Maximum Availability Server Selection Policy for Efficient and Reliable Session Control Systems

Marjan Bozinovski, Hans P. Schwefel, and Ramjee Prasad, *Senior Member, IEEE*

***Abstract*—There has been a rapid growth of services based on session control. Session-based services comprise multimedia conferences, Internet telephone calls, instant messaging, and similar applications consisting of one or more media types such as audio and video. Deployment examples include session control services as part of the IP multimedia subsystem (IMS), in the third-generation mobile networks. High service dependability in session control systems is achieved by introducing redundancy, e.g., through reliable server pooling (RSerPool) or clustering. Namely, session control servers are multiplied in server sets. Performance of such replicated session control servers is quantified by transaction control time. Thus, reducing transaction control time enhances performance. Server selection policies (SSP) are crucial in achieving this goal. The maximum availability (MA) SSP is proposed to improve session control performance in scenarios with server and communication failures. Based on a status vector, MA aims at maximizing the probability of successful transaction with the current transmission, thereby minimizing the average number of attempted servers until success. MA is applicable in a broad range of IP-based systems and services, and it is independent of the fault-tolerant platform. A simple protocol extension is proposed in order to integrate MA into the RSerPool fault-tolerant architecture. In addition, an analytic model is derived based on certain system model assumptions. Analytic and simulation results show that transaction control time is considerably reduced with MA as opposed to when using traditional round robin.**

***Index Terms*—Fault-tolerance, performance, server selection policies (SSP), session control.**

I. Introduction

RECENTLY, Internet services based on the session notion have grown in number and popularity. Examples of such services are multimedia conferences, IP telephony, instant messaging, and similar multimedia applications. Deployment scenarios include session control services within the IP multimedia subsystem (IMS), in the third-generation mobile networks [1]. In the IMS, the call session control function (CSCF) servers perform session management, based on the session initiation protocol (SIP) [2]. Session control protocols such as SIP are transactional protocols. In general, a transaction consists of a single request, any intermediate provisional response, and a final response to that request.

Manuscript received August 19, 2004; revised October 31, 2005; approved by IEEE/ACM Transactions on Networking Editor H. Schulzrinne. This work was supported by Siemens, Munich, Germany.

M. Bozinovski was with the Center for Teleinfrastruktur, Aalborg University, 9100 Aalborg, Denmark. He is now with AD Makedonski Telekomunikacii, 1000 Skopje, Macedonia (e-mail: marjan.bozinovski@mt.com.mk).

H. P. Schwefel and R. Prasadis are with the Center for Teleinfrastruktur, Aalborg University, 9100 Aalborg, Denmark (e-mail: hps@kom.aau.dk; prasad@kom.aau.dk).

Digital Object Identifier 10.1109/TNET.2007.892875

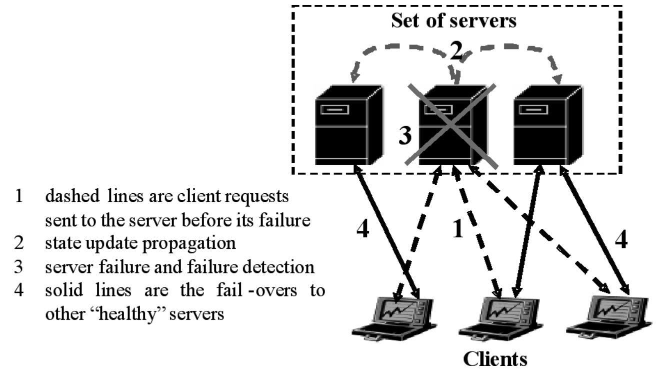

Set of servers
2
3
1 dashed lines are client requests sent to the server before its failure
2 state update propagation
3 server failure and failure detection
4 solid lines are the fail -overs to other "healthy" servers
4
1
4
Clients

Fig. 1. Fault-tolerant replicated session control system.

Fault-tolerance in session control systems is achieved by introducing redundancy, e.g., through reliable server pooling (RSerPool) [3] or clustering [4]. Namely, session control servers are multiplied in server sets (Fig. 1). Session control is a time-critical application. Performance of session control is quantified by transaction control time. Transaction control time is defined as the mean time between the moment of request sending and the moment of final response receipt at the transaction initiator (including possible multiple fail overs to different servers). One important challenge in such replicated session control systems is how to enhance performance, i.e., how to reduce transaction control time. The SSPs are crucial in reducing transaction control time.

The main problem addressed in this paper is maximizing the probability of a successful transaction with the current transmission, thereby minimizing the average number of attempted servers until success. The ultimate achievement is reduced transaction control time.

The algorithm proposed in [5] solves the problem by exploiting the dynamically obtained information on the last server access moments and the corresponding activity status of servers in a server set. The algorithm selects the server with the largest last known up time, if any. If there is no such server, the scheme selects the server with the smallest last known down time. This paper extends the work presented in [5] in the following aspects: 1) it proposes a protocol extension that integrates the developed server selection algorithm into RSerPool; 2) it develops enhanced system model assumptions resulting in more advanced simulation programs; and 3) it provides more comprehensive numerical evaluation and analysis of the proposed methods.

The proposed method is most useful in cases of frequent interaction between the client and the same server set (e.g., in messaging sessions utilizing SIP servers) as the status for a given server set stored in the client is in that case always maintained up-to-date. Note, however, that depending on the SIP architecture and the deployed fault-tolerance mechanisms, it is very well possible that the role of the client is taken by a proxy server, e.g., the P-CSCF in 3GPP IMS, that as a consequence may interact very frequently with the deployed replicated server set; in case of 3GPP IMS the latter could correspond to the S-CSCF. Second, the scheme requires adaptations to the clients, but large parts of those can be implemented in standardized protocol layers, see, e.g., the approach in Section IV.

It is worth pointing out that the method is not limited to only session stateful servers, i.e., stateless servers multiplied in a server pool for fault-tolerance reasons (higher availability/reliability) also benefit from this method. This means that the servers in the same server pool do not necessarily need to maintain and replicate session states. Thus, the method is applicable to both stateless server sets and session stateful server sets.

This paper is organized as follows. The existing server selection policies are discussed in Section II. In Section III, a formal description of the proposed MA server selection algorithm as well as its implementation details are presented. Furthermore, alternatives of the proposed MA SSP are outlined. The protocol extension to integrate MA into the RSerPool architecture is proposed in Section IV. The system model assumptions used for derivation of analytic expressions and simulation programs are described in Section V. The definitions of the evaluation metrics, the actual derivation of the analytic model, and the description of the event-driven simulation programs are presented in Section VI. Evaluation of the novel MA SSP and discussion of numerical results obtained via analytic expressions and simulations are performed in Section VII. Concluding remarks and directions for future work are presented in Section VIII.

II. Related Work

Server selection policies have been extensively studied in the literature. Some currently existing static and dynamic algorithms are outlined in this section. Existing static server selection policies use predefined schemes for selecting servers. Examples of static SSPs are as follows.

Round robin (RR) is a cyclic policy, where servers are selected sequentially in cycle [6].

Weighted RR is a simple extension of round robin. It assigns a certain weight to each server. The weight indicates the server's processing capacity. This SSP may also be dynamic if it can evaluate individual servers' capacities and their loads occasionally [6].

The unawareness of dynamic system states leads to low complexity, however, at the expense of potentially degrading performance and service dependability. Dynamic (adaptive) SSPs make decisions based on changes in the system state and dynamic estimation of the best server. Examples of dynamic SSPs are as follows.

Least used SSP [6]. In this SSP, each server's load is monitored by a central monitoring entity or by the client itself. Based on monitoring the loads of the servers, each server is assigned the so-called *policy value*, which is proportional to the server's load. According to the least used SSP, the server with the lowest policy value is selected as the receiver of the current message. It is important to note that this SSP implies that the same server is always selected until the policy values of the servers are updated and changed.

Least used with degradation SSP [6] is the same as the least used SSP with one exception. Namely, each time the server with the lowest policy value is selected from the server set, its policy value is incremented. Thus, this server may no longer have the lowest policy value in the server set and the policy may over time converge towards RR. Every update of the servers' policy values brings the SSP back to least used with degradation.

The effectiveness of a dynamic SSP critically depends on the metric that is used to evaluate the best server. The research on SSPs has been mainly focused on the replicated Web server systems. In such systems, the typical metrics are based on server proximity including geographic distance, number of hops to each server, round trip time (RTT), and HTTP response times [7]–[9]. The main objective of SSPs in Web systems is providing small service latency and high throughput (i.e., successfully transmitted data volumes in short time periods). These metrics are particularly important as the Web service is based on downloading files from Web servers over the Internet. Downloads can take substantial amounts of time if server responsiveness and link bandwidth are not optimized.

While SSPs in Web systems aim to provide high throughput and small service latency, session control protocols such as SIP deal with messages being rather small in size (500 bytes on average [10]). Thus, throughput as measured in data volumes per time unit is not as significant a metric as in the Web systems. However, service latency remains an important performance parameter. To the best of the authors' knowledge, SSPs have not been extensively investigated in the context of replicated session control systems.

III. Maximum Availability Server Selection Policy

Distributing SIP transactions among replicated servers is an increasingly significant issue. In particular, a SIP transaction consists of a single request, any intermediate provisional response, and a final response. The transaction definition is thus given as $\mathbf{SIP\ transaction} = \langle \mathbf{Request} \Rightarrow (\mathbf{Provisional\ Responses}) \Rightarrow \mathbf{Final\ Response} \rangle$.

