A Session Aware Admission Control Scheme for Next Generation IP Services

Narjess Ayari and Denis Barbaron
Orange Labs - France Telecom R&D
2, Avenue Pierre Marzin, 22307 Lannion, France
{narjess.ayari,denis.barbaron}@orange-ftgroup.com

Laurent Lefèvre and Pascale Primet
INRIA RESO - LIP (UMR 5668 CNRS, ENS, INRIA, UCB)
Université de Lyon, 46, allée d'Italie - 69364 LYON, France
{laurent.lefevre,pascale.primet}@ens-lyon.fr

***Abstract*-Under overload condition, a processing server contributes to the poor QoS through sustaining heavy request queuing delays and prolonged processing time. Admission control is a well known mean to prevent a server overload. Most state of the art research advocate session oblivious mechanisms where the dropping of the requests pertaining to an accepted session can occur at any time during the session lifespan. From an operator perspective, this means that the server, which seems to sustain a high throughput, is in reality wasting its resources on failed or on reduced QoS sessions. In this work, we advocate an innovative architecture for session aware admission control of an offered IP traffic to a cluster-based server. The proposed system is enhanced with means for maximizing the useful throughput in terms of completed sessions per unit of time. It particularly achieves an improved responsiveness and a better stability. Finally, it is open to adapt to any multiple-flow based NGN service such as voice over IP or streaming video.**

I. INTRODUCTION

The over provisioning of the core network has contributed in reducing the network Quality of Service failure rate even for large rich media and time constrained flows. However, the measure of the end-to-end QoS involves the edge processing server as well. Indeed, under overload condition, a processing server has no sufficient resources to provide the service to all the clients. Hence, it contributes to the poor QoS through sustaining heavy request queuing delays and prolonged processing time.

Since a poor perceived performance is a foremost impediment for the success of any service, an operator needs to provide an acceptable QoS for the admitted network traffic. Meanwhile, the processing server does not only require having enough processing resources, it requires also to prevent its resources from overloading.

The concept of admission control is a well known mean to prevent a server or a network route from overloading. It consists of regulating the acceptance of the offered network traffic according to the usage of the controlled resources.

In previous works [1], we questioned the appropriateness of the flow-aware processing when dealing with multiple-flow based Internet services. We showed that some services are built upon a session model which involves multiple and heterogeneous flows required for the signalling and for the data exchange all along the session lifespan. Typical examples include some of the current regular services such as file transfer using FTP as well as most of the next generation Internet services such as video streaming using RTSP/RTP/RTCP and voice over IP using SIP.

When a server is admission control oblivious or when it uses a session unaware admission control policy, the dropping of the requests pertaining to an already accepted session is likely to occur at any time during the session lifespan. This leads the concerned sessions to experience either QoS degradation or at an extreme, the interruption of the service [1].

From an operator's perspective, this means that a server, which seems to be fully satisfying the client demands at a high throughput, is in reality wasting its resources on failed or on reduced QoS sessions. Indeed, the server throughput has been usually defined as the number of connections processed per unit of time. Hence, session aware admission control is required in order to reduce the number of aborted sessions while preventing the overload of the server resources.

On the other hand, Internet server clustering is widely used by operators to improve the scalability of the rendered services under heavy load condition. Indeed, a cluster consists of a set of networked servers which transparently offer to clients a single system image while providing additional processing capabilities. The network traffic is offered to the entry point to the cluster where a load balancer diverts each incoming request to the appropriate processing server. Server clustering adds a further dimension to the QoS-aware resource management. Indeed, the session aware admission control must take into account the usage of the available resources both of the cluster entry point and the cluster internal nodes.

In this work[1], we advocate an innovative architecture for session aware admission control of an offered network traffic to a cluster of servers. By session awareness, we raise the challenge of considering both the session integrity constraint of the offered network traffic as well as its characteristics in terms of volume, rate or duration while admitting or while rejecting the offered network traffic to the cluster.

The remainder of this paper is organized as the following. In section II, we describe the general architecture of the proposed system. A detailed overview of its operations is provided in section III. In section IV, we outline the limitations of the related work on Internet server admission control. Finally, we conclude by describing the perspectives of this work.

[1] Parts of this work are protected by the Intellectual Property National Institute (INPI) patent disclosure N°FR0756191.

