Copyright Notice

Institute of Communication Networks and Computer Engineering
University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany
Phone: ++49-711-685-68026, Fax: ++49-711-685-67983
email: mail@ikr.uni-stuttgart.de, http://www.ikr.uni-stuttgart.de

Load Sharing in a Distributed IMS Architecture

Jochen Kögel*, Stefan Wahl†, Michael Scharf* and Marc C. Necker*
*Institute of Communication Networks and Computer Engineering (IKR), Universität Stuttgart, Germany
Email: {jochen.koegel, michael.scharf, marc.necker}@ikr.uni-stutgart.de
†Bell-Labs Germany, Alcatel-Lucent Deutschland AG
Email: stefan.wahl@alcatel-lucent.de

***Abstract*—The IP Multimedia Subsystem (IMS) serves as universal platform for fast and standardized creation of mobile services. Typical deployment models for the IMS favor centralized session control and application servers. Furthermore, they rely on sophisticated border elements for, e.g., policy enforcement. One step towards a simpler architecture is the integration of call control, application server, and media functions into border elements. This leads to a distributed IMS architecture with equal processing nodes, which drastically reduces system complexity and scales on functional module instantiation basis.**

However, the distributed nature of the architecture reduces the statistical gain. In this work, we study a suitable load sharing concept that counters or even over-compensates this problem. In principle, this mechanism realizes one large virtual central server with small control and message overhead. Our simulative evaluation shows the feasibility and performance of our approach.

I. Introduction

Public voice telephony is more and more realized by Next Generation Networks (NGN) that are based on Internet Protocol (IP) technology. The 3GPP IP Multimedia Subsystem (IMS) and its extension ETSI TISPAN are NGN frameworks that standardize converged session control functions for end-to-end multimedia conversational services over an all-IP network. These NGN frameworks are expected not only to replace the existing PSTN, but also to provide a platform for rapid service creation. IMS defines functions for session control, application servers, and media transport, which can be implemented either in a centralized or a distributed architecture. In most deployment scenarios, Session Border Controllers (SBC) are placed at the domain boundaries. Traditional SBC elements only perform a certain part of the session control functions. The IP Multimedia eXchange (IMX) concept [1] proposes an alternative architecture that integrates call control, application server, and media functions into the border elements and thereby reduces the number of central components. The resulting platform consists of equal general-purpose nodes that realize the IMS functions at the edge. This integration not only simplifies the platform management and operation, but it can also reduce the signaling complexity inside the system.

However, such a distributed IMS architecture comes at some cost: Some of the distributed nodes can get overloaded because of unpredicted traffic patterns, flash crowd effects, value-added services such as televoting, or platform failures. As a result, calls may have to be dropped even though other nodes in the platform still have significant spare capacity. In theory, such overload could be avoided by dimensioning each element for the worst-case scenario, but this is not a cost effective solution. It would result in a much larger total amount of resources compared to a centralized design that benefits from statistical multiplexing gains. This is why a distributed IMS architecture essentially needs load sharing mechanisms that can distribute processing efforts among the different nodes.

Load sharing is not a new concept and is already widely used with PSTN equipment (e.g., [2]) and large-scale server clusters [3]. Local load balancing schemes for IMS servers have also been studied [4], and there are ongoing standardization activities concerning overload signaling [5]. However, the realization in a highly distributed IMS system imposes a number of unique constraints. Therefore, this paper systematically analyzes load balancing concepts for such environments. We introduce different realization alternatives and discuss design issues. Furthermore, we develop a scalable load balancing architecture and develop a lightweight relocation strategy for the IMX platform. The benefits of our solution are illustrated by an analysis and selected simulation results.

The rest of this paper is structured as follows: Section II introduces the IMX concept and derives the requirements for load balancing. In Section III, we discuss the degrees of freedom, and differences of IMX to server clusters. Section IV then presents our load balancing architecture for a distributed IMS realization. A numerical analysis of our concept and the results of simulation studies can be found in Section V. Finally, Section VI concludes this paper.

