WEB ADMISSION CONTROL: IMPROVING PERFORMANCE OF WEB-BASED SERVICES

B.M.M. Gijsen [a], P.J. Meulenhoff [a] , M.A. Blom [a], R.D. van der Mei [b,c] and B.D. van der Waaij [a]
[a] TNO Telecom, Delft, The Netherlands
[b] Free University, Amsterdam, The Netherlands
[c] CWI, Amsterdam, The Netherlands

Nowadays, services based on web technology are "hot" and e-commerce revenue is growing fast. In this context it is striking that popular web servers often suffer from severe temporary overload, causing web browsing performance to degrade dramatically. Therefore controlling browsing performance is essential in the e-commerce market. Server capacity planning is important for controlling browsing performance, but more is required to resolve the problem of a total performance collapse during temporary server overload. In this paper we present a novel intelligent request discard mechanism that resolves this problem.

1 Background and Motivation

Today, services based on web technology are a hot topic and popularity of e-commerce is growing. The increasing role of web technology is particularly strong in business areas such as travel and ticket booking and online banking. In this context it is striking that the maturity level of web browsing performance is still low, or at least unstable. In current practise, popular web servers can suffer from severe, and often only temporarily, server overload. Server overload causes web browsing performance to degrade, in terms of increasing response times or even server unavailability. In the e-commerce context, customer dissatisfaction about overly long response times or unavailable servers will directly cause a decrease in revenues for e-commerce applications. It was estimated that in 1998 about 10% to 25% of e-commerce transactions were aborted owing to long response delay, which translated to about 1.9 billion dollars loss of revenue [1]. Therefore controlling web server performance (even under temporarily overload situations) is essential for companies whose business relies on web-based services. This problem is acknowledged by for instance Intel Corporation [2], HP Labs [6] and illustrated by numerous web server incidents all over the world.

Web server capacity planning is one important 'means' to control web browsing performance. But even with good capacity planning one will not be able to avoid overload circumstances every now and then, due to peaks of request arrivals. Especially, in case of growing server utilisation, temporary overloads will occur more frequently just before the server capacity is expanded. Thus, effective control over web browsing performance requires more than server capacity planning. Most contemporary web servers do not feature intelligent request discard mechanisms to handle temporary overload. As a result requests are randomly discarded during temporary overload. In [6] it is experimentally shown that the impact of random discarding on web-browsing performance is dramatic.

An effective technique for resolving this problem of a total performance collapse is admission control. In essence, admission control is an 'intelligent discard' mechanism. For example, the impact of server overload on user perception could be reduced by deploying session based, or user selective, admission control. User selective admission control means that server-access is denied to requests from some users in order to let the other users browse at a satisfactory performance level. With such a mechanism one will improve the average user perceived performance level and one would be able to postpone web server investments, leading directly to lower server cost.

Admission control is well known in the field of telecommunication, but it is hardly used in the field of web servers (see also [2]). In the field of web servers alternative mechanisms such as load balancers and (extended) firewalls are much more popular, see for example [7-10]. However, load balancing and firewall functionality are much less effective for protection against server overload than admission control.

Although less popular than these load balancing and firewall products, there do exist web server admission control products. In the literature a number of admission control concepts can be found, some of which are correlated to user selective admission control, see [2-6] (a survey of these references is given in the next section).

In this paper we present the state of the art regarding overload control for web-based services by means of admission control schemes and we highlight several opportunities for improvement. Based on these improvements we present a novel web admission control scheme and demonstrate its effectiveness.

The remainder of this paper is organized as follows. In Section 2 we highlight the key issues of web admission control and we present the state of the art. In Section 3 we present our web admission control scheme, for which we demonstrate its effectiveness in Section 4. In the final section we present our conclusions and topics for further research.