II. General Architecture

The system we advocate achieves session awareness by means of the explicit identification of the flows pertaining to a single user session. As shown in Fig. 1, the rejection of an offered datagram is based both on the information conveyed by that datagram and on the estimation of the cluster resources usage.

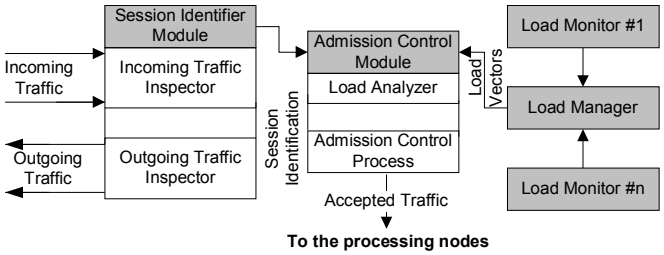

Figure 1. The functional architecture of the proposed system.

The incoming traffic, i.e. the traffic flowing from the clients to the servers, is first offered to the entry point to the cluster. It is first processed by the session identifier module which associates each incoming datagram either to a new session or to an already established one. The traffic is then delivered to the admission control module which is responsible for its acceptance or its rejection. Under overload, the admission control module aims for maximizing the cluster useful throughput by implicitly giving a higher priority to the already established sessions against any new incoming one. A load monitor at the entry point to the cluster and at each cluster node periodically collects the local load information and sends it to the peer component located at the cluster head. Finally, the accepted traffic is forwarded to a processing node inside the cluster. Fig. 2 below describes the topology on which the proposed system is deployed.

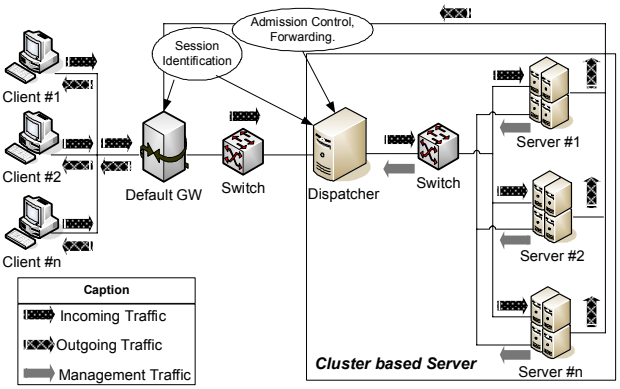

Figure 2. The general architecture of the proposed system.

All the processing servers are networked within the same cluster which is reached through its single entry point, the dispatcher. For the sake of an improved scalability, the system we advocate is built upon a one-way architecture where the outgoing traffic, i.e. the traffic flowing from the servers to the clients, bypasses the dispatcher node. Instead, it is forwarded to a default gateway configured as the default route for each processing server inside the cluster. The management traffic corresponds to the traffic carrying the load information. It is required by the admission control module to trigger responsive admission control decisions.

Although the dispatcher stands for a potential single point of failure, this work focuses on the failure free case and assumes that all the cluster entities are available at any time.

III. Detailed Architecture

A. *The session identifier engine*

The session identifier module is built as a stateful engine which inspects the payload of both the incoming and the outgoing datagrams searching for given patterns. Datagrams which are subject to content inspection are those exchanged over the signalling flows. Indeed, in [1] we showed that a multiple-flow based session, such as a video streaming or a voice over IP session, negotiates data flow identifiers and control data flow identifiers using messages sent over the main signalling flow. Hence, assuming a typical multiple-flow based session model [1], a searched pattern corresponds to a particular application level protocol header field holding the information required to identify any expected incoming flow associated to the already established session. The datagram content inspection is application layer specific and is done with respect to the syntax of the used signalling protocol.

The session identifier module maintains an in-memory session table which is updated with the receiving of an incoming traffic or with the inspection of the outgoing traffic, either by adding new entries or by updating the already existing ones.

A session is identified as the set of the transport level flows used for the signalling and for the data exchange all along its lifespan. For IPv4, a given flow is identified using a 13 bytes length vector denoted as *<IPsrc,IPdst,Portsrc,Portdst,Prot>*, defining respectively the source and the destination IP addresses, the source and the destination port numbers as well as the transport protocol. The flow state maintained within the session table includes moreover a set of variables necessary to track a given flow all along its lifespan. These variables include a timeout, a timestamp, an identity flag and a status flag. The timeout and the timestamp are used to detect the inactivity of a tracked flow. The status flag marks new flows, already established flows and inactive flows. The identity flag tells whether the handled flow is a signalling flow, an announced flow or a secondary flow.