II. IMX: Distributed IMS of equal nodes

The mapping of IMS functions to components is not standardized and offers a variety of possibilities, which are discussed here together with resulting constraints for load sharing.

A. Degrees of freedom for IMS deployment

The IMS architecture defines different Call Session Control Functions (CSCF), which handle signaling messages and call state, and Media Resource Functions (MRF). The service logic is realized in Application Servers (AS). Within the IMS/TISPAN architecture, the Proxy-CSCF (P-CSCF) function is located at the border between access network and core network in order to perform security functions and registration handling. In many deployment scenarios, P-CSCF functions are integrated into Session Border Controllers (SBC), which also integrate e.g. policy decision functions (PDF) on the media path. Similarly, Interrogating-CSCF (I-CSCF) placed at the border to other networks may be realized by SBCs, too.

Extending the idea of SBCs, a further reasonable integration step is to enhance the border element towards fully integrated IMS nodes, so that they implement also other IMS call control

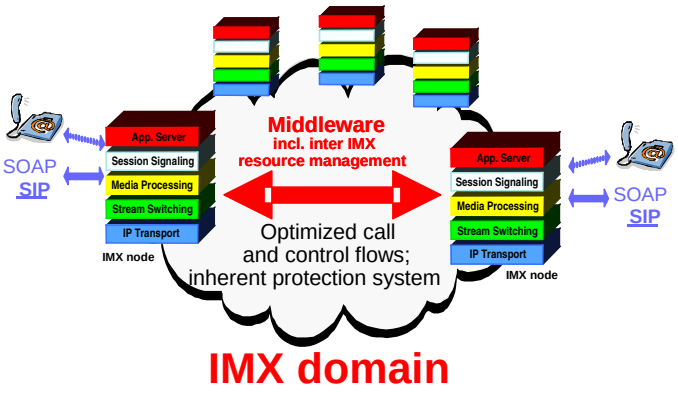

Fig. 1. Illustration of the IMX concept

functions, including I-CSCF and Serving CSCF (S-CSCF), basic application server (AS), and media capabilities [1]. This IMX concept substitutes the central servers that handle requests of all clients by several identical nodes that each processes the requests [6]. As shown in Figure 1, an IMS platform then consists of several distributed IMX nodes.

One advantage of IMX is that a significant portion of the SIP signaling between P-, I- and S-CSCF and Application Server (AS) is intra-cluster communication and can thus be realized by an optimized communication middleware that also handles transfer and access to session and user information (e.g. call state or user policies in the Home Subscriber Server). To the outside world the cluster offers standard interfaces like any other IMS implementation. This enables mixed deployment scenarios and smooth migration paths, where one part of a platform is realized as IMX cluster and the other part by IMS components. Another feature is that the IMX nodes can make use of advanced application server technology, i.e., the different functions can be realized as modular software components. Each registered subscriber corresponds to a certain number of instantiated components on an IMX node. By separating state from function instantiations and virtualization techniques, even a live migration of the components among different nodes might be possible.

B. Benefits of load sharing

Typical deployment models for NGN platforms favor a small number of server locations with several tightly coupled servers in clusters. Load balancing mechanisms distribute incoming calls across the servers of one site to achieve equal load distribution and high availability [2], [7]. In such scenarios, load originating from several access networks concentrates on one server cluster leading to statistical gain. Thus, the server cluster can handle load peaks from single access networks appropriately.

In the IMX, processing resources are not centralized but distributed across a high number of border nodes. If each IMX node handles only calls of directly connected access networks, statistical gain will be significantly reduced. In order to avoid expensive overprovisioning, load sharing mechanisms that move processing load from highly loaded to underutilized nodes are essential. Additionally, load sharing across the complete platform even leads to better statistical gain than a centralized solution with several clusters. However, load sharing mechanisms of typical server clusters differ from mechanisms required for IMX: In the former case, the goal of equal load distribution is often realized by a central dispatcher that associates clients with servers. In IMX, the goal is not equal load distribution but reacting before overload situations occur. Additionally, in IMX the offloading IMX nodes still stays in the path between client and processing IMX node and keep on handling registrations. Thus, established solutions with central dispatchers cannot be used.

