying Think Time for Overloaded Server with Fixed Session Length=5.

First, we assume that client does not timeout, i.e. it does not retry to resend the request. It is equivalent to an assumption that the client timeout is infinite. In this case, there is only one reason for a request, and a session it belongs to, to be aborted:

- a request arrived to a server, and a server listen queue is full.

Let us see whether the think time parameter in a client model impacts the server performance and simulation results. A particular think time during a simulation is defined by exponential distribution with a given mean think time.

Figures 2, 3 show the server throughput in completed sessions, for running a workload with a fixed session length: 5 and 50 correspondingly. Mean think times of 1 second, 5 seconds and 10 seconds do not affect the server throughput for short sessions at all (see Figure 2 for session length = 5).

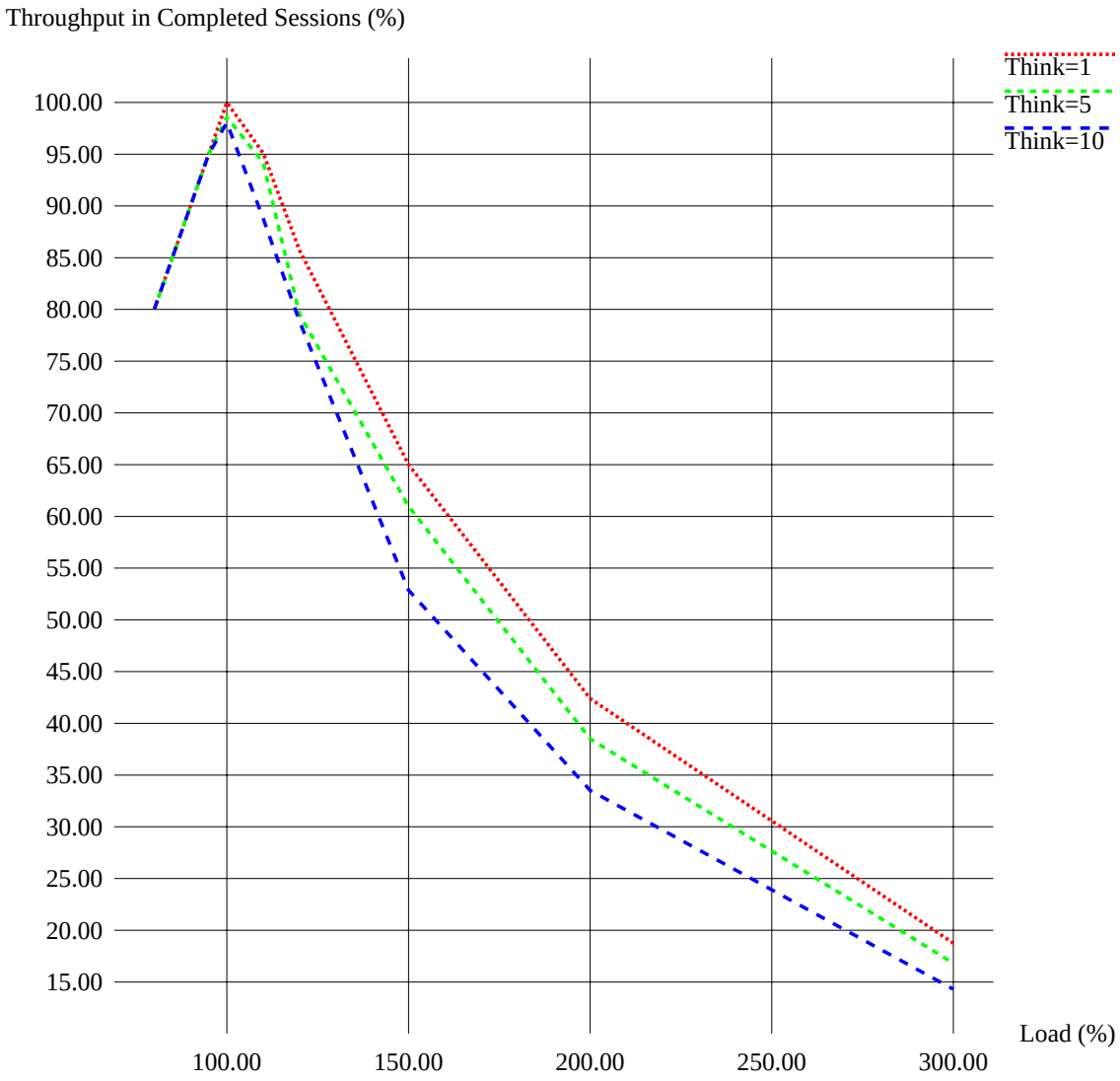

Figure 3: Varying Think Time for an Overloaded Server with Fixed Session Length=50.

Longer think times only slightly decreases server throughput for longer sessions (see Figure 3 for session length = 50). In general, the impact of think time is insignificant and independent of session length. Hence we can narrow a simulation space by assuming a client with a fixed mean think time.

For the rest of the paper we consider a client with a mean think time of 5 seconds.

Figure 4 shows a server throughput in completed sessions for a workload with a fixed session length: 1, 5 , 15 and 50 and a simplified client model: mean think time = 5 seconds, no retry. We introduce a session length of one as a special case when all the requests are independent. This represents, in fact, a server running a regular SpecWeb96 benchmark as a workload. Figure 4 clearly illustrates the difference in performance for an overloaded server running

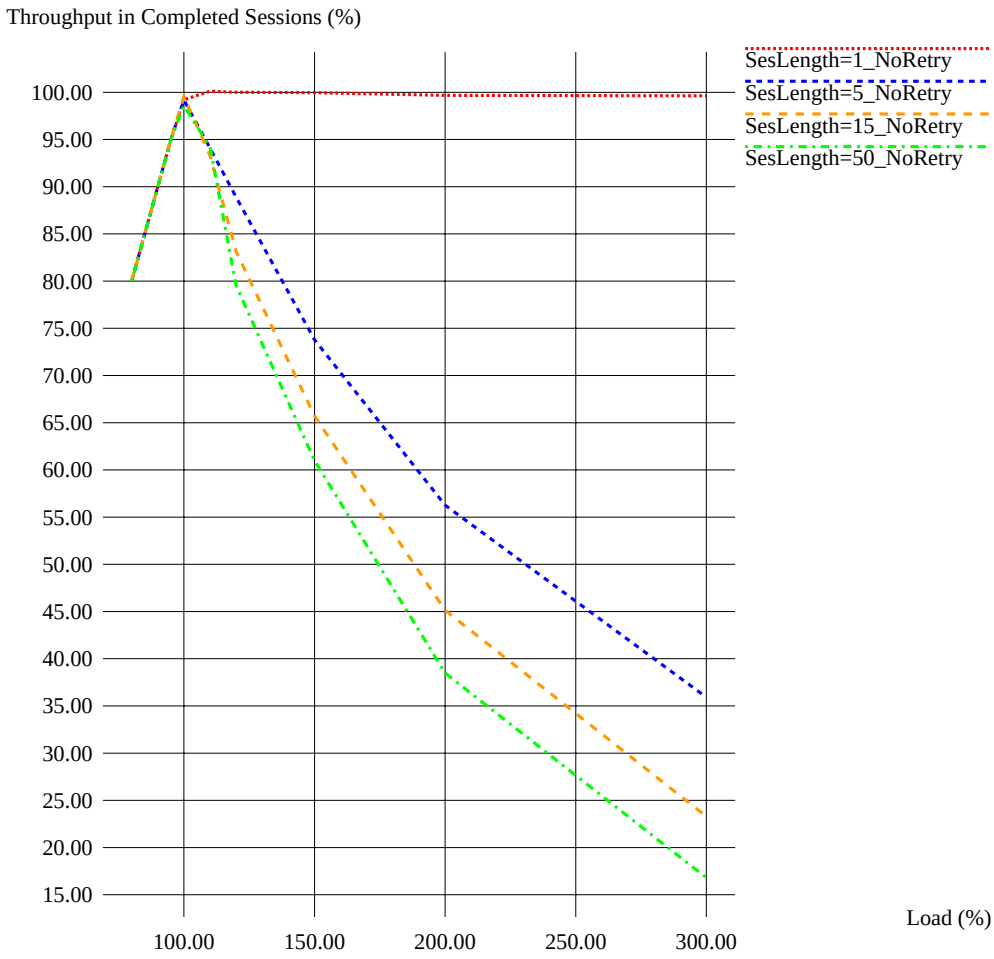

Figure 4: Throughput in Completed Sessions for an Overloaded Server with a Fixed Session Length and No Retry.

a "single request workload" and a server running a session-based workload. In case of a single request workload, a server throughput is reaching its maximum capacity of 100% at 100% of load and stays steady at 100% for higher load. While for session workload - the server throughput in completed sessions is dramatically decreasing with a higher load and longer sessions. The curves of this figure provide a most pictorial motivation for studying the overloaded web server performance with respect to session-based workloads.

Figure 5 shows a server throughput in completed sessions for a workload with fixed session length: 1, 5 , 15 and 50 and a following typical client model:

- mean think time = 5 sec,
- timeout = 1 sec,
- number of retries = 1.

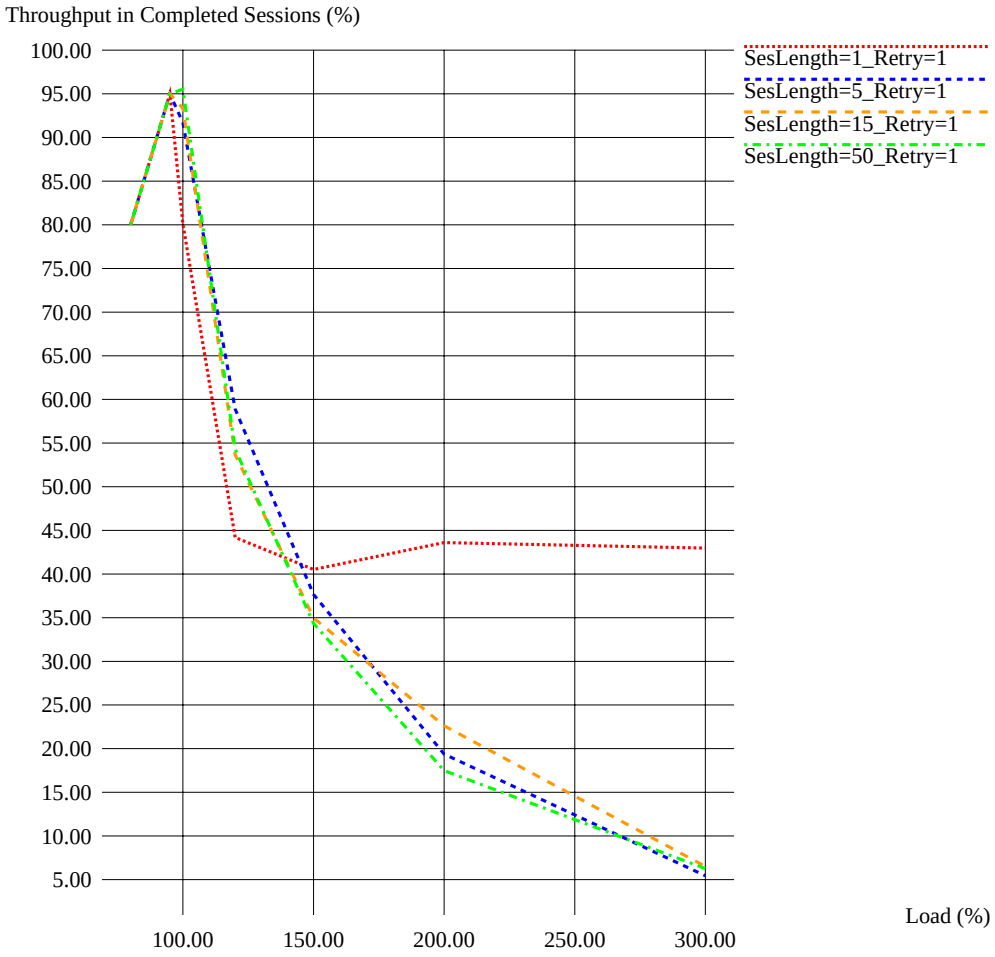

Figure 5: Throughput in Completed Sessions for Overloaded Server with a Fixed Session Length and One Retry with Timeout = 1sec.

The server performance in the overloaded region has dropped significantly in both cases: a server running a "single request workload", and a server running a session-based workload.

There is a simple relation based on:

- the server capacity,
- listen queue size and
- the client timeout value

which explains the probability of retries issued by the client.