A flow is considered new during the receiving of the first datagram asking for its establishment. The activity of each established flow is tracked in time. Hence, when no data is exchanged over an already established flow for a given duration, its status flag is set to the inactive value.

We define a signalling flow as a flow carrying application level signalling messages used to establish an end-to-end user session. The announced flows are either data or control data flows expected during a session lifespan but which are not yet established. The goal of the session identifier engine is to guess

the identifiers of these flows by inspecting the content, either of the incoming or the outgoing signalling messages. The secondary flows are the announced flows which have been successfully confirmed.

For a given service, the operations of the session identifier engine are summarized in Fig. 3 below.

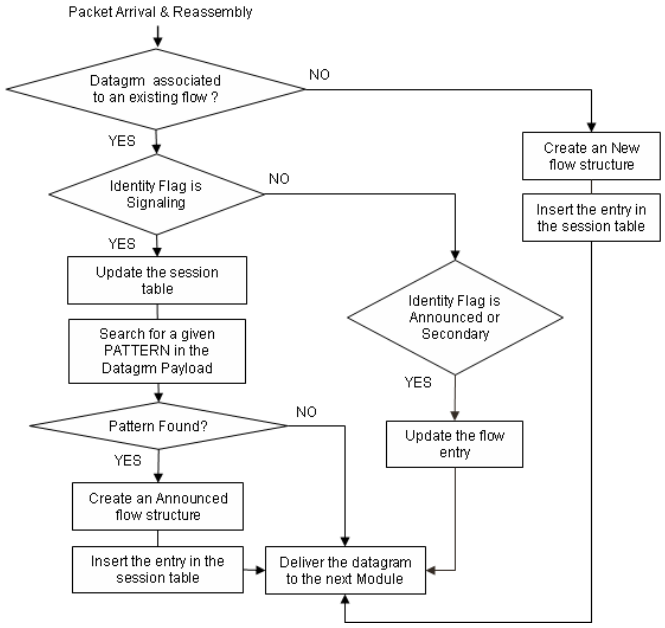

Figure 3. The session identifier module activity diagram.

When a datagram is received by the session identifier module, it is searched against the already maintained flows. If the offered datagram does not pertain to any recorded flow, it is assumed to hold a request for the establishment of a new signalling flow. A new flow structure is therefore created and added to the session table. In particular, the flow is marked as new, its identity flag is set to reference a signalling flow and its timeout is armed. The datagram is then delivered to the next engine.

If on the other hand the offered datagram is associated to an already recorded flow, the identity flag of this flow is checked. The first alternative is that the offered datagram pertains to an already established signalling flow. The corresponding entry is then updated. In particular, the flow is marked as established, its timeout is restarted and its timestamp is updated. The datagram payload is then inspected searching for a given pattern. If the searched pattern is not found, the datagram is delivered to the next engine. Otherwise, the found pattern is used to build a new entry referencing the expected flow. The datagram is then delivered to the next engine. In most cases, the outgoing traffic is required to be inspected so as to complete the identification of the announced flows. Recalling that the system we advocate is built upon a one-way architecture, the outgoing traffic is inspected at the default gateway. For this reason, we maintain at the default gateway a process which inspects the outgoing signalling traffic and which sends the useful information to its peer at the dispatcher. Once the entire identity of the announced flow is built, the corresponding flow entry is inserted into the session table.

The second alternative is that the offered datagram pertains to an announced flow or to a secondary flow. In this case, the corresponding entry is updated. In particular, an announced flow is set to a secondary flow. In both cases, the timeout is restarted and the timestamp value is updated. The datagram is then delivered to the next engine.

In order to incur a minimal latency to the end-to-end delay of the handled flows, we require a session table structure which provides good search and insertion times. On the other hand, since the number of the handled flows can reach up to some thousands, each flow is abstracted using a hash transformation which quickly computes a flow digest while avoiding collisions. Indeed, hash transformations are well known techniques to achieve relatively small memory space occupancy while allowing to uniquely identify a given flow [2]. Moreover, the detection of the inactive flows is particularly critical for the session identifier module because it affects the session table size and therefore the search and insertion times. In practice, the timeout value ranges between 10 and 60 seconds. Finally, in order to take into account any possible packet delay inside the network, the inactive flows are not immediately flushed from the session table when their timeout expires. Instead, a purging process periodically removes their entries when at least twice the corresponding timeout value elapses [3].