2 Web admission control: state of the art

2.1 The web admission control challenge

By default, contemporary web servers implement simple tail-dropping mechanisms to handle temporary overload. With such mechanisms requests are randomly discarded during temporary overload. The impact of random discarding on web-browsing performance is dramatic. In general a web-page (URL) download requires more than one browser request towards the server. If the server does not respond to one of the requests, for example due to random discarding of a request, then the browser will show a failure message instead of the requested web page (see Figure 1) and the requests that were responded correctly have become useless. This random request discarding (during temporary server overload) is a major drawback of current web server technology. Figure 1 illustrates the impact of random request discard on a page download for an individual user. Typically, a large number of users will be downloading pages from a web server simultaneously, during temporary server overload. In that case, random request discard will typically lead to degradation of browsing performance for all active users during overload

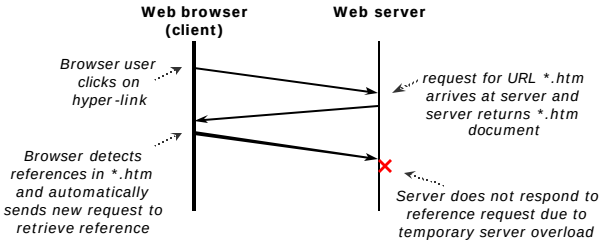

Figure 1: impact illustration of random request discard on page download

An effective technique for resolving this problem of a total performance collapse is admission control. In essence, admission control is an 'intelligent discard' mechanism. For example, the impact of server overload on user perception could be reduced by deploying session based, or user selective admission control. User selective admission control means that server-access is denied to requests from some users in order to let the other users browse at a satisfactory performance level. With such a mechanism one will improve the average user perceived performance level and one would be able to postpone web server investments, leading directly to lower server investments.

Although the general idea of web admission control is very attractive, the details of how to apply admission control are far from trivial. In particular, admission control is potentially dangerous: one does not want an admission control mechanism that unnecessarily discards requests. In this respect a good choice of congestion detection criteria is very important. A complicating factor is the fact that 'server overload' may be the result of one or more causes, such as CPU overload, database contention or transmission overload. Overload on the server's CPU will typically be caused by a large number of requests for (CPU demanding) server scripts, e.g. cgi or asp scripts. Transmission overload will typically be caused by downloads of large files. It will be intuitively clear that it is more effective to discard cgi and asp requests, during CPU overload, instead of discarding requests that are less CPU demanding. This observation raises a number of challenges:

a) at which load (or congestion) level should requests be discarded to reduce server congestion level?
b) how can specific types of congestion, based on the cause of congestion, be detected?
c) which requests should be discarded during congestion, based on the congestion cause (i.e. effective enforcement)?

Further, it should be noted that the request discard policy, also called enforcement, might be based on more information than the cause of congestion only. For example, if a web master indicates that a specific group of URLs that support business critical transactions should only be discarded under extreme load conditions, then one may include this in the enforcement policy. Similarly, one may distinguish request discarding priorities based on the user that sent the request. For example, one may prioritise access to web-based services for subscribed customers over access for non-subscribed (best-effort) customers. In this case information regarding service level agreements (SLA) with customers is used in the

enforcement policy. We will refer to this as SLA checker functionality. We emphasise that flexibility to incorporate this kind of information in the enforcement policy is important.

Admission control is not entirely new in the field of web-based services. Below we present a survey of the state of the art in web admission control.

2.2 Literature

In [3] Chen et al. propose a "Periodical Admission Control based on Estimation of Request rate and Service time" (PACERS). The PACERS mechanism discriminates between high and low priority requests and 'guarantees' performance for high priority requests by rejecting low priority requests during overload. The service time estimate is based on a priori configured "computation quantums" for each of the "object types". For example, Chen et al. state that cgi scripts require one order of magnitude higher CPU service times than processing of plain html requests. The request rate estimation (per object type) is based on measured values. The PACERS mechanism is not a user selective admission control mechanism, nor does it take the cause of contention into account explicitly.

Rumsewicz et al. [4] developed the Eddie admission control scheme, which is available as an open source prototype. The Eddie admission control scheme is aimed at improving web performance during overload, by:

- monitoring critical resource usage on the web server,
- determining the rate at which new requests can be accepted while still providing acceptable performance for already accepted requests,
- maintaining a table of currently active 'sessions',
- accepts an appropriate rate of new requests while rejecting other new requests.