N servers are assumed in the replicated server set. A SIP server is declared failed if the SIP client has not received a final response to a request within T_1 s. Upon failure detection, the fail-over mechanism resends the SIP request to the new server selected according to an SSP. This may potentially be repeated until all servers have been attempted.

The goal of the proposed algorithm is to reduce transaction control time. The algorithm is based on maximizing the probability of successful transaction with the nth request retransmission, under the condition that $(n-1)$ attempts have been unsuccessful. This leads to minimizing the number of attempted servers until success, which is shown to impact the transaction control time (see Section VI). The algorithm is referred to as MA SSP and it is developed here in the setting of server nodes that are homogeneous in the sense that they are subject to the same failure and repair model; furthermore, the failure and repair model has to fulfil some monotony requirements which hold for a large set of candidate failure models, including in particular exponential ON/OFF models. See the end of the Section III-A for details.

A. Description of the MA Algorithm

The MA SSP makes use of the assumption that the server whose last known up time is closest to the actual time, is most likely to be up at the actual time. The algorithm dynamically obtains information on the last server access moments and the corresponding activity status of servers in the server set, and selects the server with the largest last known up time, if any. If there is no such server, the MA SSP selects the server with the smallest last known down time. Thus, the MA algorithm aims at maximizing the probability of successful transaction with the current transmission.

We observe the mechanism for handling request transmissions/retransmissions at an SIP client [11]. In the case of transaction failure with one server, the client retransmits to another server and for one transaction this procedure may be repeated until the client has attempted all servers. Let us make the assumption that by some means (abstracted for the time being), at any moment a request is to be transmitted or retransmitted, the client is provided the so-called *server up status* (SUS) vector and *server down status* (SDS) vector. The SUS and SDS vectors are denoted by $\mathbf{u}$ and $\mathbf{d}$, respectively, and are defined as follows:

$$\mathbf{u} = [u_1(t_n), u_2(t_n), \ldots, u_N(t_n)] \tag{1}$$

$$\mathbf{d} = [d_1(t_n), d_2(t_n), \ldots, d_N(t_n)] \tag{2}$$

where $n(n = 1, \ldots, N)$ is the index of the nth transmission of the SIP request in a given transaction ($n > 1$ is a retransmission); t_n is the moment at which the request arrives at the selected server; $u_i(t_n)$ is the last status moment of server S_i known to be up (available); $d_i(t_n)$ is the last status moment of server S_i known to be down (unavailable). It should be noted that the inequality $u_i(t_n) \neq d_i(t_n) (\forall i = 1, \ldots, N)$ always holds because a server cannot be in ON and OFF state at the same time.

Let the vector $\mathbf{f_n}$ represent the servers, which failed while starting or completing the transaction in any of the first $(n-1)$ attempts

$$\mathbf{f_n} = \left[f_1^{(n)}, f_2^{(n)}, \ldots, f_N^{(n)}\right] \tag{3}$$

where $f_j^{(n)} = 1$ if the transaction failed with the server j in one of the first $(n-1)$ attempts, and $f_j^{(n)} = 0$ if the transaction was not sent to the server j in any of the first $(n-1)$ attempts. Thus, immediately before the nth attempt, $\mathbf{f_n}$ has $(N-n+1)$ elements with value 0, and $(n-1)$ elements with value 1. Since it gives information on which servers have not been attempted yet, $\mathbf{f_n}$ is referred to as the *remaining servers* (RS) vector. Every transaction is associated with its own RS vector, which is maintained until the (successful or unsuccessful) transaction completion. It is deleted once a transaction finishes (successfully or unsuccessfully).

We define the *conditional transaction dependability* with the attempt n as the probability of successful transaction with the nth request (re)transmission, under the condition that $(n-1)$ attempts were unsuccessful. The crucial question to pose is: How to select a server with the nth attempt, which will maximize the conditional transaction dependability?

Let us denote the conditional transaction dependability with the attempt n by $P_r(T_n|\overline{T_{n-1}})$, where T_n represents the event "*transaction is successful with the* nth *attempt*," and $\overline{T_{n-1}}$ is the event "*transaction was unsuccessful with the* $(n-1)$th *attempt*." For clarification, $\overline{T_{n-1}}$ naturally implies that all the attempts up to and including the $(n-1)$th attempt were unsuccessful. If $n = 1, \overline{T_{n-1}}$ does not exist as an event.

Let us find the expression of $P_r(T_n \,|\, \overline{T_{n-1}})$ when making the decision for the nth attempt. Let $\mathbf{u}$ and $\mathbf{d}$ be assumed to be known with certain values of their elements. It should be noted that only $(N-n+1)$ servers are still available for attempt. We will denote them by indexes $j \in \{j_n, \ldots, j_N\}$. Thus, the possible values of $P_r(T_n \,|\, \overline{T_{n-1}})$ are given as follows:

$$\begin{aligned} &P_r(T_n \,|\, \overline{T_{n-1}}) \\ &\quad \in \left\{P_r^{(j_n)}(T_n \,|\, \overline{T_{n-1}}), \ldots, P_r^{(j_N)}(T_n \,|\, \overline{T_{n-1}})\right\} \end{aligned} \tag{4}$$

$$\begin{aligned} &P_r^{(j)}(T_n \,|\, \overline{T_{n-1}}) \\ &\quad = \begin{cases} P_j(t_n) \cdot p_{\text{on}}^{(j)}(t_n|u_j(t_n)), & u_j(t_n) > d_j(t_n) \\ P_j(t_n) \cdot p_{\text{on}}^{(j)}(t_n|d_j(t_n)), & u_j(t_n) < d_j(t_n) \end{cases} \\ &\qquad\qquad\qquad\qquad\qquad\qquad j \in \{j_n, \ldots, j_N\} \end{aligned} \tag{5}$$

where $P_r^{(j)}(T_n \,|\, \overline{T_{n-1}})$ is the conditional transaction dependability with the attempt n provided that server j is chosen; $p_{\text{on}}^{(j)}(t_n \,|\, u_j(t_n))$ is the probability of finding server j in state ON at t_n provided that it was in state ON at $u_j(t_n); p_{\text{on}}^{(j)}(t_n \,|\, d_j(t_n))$ is the probability of finding server j in state ON at t_n provided that it was in state OFF at $d_j(t_n); P_j(t_n)$ is the probability that the transaction is successful provided that the server is in ON state at t_n.

Logically, $P_r(T_n \,|\, \overline{T_{n-1}})$ is maximized if the server with the maximal $P_r^{(j)}(T_n \,|\, \overline{T_{n-1}})$ is selected. However, nothing is known about $p_{\text{on}}^{(j)}(t_n \,|\, u_j(t_n)), p_{\text{on}}^{(j)}(t_n \,|\, d_j(t_n))$ and $P_j(t_n)$. In order to illustrate the analysis, let us assume that each server's activity follow a Markov ON/OFF process [12], [13]. Let the mean ON interval $1/\lambda_{\text{on}}$ and the mean OFF interval $1/\lambda_{\text{off}}$ be identical with all servers. By applying the Markov model [14], (5) turns into (6) shown at the bottom of the page, where $P_j = c$ for each $j \in \{j_n, \ldots, j_N\}; p = \lambda_{\text{off}}/(\lambda_{\text{off}} + \lambda_{\text{on}})$ and

$$P_r^{(j)}(T_n \,|\, \overline{T_{n-1}}) = \begin{cases} c \cdot \{p + q \cdot \exp[-(t_n - u_j(t_n)) \cdot (\lambda_{\text{on}} + \lambda_{\text{off}})]\}, & u_j(t_n) > d_j(t_n) \\ c \cdot p \cdot \{1 - \exp[-(t_n - d_j(t_n)) \cdot (\lambda_{\text{on}} + \lambda_{\text{off}})]\}, & u_j(t_n) < d_j(t_n) \end{cases} \quad j \in \{j_n, \ldots, j_N\} \tag{6}$$

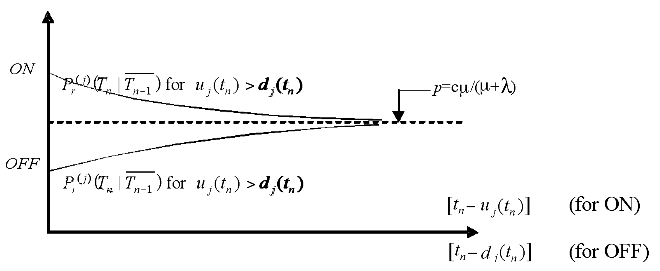


Fig. 2. Conditional transaction dependability as a function of the up/down status moments.