B. The session aware admission control engine (SA2C)

The admission control engine is responsible for the acceptance and for the rejection of the incoming traffic. Its main objective is to prevent the overload of the cluster resources while maximizing the operator profitability by maximizing the cluster useful throughput, in terms of completed sessions per unit of time. The system we advocate prevents the overload of the cluster resources by triggering, for each offered datagram, a decision which considers both the information it conveys and an estimation of the short-term usage of the cluster resources. In order to provide an improved responsiveness, the proposed system involves two moving thresholds, T_2 and T_3, as illustrated in Fig. 4 below.

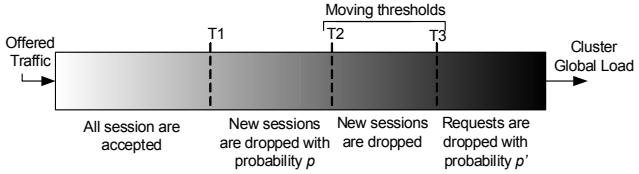

Figure 4. The proposed admission control mechanism.

SA2C applies a probabilistic dropping of the offered network traffic such that under heavy load, the datagrams pertaining to the already established sessions are granted a higher priority than those holding requests for the establishment of new sessions. The key features of the proposed system are the following two equations which together specify the dropping probability of a given offered datagram in time.

First, SA2C maintains a measurement based estimation of the cluster global load, updated with the receiving of the instantaneous cluster node load vectors each fixed time interval t and computed at time I_i according to (1).

$$l_c^i := \varphi\left(l_j^i\right) = \alpha * \bar{l}_j^i + (1-\alpha) * \sigma\left(l_j^i\right)^2 \quad (1)$$

where:

- $1 \le j \le N$, N is the number of the cluster servers,
- $I_0 = 0$ and $I_{i+1} = I_i + t$,
- $\bar{l}_j^i$ is the mean load of the cluster nodes,
- $\sigma\left(l_j^i\right)^2$ is the load variance of the cluster nodes,
- α is a smoothing factor having a value within [0,1] and used to better reveal the load distribution inside the cluster.

The instantaneous cluster global load l_c^i and the cluster head load l^i determine the drop probability p of the offered datagrams according to the following equation $p = p\left(l^i, l_c^i\right)$ (2)

$$p = \begin{cases} 0 & , \text{if } \max\left(l^i, l_c^i\right) \le T_1 \\ 1 - f\left(\max\left(l^i, l_c^i\right)\right), f(x) = \left|\dfrac{x - T_1}{T_2 - T_1}\right| & , \text{if \{new session\} and } T_1 < \max\left(l^i, l_c^i\right) \le T_2 \\ 1 & , \text{if \{new session\} and } T_2 < \max\left(l^i, l_c^i\right) \le T_3 \\ \max\left(p^u, 1 - p^u\right), \text{where } p^u = 1 - g\left(\max\left(l^i, l_c^i\right)\right), g(x) = \left|\dfrac{x - T_3}{C - T_3}\right| & , \text{otherwise} \end{cases}$$

When the cluster experiences a load value under the first threshold T_1, all the incoming traffic is accepted and forwarded to a processing node inside the cluster. Once the cluster load goes beyond T_1, the incoming traffic holding requests for the establishment of new sessions is dropped with a probability p computed as described above. When the cluster load exceeds the second threshold T_2, only the traffic pertaining to the already established sessions is admitted at the entry point to the cluster. This rule aims mainly to avoid the interruption or the QoS degradation of the already established sessions. Particularly, it reduces the discrimination against long lived sessions since short lived sessions always have a higher chance to complete normally.

The cluster global load is considered as critical when it goes beyond the third threshold T_3. Rather than interrupting the already established sessions leading them to be restarted from scratch, we suggest to instantaneously degrade their QoS. The associated datagrams are dropped with a maximized probability p as described above. Finally, when the cluster runs close to its edge capacity C, all the incoming traffic is rejected waiting for some of the cluster resources to be released.