III. Load Sharing Concepts

This section introduces the design space for load sharing and shows the differences between IMX and server clusters.

A. Design space

Load sharing mechanisms that distribute work according to the current load situation require mechanisms for load information gathering as basis for resource allocation (see Figure 2). The two basic steps of load information gathering and resource allocation can be further subdivided in a measurement and update policy as well as a transfer, location and redirection policy (see also [2]). The measurement policy defines how often the system performs load measurements, their granularity and aggregation strategy of values (mean, peak,...) resulting in a node-local load view. How to distribute the load information is determined by the update policy that defines mechanisms like an update protocol. Based on the global load view, the transfer policy specifies when to change a task assignment pattern or to relocate tasks. When relocation or reassignment is triggered, the location policy determines how load is shifted between nodes and the redirection policy determines how much load (e.g. how many tasks or calls) are relocated.

All previously mentioned steps of load sharing offer several degrees of freedom in their realization and form the design space for load sharing. At the two stages where inter-node communication is required for updating state, namely obtaining the global load view and assigning workload (sessions) to nodes, there are three design choices. It is beyond the scope of this paper to discuss the design space in detail, however, we give a basic classification illustrating our choice for IMX.

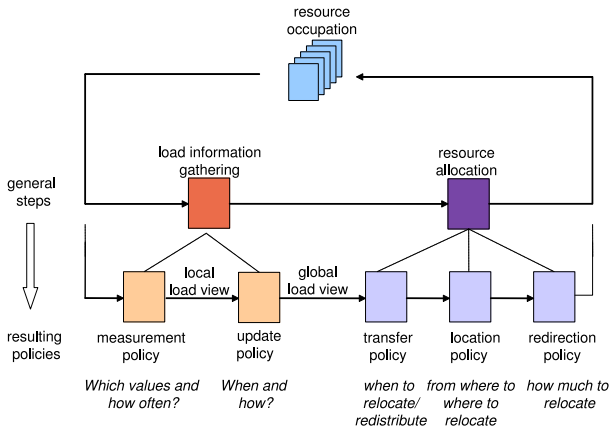

Fig. 2. General scheme for load sharing and derived policies

- **Trigger: value or time based**
  Updates can be triggered periodically (time based) or only if values exceed certain thresholds or trend changes. Value based mechanisms reduce the communication effort, but reduce the possibilities for achieving fine-grained view or control.
- **Triggering entity: push or pull model**
  Either the source or the sink of information (or workload, respectively) can trigger updates.
- **Architecture: centralized or distributed**
  The centralized model forms a star topology where all traffic traverses a dedicated central node. Contrary, the distributed model results in a full mesh with more messages but similar traffic at every node.

As we will show in Section IV, we chose for IMX an update mechanisms that is centralized, time based, and pull model based. Concerning work assignment, we chose a distributed mechanism that is value and push model based.

B. Differences to server clusters

The typical setup for web server clusters is a high number of servers hidden behind a load balancer [3]. In [2] such concepts are transferred to centralized call server clusters, however with static distribution schemes and a small number of call servers only. The IMX scenario is different from web and call server scenarios, since IMX nodes will always stay in the path for processing or policing tasks. A problem closer to the IMX scenario is presented in [8], where web servers forward new connections to other servers by rewriting addresses directly in the network stack. However, the resource allocation mechanism statically forwards a fix rate of requests, which is inappropriate for IMX where relocation is more costly. In summary, there is no approach close to the IMX scenario, which is why we developed a suitable concept.

IV. Load Sharing in a Distributed IMS Platform

This section covers function relocation support, resource management, and the load sharing approach developed on top.

A. Platform-supported function relocation

As introduced in Section II-A, the IMX architecture allows dynamically creating and assigning instantiations of functional components , while state information is kept in objects separated from the actual instances. These instantiations are managed by a Local Resource Manager (LRM) belonging to the IMX node. When created, these instances mainly allocate memory and no CPU resources, since they are not yet assigned to users and are thus inactive and stateless.

