SARA: A Session Aware Infrastructure for High Performance Next Generation Cluster-based Servers

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/CNRS, ENS, INRIA, UCB)
46, allée d'Italie - 69364 LYON Cedex 07, France
{laurent.lefevre,pascale.primet}@ens-lyon.fr

***Abstract*-Internet server clustering has been widely used by operators to improve the scalability and the availability of the rendered services under heavy load condition. Load balancing is a well known mean to optimize the usage of the cluster available resources by fairly allocating them to the offered network traffic. Most state-of-the-art research advocate either session oblivious or coarse grained session aware load balancing architectures. While guaranteeing session integrity, the coarse grained session awareness leads to an unfair allocation of the cluster resources. In this work, we advocate an innovative fine grained session aware load balancing architecture of an offered IP traffic to a cluster of Internet servers. The proposed architecture aims to maximize the cluster useful throughput in terms of completed sessions per unit of time. It provides high availability capabilities in case of failure of the legitimate entry point to the cluster. Finally, it is fully client transparent and is open to adapt to any multiple-flow based NGN service such as voice over IP or video streaming.**

I. INTRODUCTION

Performance critical next generation services are heavily dependant both on scalability and on high availability for guaranteed QoS provisioning. Server clustering is a well known mean to provide additional processing capabilities and improved failure recovery facilities. Previous works on cluster-based network traffic load balancing [1,2] provide exclusively flow aware solutions which do not guarantee the building of robust and scalable IP service frameworks. Conversely, most of the next generation IP 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 [3]. Typical examples include media streaming and voice over IP. On the other hand, the currently proposed high availability models for cluster-based servers are transport level unaware [4]. These models grant a little interest to avoid the interruption of the already established end-to-end sessions in case of failure of either the legitimate cluster head or the processing server. From an operator's perspective, session unawareness means that a cluster-based server which seems to be fully satisfying the client demands at a high throughput is in reality wasting its resources either on failed or on reduced QoS sessions.

In this work[1], we advocate an innovative architecture for fine grained session aware load balancing of an offered IP traffic to a cluster of Internet servers. The proposed system uses original concepts to guarantee the application level session integrity while efficiently spreading the incoming network sessions among the cluster available resources. The cluster resource's overload is prevented by using a session aware admission control which grants, during overload, a higher priority to the already established sessions against the new incoming ones. The involved mechanisms achieve particularly an improved responsiveness and a better stability through adjusting all their decisions according to an estimation of the cluster short term load rather than considering only instantaneous load feedbacks. The proposed architecture is enhanced with means to provide fine grained high availability capabilities. Indeed, it uses an active replication based framework [5] to allow the recovery of both the connection-oriented and the connectionless active sessions in case of failure of the legitimate entry point to the cluster. Finally, it is open to adapt to any multiple-flow based NGN service such as voice over IP and streaming video.

The remainder of this paper is organized as follows. In section II, we outline the general architecture of the proposed system. A detailed overview of its operations is provided in section III. Finally, in section IV, we conclude the paper by describing the perspectives of this work.

II. GENERAL ARCHITECTURE

SARA is built upon a functional separation of its operations into control plane, data plane and management plane functions. This functional grouping is summarised in Table I below (see Table I).

The control plane functions provide the processing required for controlling the end-to-end communication between the clients and the processing servers. It includes the session identification, the admission control, the target server computation and the fault recovery. The session identification provides means to associate each incoming datagram either to an already established session or to a new incoming one. The admission control provides means to accept or to reject the incoming datagrams according to the load sustained within the cluster. The server selection associates each new incoming session to an available processing server inside the cluster. Finally, the fault recovery provides means to recover the already established sessions when the legitimate cluster head fails.

The data plane functions provide the processing required on the data to spread inside the cluster. It includes mainly the data

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

forwarding function which keeps routing the data associated to a given session to the right cluster processing server.

TABLE I
FUNCTIONAL GROUPING OF THE OPERATIONS OF SARA

FUNCTIONS	CONTROL PLANE	DATA PLANE	MANAGEMENT PLANE
SESSION IDENTIFICATION	X		
ADMISSION CONTROL	X		
SERVER SELECTION	X		
FAULT RECOVERY	X		
FORWARDING		X	
LOAD ACCOUNTING			X
AVAILABILITY MONITORING			X

The management plane functions provide the processing needed to locally account and monitor the cluster resources in terms of load and availability. The information provided by the management plane functions are necessary to the control plane functions to enable the adaptive processing.