The server listen queue is limited to 1024 buffers. If a listen queue gets full, then any coming requests get a "connection refused" message. Since a server processes 1000 connections per

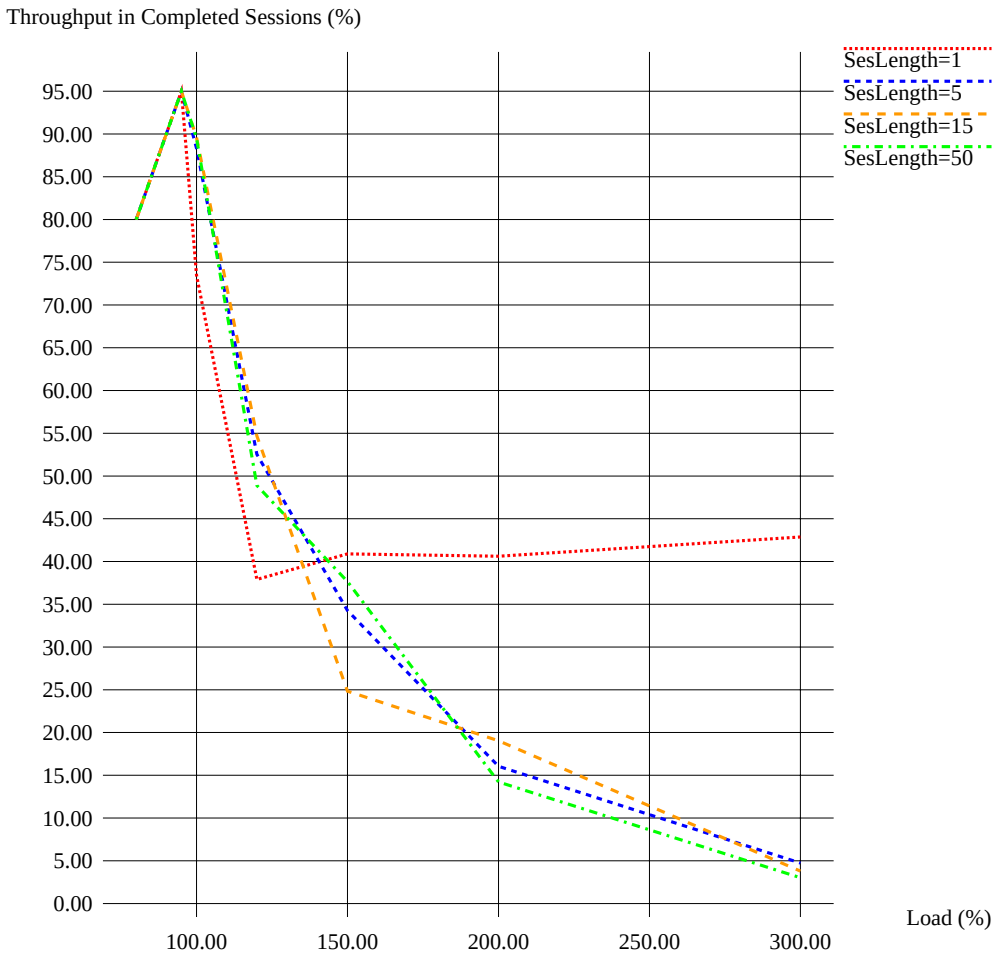

Figure 6: Throughput in Completed Sessions for Overloaded Server with Fixed Session Length, Timeout = 1sec and Two Retries.

second but the listen queue length is 1024 (1024 > 1000), it creates a possibility that the requests which are currently last in the listen queue, will be processed by the the server later than one second after they have been issued by the clients. If this happens, a client will abort a previous connection and send a retry. It leads to a server being even more overloaded. Many sessions (especially, a single request long) will be aborted because they are not able to meet the timeout and retry requirements. A client timeout of 1 second might be considered as an additional quality of service requirement: it sets a limit on a request latency to 1 second. If this requirement is not met (after a given number of retries) the session is aborted.

Figure 6 shows a server throughput in completed sessions, for a workload with a fixed session length: 1, 5 , 15 and 50 and a typical client model:

- mean think time = 5 sec,
- timeout = 1 sec,

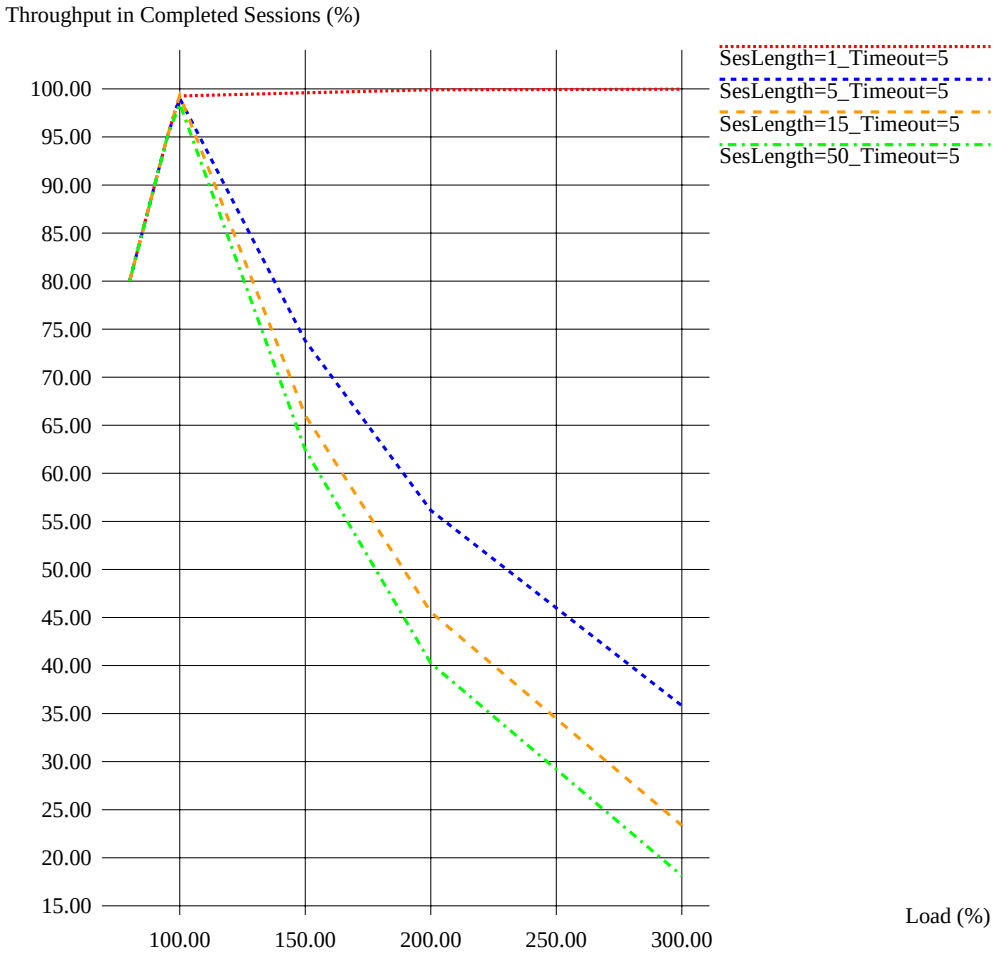

Figure 7: Throughput in Completed Sessions for an Overloaded Server with a Fixed Session Length and One Retry with Timeout = 5sec.

- number of retries = 2.

The server performance results are the same as for the client with one retry (shown in Figure 5) since it is very likely for the reissued requests either to see the listen queue being already full, or again to get at the very end of the listen queue and be not able to meet the latency requirements for the second time.

Figure 7 shows a server throughput in completed sessions, for a workload with a fixed session length: 1, 5 , 15 and 50 and a typical client model:

- mean think time = 5 sec,
- timeout = 5 sec,
- number of retries = 1.

Since a server processes 1000 connections per second and the listen queue length is 1024, the latency to process any accepted request is less than 2 seconds. Since the client timeout is 5 seconds, as specified above, it means that a client is never going to retry. Indeed, the results shown in Figure 4 and Figure 7 are the same. If the client timeout is greater than a server time needed to process all the requests from the listen queue then it eliminates the possibility of client timeouts and retries.

Since we are interested in studying a model with a full range of possible client-server interactions, we carefully select the model parameters allowing them.

Without loss of generality for the rest of the paper, we assume a model with the following client parameters:

- a think time between the requests of the same session is exponentially distributed with a mean of 5 seconds;
- a timeout - the time client waits for a reply before resending the request - is set to 1 second;
- a number of retries to resend the request after timeout is 1.

4.2 Extended Model: Exponentially Distributed Length Sessions

In this section, we will lift the fixed session length assumption which we used in Section 4.1, to analyze and to justify the client parameters and will introduce a more general session model.

For the rest of the paper, we assume the session lengths to be exponentially distributed with a given mean. In order to analyze the server behavior depending on a session length, we have performed experiments for session lengths with a mean of 5, 15 and 50.

Figure 8 shows throughput in completed sessions for an overloaded web server.

These results are quite different from fixed session length case, shown in Figure 5. First of all, the server throughput in completed sessions is much higher. Second, the ordering is somewhat counterintuitive: web server performance is better for workloads with a longer session mean.

How can it be explained?

First of all, the server throughput is measured as a number of completed sessions. Now, we have sessions of different length, since the session lengths are exponentially distributed. Our first explanation of the above phenomenon is that shorter sessions have higher chances to complete. Thus, the better "quantitative" value of throughput can be obtained at a price of "lower quality" of this throughput.

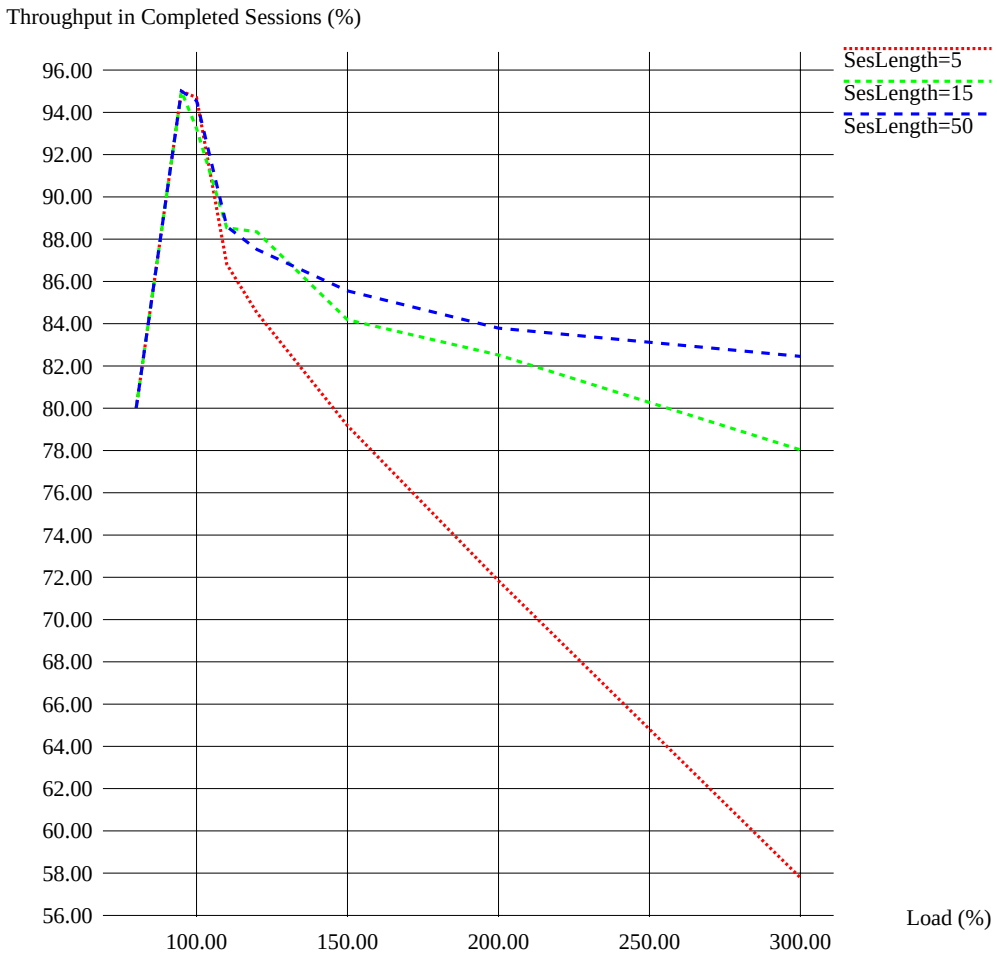

Figure 8: Throughput in Completed Sessions for Overloaded Server.