Figure 3 illustrates how IMX handles user registrations and sessions. As shown at IMX1, the functionality of an IMX node consists of a relocatable portion of stateless function instances and non-relocatable functions that are transport related (e.g. forwarding, policing, redirections). The latter have always to be performed at the ingress node (iNode). With the registration of a user, the dispatcher on the IMX node takes a User Control Manager (UCM) instance and assigns it semi-statically to this user. This results in registration state on the iNode, as indicated

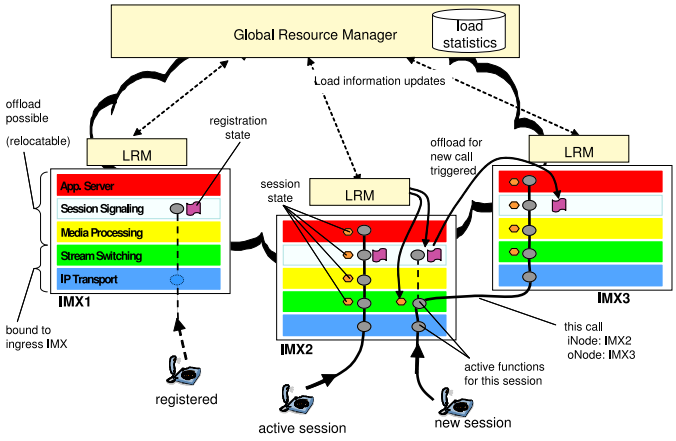

Fig. 3. Session offloading by function relocation with IMX

in Figure 3. Depending on the user's activities, the UCM takes appropriate instantiations of functional components from the LRM and controls the appropriate functional chaining as well as the session state of these components. If a new session is initiated and resources on the iNode are scarce, the UCM will not get functional instances from the LRM, but the LRM will trigger function relocation. In this case, the UCM will associate instances on a remote node with the user and transfer the required state information, while traffic is redirected accordingly. The iNode will remain the contact point for this user and continues to handle its registration. The relocation principle is also shown in Figure 3, where a new session arrives at IMX2 and is relocated to IMX3, while the registration will still be handled at IMX2. Functions of existing sessions will not be relocated. We will label IMX nodes to which functions are relocated as oNode in the context of a session.

B. Load information gathering and update

In our load sharing concept, each IMX node takes relocation decisions on its own, i.e., the resource allocation decisions are performed in a distributed fashion and value based. This enables fast reaction in situations of high load and timely function relocation when the call arrives without the necessity to query a central instance for relocation decisions. We describe the resource allocation mechanism in Section IV-C.

In contrast to the distributed resource allocation mechanism, we chose a centralized load update strategy, which is designed for load update periods of few seconds enabling suitable reaction times. With the Global Resource Manager (GRM) as central hub for load information, a star topology is realized that reduces the amount of load update messages the IMX nodes have to process. In terms of trigger we decided to use periodic time-based updates, which can deliver timely information. Since IMX platforms will base on high-bandwidth core networks, there is no point in saving message transfers between IMX nodes. Concerning the triggering instance we chose a pull model based on a three-way handshake. An advantage of the pull model is the fact that load information updates are faster and provide fresh information at the same time to all nodes.

Figure 4 shows the IMX update strategy. The GRM triggers the update by sending an `updateRequest` Message to the

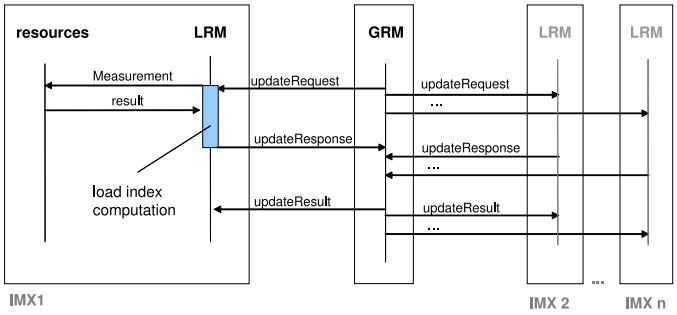

Fig. 4. Measurement and update strategy

LRMs of all IMX nodes at once. On its arrival, IMX nodes measure the mean resource occupation and translate it into a load index as detailed in the next section. Each node sends an `updateResponse` containing its load index to the GRM. Once the GRM received all `updateResponse` messages, it sends an `updateResult` message containing load information about all nodes to every IMX node. With this mechanism, all results will arrive at the GRM within a short time slot and can directly be included in one `updateResult`.