$q = 1 - p$. The general shape of this function is depicted in Fig. 2. The selection decision is thus formulated as follows:

$$
\begin{aligned}
&\text{select } j_{\max} \text{ such that:}\\
&P_r(T_n \mid \overline{T_{n-1}}) = P_r^{(j_{\max})}(T_n \mid \overline{T_{n-1}})\\
&\qquad = \max\left\{P_r^{(j_n)}(T_n \mid \overline{T_{n-1}}), \ldots, P_r^{(j_N)}(T_n \mid \overline{T_{n-1}})\right\}\\
&j_{\max} \in \{j_n, \ldots, j_N\}; n \in \{1, \ldots, N\}. \qquad (7)
\end{aligned}
$$

From (7), it is clear that the server maximizing $P_r(T_n \mid \overline{T_{n-1}})$ is the one, which has the largest $u_{j_{\max}}(t_n)$ such that $u_{j_{\max}}(t_n) > d_{j_{\max}}(t_n)$. In the case that for all servers $j \in \{j_n, \ldots, j_N\}, d_j(t_n) > u_j(t_n)$, then the server $j_{\max}$ with smallest $d_{j_{\max}}(t_n)$ is selected. Hence, the decision making procedure at the nth attempt is formalized as follows:

$$
\begin{aligned}
&\textbf{construct the vector } p \textbf{ such that}\\
&p_i = u_i(t_n) \cdot s(u_i(t_n) - d_i(t_n)) - d_i(t_n) \cdot s(d_i(t_n) - u_i(t_n))\\
&\qquad\qquad i \in \{1, \ldots, N\}\\
&\text{select } j_{\max} \text{ such that:}\\
&p_{j_{\max}} = \max(\mathbf{p}) \text{ such that } f_{j_{\max}}^n = 0\\
&\text{where}\\
&j_{\max} \in \{j_n, \ldots, j_N\}; n \in \{1, \ldots, N\} \qquad (8)
\end{aligned}
$$

where p is referred to as the *status vector* consisting of N elements and p_i is its ith element; $s(x)$ is the unit step function of the scalar argument x; $\max(\mathbf{p})$ is the maximum element of $\mathbf{p}$.

The MA SSP is based on the assumption that the server whose last known up time is closest to the actual time is most likely to be up at the actual time. The class of ON/OFF models that justify this assumption ensure that $P_r^{(j)}(T_n \mid \overline{T_{n-1}})$ is a monotonously decreasing function, independent of the current moment t_n and identical for all servers in the set. The Markov ON/OFF model is such an example. In the cases of, e.g., periodic (deterministic) ON/OFF models, the MA algorithm would need to be modified. However, the Markov ON/OFF model is the most common assumption in the literature for modeling up time and down time [12], [13].

To summarize, the MA algorithm maximizes the probability of a successful transaction with the current transmission. The drawing that summarizes the basic principle of MA is shown in Fig. 3.

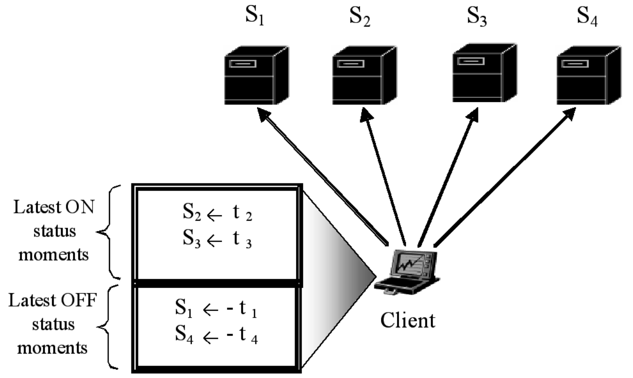


Fig. 3. Client makes a decision on which server is to be selected. In this example, S_2 has the largest (positive) time stamp, while S_4 has the smallest (negative) time stamp. Hence, S_2 is selected for serving the current transaction.

B. Implementation Details

It is important to note that the MA algorithm does not need to use three vectors (i.e., $\mathbf{u}, \mathbf{d}$, and $\mathbf{f_n}$) for its operation. Namely, the vector $\mathbf{f_n}$ is transaction associated[1] and is mandatory as one has to know which servers have already been unsuccessfully attempted out of the whole set. Nevertheless, instead of keeping the status vectors $\mathbf{u}$ and $\mathbf{d}$ each with dimension N, the *status vector* $\mathbf{p}$ of dimension N [as defined in (8)] can initially be created and then dynamically be updated. The status vector is updated when a transaction or a heartbeat (HB) sent to a given server is successfully completed or failed. A transaction (or an HB) is failed if the client has not received a response to the SIP request (or the HB request) within a time interval defined by a timeout. Whenever a new status moment t_i associated to server S_i, is obtained (when a transaction or HB having been sent to a given server is successfully completed or failed), the entry associated to server S_i in the vector $\mathbf{p}$ is updated as follows:

$$
p_i = \begin{cases} t_i, & \text{server is reported to be up at } t_i \\ -t_i, & \text{server is reported to be down at } t_i \end{cases} \qquad i \in \{1, \ldots, N\}. \quad (9)
$$

Thereby, the vector $\mathbf{p}$ in (8) is maintained, while keeping two separate vectors for up and down status moments is avoided. The same algorithm in (8) is used with the exception for the $\mathbf{p}$ calculation, which is dynamically done by applying (9).

C. Alternative Server Selection Policies

The following other SSPs that are related to the MA SSP are also proposed.

- Smart RR (SRR): In this SSP, a new request is sent to a server by applying RR on the current subset of servers that have been reported to be alive. If no server has been reported to be alive, RR is applied to the whole server set. The major advantage of this SSP is that it combines the good features of RR and MA SSP. Namely, due to RR, there is a fair load balancing in the system, whereas requests are sent to servers that are reported to be alive, thus increasing the instantaneous probability of successful transaction.

[1]Generalizations taking into account knowledge from concurrent transactions are possible but not considered further in this paper.

- SRR per Session (SRR-S): This is a variant of SRR, which is only applied to select a server for INVITE requests (new sessions) and for midsession requests that need to fail over due to a missing final response. Once a server is selected, all the next requests within the session are sent to the same server until the session ends or a request failure is detected. SRR-S aims to provide fair load balancing in a long run (due to RR on a session level), however, there is no load balancing within a session. It also aims at increasing the probability of successful transaction with the current transmission by sending the next request to the same server that successfully completed the previous transaction.

IV. Protocol Extension for Integrating MA Into RSerPool

The problem addressed in this section is the integration of MA into the RSerPool [3] fault-tolerant platform, which is being defined in the IETF. The solution to this problem is an RSerPool protocol extension. An overview of RSerPool is given in Section IV-A.

A. *RSerPool Fault-Tolerant Platform*

The RSerPool architecture is being standardized within the IETF RSerPool working group [3]. RSerPool defines three of the following types of elements:

- Pool Elements (PEs): servers that provide the same service within a pool;
- Pool Users (PUs): clients served by PEs;
- Name Servers (NSs): servers that provide the translation service to the PUs and monitor the health of PEs.

In RSerPool, pool elements are grouped in a *pool*. A pool is identified by a unique *pool name*. To access a pool, the pool user consults a name server. RSerPool consists of two protocols, namely, the *aggregate server access protocol* (ASAP) and the *endpoint name resolution protocol* (ENRP). ASAP uses a name-based addressing model which isolates a logical communication endpoint from its IP address(es). The name servers use ENRP for communication with each other to exchange information and updates about the server pools (note that name servers are also called ENRP servers). The instance of ASAP (or ENRP) running at a given entity is referred to as *ASAP (or ENRP) endpoint* of that entity. For example, the ASAP instance running at a PU is called *the PU's ASAP endpoint*.

Each time a PU sends a message to a pool that contains more than one PEs, the PU's ASAP endpoint must select one of the PEs in the pool as the receiver of the current message. The selection is done according to the current server selection policy. The example in Fig. 4 illustrates the event sequence when the PU's ASAP endpoint does a *cache population* [6] for a given pool name. Cache population (update) is updating of the *local name cache* with the latest name-to-address mapping data as retrieved by the ENRP server.

B. *RSerPool Extension for Cache Population*

We define an extension of the RSerPool architecture that supports the novel MA SSP. Here, the failure-detection intelligence is distributed in the pool user and the name server. The pool user makes use of the application layer and transport layer timers to detect transport failure, while name servers provide the keep-alive mechanism to periodically monitor PE's health. In the further text, the terms keep-alive and heartbeat are used interchangeably.

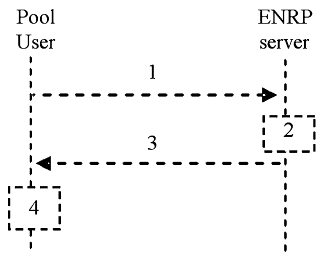


Fig. 4. Cache population as currently defined in [6]. The steps are explained as follows. 1) The ASAP endpoint of the PU sends a NAME RESOLUTION query to the ENRP server asking for all information about the given pool name. 2) The ENRP server receives the query, and locates the database entry for the particular pool name. The ENRP server extracts the transport addresses information from the database entry. The ENRP server creates a NAME RESOLUTION RESPONSE in which the transport addresses of the PEs are inserted. 3) The ENRP server sends the NAME RESOLUTION RESPONSE to the PU. 4) The ASAP endpoint of the PU populates (updates) its local name cache with the transport addresses information on the pool name.

The PU's ASAP endpoint accomplishes the updating of its status vector denoted as $\mathbf{p^{(u)}}$. The extension is rather simple and easy to introduce in RSerPool. It affects the communication between the ENRP server's and the PU's ASAP endpoint. It is assumed that both the functionality of the PU and the ENRP servers is extended to support the MA algorithm. The extended functionality in the ENRP server creates a status vector for a given server pool. This status vector is updated periodically by using the existing ASAP's keep-alive mechanism [6]. The name server's status vector is denoted as $\mathbf{p^{(s)}}$. The $\mathbf{p^{(s)}}$ vector for a given pool is stored in the same database entry in the name server reserved for that pool.