'Sessions' may be determined by means of cookies, or based on the IP source address. The state of a 'session' can be active or inactive. Based on a time-out an active 'session' will change its state to an inactive 'session'. The accept/reject decision is based on resource utilisation, such as CPU utilisation, memory used, number of disk input/output operations and average disk delay. For each resource a congestion (high and low) threshold is defined. Once the resource utilisation exceeds the (high) threshold, the resource is marked as congested. The more resources are congested the higher the web server congestion level. The admitted rate of new 'sessions' is inversely proportional to the web server's congestion level. In this sense the Eddie algorithm does not take into account the cause of the congestion.

Eggert and Heidemann [5] propose a server side, application-level mechanism to provide two different levels of web-based service: regular and low priority. The idea is that not all requests are equally important. As examples Eggert and Heidemann mention that prefetching web page requests by proxies are less important than user-initiated requests. Therefore the terms foreground (e.g. user-initiated) requests and background (e.g. proxy prefetching) requests are introduced. The mechanisms to enforce the differentiated web-based service levels are: limiting process pool sizes, lowering process priority and/or limiting transmission rate via operating system mechanisms.

In [6] Bhatti and Friedrich propose the Web QoS architecture for supporting server QoS and in particular tiered (i.e. differentiated) QoS levels. The Web QoS architecture contains several modules. The request classification module sets QoS attributes based on the filters and policies defined by the system administrator. After classification the admission control module determines whether the request is rejected or executed according to the scheduling policy of the class. If executed, then the classification attributes are used for request scheduling to determine how to enqueue and dispatch the request. Session management is used to provide session semantics and maintain session state for the stateless HTTP protocol. Session management is both based on a combination of cookies and IP source addresses. Resource scheduling ensures that high priority tasks are allocated and executed as high priority processes by the host operating system. The Web QoS architecture also supports integration with network QoS mechanisms such as reading and marking of IP Type of Service (ToS) header bytes, or Differentiated Services (DiffServ) fields.

From the literature we conclude that the idea of *session based* admission control is not available in all implementations. For example, even in [3] and [5] the admission control schemes do not feature session based admission control. In the literature we did not find any reference to a web admission control mechanism that discards only 'appropriate' requests, based on the cause of congestion. Nor did we find references stating that user subscription information can be taken into account regarding decisions about which sessions should be admitted or restricted. In the next section we present our admission control scheme that does take these aspects into account.

3 New Web Admission Control Scheme

In this section we present our new web admission control scheme (WAC) and its goal. The best one can do under overload circumstances is to minimise the number of users that will suffer from the overload. Therefore, the goal of our WAC scheme is to maximise

the number of users that can be handled with satisfactory performance, during server overload periods.

Web admission control can be implemented as software on a web, application or proxy-server or as hardware that is placed directly in front of a web server, similar to the implementation of a firewall. In any case the WAC functionality should be executed before the 'regular' web server functionality, as indicated in Figure 2. Figure 2 also indicates that SLA checker functionality can be added to the WAC functionality, in order to include customer subscription information in the enforcement policy.

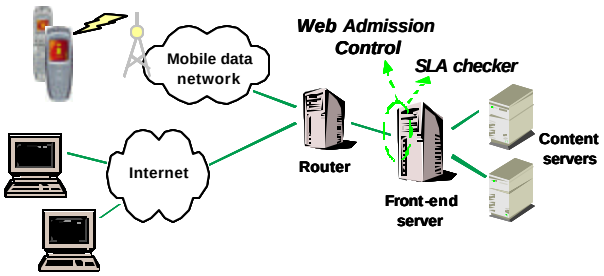

Figure 2: architectural location of WAC and SLA checker functionality

The WAC module consists of congestion detection and admission enforcement functionality. In the following subsections several implementation guidelines for WAC are specified.

3.1 Congestion detection module

The congestion detection sub-module detects either one or a combination of the following criteria:

a) the overall frequency of requests exceeds a threshold value;
b) the frequency of a specific type of requests exceeds a threshold value;
c) the overall response time of requests exceeds a threshold value;
d) the response time of a specific type of requests exceeds a threshold value;
e) the number of simultaneously active TCP/IP sessions exceeds a threshold value;
f) the number of concurrent (web)sessions exceeds a threshold value;
g) the server's CPU utilisation exceeds a threshold value;
h) another 'application' triggers activation of admission control.