The interaction between the control plane, the data plane and the management plane functions is illustrated in the following Fig. 1.

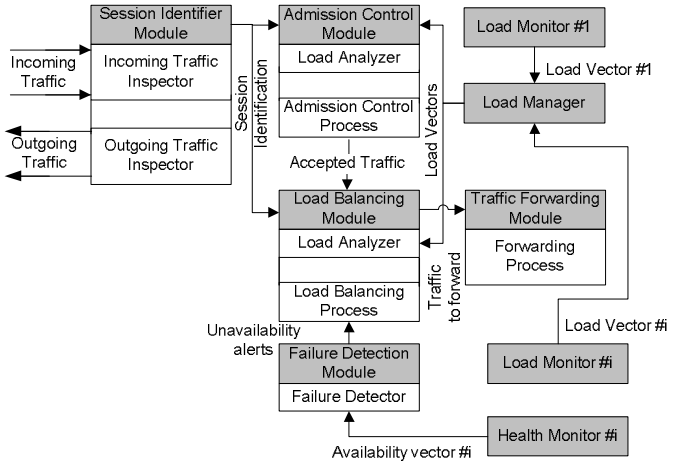

Figure 1. SARA functional architecture.

The topology on which SARA is deployed is on the other hand described by Fig. 2 below. The incoming traffic is first handled by the dispatcher where a session identifier engine tracks each session state and associates the offered traffic either to a new session or to an already established one. SARA achieves fine grained session integrity by means of the explicit identification of the flows pertaining to a single user session. The offered traffic is then delivered to the admission control engine which is responsible for the acceptance or for the rejection of the offered sessions. The admitted network traffic is associated to a processing server inside the cluster. In particular, a processing server is selected for each new incoming session while the traffic pertaining to an already established session is forwarded to the processing server to which that session was associated.

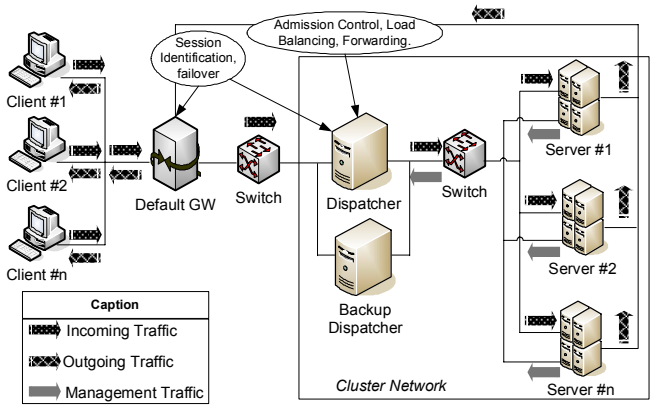

Figure 2. SARA general architecture.

For scalability concerns, the outgoing traffic is forwarded to a default gateway configured as the default route for each cluster node. Finally, since the dispatcher stands for a potential single point of failure, it is actively replicated [4,5]. In case it fails, a backup dispatcher transparently recovers the new incoming sessions while avoiding the interruption of the already established ones.

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 being exchanged over the signalling flows. Indeed, in [3] we showed that multiple-flow based sessions such as video streaming sessions negotiate data flow identifiers and control data flow identifiers using messages exchanged over the main signalling flow. A searched pattern is a particular application level protocol header field holding the information necessary to identify the possible next incoming flows associated to an already established session. The datagram content inspection is application layer specific and is done with respect to the syntax of the used signalling protocol. For instance, let us assume a trivial RTSP based unicast video streaming session. First, a client establishes a TCP flow with the streaming server. Upon this flow will be exchanged the RTSP signalling messages holding either requests sent from the client or replies sent from the streaming server. A client uses a typical SETUP message to ask the server for a particular media while specifying on which ports it wishes to receive it as well as the transport protocols it potentially supports. The server replies the client by announcing the identifiers of the next RTP/RTCP data flows associated to that session. Finally, the media is sent to the client.