Steps 1)–4) (with reference to Fig. 4) for cache population with the MA RSerPool extension are defined as follows.

The ASAP endpoint of the PU sends a NAME RESOLUTION query to the ENRP server asking for all information about the given pool name.

The ENRP server receives the query, and locates the database entry for the particular pool name. The ENRP server extracts from the database entry the transport addresses information as well as the $\mathbf{p^{(s)}}$ vector. The ENRP server creates a NAME RESOLUTION RESPONSE in which the transport addresses of the PEs and the $\mathbf{p^{(s)}}$ vector are inserted.

The ENRP server sends the NAME RESOLUTION RESPONSE to the PU.

The ASAP endpoint of the PU populates (updates) its local name cache with the transport addresses information on the pool name. The PU's ASAP endpoint applies the following simple procedure in order to update the status vector $\mathbf{p^{(u)}}$:

$$p_i^{(u)} = \begin{cases} p_i^{(s)}, & \left|p_i^{(s)}\right| > \left|p_i^{(u)}\right| \\ p_i^{(u)}, & \left|p_i^{(s)}\right| \leqslant \left|p_i^{(u)}\right| \end{cases} \quad i \in \{1, \dots, N\}. \tag{10}$$

It should be noted that this works well under the condition of synchronized time clocks in pool users and name servers.

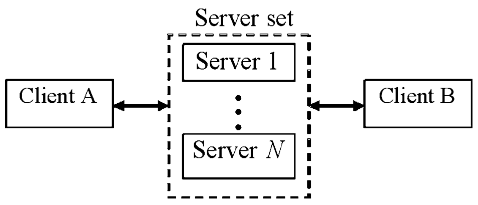


Fig. 5. General considered system.

Employing the network time protocol (NTP) provides acceptable levels of synchronization (maximum 20–50 ms clock drift if the NTP server is reached via WAN and less than 10 ms if the NTP clients and servers are part of the same LAN [15]). Note, however, that the MA algorithm does not really use the distance between time stamps, but only their (total) ordering. Hence, the synchronization requirements really reduce to monotony requirements for time stamps. The investigations whether the partial ordering as introduced by, e.g., logical clocks (see [16]), may be sufficient to achieve substantial performance improvement are outside the scope of this paper.

In Section V, system model assumptions used for developing the analytic model and simulation programs are presented.

V. System Model Assumptions

The general system considered is based on RSerPool as the underlying fault-tolerant platform shown in Fig. 5. The system model assumptions are presented in the following subsections.

A. Server Model Assumptions

Failures of hosts and networks follow certain failure/repair models. A random variable that accounts for system reliability is the time to failure (TTF). In reversible systems, where the system can be repaired after failure, another random variable is defined, i.e., time to repair (TTR). When the TTF and the TTR follow exponential distributions with mean time to failure (MTTF) and mean time to repair (MTTR), respectively, the overall reversible failure/repair process is referred to as two-state Markov ON/OFF model [14], which is used in the context of this paper.

There are N servers in the pool and this number is referred to as pool size. Each server is denoted by $S_n (n = 1, \ldots, N)$. MTTF and MTTR of S_n are denoted as $1/\lambda_n$ (ON) and $1/\lambda_n$ (OFF), respectively. Note that the MA SSP has been developed for homogenous server sets, however, the simulation experiments also investigate the case of nonhomogenous server sets (see Section VII).

There is a FIFO queue of size Q_n for incoming messages (i.e., the extreme case is: $Q_n - 1$ SIP messages are in the queue, and one SIP message is being processed). We will assume that S_n can only process one SIP message at a time. Once a SIP message is accepted for processing, it occupies the server's processor for a certain fixed period of time denoted by T_n (PSIP). Once the server finishes processing a SIP message, another SIP message, if any, is pulled from the queue for processing. An incoming SIP message is dropped if the current number of SIP messages in the server is equal to Q_n.

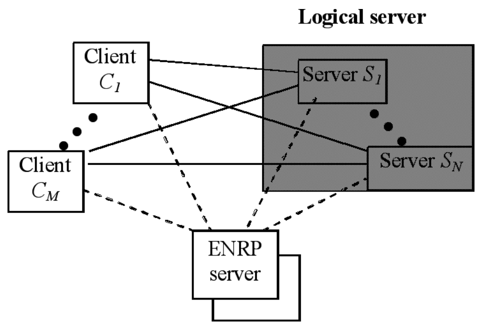


Fig. 6. Network links in an RserPool-based session control system.

When S_n fails, the SIP message that is currently being processed is dropped. Also, all the SIP messages that are currently in the server are dropped.

B. Network Model Assumptions

The different links present in the RSerPool-based session control system are depicted in Fig. 6. They interconnect clients and servers. SIP messages are sent and received over these links.

SIP is assumed to run over SCTP. Each host is dual-homed (i.e., it has access to two different networks) and a message is by default sent via the primary path, which is set to be the faster link. The timeout expires after the round trip time of the primary path. The RTT of the primary path is considered constant and its value is the doubled value of the message propagation time of the primary path. If a message is not acknowledged within the RTT of the primary path, it is retransmitted over the secondary path. The timeout being equal to RTT is a very tight assumption. However, it is justified taking into account that if the message delivery is successful, the acknowledgment arrives exactly in RTT time units.

There are M clients in total. From the RSerPool point-of-view, as shown in Fig. 6, clients represent PUs and servers represent PEs. For PU-to-PE links, the delivery time of a SIP message over the primary link under the condition of successful delivery is denoted as t_{sip}. For PE-to-ENRP links, the delivery time of an ASAP message over the primary link under the condition of successful delivery is denoted as t_{pe}. For PU-to-ENRP links, the delivery time of an ASAP message over the primary link under the condition of successful delivery is denoted as t_{pu}. Each link is also assigned fixed packet loss probability p_{loss}.

Note that in the parameter settings of the simulation experiments, the SIP timeout is much larger than the SCTP timeout, consequently, in case of a host failure, an unsuccessful SCTP retransmission on the secondary link will happen first, but the SCTP retransmission mechanism is not performance relevant for the host failure case. Only in scenarios of packet loss the SCTP retransmission is beneficial.

C. Traffic Model Assumptions

SIP sessions arrive following a *Poisson* process with intersession arrival rate λ_s. This assumption is expected to be very accurate (according to the central limit theorems for stochastic processes) if the sessions are created by a large population of independent users as it would be the case if the "client" is the

P-CSCF in a 3GPP IMS system. The number of transactions in a session is assumed to be uniformly distributed within the integer set $\{2, \ldots, 2K_t + 2\}$ (the simplest session consists of an INVITE and BYE transaction, e.g., a basic telephony call), where K_t is the mean number of non-INVITE and non-BYE transactions per session. A session proceeds with the next transaction even if the current transaction has not been successfully completed, unless the current transaction is the INVITE transaction. The inter-transaction time (i.e., the interval between two subsequent SIP requests) is exponentially distributed[2] with mean value $1/\lambda$. A non-INVITE transaction consists of a request and a final response, while an INVITE transaction consists of INVITE, 200OK, and ACK. The requests between INVITE and BYE are assumed to represent the MESSAGE method used for instant messaging (IM) application [17]. Thus, the transactions between INVITE and BYE are referred to as IM transactions.

The RSerPool architecture allocates an ENRP server (or a set of replicated ENRP servers) to a given operation scope. In this model, the operation scope comprises a single server pool (server set). The ENRP server employs the ASAP keep-alive messages to periodically monitor the availability status of all the pool elements, or a subset of them, in the server pool. A new keep-alive request is sent once the previous one has been either acknowledged by each server or timed out for each server due to a missing keep-alive response.

When a keep-alive request arrives at a server, it is immediately processed if the server's buffer size is not exceeded. A HB request is dropped if the current number of SIP messages in the server is equal to the server buffer size.

The RSerPool extension proposed in Section IV is assumed. Furthermore, each pool user periodically initiates cache update, i.e., name resolution with the ENRP server. The following are the summarized quantitative traffic assumptions for the RSerPool-based system:

- keep-alive period in the ENRP server is denoted as T_{hb};
- timeout per keep-alive request is denoted as T_0;
- processing time of a keep-alive request in a pool element is fixed and is denoted by T_{phb};
- cache update period is denoted as T_c;
- timeout per cache update is T_0 (identical as the timeout per keep-alive request);
- processing time of a name resolution request in the ENRP server is fixed and is denoted by T_{pnr}.

The previous time parameters are illustrated in Fig. 7.

D. Timeouts and Server Selection Policies

The SIP server is declared failed if the SIP client has not received a final SIP response to a SIP request within T_1 seconds. Upon failure detection, the fail-over mechanism resends the SIP request to the new server selected according to the SSP. The request is not retransmitted to the currently failed server because the server is assumed to remain failed for a longer period of time.

A number of SSPs are employed at a client. The following SSPs are used in the evaluation: 1) RR; 2) SRR-S; 3) MA.

[2]This assumption is made for the ease of modeling. Although the results in Section VII give a first indication of the impact of different types of inter-transaction time processes, detailed investigations need to be performed in future work.