In these criteria the term 'frequency' denotes a number of events per fixed time period. For example, the frequency of URL requests means the number of arriving URL requests per time period. The time period should be a configurable time parameter and the frequency can be implemented as the (exponentially weighted) ratio between the time period and the event-interval. The term 'request' can denote arriving IP packets, TCP/IP session set-up requests, or URL requests. By 'frequency of a specific type of requests' we mean the frequency of IP packets, or TCP/IP session set-ups with a particular IP address, or the frequency of a particular URL. The threshold mentioned above can be either pre-specified thresholds, or adaptive thresholds. An adaptive threshold should adapt itself to a value based on measured values. An example of an adaptive threshold is the well-known Retransmission Time-Out (RTO) parameter from the TCP protocol. Based on measurements of round trip times (RTT) of TCP/IP packets, the TCP algorithm calculates the exponentially weighted mean and standard deviation of the RTT. From that mean and standard deviation the TCP algorithm calculates the adaptive parameter value for the RTO. Further, the term 'response time' can denote the average response time, or variation in response time, etc. Finally, the server's CPU utilisation can typically be obtained from the server's operating system.

Each of the criteria presented above can be implemented in the form of 'resource' objects. Each 'resource' object represents the resource which is protected by WAC, e.g. CPU utilisation, number of concurrent sessions, request rate for a (group of) web-page(s), etc. and must implement two interfaces:

1. An `update()` procedure is used to update the internal data in the resource object. This can either be done at the arrival of a new HTTP request (web-resources) or initiated by another process operating at a different timescale.
2. A `congestion()` function that returns a parameter indicating the level of congestion of the resource. The type of the congestion-parameter is boolean (true, false), enumerated (green, orange, red) or fraction α ($0 \leq \alpha \leq 1$).

The result of the congestion function of each resource is grouped in a list, and forwarded to the decision/rules module which implements the logic in order to accept/reject a new request.

3.2 Decision module

Once the congestion detection module has detected either one or a combination of the congestion criteria, the admission enforcements module will either accept or reject the arriving request. The decision to accept or reject is based on decision rules, implemented in the decision module. The decision rules specify what enforcement policy is executed for specific combinations of congestion criteria. Possible enforcement policies are (this list is not exhaustive; more policies can be implemented):

a) reject newly arriving 'web-sessions';
b) reject newly arriving 'web-sessions' with exception of several URLs;
c) reject all arriving requests;
d) reject arriving requests of a specific type;
e) for any URL forward only one URL request, per configurable time-period, towards the web server and broadcast the server's response to all requests (i.e. 'URL multicast');
f) poll other 'applications' for additional enforcement information.

In the above policies the notion of 'web-session' can be directly correlated to the establishment of a TCP/IP session on the web-server, or one can implement a list of 'active IP addresses' based on the time-stamp of latest packet arrival per IP address. Alternatively, the notion of 'web-session' can be implemented by using application level 'cookies'. The last method is preferable, especially for clients that are browsing with multiple established TCP/IP sessions and for requests that arrive through a proxy server (in that case all IP addresses are the same). Further, the notion of a session should be configurable with a parameter specifying after how much time a 'session' is removed form the list of active sessions.

Accepting and rejecting at 'web-session' level, instead of at URL request level has the advantage that it will enable to control the performance level at user level, instead of at request level. In fact, with web-session level admission control some of the users will experience normal web-browsing performance, while others will not be allowed to browse on the server at all. In this sense web-session level admission control maximises the number of users that can be handled with satisfactory performance, during overload.

To enable appropriate tuning of WAC to any specific web-based service, that is independent of the used internet access medium, the decision module enables non-real-time configuration of any (combination of) condition detection criteria to initiate any (combination) of the described admission enforcement policies. This enables that if, for example, the CPU utilisation exceeds its threshold, then newly arriving requests of URL-types *.cgi and *.asp can be rejected.

3.3 Admission enforcement module

Once the decision module has determined which enforcement policy should be executed, based on the input from the congestion detection module, the enforcement module executes the policy by performing one of the following actions:

a) Reject the request at application level;
b) Reroute to a less resource intensive response;
c) Send an 'overloaded' response;
d) Reject the request at connection level;
e) Terminate the active connection;
f) Immediately stop sending any data.

