A Workload Model for Benchmarking IMS Core Networks

George Din
Fraunhofer FOKUS
Berlin, Germany
george.din@fokus.fraunhofer.de

Razvan Petre
Fraunhofer FOKUS
Berlin, Germany
razvan.petre@fokus.fraunhofer.de

Ina Schieferdecker
Technical University of Berlin, and,
Fraunhofer FOKUS
Berlin, Germany
ina.schieferdecker@fokus.fraunhofer.de

***Abstract*—Performance benchmarking is a method to assess performance characteristics of various systems across different chip/system architectures under realistic conditions, to collect measurements such as success/fail rate, response times or round-trip delays and to identify problems for scalability or usability aspects under heavy load.**

In this paper, a benchmarking workload model for IP Multimedia Subsystem (IMS) is introduced. Our performance methodology is based on simulation of real world traffic conditions by defining a set of requirements which best characterize a population of UEs (user equipment).

I. Introduction

The IP Multimedia Subsystem (IMS) [1] is a standardised Next Generation Networking (NGN) architecture for the integration of mobile and fixed telecommunication, Internet and multimedia services. It uses a Voice-over-IP (VoIP) implementation, based on a 3GPP standardised implementation of SIP [2], and runs over the standard Internet Protocol (IP). World-wide more and more operators run experiments with the IP Multimedia Subsystem (IMS) prototypes as overlay architecture for different IP entrance networks and as docking station for a multiplicity of service platforms.

As the communications technology progresses, consumers have come to demand an increasingly varied and individualized set of services and content. QoS is becoming essentially important, as many of the new multimedia services are more sensitive to network performance and service degradation.

Performance benchmarking and, in general, performance evaluation is the activity that validates the system performance and measures the system capacity [3]. We can distinguish between three major objectives. Firstly, performance testing is used to validate the system ability to satisfy the performance requirements. The performance requirements are expressed as a) time intervals in which the tested system must accomplish a given task, or as b) performance throughput, which is the number of successful transactions per unit of time or as c) resources utilization. A second goal is to find the capacity and the boundary limits which is extremely important to know before deploying the system. However, the ultimate goal of performance testing is to assist the system designers and developers in finding performance issues, bottlenecks and/or to further improve the system.

A benchmark is executed by a test system (TS) which is a combination of hardware and software application that emulates the user equipment for a huge number of users. The test system generates system load to the System under Test (SUT), responds to the output of the SUT in a relevant manner, and collects events and other data that are required to produce the results of the benchmark test. All these actions perform under the control of a test script that determines for how long each phase of a test should execute and when the test should terminate. The traffic load conditions to verify the performance objectives are called *benchmark workload*. The workload design rely on a traffic model which characterizes various sets of traffic parameters and services.

Recently, the ETSI TISPAN working group 6 [4] released a performance benchmark standard for IMS/NGN networks, introducing a common methodology for benchmarking the performance of IMS implementations. The standard defines a set of test cases that can be applied to an IMS SUT, in order to evaluate its performance. The implementation of this specification has been started in the IMS Benchmarking Project [5]. In this paper, we present and characterize the first results running an implementation of the ETSI standard. We implemented several benchmarking scenarios which are then combined into a traffic set (a mix of the scenarios).

The paper is structured as follows. The next section introduces IP Multimedia Subsystem and Section III describes the workload elements used to create a performance benchmark. Some technical details about the benchmark implementation are presented in Section IV. The first results of running the benchmark against an open source IMS core network are presented in Section V. The conclusion section finalizes the paper.

II. IP Multimedia Subsystem

The IP Multimedia Subsystem (IMS) is the 3GPP standard architecture and protocol specification for deploying real-time IP Multimedia services in mobile networks. It provides the basis platform of a multimedia service model for core voice services (a.k.a. VoIP) and for new services based on voice, but including both video (e.g. video conferencing) and data services (e.g. location). The architecture has been extended by ETSI TISPAN to support the deployment of IP services in all communication networks (fixed, cable, etc.).

The IMS architecture is presented in Fig. 1, where the call control signaling entities are highlighted. The key technology

1930-529X/07/$25.00 © 2007 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2007 proceedings.