The second explanation is of a different nature. Let us consider a session length distribution with a mean of 50. When a long session gets aborted, it creates a potential amount of unused server resources, big enough to service several short sessions. Applying the same reasoning to a session length distribution with a mean of 5 – we can see that the difference between "long" and "short" session lengths for this distribution is less significant: often several sessions are aborted before it creates enough "unused" server resources to complete an additional session.

Let us analyze the simulation results in detail. Figure 9 shows the average session length of completed sessions against the average session length of all generated sessions as the model input. As it is clearly seen, the average session length of completed sessions is significantly lower than the original,input distribution. For the load of 300% and the original average session length of 5, 15 and 50, the average length of completed sessions is 1.7, 4.3 and 13.4 correspondingly.

The session lengths are defined to be exponentially distributed with a given mean m. In order

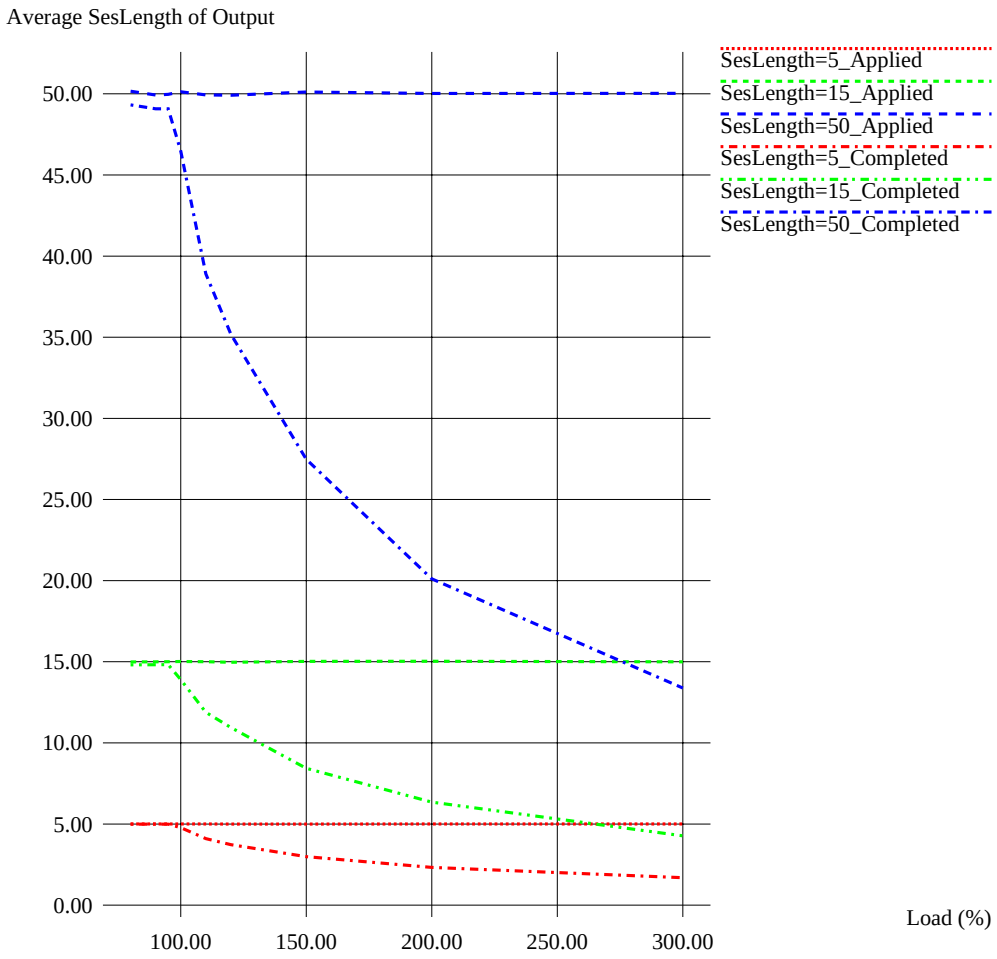

Figure 9: Average Length of Completed Sessions.

to analyze the distribution of the completed sessions in more detail, we will partition them in the following three bins:

- first bin: the sessions shorter or equal to m;
- second bin: the sessions longer than m but shorter or equal to $2 * m$;
- third bin: the sessions longer than $2 * m$;

Figure 10 shows the percentage of original and completed sessions in three bins by length for an overloaded server running a session-based workload with a mean of 50.

The original distribution by session lengths is the following:

- first bin: 63%;

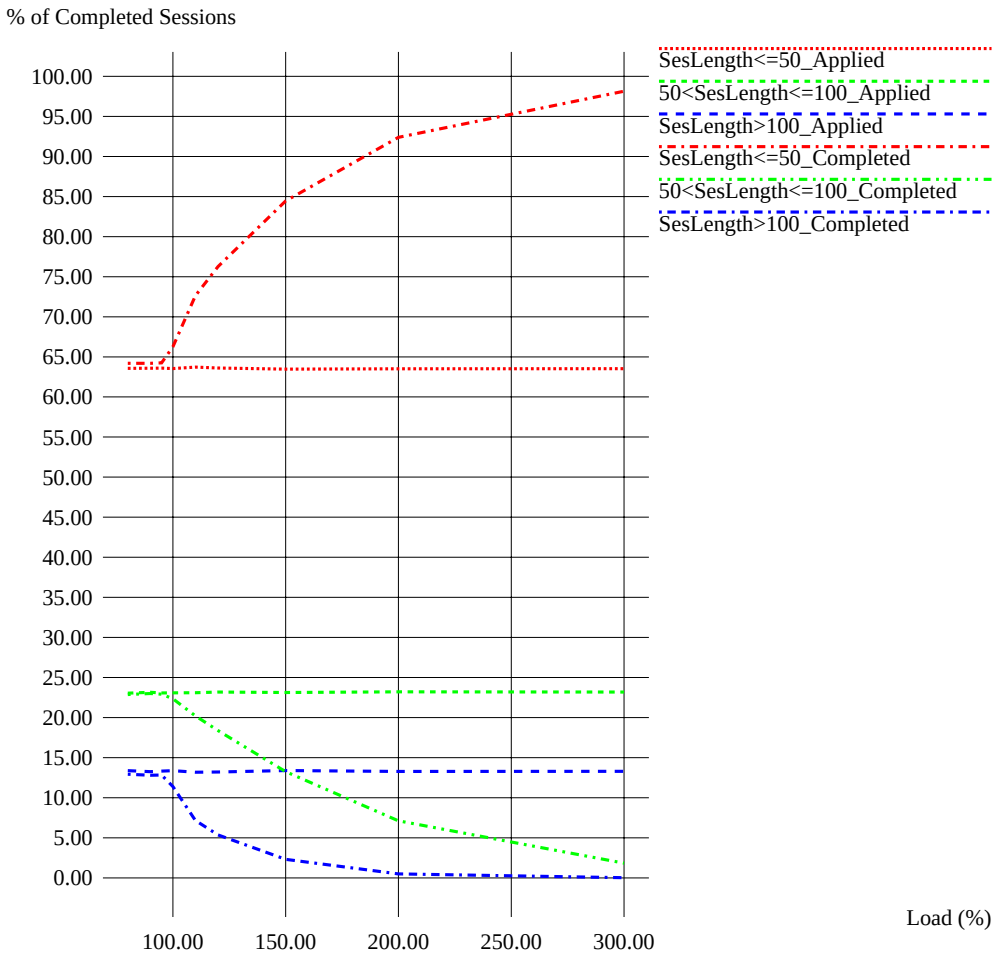

Figure 10: Percentage of Completed Sessions in Three Bins by Length for an Overloaded Server Running Session-Based Workload with Mean = 50.

- second bin: 23%;
- third bin: 14%.

The distribution of completed sessions under 300% load changes dramatically:

- first bin: 98.14%;
- second bin: 1.83%;
- third bin: 0.03%.

Indeed, the overloaded web server discriminates against the long sessions in a quite severe way: almost all the completed sessions fall in the first bin, the sessions from the second and the third bins are practically absent.

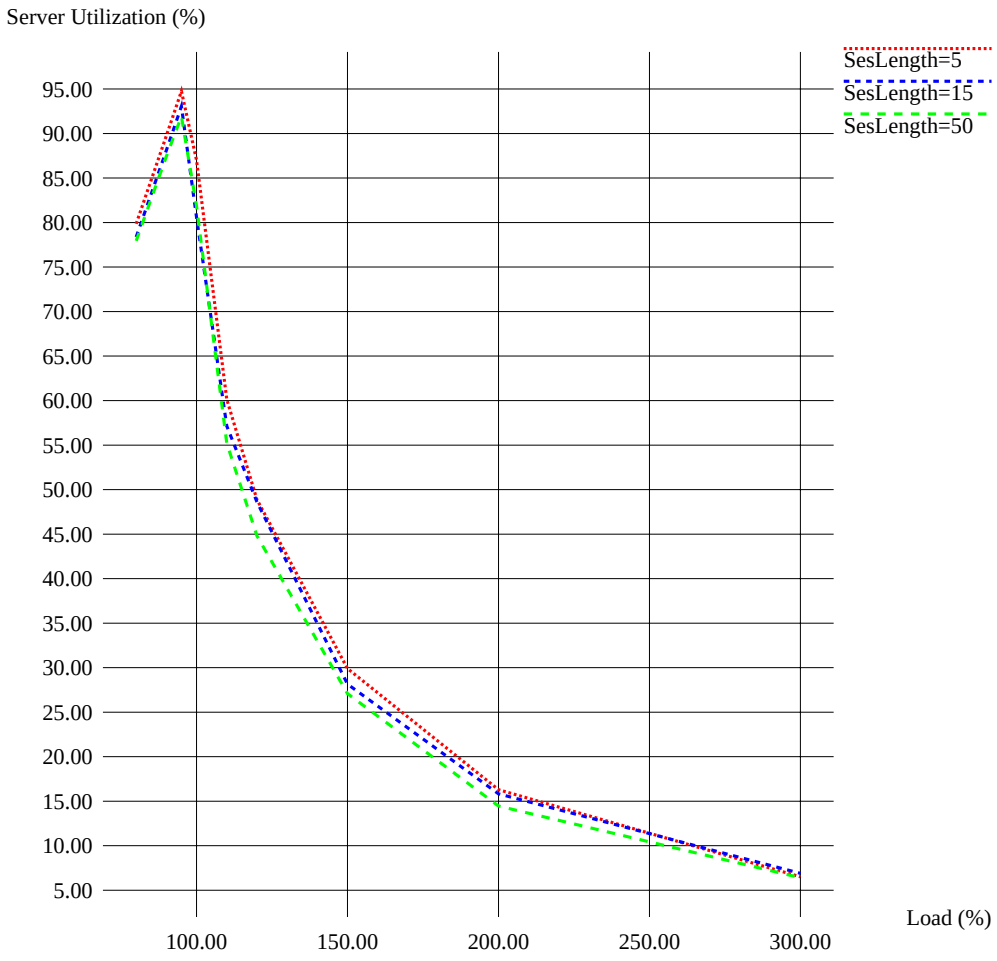

Figure 11: Server Useful Utilization of Processing Sessions which Complete.

To complete the analysis of an overloaded web server running a session-based workload we introduce a new performance measure: *useful server utilization.* Traditionally, a server performance is characterized by its throughput and utilization. We have shown a difference in throughput of an overloaded web server, when measured in percentage of completed requests and in percentage of completed sessions. We apply the same idea to characterize server utilization. We define *useful server utilization* as server busy time spent processing only sessions which complete.

Figure 11 shows useful server utilization as a function of load and session length. The results are overwhelming: the overloaded, "busy looking" server produces an amazingly small amount of useful work: around 15% for a 200% load; less than 7% for a 300% load.