3.4 WAC guidelines

As illustrated in Figure 2 the WAC functionality should be implemented directly in front of the 'regular' web server functionality. Figure 3 illustrates the sequence of accept / reject actions, depending on the congestion status, by the WAC module.

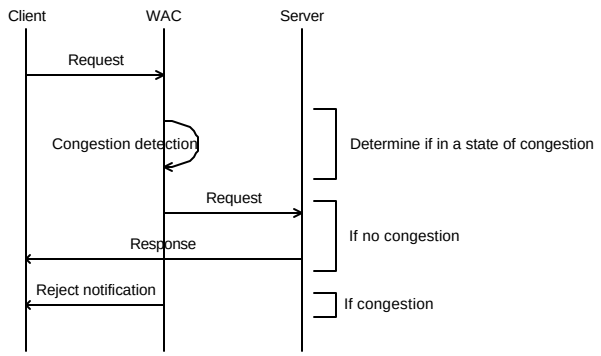

Figure 3: Sequence diagram of accept / reject actions by WAC

A more detailed view on the sub-modules in WAC is presented in Figure 4. This figure illustrates how requests that arrive at the WAC module flow through the WAC module.

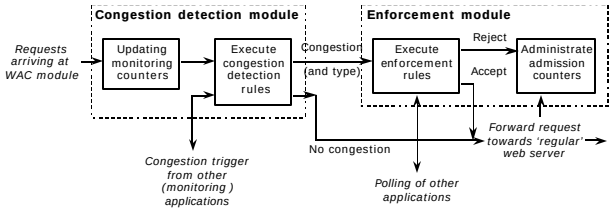

Figure 4: flow chart of requests through WAC module

Besides executing the actual congestion detection rules, the congestion detection module will constantly update monitoring counters (such as time since last request arrival, for calculation of request arrival frequency) that are input for the congestion detection rules. Similarly, the enforcement module does not only execute the enforcement rules, but it will also keep track of admission statistics, such as number of accepted and rejected requests. Note, that if none of the congestion criteria indicate congestion (denoted by "No congestion" in Figure 4), then the enforcement rules need not be executed.

Since the detection, decision and enforcement actions are executed in real-time, all steps are implemented such that it can be done very fast (say within 0.01 seconds). Further, the WAC module should be scalable in order to prevent overload on the admission control module. To this end, the congestion detection

sub-module features a configurable sampling routine. For example, by letting the WAC module verify a criterion for only one in every two (or more) requests instead of verifying a criterion for each arriving request. Typically, the sampling routine depends on the detection criterion: for a URL request arrival criterion the sampling frequency shall be based on the number of arriving URL requests, while for a TCP/IP session based criterion the sampling frequency shall be based on the number of initiated TCP/IP sessions. Possibly the sampling routine could be configured dynamically. This can for example be achieved by letting the sampling frequency depend on the measured number of IP packets per second, or on the number of requested URL, etc.

4 WAC improvement results

In order to validate the effectiveness of the proposed web admission control scheme we conducted numerous experiments. Below we present some of te results.

4.1 Experiment set-up

For the experiments we used the configuration depicted in Figure 5. We used a number of Pentium III (400 Mhz) PCs to execute client browsing scripts, an HTTP proxy and a web server. All requests generated by the clients are directed towards the proxy server which is equiped with WAC functionality.

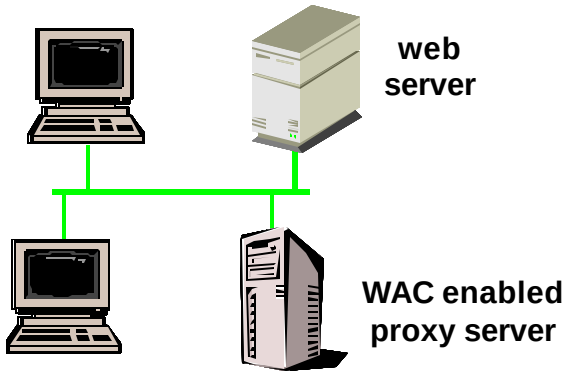

Figure 5: configuration of experiment set-up

In case WAC functionality is switched off, then the proxy server will forward all requests towards the web server. If the WAC functionality is switched on, then the WAC module decides whether the request is forwarded to the web server or not.