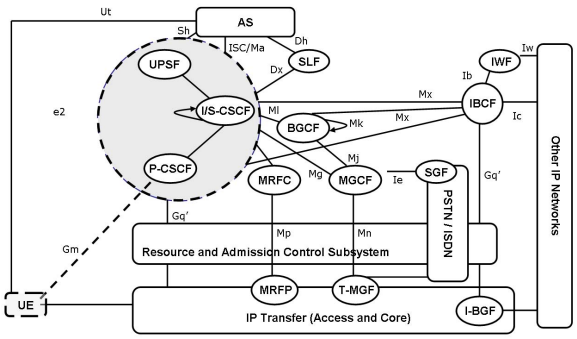

Fig. 1. ETSI TISPAN IMS Architecture

behind IMS is the SIP protocol which underlays many of the important interfaces between elements in an IMS-based network. The entry point to the core networks is the P-CSCF (Proxy-Call/Session Control Functions) which is a SIP proxy between UEs and IMS network. A UE is assigned to an IMS terminal during registration, and it does not change for the duration of the registration. The I-CSCF (Interrogating Call/Session Control Functions) queries the UPSF (User Profile Server Functions) to retrieve the user location, and then routes the SIP request to its assigned S-CSCF (Serving Call/Session Control Functions). S-CSCF is the central node of the signalling plane; it decides to which application server(s) the SIP message will be forwarded to, in order to provide their services.

Our approach for IMS performance testing is to measure the capacity of the whole core network, also called control plane. The control plane consists of many components that perform SIP-based calls to each other, and to the database. A call is a relationship between one or more subscribers, signalled and managed through the IMS system. The intent of a subscriber when using IMS is to make calls, so the obvious way to obtain realistic behaviour in an IMS test is to model the behaviour of calls.

The control plane is the part responsible for interacting with users, therefore it has to be highly available and robust. It represents also the part which handles most of the traffic routed through the IMS network. This is the reason for selecting it first as target of performance evaluation. However, in future, we plan to extend the framework also for other signaling entities (i.e. Application Server) and media plane.

III. Workload Model

The workload model used in our benchmark design consists of three main elements [6]: *use-cases* structured in scenarios, *benchmark tests* which instantiate the scenarios, and *benchmark test reports* generated out of execution traces.

Use-cases define interaction models between users and the IMS network. We defined call models for three very general use-cases: registration, voice call set-up and page-mode messaging. An individual interaction path is called a *scenario*.

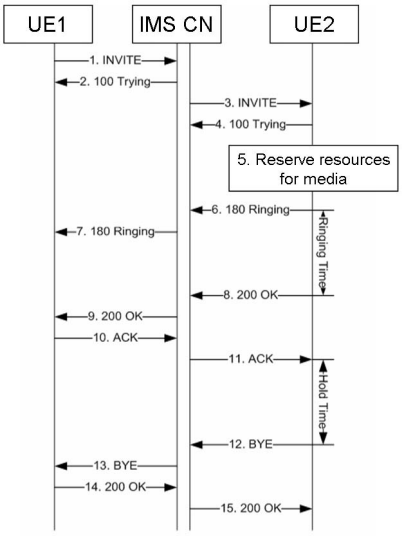

Fig. 2. Sequence of messages for voice call establishment

Each scenario is described by its message flow between the talking entities. An example for a scenario message flow is presented in Fig. 2 where the sequence of messages for a voice call is presented. The call scenario implies two UEs which establish a call session. One of them plays the role of caller by sending the INVITE message to the IMS network and the other one plays the role of callee by accepting the call invitation and responding accordingly. To be more realistic, we simulate also the ringing time and the hold time.

The benchmark has to simulate a similar behaviour by providing precise definitions of transaction types and contents. Additionally, the benchmark should regard the statistical distributions for transaction types, arrival rates, and other relevant parameters. The tests specify how the traffic is to be provided to the SUT and define the measurements to be made from the SUT and how they are to be reported.

Therefore, for each scenario, *metrics* and *design objectives* are defined. Typical *metrics* include scenario outcome, response times, and message rates. An instance of a scenario that experiences an error, failure, or timeout is referred to as an *inadequately handled scenario attempt* (IHSA). *Design objectives* (DO) describe the acceptable rate of inadequately handled scenario attempts for a use-case. If during the benchmark execution, at least one of the use-cases exceeds its DO we call that the tested system has reached its Design Objective Capacity (DOC). The DOC indicates the overload performance limit of the SUT and it represents the threshold for the accepted QoS. It is however, the output number which globally characterizes the performance of the tested system. The DOC can be used as capacity indicator for the overall performance of the IMS system but also for comparison with other systems.