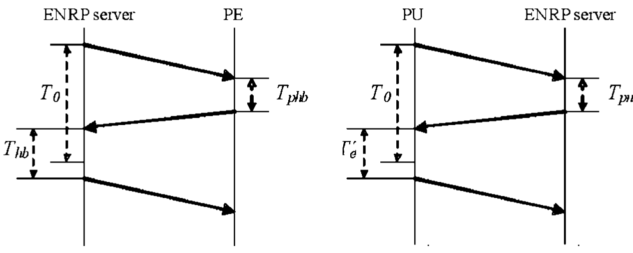


Fig. 7. Time parameters: (a) HB and (b) cache update.

VI. Analytic and Simulation Models

In this section, analytic expressions are derived for selected metrics and certain assumptions, and simulation programs are presented. First, the evaluation metrics are defined. Second, the analytic expressions for transaction dependability and transaction control time are determined. Third, the simulation programs developed for the evaluation purpose are briefly explained.

A. Evaluation Metrics

The set of evaluation metrics are transaction dependability and transaction control time. The transaction dependability is a service dependability metric, while transaction control time is a performance metric.

Transaction dependability at a client is defined as the probability that the transaction is successfully completed as seen by the SIP client that initiates it. One distinguishes between INVITE transaction dependability, IM transaction dependability, and BYE transaction dependability.

Transaction control time at a client is defined as the average duration of the interval between the moment of sending the request and the moment of receiving either a final response (200 OK) in case of IM and BYE, or the moment when ACK is received by the destination client in case of INVITE. One distinguishes between INVITE transaction control time, IM transaction control time, and BYE transaction control time.

B. Transaction Dependability Expression

An expression for transaction dependability is derived for a general transaction, which is then adapted to a particular transaction. A general transaction is used to term any transaction. The derivation is conducted for the maximum availability SSP. The system model assumptions from Section V and certain restrictions to them are adopted for the derivation. Note that these restrictions are only valid for the derivation and the simulations to validate the following analytic results.

$\lambda_n^{(\mathrm{on})} = \lambda_{\mathrm{on}}$, and $\lambda_n^{(\mathrm{off})} = \lambda_{\mathrm{off}}$ for each $n = 1, \ldots, N$, i.e., all the servers follow identical Markov ON/OFF process (where $\lambda_{\mathrm{on}}, \lambda_{\mathrm{off}}$ are fixed constants).

$T_n^{(\mathrm{psip})} = T_{\mathrm{psip}}$ for each $n = 1, \ldots, N$. However, it is assumed that no queuing takes place at a server, i.e., messages are processed immediately once they enter the server.

There is only one session with infinite number of transactions arriving so that inter-transaction time is approximated as the interval from the last response receipt (in case of successful transaction) or timeout expiration (in the case of unsuccessful transaction) until the next transaction request. This assumption is made to simplify the derivation of the analytic model and its validation through simulation. Also, there is only one client that starts transactions and one client that receives transactions.

Network redundancy is not used, i.e., messages can only be sent over a single link.

The general expression for the transaction dependability in a stationary system state at a client is given as follows:

$$D = \lim_{i->\infty} D(i) \quad (i = 1, 2, \dots, \infty) \tag{11}$$

where $D(i)$ is the dependability of a transaction with sequence number i at a client and is found as follows:

$$D(i) = \sum_{n_i=1}^{N} D(n_i) \quad (i = 1, 2, \dots, \infty) \tag{12}$$

where $D(n_i)$ is the probability that transaction i is successful with the n_i attempt. The expression of $D(n_i)$ is given as follows:

$$D(n_i) = P_r\left(T_{n_i}^{(i)}\right) = \sum_{n_{i-1}=0}^{N} P_r\left(T_{n_i}^{(i)}, T_{n_{i-1}}^{(i-1)}\right) \qquad (n_i = 1, \dots, N) \tag{13}$$

where $T_{n_i}^{(i)}[T_{n_{i-1}}^{(i-1)}]$ is the event: "*transaction* i $[i-1]$ *is successfully completed with the* $n_i[n_{i-1}]$ *attempt and there have been unsuccessful attempts with* $n_i - 1[n_{i-1} - 1]$ *servers*;" There is an exception, namely, $T_0^{(i-1)}$ represents the event: "*transaction* $(i-1)$ *is not successfully completed because of unsuccessful attempts with all the* N *servers*;" The following notation is used: $P_r(U)$ is the probability of the event U, $P_r(U, V)$ is the probability of the joint event (U, V), and $P_r(U \mid V)$ is the probability of the conditional event $U \mid V$. The expression (13) can be rewritten as

$$D(n_i) = \sum_{n_{i-1}=1}^{N} P_r\left(T_{n_i}^{(i)} \middle| T_{n_{i-1}}^{(i-1)}\right) \cdot D(n_{i-1}) + P_r\left(T_{n_i}^{(i)} \middle| T_0^{(i-1)}\right) \cdot P_r\left(T_0^{(i-1)}\right) \qquad (n_i = 1, \dots, N). \tag{14}$$

One should note that according to (12), $P_r(T_0^{(i-1)})$ represents

$$P_r\left(T_0^{(i-1)}\right) = 1 - D(i-1) = 1 - \sum_{n_{i-1}=1}^{N} D(n_{i-1}). \tag{15}$$

For more compact analysis, let us introduce the following matrix notation. Let the $N \times 1$ vector $\mathbf{d}^{(i)}$ be defined as

$$\mathbf{d}^{(i)} = [D(1) \quad \cdots \quad D(n_i) \quad \cdots \quad D(N)]^T. \tag{16}$$

Let the $N \times 1$ vector $\mathbf{p}_0^{(i)}$ be defined as

$$\mathbf{p}_0^{(i)} = \left[P_r\left(T_1^{(i)} \middle| T_0^{(i-1)}\right) \quad \cdot \quad P_r\left(T_N^{(i)} \middle| T_0^{(i-1)}\right)\right]^T. \tag{17}$$

Let the $N \times N$ matrix $\mathbf{P}^{(i)}$ be defined as

$$\mathbf{P}^{(i)} = \begin{bmatrix} P_r\left(T_1^{(i)} \middle| T_1^{(i-1)}\right) & \cdots & P_r\left(T_1^{(i)} \middle| T_N^{(i-1)}\right) \\ \vdots & \ddots & \vdots \\ P_r\left(T_N^{(i)} \middle| T_1^{(i-1)}\right) & \cdots & P_r\left(T_N^{(i)} \middle| T_N^{(i-1)}\right) \end{bmatrix}. \tag{18}$$

Thus, combining (14)–(18), the following matrix form of (14) is obtained:

$$\mathbf{d}^{(i)} = \left[\mathbf{P}^{(i)} - \mathbf{p}_0^{(i)} \cdot \mathbf{u}\right] \cdot \mathbf{d}^{(i-1)} + \mathbf{p}_0^{(i)} \tag{19}$$

where $\mathbf{u} = [1, \dots, 1]$ is a unit $1 \times N$ vector. According to (11)

$$\mathbf{d} = \lim_{i->\infty} \mathbf{d}^{(i)} \quad \mathbf{p}_0 = \lim_{i->\infty} \mathbf{p}_0^{(i)} \quad \mathbf{P} = \lim_{i->\infty} P^{(i)} \tag{20}$$

Letting $i \to \infty$ in (19), one obtains

$$\mathbf{d} = [\mathbf{I} - \mathbf{P} + \mathbf{p}_0 \cdot \mathbf{u}]^{-1} \cdot \mathbf{p}_0 \tag{21}$$

where $\mathbf{I}$ is an $N \times N$ identity matrix, and $\mathbf{A}^{-1}$ is the inverse matrix of the square matrix $\mathbf{A}$.

Finally, the dependability of a transaction in a stationary system state at a client is given as follows:

$$D = \mathbf{u} \cdot \mathbf{d}. \tag{22}$$

It should be noted that (21) and (22) are the general expressions that are further used as the basis to develop the expressions for particular application scenarios. The derivation is completed when all the elements of $\mathbf{p}_0$ and $\mathbf{P}$ are analytically determined. Thus, the total number of elements to determine in one scenario is $N + N^2$.

The determination of $\mathbf{p}_0$ and $\mathbf{P}$ elements has been conducted for MA without HB mechanism (see [5, App.] for detailed derivation). The status vector turns out to have the crucial impact on the derivation. Two cases are considered: deterministic and exponentially distributed random inter-transaction time. The special case when $N = 2$ is analyzed and the exact expressions obtained for deterministic inter-transaction time are also used for the case of exponential inter-transaction time. The actual determination of the expressions of these elements was presented in detail in [5, App.]. The derivation for other SSPs with HB support will be conducted in some future studies.

C. Transaction Control Time Expression

The transaction control time is computed only for successfully completed transactions. The expression for the transaction control time is found in terms of the probability that a transaction is successfully completed with the nth attempt after unsuccessful attempts with $(n-1)$ servers. This probability is introduced in (13) and here it is denoted as $D(n)$. Ultimately, the transaction control time is determined using the following expressions:

$$T_c^{\mathrm{im,bye}} = 2T_{\mathrm{psip}} + 4t_{\mathrm{sip}} + T_{\mathrm{i}} \cdot \sum_{n=1}^{N} (n-1)D(n) \qquad (23)$$

where $D(n)$ in (23) is computed for IM and BYE transactions. The expression for INVITE transactions may be obtained as a simple modification of (23). As opposed to IM and BYE transaction, which consist of one request and one response, the INVITE transaction is made up of the first request (INVITE), a response (200 OK), and another request (ACK), hence, three messages in total. The INVITE transaction control time is determined as follows:

$$T_c^{(\mathrm{inv})} = 3T_{\mathrm{psip}} + 6t_{\mathrm{sip}} + T_1 \cdot \sum_{n=1}^{N} (n-1)D(n). \qquad (24)$$

Note that the sum term in (23) is proportional to the mean number of servers attempted until successful transaction. The ideal SSP provides $D(n) = 0$ for $n > 1$ thus the sum term becomes zero.