We can quantify the additional effort caused by the update strategy based on the message and data rate. For each of the n IMX nodes, three messages are sent per node in one update interval u leading to the message rate

$$m = \frac{3n}{u} \tag{1}$$

at the GRM (linear with n). With a packet size of L per `updateRequest` or `updateResponse` and `updateResult` size being $n \cdot L$, the update protocol consumes at the GRM the overall data rate at of

$$r = \frac{2n \cdot L + n \cdot n \cdot L}{u} = \frac{L}{u}(2n + n^2) \tag{2}$$

(quadratic with n). For reasonable numbers, like $u = 1$ s, $L = 50\ Bytes$ and $n = 50$, the data rate is only $1.04\ Mbps$ with $150\ messages\ per\ second$. Message and data rates for the update protocols at the IMX nodes are almost negligible: Three messages per second and a data rate of $20\ kbps$ are needed, which is very small compared to e.g. SIP signaling alone (about $120\ kbps$ at a call arrival rate of $5\ per\ second$ and ten 300-Byte SIP messages per call).

C. Resource Allocation

Since sessions cannot be completely offloaded, relocation is always costly and all sessions should be handled in the iNode as long as possible. Thus, early call relocation for achieving an equal load distribution would be counterproductive. Furthermore, suitable dimensioning of IMX platforms should assure that nodes are equipped with resources according to the size of the access network they serve. Nodes will therefore offload sessions only in case of high load. The definition of the load index reflects these conditions and requirements.

The load index is calculated based on the load level $\rho \in [0..1]$, which is the ratio of mean resource usage to available capacity in the previous measurement interval. ρ only describes the occupancy of resources assigned to per-call IMX functions. Thus, at $\rho = 0.0$ basic functions and background tasks are running, registrations of idle clients are handled, but no sessions are active. Contrary, at $\rho = 1.0$ no further incoming sessions are handled, but enough resources for resource management and overload control are available.

Our relocation strategy subdivides the load index values in three ranges that describe the node's behavior concerning load sharing and is defined by the parameters φ and ω as follows:

- `accepting`: $0 \leq \rho < \varphi$
- `passive`: $\varphi \leq \rho < \omega$
- `offloading`: $\omega \leq \rho \leq 1,0$

In our load sharing approach, each IMX node decides on its own how to relocate calls based on its current load index. Thus, according to Section III-A, our approach can be classified as distributed, value- and source-triggered. Concerning the transfer policy, IMX nodes offload new sessions as soon as they are in the state `offloading`. Our location policy specifies that sessions are offloaded to nodes in `accepting` state. The offloading node selects in a round-robin fashion `accepting` nodes from the list given in the `updateResult` message. As soon as a node starts offloading, it will relocate all new calls (redirection policy). If no node in `accepting` state can be found, a node in offloading state will try to handle new calls locally.

The load sharing mechanism adds only small effort on the IMX nodes, since besides measurement and update, no extensive processing is required. Additionally, the parameters φ and ω can be pre-calculated taking the load characteristics and system parameters into account, as we show in Section V-B. Switching from `offloading` immediately to the `accepting` state in cases of varying load can be prevented by the intermediate `passive` state. The size of the `passive` range therefore adds a hysteresis to the system. Increasing the passive range decreases the number of nodes in accepting state and therefore limits the overall system capacity. How to set the hysteresis in this trade-off depends heavily on the system implementation and its sensitivity to oscillations. Therefore, we will not consider the passive state (i.e., $\varphi = \omega$) in our evaluation.

V. Quantification of Load Sharing Benefit

Here, we consider a two node scenario for understanding the basic behavior of our load sharing mechanism. We first calculate the optimal threshold setting for the static case, then we evaluate the system by simulation with stochastic load.