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 associated 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:Portsrc, IPdst:Portdst, Prot>*, defining the source and the destination IP addresses and port numbers as well as the used 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 and which are not yet established. The idea 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.

The operations of the session identifier engine are summarized in Fig. 3 below.

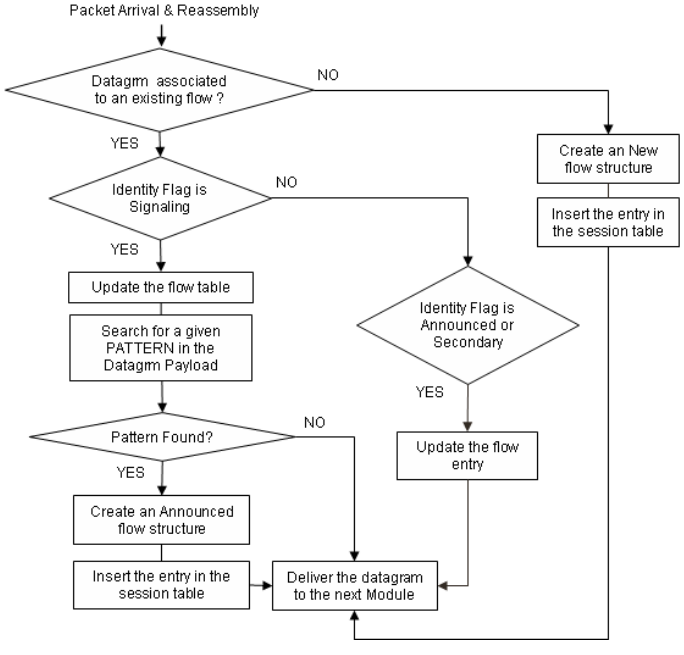

Figure 3. The session identifier module activity diagram.

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 [6]. 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 [7].

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 admission control module used in SARA involves two moving thresholds, T_2 and T_3, as illustrated in Fig. 4 below.

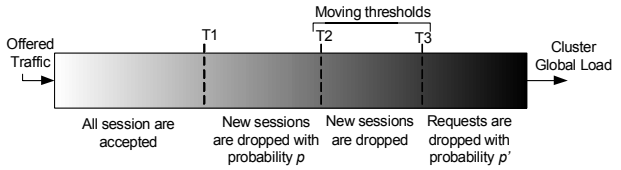

Figure 4. The proposed admission control mechanism.

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 used admission control mechanism 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).

$$l_c^i := \varphi\left(l_j^i\right) = \alpha * \overline{l_j^i} + (1-\alpha) * \sigma\left(l_j^i\right)^2 \Big/ 1 \le j \le N \quad (1)$$

where:

- $1 \le j \le N$, N is the number of the cluster servers,
- $I_0 = 0$ and $I_{i+1} = I_i + t$,
- $\overline{l_j^i}$ 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 maximised 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. In order to allow our approach to meet the stability constraint, we suggest adjusting the admission control decisions according to an estimation of the short-term cluster load instead of considering only instantaneous load feedbacks. Interested readers can see [8] for further details.

C. The load balancing engine

SARA provides a session aware load balancing of the offered network traffic to the cluster resources. A new load balancing decision is taken for each datagram holding a request for the establishment of a new session. The application level session integrity is achieved by assigning all the flows pertaining to the same session, to the same cluster processing server. Fig. 5 below abstracts the activity diagram of the distribution process of the offered network traffic among the cluster resources.

The load balancing module is built as a stateful engine which maintains an in-memory mapping between the handled sessions and the associated processing servers inside the cluster. This mapping is maintained by an association table which indexes each session by using the identifiers of its

signalling flow and which associates to it the internal IP address of the chosen processing server.

The admitted traffic is delivered to the load balancing engine which searches for the corresponding flow identifiers in the session table. If the admitted datagram pertains to a signalling flow marked as new, it is assumed to hold a request for the establishment of a new session. A server is then chosen among the pool of available servers and is associated to it. The association table is updated accordingly and the datagram is relayed to the processing node. If on the other hand the admitted datagram pertains to an already established signalling flow or to a secondary flow, the association table is looked up so as to find the server already associated to the session in hands. The offered datagram is then forwarded to that processing server.