The client browsing scripts that are executed by the PCs generate sessions of requests. In each session seven subsequent requests are sent to the proxy server, with a sleep time between the response of request *i* and the sending of request *i*+1. This sleep time emulates a user reading time. The sleep time is randomly drawn from a uniform distribution between *2* and *4* seconds. All the requests in a session are targeted at the same URL, but with different parameters. After the client receives the response to the last request the client sleeps for a time between *16* and *40* seconds before repeating itself and sending a new session of requests.

The server executes a parameterized script, where the parameter is read from the URL parameter in the request. The processing time of the script is a function of this parameter. The average processing times for each of the seven consecutive requests in a session are: *0.25*, *0.39*, *0.61*, *0.05*, *0.03*, *0.39*, respectively *0.25* seconds. Of course, the request completion times will become larger than the processing times in case the web server is handling requests simultaneously.

The session time-out parameter in the WAC module is configured at *15* seconds. Recall that a client sleeps at least *16* seconds between two sessions, so the WAC module will time-out between two sessions of a client. Due to this time-out the WAC module 'recognizes' the client sessions in this experiment set-up. Further, in these experiments the admission decision is solely based on the number of simultaneous sessions. The number of simultaneously admitted sessions is configured at 10.

The duration of the experiments was set at *20* minutes for each set of parameters. For scenarios with six clients or more this results in experiments consisting of the generation of several thousands of requests.

4.2 Numerical results

For validating the effectiveness of the WAC module we compare the average response time and the session goodput. We define the goodput as the average number of good sessions per minute, where a good session is a session for which the response times of all seven requests were smaller than 5 seconds.

Figure 6 shows the comparison of average response times between the scenarios with and without the WAC module. In this graph the response time in seconds (y-axis) is plotted against the number of configured clients (x-axis). Figure 7 shows the improvement of the WAC module regarding the session goodput (in sessions / minute).