A. System model

We focus on the fundamental properties of an IMX system and therefore use a system model that abstracts from the implementation and deployment specific details. The model covers the architecture, resource behavior and the load imposed on the system in the form of a traffic model.

We consider a platform with several access networks, each connected to an IMX node residing at the platform border. The IMX nodes are interconnected by a high-speed network. Thus, network effects on load sharing mechanisms are negligible and not part of our model. We assume that IMX nodes feature

TABLE I
MODEL PARAMETERS

Symbol	Description	Default value
A	normalized system load	none, input parameter
h	mean call holding time	$180s$
λ_i	mean call setup rate	$\frac{A_i}{h}$
c	resource capacity of an IMX node	$1,000,000$
e	resource consumption at IMX node where call is handled	$1,000$
f	resource consumption per call at iNode if call is relocated	50
b	relocation overhead	1.05 (from e and f)
x	load asymmetry	0.5

suitable overload control and drop new requests when the capacity is exceeded, leading to a blocking behavior. The nodes' load level is calculated based on integer values for resource capacity and consumption. In this work, we only consider one type of processing resources. Differentiating between resource classes (e.g. memory, specialized processors) would lead to a more complex optimization problem for placing functions.

For the simulation study, we assume the holding time of sessions to be exponentially distributed with the constant mean h. The arrival rate λ_i is assumed as exponentially distributed and its mean value is set depending on the load originating from the access network connected to node i. Both assumptions are reasonable for conversational traffic. This leads to the offered traffic for node i: $A_i = \lambda_i \cdot h$ and the normalized load for n nodes $A = \frac{1}{n}\sum_{i=1}^{n} A_i$. In order to study the load sharing mechanisms, we impose load asymmetry described by the parameter x. For simplicity, we suppose that half of the access networks impose the high load

$$A_1 = (1+x)A, \tag{3}$$

the other half the low load

$$A_2 = (1-x)A. \tag{4}$$

The resources consumed by one session are described by the effort e required to handle the call both locally or relocated and the additional relocation effort f. The latter represents the SIP-parsing, encryption and decryption as well as transport effort that is required in iNode in case of function relocation. The relocation overhead ratio is defined as

$$b = 1 + \frac{f}{e}. \tag{5}$$

Resource consumption for active sessions is assumed to be constant. Typically, there are several hundred active sessions in parallel, which will average out load peaks in resource usage. In our model, calls affect only one IMX node, which is e.g. valid for value-added service calls. End-to-end call between two clients corresponds to two service calls. Table I summarizes the model parameters and gives exemplary default values.

B. Considerations for the ideal and static case

This estimation derives a guideline for the load sharing parameter ω by considering a scenario with two IMX nodes, a load asymmetry x, and relocation overhead ratio b.

As discussed, we omit the "passive" range ($\varphi = \omega$), but extending the calculations for also taking φ into account would

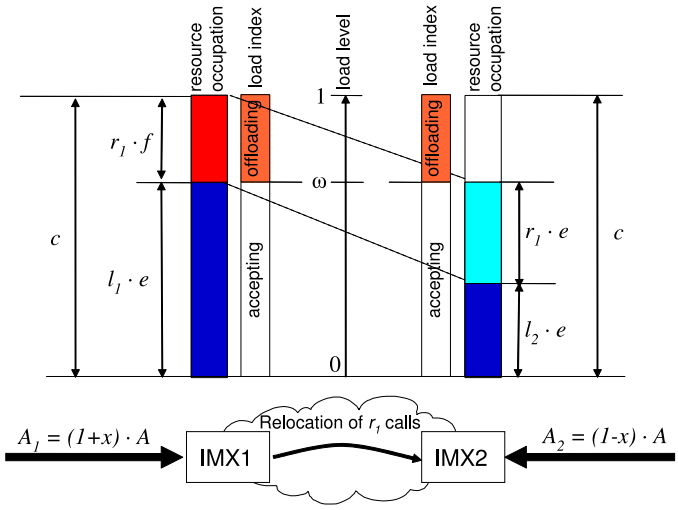

Fig. 5. Scenario for estimating the load sharing parameter ω

be straight-forward. The same value for ω is used for all nodes. For estimating an optimal value for ω, we consider the static case (infinite call holding time). Figure 5 shows two nodes, where IMX1 receives more traffic (A_1) than IMX2 (A_2). l_i calls are handled locally and r_i calls are relocated, with