In practice, the rejection of the incoming traffic is triggered each time interval T_i computed as a function of the dropping probability p as shown in (3).

$$T_{i+1} = (1-p) * T_i \quad (3)$$

In order to better prevent the persistent overload situations, we suggest improving the responsiveness of the admission control policy by involving moving thresholds rather than static thresholds holding for the whole future. Critical thresholds are moved proportionally to the instantaneous cluster global load value. However, since crossing each threshold portrays a specific overload situation, we suggest to dynamically and differently adjusting T_2 and T_3. The idea is to linearly decrease T_2 and to multiplicatively decrease T_3, as shown below (4).

$$\begin{cases} T_2 = T_2 - \Delta\left(T_2, \max(l^i, l_c^i)\right) \\ T_3 = T_3 * \Delta\left(T_3, \max(l^i, l_c^i)\right) \end{cases} \text{ where } \begin{cases} \Delta\left(T_2, \max(l^i, l_c^i)\right) = \left|T_2 - \max\left(l^i, l_c^i\right)\right| \\ \Delta\left(T_3, \max(l^i, l_c^i)\right) = \left|T_3 - \max\left(l^i, l_c^i\right)\right| \end{cases} \quad (4)$$

where Δ measures the load excess computed respectively against T_2 and T_3. Finally, once the load gets below T_1, both thresholds are reset to their initial values set by the operator.

The already described mechanism is designed such that it fastens the rejection of the new incoming sessions when the cluster experiences a high load condition. However, when sudden bursts of load occur due to short lived sessions, slowing down the rejection of the offered new sessions seems more appropriate since cluster resources are likely to be released in the short run. Consequently, in order to allow our approach to meet the stability constraint, it needs to be more sensitive to the characteristics of the load sustained within the cluster in time. To achieve this goal, we suggest to adjusting the admission control decisions according to an estimation of the short-term load that the cluster potentially experiences during a time interval T instead of considering only instantaneous load feedbacks.

In practice, we maintain a history of the load sustained within the cluster during each time interval T. The T-dimensional space spanned by the cluster node's load samples for a time period T starting at time I_i is described using a load history matrix denoted as L^i. L^i is computed as shown in (5).

$$L^i = \begin{bmatrix} l_1^{i*T} & l_1^{i*T+1} & \dots & l_1^{i*T+T-1} \\ l_2^{i*T} & l_2^{i*T+1} & \dots & l_2^{i*T+T-1} \\ \dots & \dots & l_j^k & \dots \\ l_N^{i*T} & l_N^{i*T+1} & \dots & l_N^{i*T+T-1} \end{bmatrix} \quad (5)$$

where:

$\{j : 1..N, k : i*T \dots i*T+T-1\}$, N being the number of nodes in the cluster,

and

$$\begin{cases} I_0 = 0 \\ I_{i+1} = I_i + T \end{cases}$$

The load matrix L^i is used to estimate the cluster short-term load at time I_{i+1} as shown in (6,7).

$$\hat{l}_c^{i+1} := \varphi\left(\hat{l}_j^{i+1}\right) / 1 \le j \le N \quad (6)$$

where $\varphi(x)$ is described in (1).

An estimation of the load of a given node j at I_{i+1} is calculated as shown in (7).

$$\hat{l}_j^{i+1} = \phi\left(l_j^k\right) \pm err_i = \sum_{k=i*T}^{k=i*T+T-1} \alpha_j^k * l_j^k \pm err_i, j : 0..N \quad (7)$$

where:

- $\phi(x)$ applies a simple forward linear regression model as described above in (7),
- err_i is a periodically updated error used to regulate the accuracy of the prediction model. It is computed as the normalized step between an estimated value and its effective measure as shown in (8).

$$err_i = \left| \Delta\left(\hat{l}_j^i, l_j^i\right) / l_j^i \right| \quad (8)$$

This error is used as a damp coefficient measuring the step between the stable and the responsive admission control decisions as shown in (9).

$$\breve{l}_c^i := (1 - err_i) * \hat{l}_c^{i+1} + err_i * l_c^i \quad (9)$$

Finally, l_c^i and l^i are substituted in (2) by $\breve{l}_c^i$ and $\breve{l}^i$ so as to provide adaptive and stable decisions. In particular, a value of err_i set to 1 defines an exclusively measurement based session aware admission control policy.

C. The load management engine

The load management engine involves two peer components aiming respectively to collect and to monitor the local load values of the cluster nodes. The collected load vectors are periodically sent to the monitor component at the entry point to the cluster over a UDP channel. A measure of the load includes significant indicators of the usage of the server resources such as its CPU usage, its memory usage, its network buffer usage, its I/O queue length as well as its application server's backlog queue length. Fig. 5 below describes the monitor and the manager operations.

```
The Monitor side
Loop forever
        - Catch the load vector
        - Send the load vector over the UDP channel established
        with the peer module at the dispatcher;
        - Wait δ;

The Manager side
Loop forever
        On receive {Load_Vector} from the Node I
          - Store for Analysis
        After 3*δ
          Return Load Monitor failed;
```

Figure 5. The load manager operations.