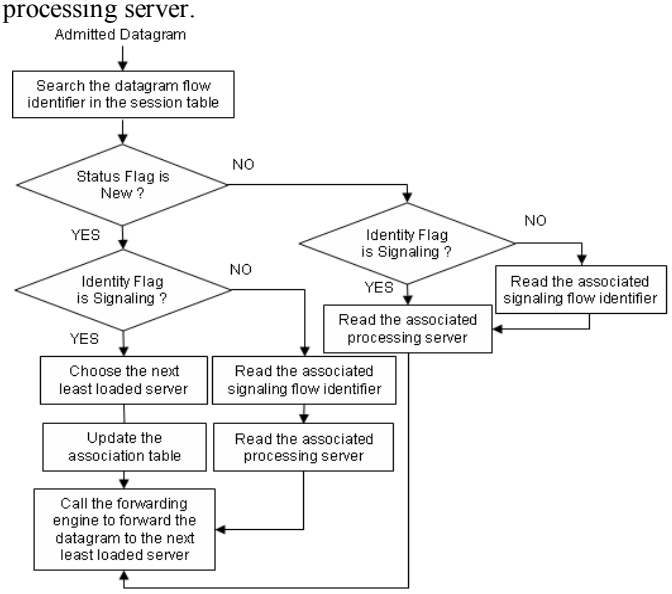

Figure 5. Network traffic dispatching process in SARA.

A processing server is chosen on the basis of its load value. SARA proposes a novel responsive load balancing algorithm baptised *shell* for *sh*ort-term forecasted *l*east *l*oaded server. *Shell* aims to achieving an improved stability while spreading the offered network traffic among the cluster available resources. Indeed, it reacts to the characteristics of the offered network sessions by adjusting its decisions according to a short term estimation of each cluster server load rather than considering only instantaneous load feedbacks. To achieve its goals, *shell* uses a load history sustained for each cluster node for each time interval *T*. Each load history matrix describes the T-dimensional space spanned by the load samples. Let us denote Lj^i the load history matrix for node j during a period *T* starting at time I_i. Lj^i is computed as shown in (5).

$$L_j^i = \left\lfloor l_j^{i*T} \quad l_j^{i*T+1} \quad \ldots \quad l_j^{i*T+T-1} \right\rfloor \tag{5}$$

where $1 \le j \le N$, N is the number of the cluster servers and

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

The load history matrix Lj^i is used to estimate the short term load of node *j* at I_{i+1} as shown in (6).

$$\hat{l}_j^{i+1} = \phi\left(l_j^k\right) \pm err_i = \sum_{k=i*T}^{k=(i+1)T-1} \alpha_j^k * l_j^k \pm err_i, j : 0..N \tag{6}$$

where:

- $\phi(x)$ applies a simple forward linear regression model,
- 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 (7).

$$err_i = \left| \Delta\left(\hat{l}_j^i, l_j^i\right) \middle/ l_j^i \right| \tag{7}$$

This error is used as a damp coefficient measuring the step between the stable and the responsive load balancing decisions as shown in (8).

$$\breve{l}_j^i := (1 - err_i) * \hat{l}_j^{i+1} + err_i * l_j^i \tag{8}$$

In particular, a value of err_i set to 1 defines an exclusively measurement based session aware load balancing. Finally, the node *j* having the minimal short term estimated load is chosen as shown in (9).

$$j \middle/ \left\{ \hat{l}_j^{i+1} = \min\left(\hat{l}_k^{i+1}\right) \middle/ 1 \le j,k \le N \right\} \tag{9}$$

D. The forwarding engine

The forwarding engine is responsible for routing the offered datagrams to a processing server inside the cluster. The forwarding process could have been built using different techniques such as the link layer encapsulation [9] or the network layer encapsulation through IP tunneling [10]. In SARA, the forwarding engine uses the link layer forwarding mechanism. Indeed, the IP tunneling technique has two major drawbacks. First, it requires that the cluster head as well as each cluster node to being tunneling aware. Second, it adds an additional overhead to the forwarding procedure both at the sender and the receiver sides. To achieve the link layer forwarding, each cluster node shares the same virtual IP address as the cluster head. On the other hand, to ensure that the incoming traffic will be first handled by the dispatcher, each cluster node disables its ARP processing. The forwarding process uses instead a static association maintained at the entry point to the cluster which tells for each cluster node its link layer address. The forwarding process consists in encapsulating each offered IP packet into a link layer frame where the destination link layer address is the internal address of the cluster processing server.