D. Event-Driven Simulation Programs

The following simulation programs were developed within this paper.

- Simulation program (referred to as *Simulation Model 1*) to validate the analytic expressions for transaction dependability and transaction control time. It is based on the system model presented in Section V and their restrictions presented in Section VI.
- Simulation program (referred to as *Simulation Model 2*) to investigate transaction dependability and transaction control time. It is entirely based on the system model presented in Section V.

The probability estimates are computed using the standard definition of relative frequency (also known as proportion). Thus, the probability of an event in a given simulation run is calculated as the ratio of the number of event occurrences and the total number of trials. Simulation probability estimates can be considered statistically significant as the number of trials in each simulation run for each estimate was in the order of magnitude of 10^8. Furthermore, the number of occurrences of the less probable event (out of two complementary events) in a simulation run was at least 10^4.

A transaction control time estimate for a given transaction is computed as an average value of all transaction control times measured for that type of transaction during a single run.

VII. Numerical Results and Discussions

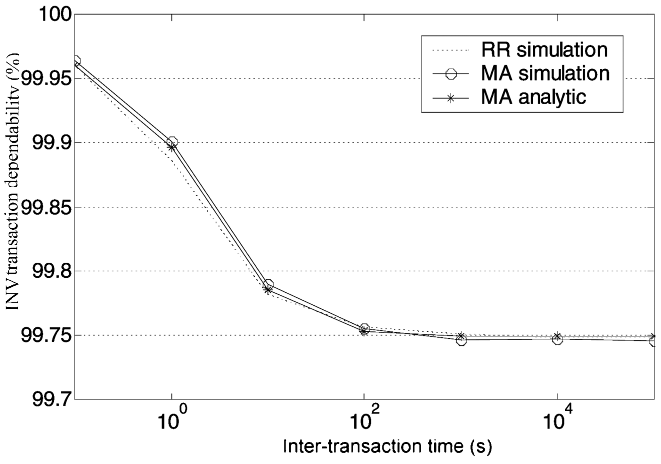
The scenarios considered can be classified into the following categories: 1) Analytic model validation (results based on analytic expressions in Section VI, and results based on Simulation Model 1) and 2) investigations on dependability/performance (results based on Simulation Model 2).

A. Analytic Model Validation

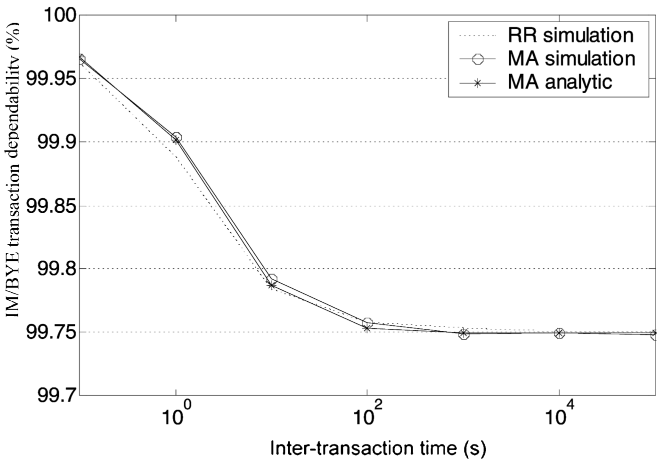
Within this section, the analytic model is validated through a detailed quantitative comparison between results based on expressions in Section VI, and results based on Simulation Model 1. The input parameters are presented in Table I.

TABLE I
Input Parameters

N	$T_n^{(psip)}$	t_{sip}, t_{pu} and t_{pe}	$1/\lambda_n^{(on)}$	$1/\lambda_n^{(on)}$	T_l
2	0.01 s	50 ms	4750 s	250 s	1 s



(a)

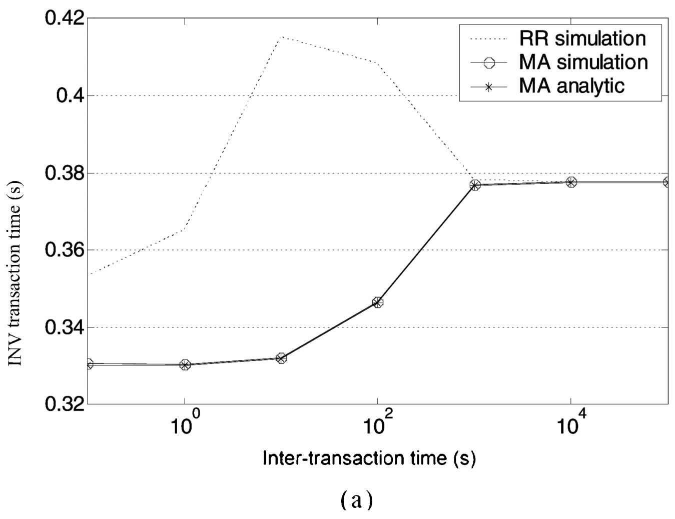


(b)

Fig. 8. (a) INVITE dependability. (b) IM and BYE dependability versus deterministic inter-transaction time.

Figs. 8 and 9 present the transaction dependability and transaction control time as functions of the inter-transaction time, respectively. The inter-transaction time is a deterministic variable in this scenario. The numerical results are obtained via simulation (for RR and MA) and through the analytic expression derived for MA with deterministic inter-transaction time. The following important conclusions are drawn.

1) In Fig. 8, it is interesting to note that the transaction dependability metrics achieve the maximum when the inter-transaction time approaches zero. In general, this is valid for such a model because the effective inter-request time is always lower-bounded to NT_1 when all the servers are in the OFF interval. On the other hand, there is no lower bound on the effective inter-transaction time when there is at least one server in the ON interval. Thus, shorter effective inter-transaction time implies that the percentage of trans-



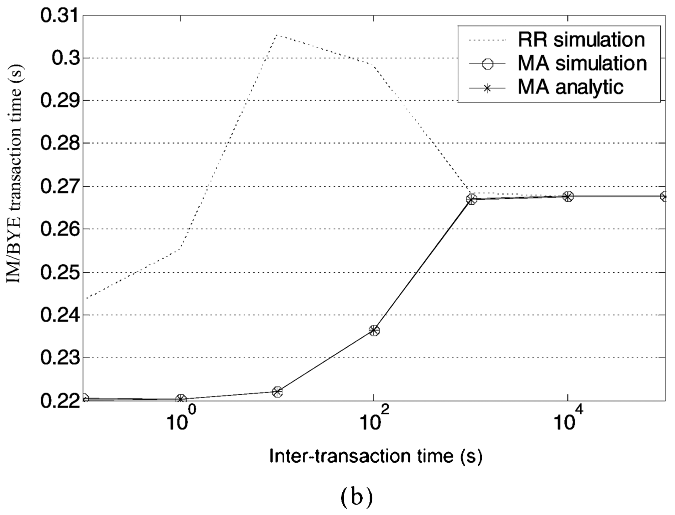
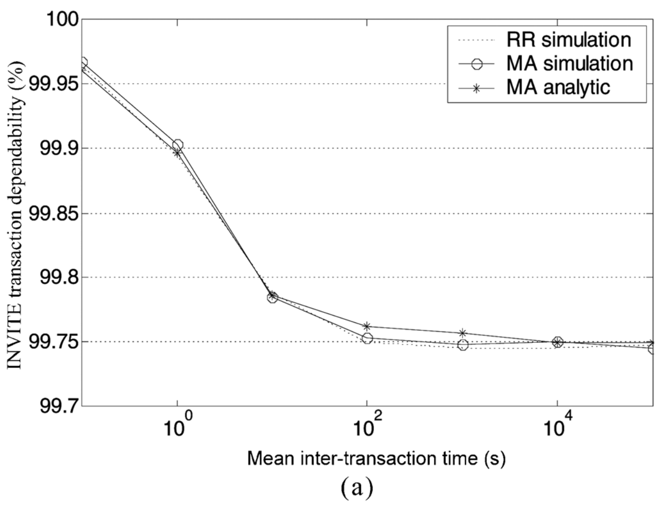


Fig. 9. (a) INVITE transaction time. (b) IM and BYE transaction time versus deterministic inter-transaction time.