IV. Related Works

Internet server admission control shares almost the same objectives with the core network admission control. Chen and Mohapatra [4] applied the ERD approach [5,6] to regulate the acceptance of web server requests. They used a double threshold based admission control to monitor the application server listen queue. Once the server utilisation exceeds a first threshold, requests of lower priority are rejected with a higher probability. All the requests are rejected when the second threshold is reached. This approach was showed to be effective to control differentiated services mainly in terms of queuing delays between lower and higher traffic priority classes. Its major drawback is that the application server queue length is not necessarily a good indicator of the server load. Abdelzaher et al. [7] assumed a linear regression method which estimates the impact of the handled requests on the system utilization. They used a linear feedback control theory which admits an appropriate number of requests while keeping bounded the system utilization. However, they didn't consider any further constraints of the handled traffic. Lee et al. [8] focused on web servers and assumed the knowledge of the request arrival rate as well as the knowledge of the maximum waiting time for each incoming traffic class. They suggested two admission control approaches. The first maximizes the potential profit of the service provider while the second leads the controlled web server to admit as many clients as possible.

Most of the state of the art research works on Internet server admission control provide facilities exclusively adapted to single-flow based sessions. Our work focuses however on meeting the specific requirements of the multiple-flow based services.

V. Conclusion and Future Works

Service aware network management is a key issue both for the current and for the future NGN networks. In this work, we advocate an innovative open architecture for maximizing the operator profitability by maximizing its cluster-based server's useful throughput in terms of completed sessions per unit of time. The proposed architecture is based on original concepts which meet the constraints of multiple-flow based sessions. Indeed, it is first enhanced with means to explicitly identify the flows pertaining to a single user session. Second, it uses a three-threshold based responsive admission control policy where critical thresholds tune themselves according to the cluster overload ratio. In order to guarantee the fair completion of sessions independently of their duration, the system is enhanced with means to ensure stability. Indeed, it adjusts all its decisions according to an estimation of the short-term cluster load instead of considering only instantaneous load feedbacks. At an extreme, during a persistent overload situation or when the handled session is single-flow based, our system reacts by slowing down the degradation of the already established sessions. Near future works will focus on evaluating our approach in providing an improved useful throughput in a cluster of Internet servers. Target applications include video streaming and voice over IP using SIP. Further works aim to address the client based service differentiation during the admission control as well.

References

[1] N. Ayari, D. Barbaron, L. Lefèvre and P. Primet, "Session awareness issues of the next generation cluster based network load balancing frameworks", Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications, AICCSA07, May 2007.

[2] Z. Cao, Z. Wang, E. Zegura, "Performance of hashing based schemes for Internet load balancing", Proceedings IEEE INFOCOM, 2000, vol. 1, pp. 332-341.

[3] K.C. Claffy, H.-W. Braun and G.C. Polyzos, "A parameterizable methodology for Internet traffic flow profiling", Proceeding of the IEEE Journal on Selected Areas in Communications, vol. 13, Oct. 1995 pp. 1481–1494.

[4] X. Chen and P. Mohapatra, "Providing differentiated services from an Internet server", Proceedings of the 8th IEEE Transactions on Computer Communication and Networks, 1999, pp. 214-216.

[5] S. Floyd and V. Jacobson, "Random Early Detection for congestion avoidance", Proceedings of the IEEE/ACM Transactions on Networking, Aug. 1993, pp. 60-64.

[6] E. Hashem, "Analysis of random drop for gateway congestion control", Technical report MIT-LCS-TR-467, Laboratory of Computer Science, 1989, pp. 60.

[7] T. Abdelzaher, K. Shin and N. Bhatti, "Performance guarantee for Web server end-systems: a control theoretical approach", Proceedings of the IEEE Transactions on Parallel and Distributed Systems, 2002, pp. 60-131.

[8] S. Lee, J. Lui, Y. Yau, "Admission control and dynamic adaptation for a proportional delay diffserv-enabled web server", Proceedings of the IEEE SIGMETRICS'02, 2002, pp. 60.