This concludes our preliminary analysis of the behavior and performance characteristics of an overloaded web server running a session-based workload. This section raises rather serious question: is such server behavior expected and acceptable for commercial sites? Since the

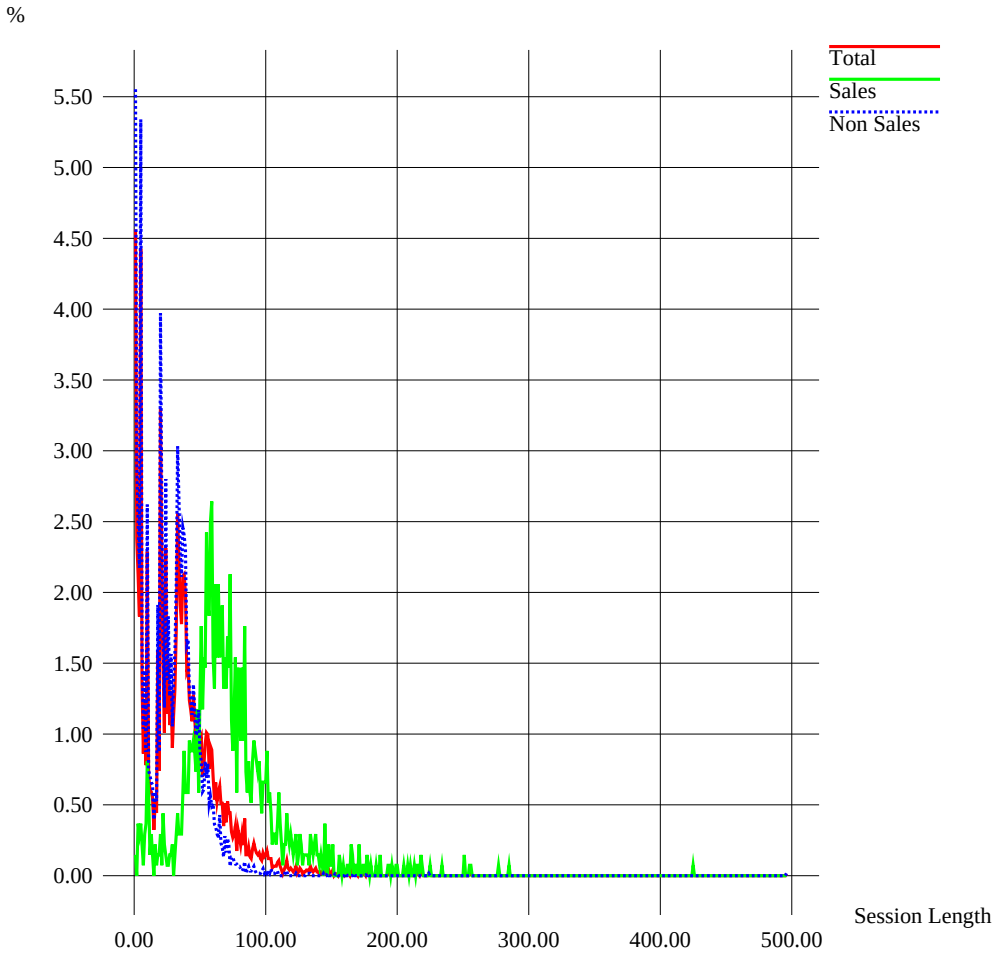

Figure 12: Commercial Workload Session Length Distribution.

answer is rather obvious, the next question to ask is: can a web server be augmented with session based admission control mechanism to prevent the server from becoming overloaded and to ensure that longer sessions are completed?

5 Sessions Length Distribution for Commercial Web Sites

In order to outline a workload space of interest and narrow the simulation space, we have analyzed web server access log data from a particular commercial site. This commercial site allows small businesses to purchase products online. This site provides the clients with product catalogues to browse, ability to add selected products to a "shopping cart", and finally to purchase the contents of the shopping cart, completing the sale. Figure 12 shows a session length distribution, specific for sales and non-sales transactions.

The distributions clearly show that the sale sessions are much longer than non-sale sessions. The following table summarizes the distribution statistics:

	Mean Session Length	Percentage of Sessions
Total	36.5	100%
Sales	73.6	18.3%
Non-Sales	28.2	81.7%

(1)

The average session length of *a sale* is more than 2.5 times longer that of *non-sale*. If we apply the partitioning in three bins proposed in Section 4.2 then sale-sessions belong to the second and third bins. As it was shown the vast majority (98%) of sessions which complete on an overloaded server fall in a first bin. As a result it would significantly impact sales and profitability of the site.

6 Web Quality of Service Requirements

An overloaded server has poor throughput for longer sessions because it is unable to sustain the level of service needed to complete these sessions. As we saw in Section 5, visitors making purchases tend to generate longer sessions and are most affected by inadequate service levels.

We introduce the term, *web quality of service*, to describe the service levels needed to complete web sessions. A web server that ensures a fair opportunity and guarantee of completion for all sessions, independent of session length, exhibits good web quality of service.

A competing requirement is the site operators desire to maximize the number of sessions completed. Server throughput should be maximized subject to providing adequate web quality of service. Our notion of useful server utilization captures this combined goal and results in the objective of maximizing useful server utilization.

One of the effects of poor web quality of service is that large numbers of sessions are aborted. Web site visitors may abort sessions because:

- at some point during their session they receive "connection refused" messages (in the case when the server listen queue is full).
- at some point during their sessions delays become intolerable and even retries (which further overload the server) fail to yield a timely response.

To summarize, a web server providing good web quality of service will have the following characteristics:

1. Visitors will have a fair chance of completing sessions, independent of session length.
2. Server will minimize waisted work in order to maximize useful server utilization.
3. There will be a minimal number of aborted sessions (ideally zero).

7 Session Based Admission Control Mechanism: Responsiveness vs Stability

To satisfy the web quality of service requirements discussed in Section 6, we introduce a *session based admission control mechanism* for a server handling a session-based workload.

The main goal of an admission control mechanism is to prevent a web server from becoming overloaded. We introduce a simple admission control mechanism based on the server CPU utilization.

The basic idea of a session based admission controller is as follows: the server utilization is measured during predefined time intervals (say, each second). Using this measured utilization (for the last interval) and some data characterizing server utilization in the resent past, it computes an "observed" utilization. If the observed utilization gets above a specified threshold then for the next time interval (i.e. the next second), the admission controller will reject all the new sessions and will only serve the requests from already admitted sessions. Once the observed utilization drops below the given threshold, the server (controller) changes its policy for the next time interval and begins to admit and process new sessions again.

Formally, the admission control mechanism is defined by the following parameters:

- U_{ac} – an *ac-threshold* which establishes the critical server utilization level to switch on the admission control policy;
- $T_1, T_2, ..., T_i, ...$ – a sequence of time intervals used for making a decision whether to admit (or to reject) new sessions during the next time interval. This sequence is defined by the *ac-interval length*;
- f_{ac} – an *ac-function* used to evaluate the observed utilization.

We will distinguish two different values for server utilization:

- $U_i^{measured}$ – a measured server utilization during T_i – the i-th ac-interval;
- $U_{i+1}^{observed}$ – an observed utilization computed using a given ac-function f_{ac} after ac-interval T_i and before a new ac-interval T_{i+1} begins, i.e. $U_{i+1}^{observed} = f_{ac}(i+1)$.

In this paper, we will consider ac-function $f_{ac}(i+1)$ defining $U_{i+1}^{observed}$ in the following way:

- $f_{ac}(1) = U_{ac}$;
- $f_{ac}(i+1) = (1-k) * f_{ac}(i) + k * U_i^{measured}$, where k is a coefficient between 0 and 1, and it is called *ac-weight coefficient*.

A web server with an admission control mechanism re-evaluates its admission strategy on specified by the time intervals $T_1, T_2, ..., T_i, ...$ boundaries. Web server behavior for the next time interval T_{i+1} is defined in the following way:

- If $U_{i+1}^{observed} > U_{ac}$ then any new session arrived during T_{i+1} will be rejected, and web server will process only requests belonging to already accepted sessions.
- If $U_{i+1}^{observed} \leq U_{ac}$ then web server during T_{i+1} is functioning in a usual mode: processing requests from both new and already accepted sessions.

There are two desirable properties for an admission control mechanism: *responsiveness* and *stability*. If a server's load during previous time intervals is consistently high, and exceeds its capacity, then responsiveness is very important: the admission control policy should be switched on as soon as possible, to control and reject newly arriving traffic. However, if the server receives an occasional burst of new traffic, while still being under a manageable load, then the stability, which takes into account some load history, is a desirable property. It helps to maximize server throughput and does not unnecessary reject newly arriving traffic.

As we can see, these two properties are somewhat contradictory:

- responsiveness leads to a more restrictive admission policy (since there is a chance of "over reacting" to occasional traffic bursts while overall a server is not yet overloaded). It aims to achieve higher web quality of service guaranties at a price of slightly lower server session throughput (in particular, when a server operates in a heavy load area but is not yet overloaded).
- stability takes into account a server's load history. In such a way, that it delays a first reaction of an admission control policy to the overload, while it still looks like an occasional burst, rather than a consistent overload. If a total server load is still around the server capacity then such a strategy allows better server session throughput to be achieved. However, if the overload is consistent then a less restrictive rejection policy inevitably leads to a higher rate of aborted sessions, and as result to poorer session completion characteristics.

The value of coefficient k in definition of f_{ac} introduces a family of admission control policies which cover the space between stable and responsive policies.

If $k = 1$ then the admission control policy is based entirely on a value of measured server utilization during the last ac-interval. Let us call this strategy *ac-responsive*.

If $k = 0.1$ then the admission control policy decision is strongly influenced by a server load prehistory, while the impact of a measured server utilization during the last ac-interval is limited. Let us call this strategy *ac-stable*.

8 Cost of Rejection

Session based admission control combines several functions:

- it measures and observes the server utilization;
- it rejects new sessions when the server becomes critically loaded;
- it sends an explicit message of rejection to the client of a rejected session.

We believe that sending a clear message of rejection to a client is very important. It will stop clients from unnecessary retries which could only worsen the situation and increase the load on the server. If the server promises to serve these clients, say in five minutes, it might be enough to resolve the current overload and provide a high level of service without loosing customers. Commercial sites might use some additional stimuli and bonuses issued in these rejection messages to keep their customers satisfied.

However, issuing an explicit rejection message imposes an additional load on a web server. The higher the load – the greater the number of rejection messages sent by the server. How large is the rejection overhead? What percentage of total messages constitutes the rejection messages?

This section derives a worst case bound to estimate the rejection overhead as a function of the applied load and average session length.

We use the following denotations:

- S_r - a server capacity in requests, i.e. number of connections (requests) per second a server can sustain.
- S_s - a server capacity in sessions, i.e. number of sessions per second a server can complete.
- $SesLength$ - an average session length.
- $Load$ - an applied load in sessions ($Load = 2$ means a load of 200% of server capacity).
- x - a number of rejected sessions per second.
- y - a number of completed sessions per second.

First of all, there is a simple relation between S_r, S_s and $SesLength$:

$$S_s = \frac{S_r}{SesLength} \qquad (1)$$

Since S_s is a server capacity in sessions and $Load$ is an applied load in sessions, $Load * S_s$ is a total number of issued sessions per second. Obviously, the sum of completed and rejected sessions per second is a number of sessions in total, a server has received per second:

$$x + y = Load * S_s \qquad (2)$$

There are two types of sessions: completed and rejected ones. Each completed session implies that a client consequently makes, on average, the number of requests defined by the $SesLength$. Each rejected session is equivalent to processing a single request - a worst case estimate of the cost of sending an explicit rejection message to the client. Thus a number of requests per second handled by a server is defined in the following way:

$$y * SesLength + x = S_r \qquad (3)$$

Replacing S_s in (2) with a formula (1), and expressing y from (2), we have the following equation:

$$y = \frac{Load * S_r}{SesLength} - x \qquad (4)$$

Replacing y with (4) in equation (3) we can express x:

$$x = \frac{S_r * (Load - 1)}{SesLength - 1} \qquad (5)$$

Since x is a number of rejected sessions (rejection messages) per second, and S_r defines a total number of requests per second processed by a server, then a percentage of rejection messages from the total number of requests is defined as follows:

$$\frac{100\% * x}{S_r}$$

Let us call this percentage the $RejectionPercentage$. Here is the final equation:

$$RejectionPercentage = 100\% * \frac{(Load - 1)}{SesLength - 1} \qquad (6)$$

The rejection overhead depends on average session length and the load received by the server. Figure 13 illustrates the rejection overhead as a percentage of rejection messages to a total number of requests per second.