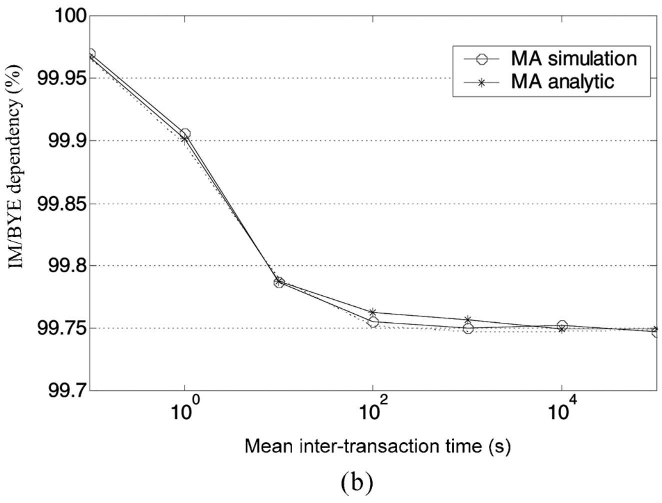


Fig. 10. (a) INVITE dependability. (b) IM and BYE dependability versus mean inter-transaction time (which is exponentially distributed random variable).

actions that arrive in an ON interval is growing as opposed to the percentage of transactions that arrive in OFF intervals.

2) The transaction dependability metrics converge towards the limit probability of 0.9975, which is obtained as $1-(1-p)^2$ when the inter-transaction time infinitely grows (Fig. 8); where p is the *a priori* availability of each particular server (i.e., 0.95 in this particular scenario). For extremely large inter-transaction times, the correlation between two subsequent transactions is negligible.
3) The curves of the transaction dependability metrics for either SSP are very close to each other (Fig. 8). This is intuitively expected, as the sequence order in which the servers are accessed only has a small impact on the probability of transaction success for the considered scenarios that the time scales of the failure/repair model are much larger than the time scales of an individual transaction. The MA SSP is slightly more efficient as it attempts to maximize the probability of successful transaction with every transmission.
4) The INVITE transaction time is generally longer than the IM/BYE transaction times due to the larger number of messages to be exchanged within the INVITE transaction (Fig. 9). Fig. 9 evidently shows that for inter-transaction times up to 1000 s, MA outperforms RR with respect to transaction control time. This is due to the inherent capability of MA to minimize the number of servers to attempt until success. Recall that MA always sends the next request to the server that was last known to be alive, thereby maximizing the instantaneous probability of success. For extremely large inter-transaction times, the transaction control times converge towards a certain limit for both MA and RR. For this range, MA performs as well as RR because there is not up-to-date information about the servers' status at the client.
5) The analytic numerical results (both transaction dependability and transaction control time) for MA are almost perfectly matching those obtained via simulations. Thus, the analytic model has been validated.
6) Fig. 10 presents the transaction dependability metrics as functions of the mean inter-transaction time, respectively. In this scenario, the inter-transaction time is an exponentially distributed random variable. The element $P_r\{A_{2\,|\,1}\}$ of the matrix $\mathbf{P}$ is approximated with the expression for the

TABLE II
INPUT PARAMETERS AND NOTATION

Server model	N	2
	Q_n	1000
	$\{1/\lambda_1^{(on)}, 1/\lambda_2^{(on)}\}$	{4750 s, 4900 s}
	$\{1/\lambda_1^{(off)}, 1/\lambda_2^{(off)}\}$	{250 s, 100 s}
	$\{T_1^{(psip)}, T_2^{(psip)}\}$	{10 ms, 50 ms}
Network model	t_{sip}, t_{pu} and t_{pe} of primary link	50 ms
	t_{sip}, t_{pu} and t_{pe} of secondary link	200 ms
	p_{loss}	10^{-4}
Traffic model	M	10
	λ	$0.1\ s^{-1}$
	K_t	10
Time constants	T_{hb}	30 s
	T_c	30 s
	T_0	1 s
	T_{phb}	10 µs
	T_{pnr}	10 µs
	T_l	1 s
Notation	MA	Maximum Availability SSP
	RR	Round Robin
	RR SMART SES	Smart Round Robin per Session SSP
	simulation	results obtained via simulations
	analytic	analytic results
	w/o	without

same element when the inter-transaction time is deterministic (see [5, App.]). There are no significant differences between the exponential and deterministic results therefore transaction control time plots for the exponential case are not shown.

The previously mentioned Points 1)–5) apply here as well. The transaction dependability metrics for the exponential case are nevertheless shown to point out that the analytically obtained dependability metrics do not perfectly match those from the simulations (see Fig. 10). This is due to the approximate expression for $P_r\{A_{2|1}\}$ in $\mathbf{P}$. Nevertheless, the analytic and simulation curves are quite close to each other, which is satisfactorily acceptable.

B. Investigations on Dependability/Performance

The following simulation scenarios are based on Simulation Model 2. The goal of these scenarios is to evaluate how servers with nonhomogenous characteristics impact transaction dependability and transaction control time. The input parameters are listed in Table II. The output parameters are plotted as functions of normalized session arrival rate (defined as $M\lambda_s T_1^{(\text{psip})}$). Results are discussed for different server selection policies. The plots are shown in Figs. 11 and 12.

In this scenario, the server set assumed consists of two servers with nonidentical hardware and software characteristics. One of the servers has higher processor speed as well as faster SIP software, while its ON/OFF model is such that the resulting availability is lower than that of the other server. HB requests are sent regularly every 30 s. With RSerPool, cache updates are also

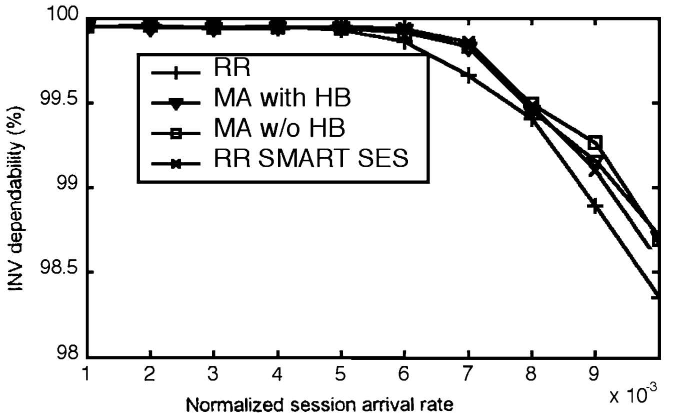


Fig. 11. INVITE transaction dependability versus normalized session arrival rate.

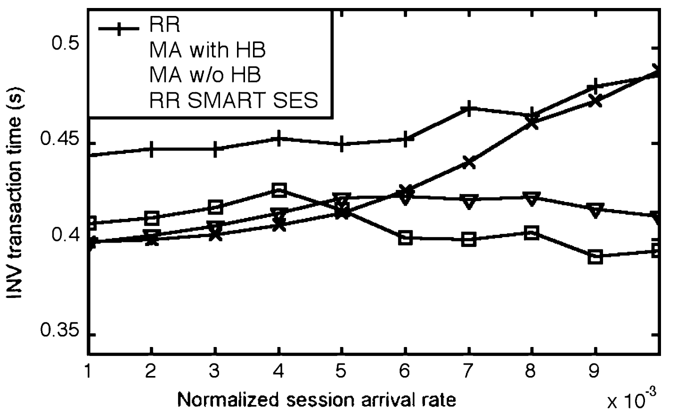


Fig. 12. INVITE transaction time versus normalized session arrival rate.

regularly done every 30 s. One can note the following results by observing Figs. 11 and 12.

1) We define the *pool capacity* as the value of the normalized session arrival rate up to which the transaction dependability is not dropping more than 0.05% below the maximum. As shown in Fig. 11, the *pool capacity* is roughly about 0.005 or 0.006 depending on the server selection policy. Further, transaction dependability goes smoothly down.
2) MA with and without HB gives the best transaction dependability (see Fig. 11). SRR-S performs better than RR regarding the INVITE transaction dependability. RR provides worst transaction dependability among the four schemes because it always sends the requests in a cyclic fashion regardless of the server's processing performance and the current availability server status. For higher session arrival rates, the probability of overloading the slower server becomes considerable. Despite its generally higher availability, the slower server easily gets congested and a certain percentage of the incoming SIP messages cannot be queued and must be dropped. Based on monitoring the up-to-date status (by HB requests as well as SIP messages) MA possesses the inherent capability to choose the instantaneously most available server from within the server set. Higher session arrival rates cause more frequent failures of the slower server, and thus, effectively decrease

its overall instantaneous availability. This information in turn dynamically maps into the clients' status vectors and it implies selecting the instantaneously more available server. This suggests that the server processing speed has an inevitable impact on the equivalent server instantaneous availability. Dynamic monitoring of the servers' status enables clients to make the proper selection decision as to increasing the instantaneous transaction dependability.

3) The same conclusions apply to SRR-S (see Fig. 11). Consequently, in some fractions of time the slower server is selected, which may not be quite an appropriate decision particularly in heavy traffic conditions. The effect is worse transaction dependability as well as worse performance.
4) Analyzing the transaction control time reveals one particularly interesting and important result (see Fig. 12). Surprisingly, MA without HB support generally shows the best performance. The critical observation is that MA without HB support performs even better than MA supported by a HB mechanism! This outcome can be explained by the following reasoning. For lower to medium session arrival rates, both servers are not overloaded, and these conditions imply that MA ensures relatively equal load distribution amongst the two servers. Further load growth gets the slow server overloaded more frequently than the fast one. Directing the transactions over the both servers according to their up status, as suggested by the regular HB health check, degrades the transaction times because the slower server is more often forced to drop incoming messages.