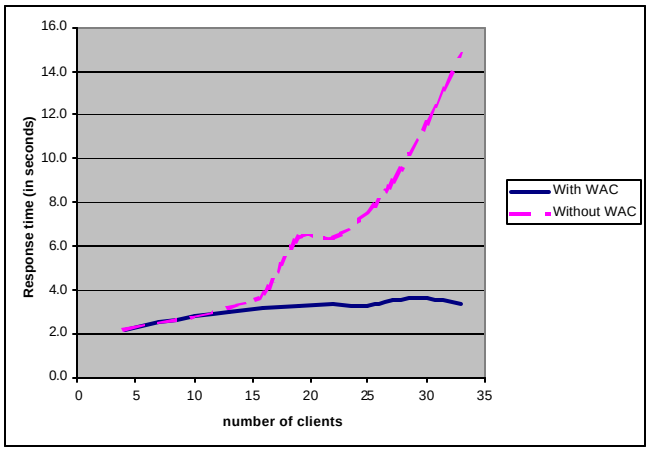

Figure 6: comparison of response times

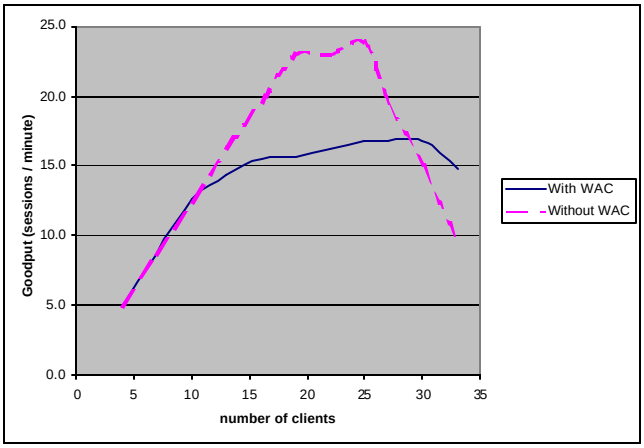

Figure 7: comparison of session goodput

4.3 Discussion

First of all, note that in an uncongested system the average session duration equals the sum of the processing and sleep times. The average sleep time equals *6* x *3.0* + *1* x *28.0* = *46.0* seconds. The sum of the processing times of all seven requests per session, averaged over all sessions, approximately equals *2.0* seconds. So, the total session duration in a light loaded scenario equals approximately *48.0* seconds. Further, each session roughly induces a processing load on the web server of: *2.0* / *48.0* = *4.2*%. Simular, the theoretical maximum number of sessions per minute that the web server can handle equals approximately *60.0* / *2.0* = *30* sessions per minute. Observe that this calculation is confirmed by the experimental results, where the experimental peak goodput lies at 24 sessions per minute.

Note that the calculation above provides a good starting point for configuration of the WAC parameter regarding the maximum simultaneously admissible sessions. In particular, the product of the maximum number of sessions per minute and the average session duration (in minutes) represents the maximum number of sessions that the web server can handle simultaneously. Given this value (for example, based on measurements) one may choose to configure the maximum number of sessions WAC parameter at, for example, 90% of this value.

Figure 6 clearly illustrates that the WAC module succeeds in maintaining the response times at a relatively low level. Without WAC the response times grow unbounded.

Figure 7 illustrates the impact of randomly discarding requests on session goodput. Once the number of clients exceeds 25 clients the goodput totally collapses. The results with the WAC module show a lower peak goodput, relative to the scenario without WAC. This is due to the choice of the WAC parameter. Limiting the maximum number of simultaneous sessions at 10 appears to be too restrictive. Fine-tuning this parameter remains a topic for further research, yet. In any case the WAC module appears to prevent the total goodput collapse, as seen for the results without WAC.

5 Concluding remarks & further research

In this paper we presented a novel web admission control scheme, called WAC. Three distinguishing features of WAC are:

1. Its session based admission control feature to maximise user perceived browsing performance;
2. The ability to detect several types of congestion and the freedom to configure any specific enforcement policy for each congestion type to ensure an effective admission control mechanism;
3. A separate implementation of congestion detection and enforcement enable flexibility with respect to the integration of other monitoring tools (for congestion detection) and/or user profile applications (as additional input for enforcement).

We demonstrated the performance improvements that can be obtained with WAC, both in terms of reduced response times and in terms of controlled session goodput under heavy overload.

As mentioned in the discussion of the numerical results investigation of the WAC configuration parameters and thresholds remains a subject for further research. Besides that we implemented the WAC module as an HTTP proxy. By implementing the software as add-on on a web server one can strongly improve the speed of the WAC module, which would make it better scalable. Finally, in the scenarios that we investigated so far we only demonstrated a small part of the WAC functionality. We emphasize that WAC can be applied in a much broader setting. Nevertheless we have shown that WAC has much

potential for improving performance of web-base services.

6 References

[1] E-biz bucks lost under ssl strain, T. Wilson, Internet Week Online, May 20 1999, http://www.internetwk.com/lead/lead052099.htm

[2] Overload Control Mechanisms for Web Servers, R. Iyer, V. Tewari, K. Kant (Server Architecture Lab, Intel Corporation)

[3] An Admission Control Scheme for Predictable Server Response Time for Web Accesses, X. Chen, P. Mohapatra, H. Chen, http://www10.org/cdrom/papers/249

[4] Admission Control Scheme: Algorithm description, prototype design details and capacity benchmarking, M. Rumsewicz, M. Castro, M. Tai Le, http://eddie.sourceforge.net/faq.html.Eddie

[5] Application-level differentiated service from an internet server, L. Eggert and J. Heidemann, World Wide Web Journal, vol. 3, no. 2, 133-142, 1999

[6] Web server support for Tiered Services, N. Bhatti, R. Friedrich (Internet Systems and Applications Laboratory, HP Laboratories), IEEE Network, 64-71, September/October 1999

[6] http://modules.apache.org provides a list of add-on modules for Apache web servers

[8] http://www.coyotepoint.com contains information about Coyote Point System's load balancer solutions

[9] http://www.availinux.com/modules.php?name=News&file=article&sid=36 contains information about a clustering and load balancing tool for Apache

[10] http://www.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml provides information about load balancing functionality in Cisco routers