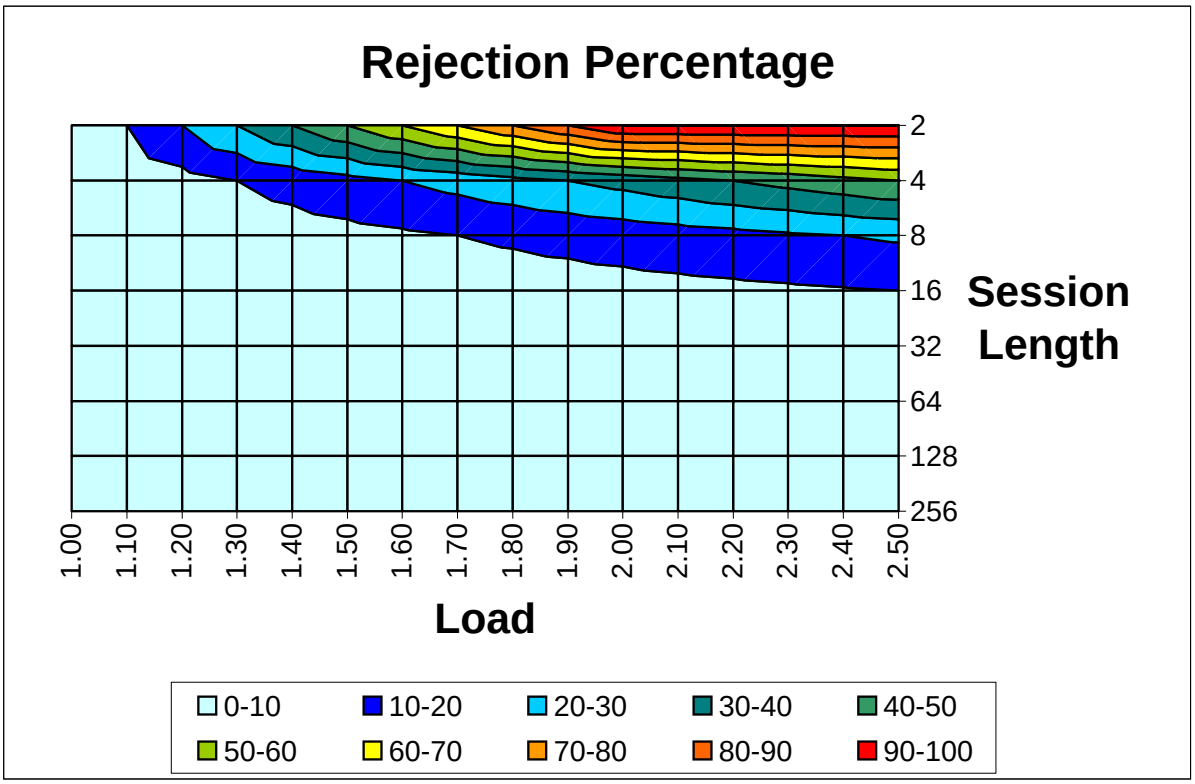

Figure 13: Rejection Cost as a Percentage of Rejection Messages to a Total Number of Requests per Second.

The rejection cost varies depending on the average session length and applied load: the higher the load and the shorter the session length – the higher the rejection overhead. However, for most of the load values and workloads of interest – the overhead is less than 10%.

REMARK: Formula (6) holds for the $Load$ and $SesLength$ values, satisfying the following condition: $Load - 1 \leq SesLength - 1$. For the other values, formula (6) is meaningless and reflects the situation that the applied load is so high that the server's capacity is not enough to send all the rejection messages.

For example, let us consider a server with a capacity of 1000 requests per second: $S_r = 1000$, and let an average session length be 5: $SesLength = 5$. Then a server capacity in sessions is 200: $S_s = 200$ accordingly to (1). Load of 500% will produce 1000 sessions per second which is a maximum request rate server can sustain. Thus all the server capacity will be consumed sending the rejection messages.

The same value is produced by the formula (6) computing 100% of rejection cost.

9 Overloaded Web Server with Session Based Admission Control: Behavior and Characteristics

This section analyzes the simulation results of an overloaded web server augmented with session based admission control.

We analyze the results produced by the ac-responsive admission control policy introduced in Section 7 (i.e. ac-weight coefficient k=1) with the following parameters:

- ac-threshold $U_{ac} = 95\%$
- ac-interval length of 1 second;

a web server augmented with such an admission control policy re-evaluates its admission strategy each second. Since the ac-responsive policy, $U_{i+1}^{observed}$ is defined entirely by the cpu utilization measured during i-th second, i.e. $U_{i+1}^{observed} = U_i^{measured}$.

If a measured cpu utilization for the previous i-th second is above the ac-threshold, i.e. $U_i^{measured} > 95\%$ then any new session arriving during the next second will be rejected, and web server will process only requests belonging to already accepted sessions. Otherwise, for the next second, the web server is functioning in a usual mode: processing requests from both new and already accepted sessions.

We performed the experiments for the average session lengths of 5, 15 and 50. We varied a load from 80% to 300%. The session workload with mean of 5 is not a realistic representative of commercial workloads. However, we included this case to cover the simulation space and understand the possible admission control limitations. The same can be said about a load of 300%: if a web server is consistently overloaded more than 200% it is a time to increase capacity and to extend it with an additional server. However, for completeness, and to understand the general behaviour of admission control mechanism we included a load of 300% too.

Figure 14 shows throughput in completed sessions. At a first glance, the only results for sessions with mean length of 50 look perfect. In fact, the curves correspondent to sessions with mean of 15 and 5 are justified and, somewhat in line with our expectations. Using formula (6) in Section 8 for the load of 300% we receive 14% and 50% of rejection overhead correspondingly to send an explicit rejection message. The percentage of completed sessions is largely offset by that amount.

One of the goals of the admission control mechanism is to minimize the number of aborted sessions (ideally, reducing them to 0) by explicit session rejection. Figure 15 shows the percentage of aborted sessions from those admitted for processing. The results for sessions with

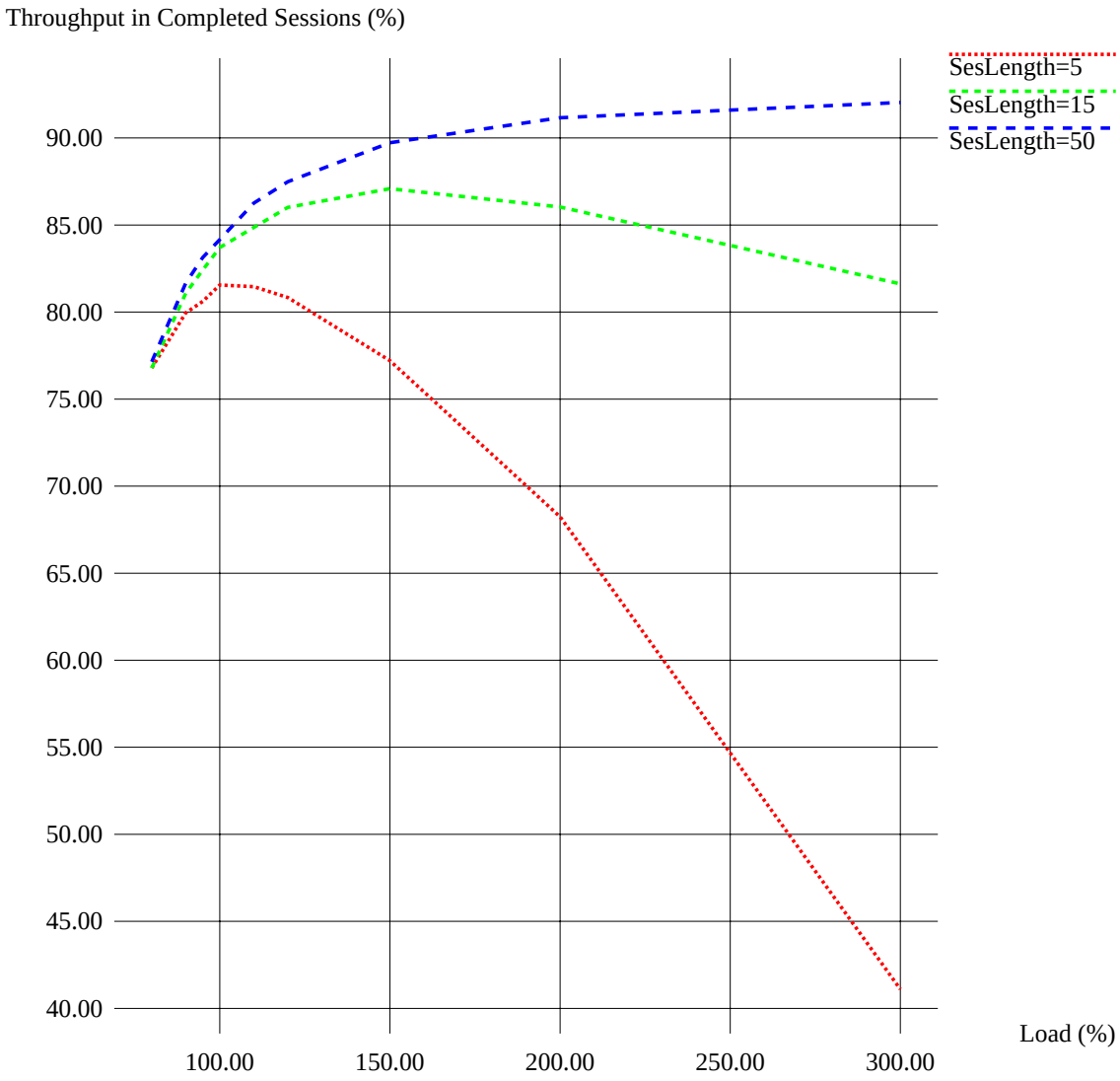

Figure 14: Throughput in Completed Sessions for Server with Admission Control.

a mean of 15 and 50 are perfect across the whole load space. They meet the desired quality of service requirement: zero aborted sessions from those accepted for service.

For a workload with mean of 5, the results are getting worse at load greater than 200%. The reason is that the shorter the average session length – the higher the number of sessions generated by the clients and accepted by the server during the ac-interval (i.e. 1 second). For example, if a web server is in "accept mode" then for a load of 300%, during one second it accepts around 600 new sessions, in addition to the sessions which are already in progress.

The main reason for aborted sessions under this scenario is that the listen queue overflows. One way to fix the problem is to reduce the ac-interval. Figure 20 shows significantly improved percentage of aborted sessions for a workload with mean of 5 and an admission control mech-

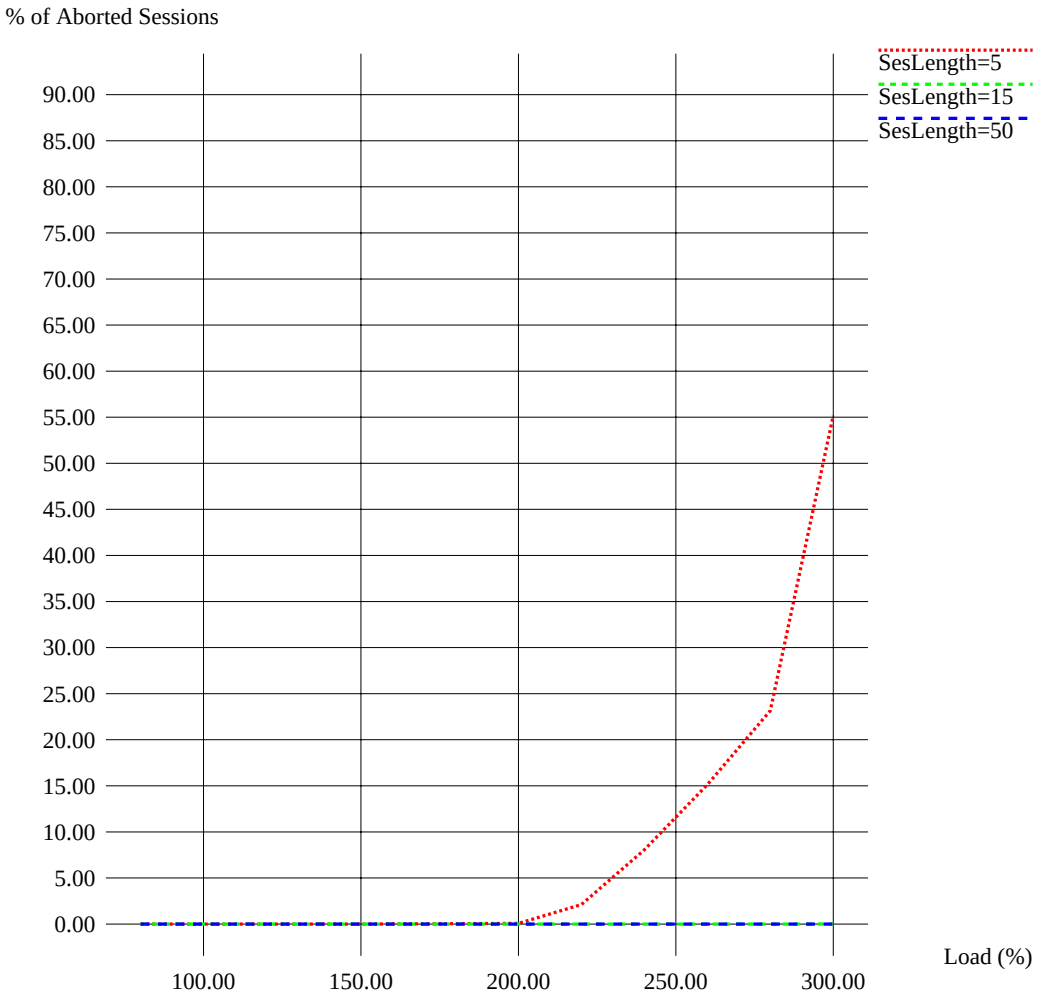

Figure 15: Percentage of Aborted Sessions from Admitted for Processing by Server with Admission Control.

anism with an ac-interval of 0.5 seconds. We will discuss further how to tune an admission control strategy for better performance in Section 11.