When MA is deployed without HB support, the servers' status is discovered solely by the SIP messages. Thus, once a single transaction is successful with the slow server, the MA algorithm is forwarding the next transactions to this server. Since the traffic load is rather heavy, the slow server rapidly goes into saturation and causes transaction failures. Consequently, the client is more often suggested to direct transactions to the fast server. Thus, the fast server proceeds the servicing until it fails. This leads us to the conclusion that the MA algorithm has a naturally embedded ability to detect the faster servers. In fact, the HB mechanism does not help the MA SSP discover the faster server. Indeed, the HB requests do not indicate the server speed instead they rather collect the availability status of the servers, which does not always tell about the server processing power. For this reason, deploying the HB mechanism for medium to heavy traffic loads deteriorates performance because a large fraction of transactions are directed to the slow server thereby increasing the transaction times. The misleading information provided by the HB health checks causes even more serious overloaded conditions at the slow server resulting in noticeably longer transaction times. There are certain maximums in the transaction time as seen in Fig. 12.

These results suggest that a HB mechanism is recommended to couple the MA SSP only for light traffic loads in order to minimize the transaction times. For medium to heavy traffic load conditions, the HB mechanism is not desired because it indirectly deteriorates performance. Thus, the HB mechanism is proven to be a helpful technique only with light traffic conditions in the network. Note that these conclusions are valid for the specific model where HB requests do not get into server's queue and are rather processed immediately by a separate software entity.

VIII. Conclusion

A novel dynamic SSP referred to as MA SSP is proposed along with a simple RSerPool protocol extension. MA is applicable in a broad range of IP-based systems and services, even though in this paper it was only considered in session control systems. The proposed MA SSP is a dynamic and adaptive algorithm that aims at maximizing the probability of successful transaction with the current transmission. Decisions are made using a simple status vector maintained at a client. MA has a low implementation complexity; a client should only keep a status vector with as many elements as servers in the server set.

An analytic model was developed and expressions of the selected evaluation metrics were derived. Also, based on the system model assumptions, event-driven simulation programs were developed. By comparing the results obtained from the expressions and simulations, the analytic model was validated. Numerical results show that MA significantly outperforms RR. Moreover, for the considered system model, MA without a heart beat mechanism support gives the best service dependability and performance in nonhomogenous server sets.

In order to integrate MA into RSerPool, a simple RSerPool extension was proposed, where a name resolution response is added an extra field containing the status vector. Nevertheless, MA is not restricted to RSerPool, and it may be used with any other fault-tolerant platform such as clusters.

In the future work, dynamic SSPs based on other metrics will be devised. For instance, in order to further enhance performance, application response time (or the round trip time) from each server can be measured to assess each server's processing power. In this paper uptime and downtime were assumed exponentially distributed. It would be interesting to also consider other failure/repair models as well as other traffic models. The performance of MA may differ for distributions other than exponential. The current analytic model was derived only for MA. In the future studies, we aim at extending it to other SSPs with or without HB support.

Comparison of the SSP to a standard setup with more realistic parameter settings (e.g., SCTP HB mechanism with appropriate parameters) for telephony signaling will be performed in the future work.

Finally, assuming synchronized time (between PU and ENRP servers) may be an unreasonable constraint. The impact of varying clock skew on the SSP, and the possible mechanisms that could alleviate the clock skew breakage will be investigated and evaluated in the future studies.

Acknowledgment

The authors would like to thank R. Seidl, Siemens, Munich, for his valuable comments. This work was carried out within the joint Siemens-CTIF research project.

References

[1] *IP Multimedia (IM) Subsystem-Stage 2*, 3GPP TS 23.228, Sep. 2003, Tech. Specification.
[2] J. Rosenberg, "SIP: Session initiation protocol," RFC 3261, Jun. 2002.

[3] M. Tuexen, "Architecture for reliable server pooling," Oct. 12, 2003, draft-ietf-rserpool-arch-07.txt.
[4] *"Resilient Telco platform, V2.0 for Linux and Solaris: RTP Overview and Programmer's Guide,"* Fujitsu Siemens Computers, 2002.
[5] M. Bozinovski, H.-P. Schwefel, and R. Prasad, "Maximum availability server selection policy for session control systems based on 3GPP SIP," in *Proc. 7th Int. Symp. Wireless Personal Multimedia Commun. (WPMC'04)*, 2004, pp. 276–281.
[6] R. Stewart, "Aggregate server access protocol (ASAP)," 2004, draft-ietf-rserpool-asap-09.txt.
[7] M. E. Crovella and R. L. Carter, "Dynamic server selection in the Internet," in *Proc. 3rd IEEE Workshop Arch. Implementation High Performance Commun. Subsyst. (HPCS)*, 1995, pp. 158–162.
[8] M. Sayal, "Selection algorithms for replicated web servers," in *Proc. Performance Evaluation Rev.—Workshop Internet Server Performance*, 1998, pp. 44–50.
[9] K. Obraczka and F. Silvia, "Network latency metrics for server proximity," in *Proc. IEEE Globecom*, 2000, pp. 421–427.
[10] G. Camarillo, R. Kantola, and H. Schulzrinne, "Evaluation of transport protocols for the session initiation protocol," *IEEE Network*, vol. 17, no. 5, pp. 40–46, Sep. 2003.
[11] M. Bozinovski, L. Gavrilovska, and R. Prasad, "Fault-tolerant SIP-based call control system," *Electron. Lett.*, vol. 39, no. 2, pp. 254–256, Jan. 2003.
[12] A. Helal, A. Heddaya, and B. Bhargava*, Replication Techniques in Distributed Systems*. Norwell, MA: Kluwer, 1996.
[13] H. Schioeler, "Towards reliable integrated services for dependable systems," presented at the Workshop on Real Time LANs Internet Age (RTLIA), Porto, Portugal, 2003.
[14] C. G. Cassandras and S. Lafortune*, Introduction to Discrete Event Systems*. Norwell, MA: Kluwer, 1999.
[15] M. Mauve, "Consistency in replicated continuous interactive media," in *Proc. ACM CSCW*, 2000, pp. 181–190.
[16] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.
[17] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle, "Session initiation protocol extension for instant messaging," RFC 3428, Dec. 2002.

Marjan Bozinovski was born in Skopje, Macedonia, in 1974. He received the Dipl. Engineer degree in electronics and telecommunications and the M.Sc. degree in telecommunications from the University of Skopje, Macedonia, in 1997 and 2000, respectively, and the Ph.D. degree in wireless communications from Aalborg University, Aalborg, Denmark, in 2004.

His current position is Senior Specialist at the Research and Development Department, AD Makedonski Telekomunikacii, Macedonia, the largest telecom operator in Macedonia. He worked as a Research Assistant at Aalborg University, where he became an Assistant Professor. He worked on a Siemens-funded research project that resulted in generating two patent applications. He is author of a number of peer-reviewed journal papers and international conference papers. His research interests include the area of dependability/fault-tolerant platforms in next generation network, optimization techniques in stateful wireless/wired IP-based systems and services, and dynamic server selection policies in IP-based server systems.

Hans-Peter Schwefel received the Doctoral degree in IP traffic and performance modeling from the Technical University in Munich, Germany, in 2000.

He is an Associate Professor and head of the IP Labs at Aalborg University, Aalborg, Denmark. His research focuses on IP-based wireless networks with a main interest in performance, security, and reliability aspects. Before he joined Aalborg University, he was a project manager at Siemens Information and Communication Mobile, supervising research projects and responsible for the development of technical concepts for next generation mobile networks. For his research activities, he also spent extended periods of time at the University of Connecticut, Storrs, and at AT&T Labs, Middletown, NJ. He is coordinating multiple research projects, among them the Targeted Research Project "Highly Dependable IP-based Networks and Services (HIDENETS)" (http://www.hidenets.aau.dk), contributing to the European Commission Sixth Framework Program.

Ramjee Prasad (SM'90) was born in Babhnaur (Gaya), Bihar, India, on July 1, 1946. He is now a Dutch Citizen. He received the B.Sc. (Eng.) degree from Bihar Institute of Technology, Sindri, India, in 1968, and the M.Sc. (Eng.) and Ph.D. degrees from Birla Institute of Technology (BIT), Ranchi, India, 1970 and 1979, respectively.

Since 1999, he has been with Aalborg University, Aalborg, Denmark, where he is currently Director of the Center for Teleinfrastruktur (CTIF), and holds the chair of wireless information and multimedia communications. He is a coordinator of the European Commission Sixth Framework Integrated Project My personal Adaptive Global NET (MAGNET). He was involved in the European ACTS project Future Radio Wideband Multiple Access Systems (FRAMES) as a DUT project leader. He is a project leader of several international, industrially funded projects. He has published over 500 technical papers, contributed to several books, and has authored, coauthored, or edited 16 books. He has served as a member of advisory and program committees of several IEEE international conferences. In addition, he is the coordinating Editor and Editor-In-Chief of the Springer *International Journal on Wireless Personal Communications* and a member of the editorial board of other international journals. He is also the founding chairman of the European Center of Excellence in Telecommunications, known as HERMES, and he is now Honorary Chair.

Dr. Prasad has received several international awards; the latest being the Telenor Nordic 2005 Research Prize (website: http://www.telenor.no/om/). He is a Fellow of the Institution of Electrical Engineers, a Fellow of IETE, a member of The Netherlands Electronics and Radio Society (NERG), and a member of IDA (Engineering Society in Denmark). He is an advisor to several multinational companies.