E. The availability monitoring engine

In order to avoid assigning the offered data to an instantaneously unavailable processing server, it is necessary to track the availability of the cluster nodes. Different failure types can occur at a given server. In SARA, we distinguish between the planned and the unplanned failures. The availability monitoring engine is built upon two peer components. A first component is attached to each cluster node to periodically broadcast availability announcements. A second component is attached to the cluster head. It takes the absence of proof of aliveness of a node as an evidence of its failure. The availability messages are exchanged over UDP channels established between the failure detector process and the health

monitor process. Fig. 6 below describes a typical availability message which spans over 11 bytes.

Figure 6. The availability message format

The Host_Identity field designates the identity of the node which sends the announcement. The Seq_Nber field is an integer used to detect and to count the lost heartbeat messages. The Flag field is an integer which identifies the availability event announced by the heartbeat message. It takes one of the three following values corresponding respectively to an available node and an available application, an available node and an unavailable application or a planned failure to occur. In the last case, the timestamp field indicates the planned failure time. Otherwise, it is used to recall the time at which the availability announcement has been sent. Fig. 7 below abstracts the failure detection process operations.

```
The Manager side
function Failure_detector(Host h)
        On receive {Availability_Announcement} from h
          wait d;
        Case Type in:
          Application_Available:
            return Up;
          Application_Unavailable or Announced_Failure:
            return Crashed;
        After n*d
          return Crashed;
```

Figure 7. The failure detection procedure.

where:

- d is the heartbeat inter-arrival period,
- and $n*\delta$ is the timeout period associated to a given health monitoring channel. A typical value of n is 3.

F. The failure recovery engine

In previous works [4,5], we described T2CP-AR, an active replication based mechanism which provides TCP based streams with high availability capabilities. SARA applies the T2CP-AR system at its entry point so as to provide high availability capabilities both to connection-oriented and to connectionless services. The cluster head is actively replicated so as to enable the failover of the offered network traffic on an available replica while avoiding the interruption of the already established sessions.

G. 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. 8 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 8. The load manager operations.

IV. Conclusion and Future Works

This paper advocates a scalable and a robust session aware infrastructure baptized SARA to efficiently process the next generation multiple-flow based services. SARA is fully client transparent. It provides the intelligence required by an operator to maximize its profitability by maximizing the useful throughput of its cluster-based servers in terms of completed sessions per unit of time. Original concepts are used to guarantee the application level session integrity while efficiently spreading the incoming network traffic inside the cluster. At an extreme, a persistent overload situation leads our system to slow down the degradation of the already established sessions. High availability capabilities are moreover provided to enable the recovery of the already established sessions in case of failure of the legitimate cluster head. Current works focus on evaluating the proposed mechanisms in a real operator context to provide an improved useful throughput in a cluster of voice over IP SIP-based servers. Future target applications include improving the scalability of streaming video servers as well.

References

[1] Network Working Group, "RFC 2782 - A DNS RR for specifying the location of services", Feb. 2000.

[2] www.linuxvirtualserver.org.

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

[4] N. Ayari, D. Barbaron, L. Lefèvre and P. Primet, “A survey on fault tolerance approaches and issues for highly available services”, submitted to IEEE Surveys and Tutorials.

[5] N. Ayari, D. Barbaron, L. Lefèvre and P. Primet, “T2CP-AR: A system for Transparent TCP Active Replication”, Proceedings of the IEEE 21st International Conference on Advanced Information Networking and Applications, AINA 2007, pp xx.

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

[7] 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.

[8] N. Ayari, D. Barbaron, L. Lefèvre and P. Primet, “A session aware admission control scheme for Next generation IP services”, Proceedings of CCNC'08, 2008.

[9] C.Z. Xu, "Scalable and secure Internet services and architectures", Chapman & Hall, 2005.

[10] C. Perkins, "IP Encapsulation witsin IP", Internet RFC, October 1996.