One of the main goals of the admission control mechanism is to ensure completion of any accepted session, independent of a session length.

Figure 16 shows the average session length of completed sessions against the average session length of all generated sessions as the model input. The results are perfect for sessions with a mean of 15 and 50 across the whole load space. For a workload with mean of 5, the results are getting slightly worse at load around 300% of server capacity.

The admission control mechanism dramatically improves the "quality" of the web server output compared with the similar results for a web server with no admission control shown in Figure 9.

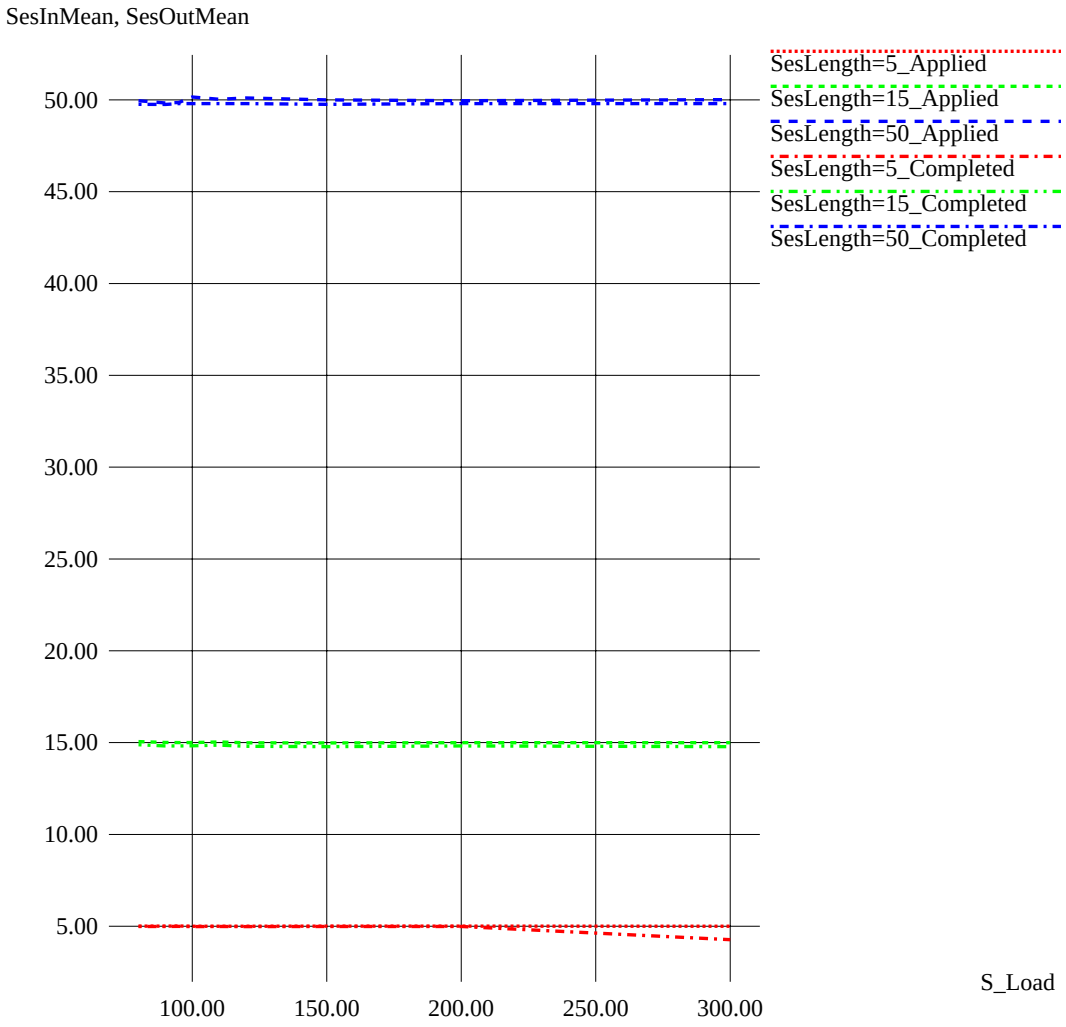

Figure 16: Average Length of Completed Sessions by Overloaded Server with Admission Control.

Finally, Figure 17 shows useful server utilization as a function of a load and a session length. Again, for sessions with a mean of 15 and 50 the results are improved almost an order of magnitude in overloaded area comparing with the similar results for a web server with no admission control shown in Figure 11. A slight decline for a curve, characterizing a server running sessions with a mean of 15, is due to occasional retries in an overloaded area (but no aborted sessions yet). Useful server utilization for a workload with mean of 5, is expectedly lower for 300% load due to a number of aborted sessions and related problems discussed above.

This concludes the analysis of an overloaded web server augmented with a session based admission control policy. It convincingly shows that an admission control mechanism is able to provide the web-quality of service guaranties discussed in Section 6 – critical for success of e-commerce retail sites.

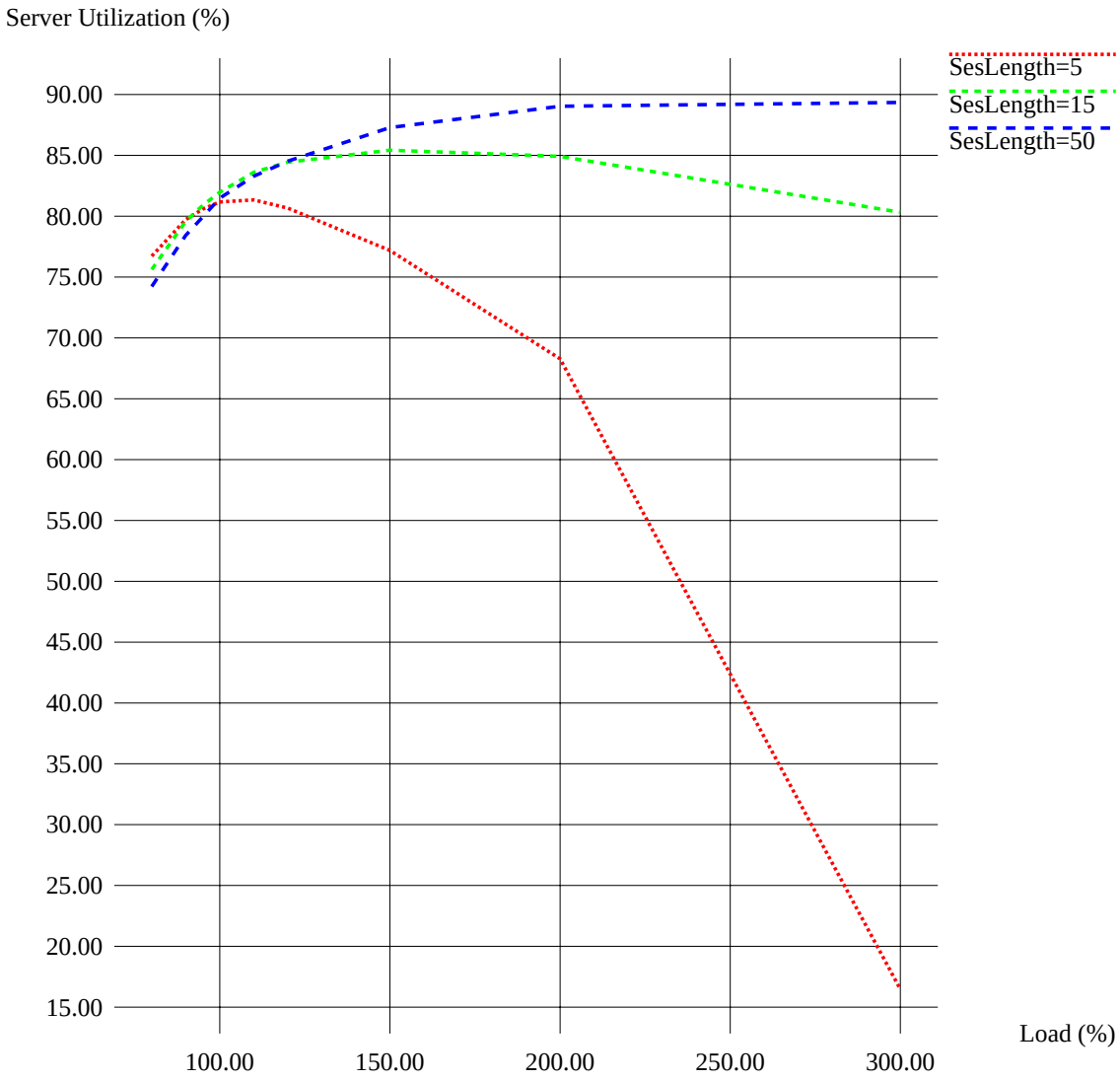

Figure 17: Useful Utilization for a Server with Admission Control.

10 Performance Metrics to Compare Different Strategies

Definition of an admission control mechanism, given in Section 7, uses the following parameters:

- ac-threshold U_{ac}
- ac-interval length;
- ac-function f_{ac}.

By assigning different values to ac-threshold and ac-interval length, as well as a varying ac-function between *ac-responsive* and *ac-stable*, a whole family of admission control policies can be introduced.

How do we compare different admission control policies? What are the metrics for their evaluation?

Let us define a function f^{ideal}, producing an "ideal throughput", in the following way:

$$f^{ideal}(Load) = \begin{cases} Load, & \text{if } Load < 100, \\ 100, & \text{otherwise.} \end{cases}$$

Let us consider a web server augmented with an admission control policy F_{ac} and processing a workload defined by a session mean *SesLength* and a load *Load*.

Let *Th(F_{ac},Load,SesLength)* denote the throughput of this web server in completed sessions, and *Ab(F_{ac},Load,SesLength)* denote a percentage of aborted sessions from a total amount of sessions admitted for processing.

Additionally, let $Load_1, Load_2, ...Load_k = MaxLoad$ be the (equally spaced) load points.

Let *Dif(F_{ac},SesLength)* denote a normalized difference between the ideal throughput f^{ideal} and throughput of the F_{ac} across the same load points:

$$Dif(F_{ac}, SesLength) = \frac{\sum_{i=1}^{k} f^{ideal}(Load_i) - Th(F_{ac}, Load_i, SesLength)}{k} \qquad (7)$$

A value of normalized difference, defined by (7), allows to approximate how far the evaluated throughput from the ideal one. For example, let *Dif(F_{ac},SesLength)=7*. It means that, in average, throughput of F_{ac} is about 7% less than the ideal one.

The "quality" of the admission control mechanism is strongly reflected by the number of aborted sessions. A new metrics reflects it.

Let F_{ac}^1 and F_{ac}^2 be two admission control policies.

F_{ac}^1 is *p-ac-better* than F_{ac}^2 ($p \geq 0$) while processing a workload defined by a session mean *SesLength* and across the same load points up to *MaxLoad* if either of the following conditions hold:

1.

$$Ab(F_{ac}^1, MaxLoad, SesLength) \leq p$$

and

$$Ab(F_{ac}^2, MaxLoad, SesLength) \leq p$$

and

$$Dif(F_{ac}^1, SesLength) \leq Dif(F_{ac}^2, SesLength),$$

or

2.

$$Ab(F_{ac}^1, MaxLoad, SesLength) \leq p < Ab(F_{ac}^2, SesLength),$$

If an application can tolerate some percentage of aborted sessions (say, up to 5%) then for the rates of aborted sessions below or equal to 5%, the two strategies are compared with respect to their throughput in completed sessions, i.e. F_{ac}^1 is *5-ac-better* than F_{ac}^2 if the percentage of aborted sessions for both strategies is less than 5%, and F_{ac}^1 has better throughput, in average, than F_{ac}^2.