$$A_i = l_i + r_i. \tag{6}$$

Figure 5 shows resource occupation in the ideal case: All resources of IMX1 are occupied when IMX2 reaches the `offloading` state. IMX1 handles l_1 calls locally and forwards r_1 calls to IMX2, while IMX2 handles all l_2 calls locally. Setting ω higher would lead to a sub-optimal case, since IMX1 starts relocation too late and will block while IMX 2 is still in accepting state. Contrary, a lower ω would cause IMX2 to leave the "accepting" stage too early and in the end IMX1 would have to handle calls locally while being in state `offloading`. Even in the ideal case, there are free resources at IMX2 when IMX1 blocks, which is an intended property that avoids overloading remote nodes by relocating calls to them up to their limit. From Figure 5, we can derive the following relations: Full utilization in IMX1 (7), utilization in IMX2 (8), and (9) from comparing occupation of IMX1 and IMX2.

$$c = l_1 \cdot e + r_1 \cdot f \tag{7}$$

$$\omega \cdot c = l_2 \cdot e + l_1 \cdot e \tag{8}$$

$$l_1 = l_2 + r_1 \tag{9}$$

Equations (6), (7), (8), and (9) yield together with the definitions of x (2,3), and b (5), the optimal setting as

$$\omega = \frac{1}{1 - x + bx}. \tag{10}$$

Considering the definitions of x and b, ω is limited to values in $[0..1]$. The exemplary values of Table I yield $\omega = 0.975$.

C. Simulation environment

For gaining performance insights into the load sharing approach under stochastic traffic load, we performed simulations of the IMX system using an event-driven simulation library

[9] with signaling extensions. The simulation environment allows implementing call setups and load updates directly as messages flows between nodes (e.g. between IMX and GRM). Access networks are modeled as nodes containing call message generators that generate call setup messages according to a predefined IAT distribution and add the call duration as message parameter.

On arrival of call setup messages, IMX nodes check whether they have enough resources left for the required call processing. If it is possible to handle the call completely locally, or only the non-relocatable portion, they register the resource utilization in their load schedule. Otherwise they drop the call. The simulation keeps per-node and global statistics on drop events. Depending on the current load index and the availability of nodes in `accepting` state, IMX nodes forward call messages originating from directly attached generator nodes to other nodes. If the outgoing node blocks despite the load view of the ingress node was `accepting`, the resource consumption for the non-relocatable part is removed. LRM functions exchange load update messages with the GRM according to the load update protocol. For each parameter set, we ran 10 simulation batches with 1 or 10 Million call setups each.

D. Simulation results

We show the influence of the load sharing parameter ω on the overall blocking probability for different load values and constant asymmetry ($x = 0.5$). Figure 6 depicts the blocking probability for different settings of the load sharing parameter ω and for varying normalized load. For understanding the basic effects, we consider here a two node scenario. There is a minimum blocking probability for a certain load sharing parameter. This minimum drifts to higher values of ω when load is increased. As a result, there is no general optimal setting for ω.

If ω is set too high, IMX1 starts relocation too late and has too few resources left for handling the non-relocatable part, while IMX2 would still be capable of handling offloaded calls. A slight increase of ω can already correspond to a considerable amount of resources not available for handling the small non-relocatable part. If ω is set too low, IMX2 leaves the `accepting` state earlier. Since e is much higher than f, slight changes of ω only affect a smaller number of calls, thus a setting ω too low does not cause such dramatic degradation than setting ω too high. Thus, we can derive the guideline that rather a too low parameter value should be chosen. Additionally, the estimation for the ideal and static case can serve as a good starting point, while the maximum tolerable blocking probability and expected traffic load must be known for fine tuning. Simulations with more IMX nodes delivered similar results with certain statistical gain.