If the percentage of aborted sessions for F_{ac}^2 is greater than 5% (i.e., F_{ac}^2 hass failed to satisfy the application requirement on acceptable percentage of failed sessions) while the percentage of aborted sessions for F_{ac}^1 is less or equal to 5% then F_{ac}^1 is *5%-ac-better* than F_{ac}^2 independent on the throughput.

Using this metrics, we compared the ac-responsive and ac-stable strategies (F_{ac}^{resp} and F_{ac}^{stable} correspondingly) defined by the same ac-threshold $U_{ac} = 95\%$ and the same ac-interval length of 1 second. The comparison has shown that F_{ac}^{resp} is *p-ac-better* than F_{ac}^{stable} across the different average sessions lengths and maximum load of 300%.

F_{ac}^{resp} has consistently less number of aborted sessions than F_{ac}^{stable}. This is an expected result, because by definition an ac-responsive strategy has a more restrictive admission policy.

However, if a workload of interest has an average session length of 50, i.e. *SesLength=50*, and the load of interest is limited to 200%, i.e *MaxLoad=200%*, then the comparison changes: F_{ac}^{stable} becomes *0-better* than F_{ac}^{resp}. Both strategies have no aborted sessions across the new data of interest. However, the throughput in completed sessions of F_{ac}^{stable} is higher than throughput of F_{ac}^{resp} as shown in Figure 18.

Proposed *ac-better* metrics for comparison of different admission control policies can be refined to reflect the application goals and quality of service requirements.

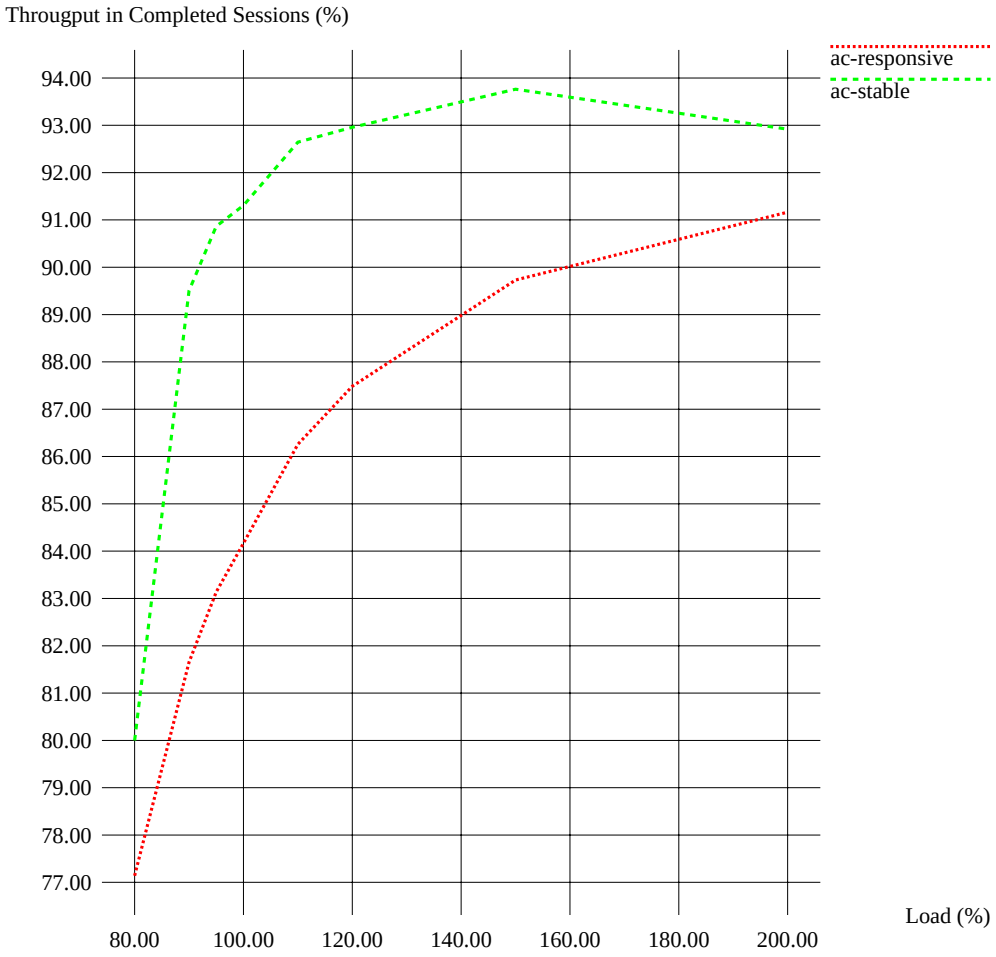

Figure 18: Throughput in Completed Sessions for AC-Responsive and AC-Stable Strategies, Workload with Average Session Length of 50.

11 Tuning the Admission Control Mechanism for Better QoS

Choosing the right parameters for the admission control mechanism is very important. In Section 9, we analyzed the results produced by the ac-responsive admission control policy with the following parameters:

- ac-threshold $U_{ac} = 95\%$
- ac-interval length of 1 second;

How do the results depend on the length of the ac-interval? What will happen if an ac-interval is set to 5 seconds?

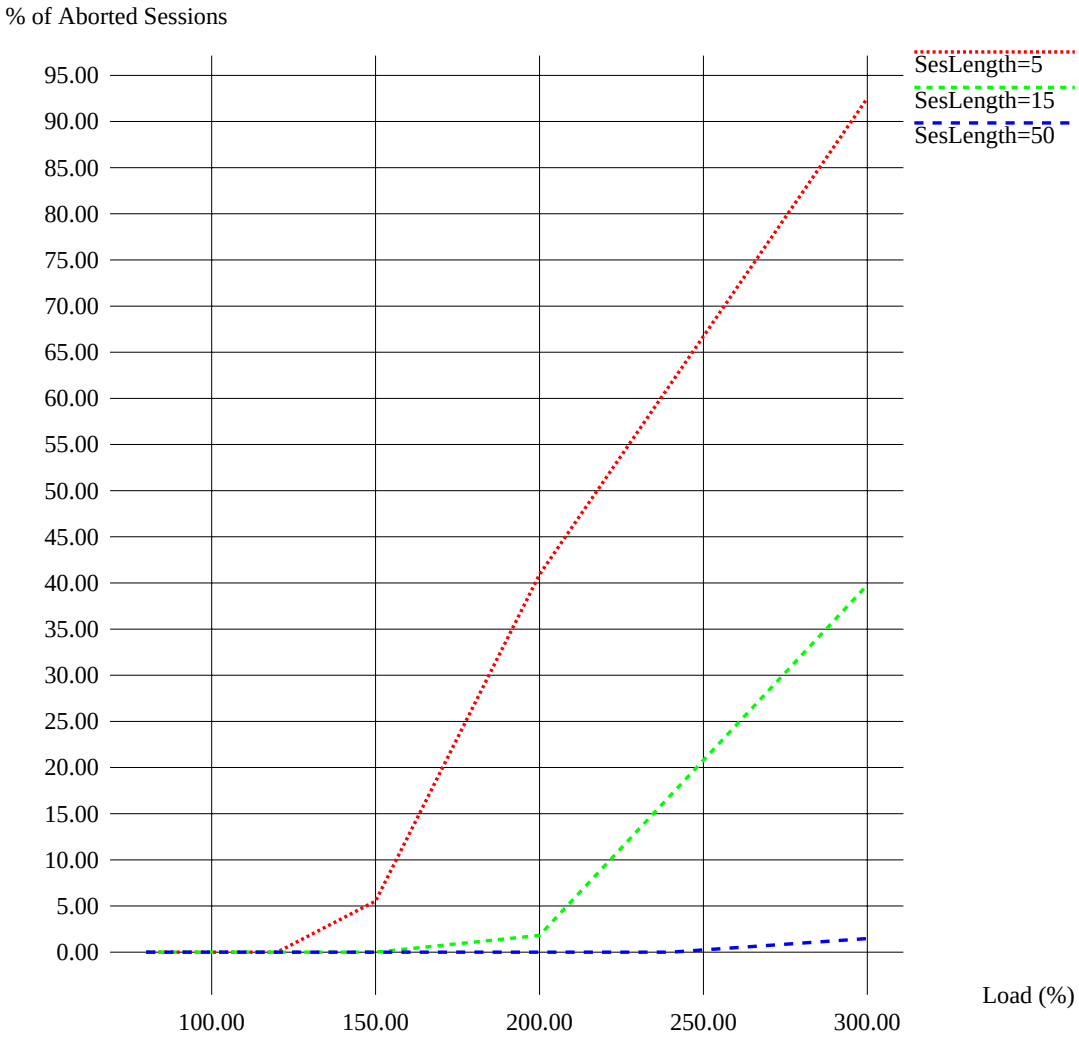

Figure 19: Percentage of Aborted Sessions from Admitted for Processing for Server with Admission Control, AC-Interval = 5sec, Workload with Average Session Length of 5.

Figure 19 shows the percentage of aborted sessions from those admitted for processing for an admission control mechanism with an ac-interval of 5 seconds. Results are significantly worse than similar ones shown in Figure 15 for an admission control mechanism with an ac-interval of 1 second. The reason is that the shorter the average session length – the higher the number of sessions generated by the clients, and accepted by the server during the ac-interval. For example, if a web server is in "accept mode" then for the load of 300% during 5 seconds it accepts around:

- 3000 new sessions ($5sec * 3load * 200sessions$) for a workload with an average session length of 5;

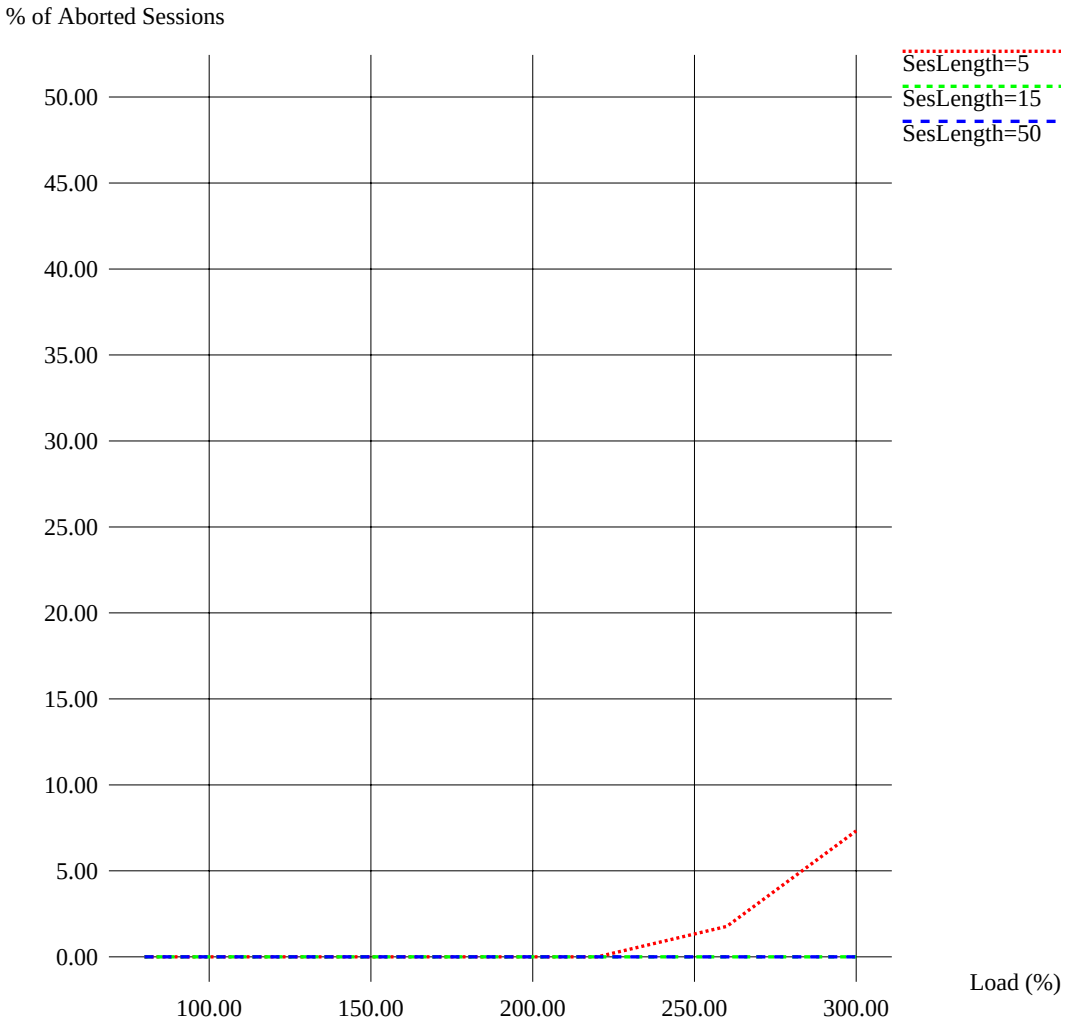

Figure 20: Percentage of Aborted Sessions from Admitted for Processing for Server with Admission Control, AC-Interval = 0.5sec, Workload with Average Session Length of 5.