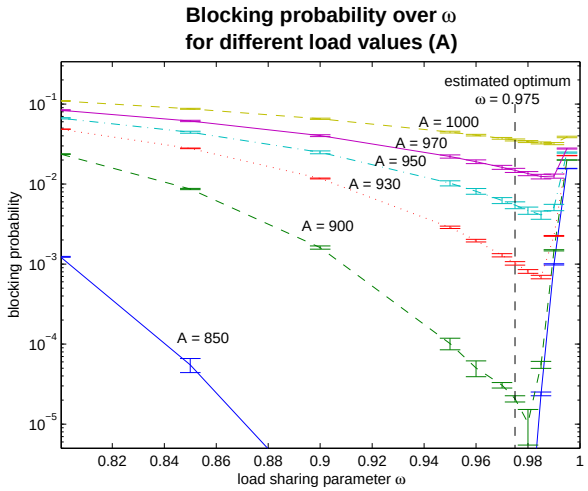

Fig. 6. Simulation results. The estimated optimum results from the calculations for the static case in Section V-B

VI. Conclusions

Distributed IMS approaches can integrate all functions into edge and border nodes. They offer a great flexibility and scalability while simplifying deployment. However, load sharing mechanisms are essential for shifting load between nodes for handling sudden load changes. In this paper we presented a load sharing mechanism for a distributed IMS. We analyzed the design space and identified possible solutions. Based on this, we develop a strategy that fits best the requirements of a distributed IMS realization. Our load sharing mechanism is light-weight and imposes minimal overhead on the overall system. The mechanism can be tuned using our formula for estimating the main load sharing parameter, which takes the relocation overhead and load asymmetry as input. Simulation results show the fundamental characteristics of our algorithm and illustrate the benefit of load sharing. Our results can serve as basis for further implementation-specific refinements of the load sharing concept and general dimensioning guidelines. Furthermore, extensions towards systems that calculate load sharing settings from observed behavior are possible.

Acknowledgment

Special thanks to Christian Müller, who developed the signaling simulation extensions. This research is partially funded as part of the ScaleNet project by the German Federal Ministry of Education and Research (BMBF).

References

[1] S. Wahl, K. Oberle, M. Kessler, P. Domschitz, *The next Step in IMS Architecture – a Comprehensive IMS Network Element*, Proc. Broadband Europe 2006, Geneva, Switzerland, Dec. 2006.

[2] M. Asif, S. Majumdar, G. Kopec, *Load sharing in Call Server clusters*, Computer Comm., vol. 30, nr. 16, pp. 3027-3045, 2007.

[3] V. Cardellini, E. Casalicchio, M. Colajanni, P. S. Yu, *The state of the art in locally distributed Web-server systems*, ACM Computer Surveys, vol. 34, nr. 2, pp. 263-311, 2002.

[4] T. Bessis, *Improving performance and reliability of an IMS network by co-locating IMS servers*, Bell Labs Technical Journal, vol. 10, nr. 4, pp. 167-178, 2006.

[5] V. Hilt, I. Widjaja, D. Malas, H. Schultzrinne, *Session Initiation Protocol (SIP) Overload Control*, IETF draft-hilt-sipping-overload-05, Jul. 2008.

[6] S. Wahl, K. Oberle, M. Stein, J. Riemer, C. Peña, *An implementation of application layer based continuous mobility for conversational services*, Proc. of Broadband Europe 2007, Antwerp, Belgium, Dec. 2007.

[7] K. Singh, H. Schulzrinne, *Failover, load sharing and server architecture in SIP telephony*, Computer Comm., vol. 30, nr. 5, pp. 927-942, 2007.

[8] A. Bestavros, M. Crovella, J. Liu, D. Martin, *Distributed Packet Rewriting and its Application to Scalable Server Architectures*, Proc. Sixth International Conference on Network Protocols, 1998.

[9] IKR SimLib, http://www.ikr.uni-stuttgart.de/Content/IKRSimLib/