- 1000 new sessions ($5sec * 3load * 66.7sessions$) for a workload with an average session length of 15;
- 300 new sessions ($5sec*3load*20sessions$) for a workload with an average session length of 50.

These new sessions are accepted in addition to the sessions which are already in progress. The main reason, for aborted sessions under this scenario, is that the listen queue overflows: it has a limited size of 1024 entries. As a snowball effect: when the listen queue gets full, it also triggers a set of retries for the requests at the end of the listen queue. The timeout value is 1 second, and the server can only process 1000 requests per second (see related discussion in Section 4.1). One way to fix the problem is to define a shorter ac-interval.

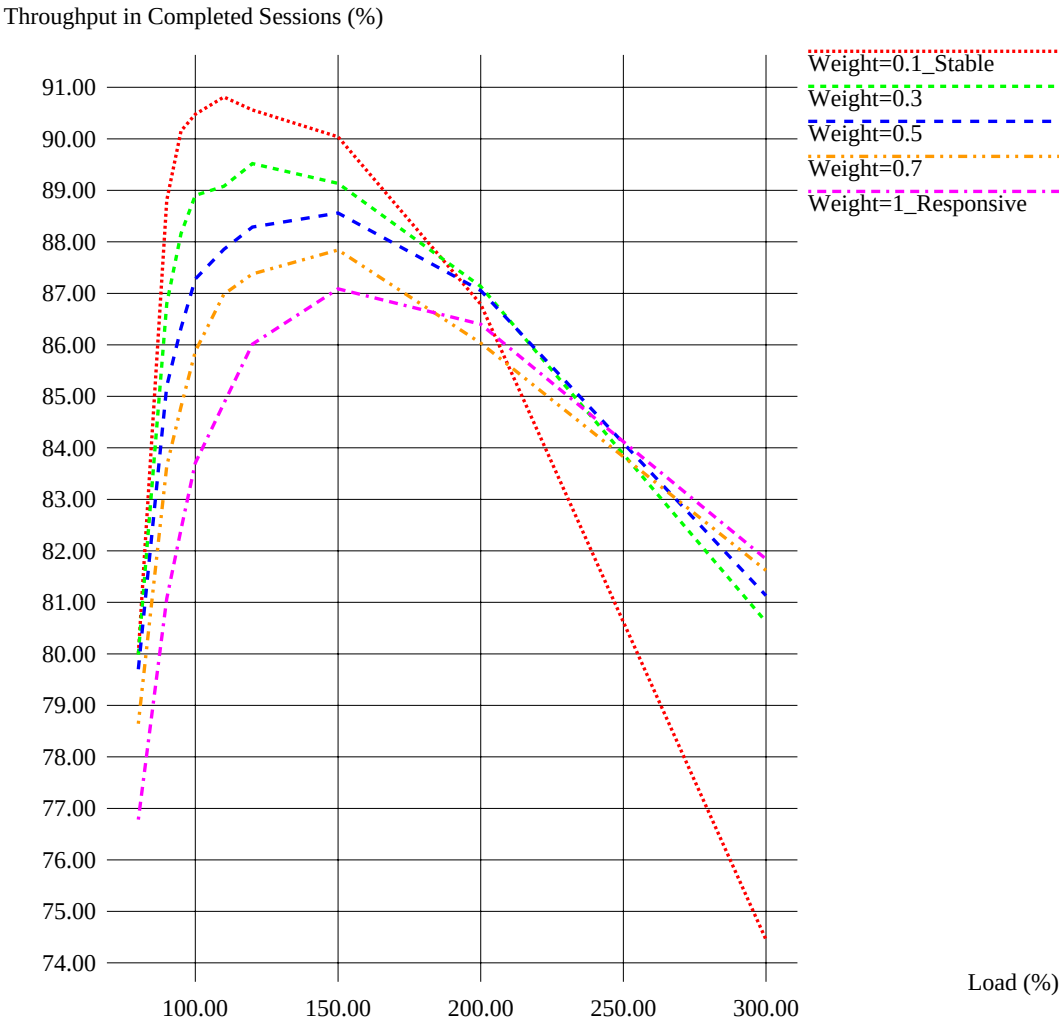

Figure 21: Throughput in Completed Sessions for Family of AC-Functions: from AC-Stable to AC-Responsive, Workload with Average Session Length of 15.

Figure 20 shows the percentage of aborted sessions for an admission control mechanism with an ac-interval of 0.5 seconds. The number of aborted sessions, for a session length of 5, is significantly less than for an admission control mechanism with an ac-interval of 1 second (see Figure 15 for comparison).

Varying an ac-threshold U_{ac} from 95% to 97% will slightly increase throughput in completed sessions at a price of greater number of aborted sessions too, especially for workloads with shorter average session length. Conversely, decreasing an ac-threshold U_{ac} from 95% to 93% will improve the quality of output, decreasing the number of aborted sessions, but at a price of slight decrease of throughput in completed sessions.

Similar impact has an ac-weight parameter in definition of ac-function allowing to define a family of ac-functions: from ac-stable one to ac-responsive one. Figure 21 shows the server

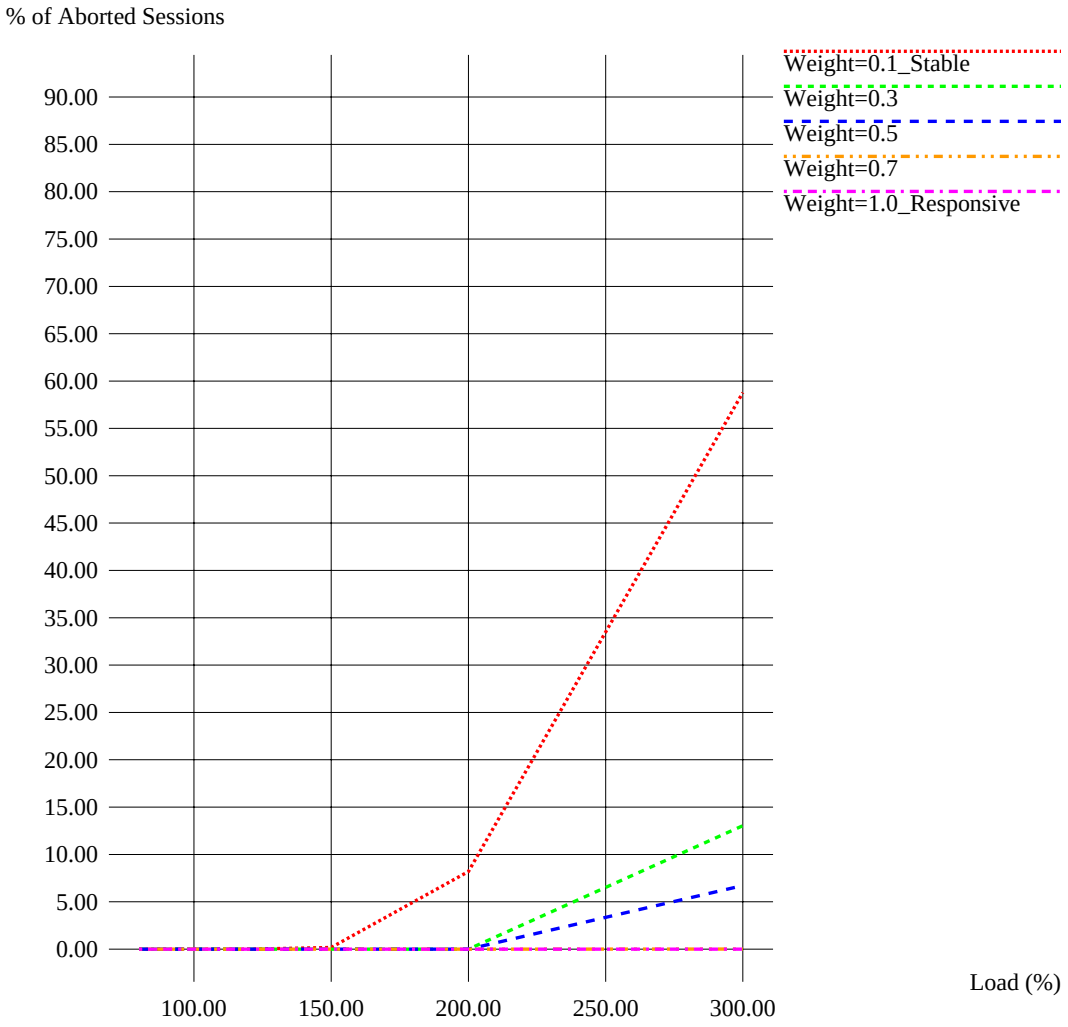

Figure 22: Number of Aborted Sessions for Family of AC-Functions: from AC-Stable to AC-Responsive, Workload with Average Session Length of 15.

throughput while running workload with average session length of 15, depending on ac-weight k used in ac-function f_{ac} definition (see Section 7).

As expected, the server throughput is higher under "more stable" ac-functions for a load below 170%. The situation changes for higher load in favor of "more responsive functions". The rates of aborted sessions are worse for "more stable functions" in higher load area as shown in Figure 22.

This shows again that ac-stable admission control functions achieve better throughput in the load range of 85%-120% at a price of higher number of aborted sessions under higher loads. While ac-responsive admission control functions lead to more restrictive admission policies and achieve higher quality of service guaranties, especially at high loads but at the price of slightly lower server session throughput (in particular, when a server operates at loads in the

range 85%-120%).

Obviously, a hybrid admission control strategy is a desirable goal. Developing a hybrid strategy is one of the goals of the future work.

Another interesting question for future research is the following. For a given web server and workload characteristics, define an optimal or nearly optimal admission control mechanism.

12 Conclusion

In this paper, we introduce a new, session-based workload for measuring a web server performance. We show that an overloaded web server can experience a significant loss of throughput as a number of completed sessions comparing against the server throughput measured in requests per second.

However, this loss is not always easy to recognize. When the session lengths are exponentially distributed (in other words, there is enough variability in session lengths) the throughput in sessions for overloaded server decreases slightly, but not dramatically.

Analysis of the completed sessions reveals, however, that the majority (up to 98%) of completed sessions are short: the overloaded web server discriminates against the long sessions. This could significantly impact sales and profitability of commercial web sites because the sale-sessions are typically 2-3 times longer than non-sale ones. Based on this analysis, we formulate the web-quality of service requirements a web server has to support. In particular, a fair guarantee of completion, for any accepted session, independent of a session length - is a crucial requirement for commercial web site to be successful.

We show that a web server augmented with admission control mechanism is able to provide required web-quality of service guaranties. Incorporating this technique into product allows HP to offer solutions to customers that enables them to migrate core business functions onto web based technologies, and to use web applications for strategic advantage.

13 Acknowledgements

The authors would like to thank Sky Golightly for his help in obtaining web server log files from e-commerce web sites. We would also like to thank Scott Jorgenson for his work in developing a production version of session based admission control and for providing data that helped validate the technology.

Josep Ferrandiz deserves a special acknowledgement for active promoting session based admission control ideas and technology during all the stages.

14 References

[HP-98] HP Enterprise Computing. URL http://www.hp.com/go/domain/

[HPSD-98] HP Software Depot. URL http://www.software.hp.com/

[Schwetman95] Schwetman, H. Object-oriented simulation modeling with C++/CSIM. In Proceedings of 1995 Winter Simulation Conference, Washington, D.C., pp.529-533, 1995.

[SpecWeb96] The Workload for the SPECweb96 Benchmark. URL http://www.specbench.org/osg/web96/workload.html

[WebStone] WebStone: The Standard Web Server Benchmark. URL http://www.mindcraft.com/benchmarks/webstone/