A *benchmark test* combines scenarios from different use-cases into a *traffic set*. Within a traffic set, each scenario has an associated relative occurrence frequency, interpreted

as its probability of occurrence during the execution of the test procedure. This frequency indicates how often a scenario should be instantiated. To obtain relevant results, the occurrence frequency has to reflect reality, i.e. voice calls occur often than registrations. In our implementation, it is defined as parameter, therefore, various configurations can be easily experimented.

The execution of a benchmark test implies that the selected scenarios from various use-cases are executed at the same time. Each started scenario becomes a *scenario attempt*. The load rate applied to the SUT is called *Scenario Attempts per Second* (SApS). A scenario which is not handled correctly by the IMS core is called *inadequately handled scenario attempt*. When the frequency of IHSAs of a specific use-case exceeds the design objective, then the *design objective capacity* (DOC) has been exceeded.

The benchmark execution consists of a sequence of several benchmark steps in order to measure the DOC of the tested system. After the execution, we validate whether the threshold for the DOC has been reached by investigating if the rate percentage of inadequate handled scenarios (IHS) goes above a threshold (i.e. 0.1%). We extend the execution trials until we find a load at which the error rate is below the threshold and another load at which the error rate exceeds the threshold. At this moment we are sure that we found the overload capacity of the tested system.

The *benchmark test parameters* are used to control the behaviour of the test script. Such parameters have to be defined for any benchmark in order to allow the tester tune the load generation before the execution. The most important parameters are: the number of subscribers, the amount by which the scenario arrival rate is increased, the number of steps in a benchmark test, the amount of time for a test to be executed with a given system load (a test step) before incrementing the load.

The *benchmark report* is generated after the execution of a benchmark test. The report contains a full description of the SUT configuration, the TS configuration, the process used to generate the system loads at each SUT reference point, and data series reporting the benchmark metrics as a function of time. Many of these graphs are presented in Section V.

IV. Benchmark Implementation

In our implementation, TTCN-3 [7] language is used to specify the behavior of the benchmark scenarios and the load characteristics.

In TTCN-3, the test component is the building block to be used in order to simulate concurrent user behaviours. The parallelism is realized by running in parallel a number of test components. For better performance, the test components are distributed over several hosts [8] by using the distributed testing platform from Testing Technologies called TTworkbench [9].

The load is generated by the `SenderComponent`. Each call created by the load generator is associated to an `EventHandler`, which will handle all required transactions for that call. The number of EventHandlers is arbitrary and depends on the number of simulated users and on the performance of the hardware running the test system. Typically, a component handler simulates a few hundreds of users.

A user may create calls of different types. Additionally, to reflect the reality, a user may run simultaneous scenarios (e.g. a voice call and a page-mode messaging). At the creation of a new call, the selection of the user is arbitrary and the selected user may call any other user. A user may be used several times if the run is long enough (for short runs it is very unlikely that a user is used two times).

During a benchmark run, the scenarios are instantiated according to an arrival distribution, which describes the arrival rate of occurrences of scenarios from the traffic set and the traffic profile. The traffic profile describes the evolution of the average arrival rate as a function of time over the duration of the test procedure. An example of such an arrival process is the Poisson process [10] employed often in simulations of telecommunication traffic.

The `EventHandler` processes events received from SUT and executes appropriate actions according to the scenario message flow (as described in section III). The event processing starts with the *identification* of the user_id for which the message is received. This information is extracted from the protocol information embedded in the message. Once the user is identified, the handler evaluates the current state of that user and *validates* if the new message corresponds to a valid state, otherwise the transaction is considered *inadequately handled scenario*. Next, the user state is *updated* in the local user information database. If the received message, requires follow-up actions on the test system side, new messages are created and sent to the SUT. At receiving or sending any message, a log event is generated with precise time-stamp for the evaluation of the SUT latency.

The connections to the SUT are implemented via ports and an adaptation layer which handles the SIP messages interchanged with the SUT. The adaptation layer is based on the NIST Jain SIP [11], a Java interface to a SIP signaling stack, and it implements the TRI (TTCN-3 Runtime Interface) interface [7] for TTCN-3 test adaptation. It provides a standardized interface for handling the SIP events and event semantics, and offers full transaction support for SIP-based calls. The test logic (i.e. state handling) is executed by the TTCN-3 parallel test components.

The SIP protocol messages are bound to carry state-full information which has to be correlated with information retrieved from other messages. We implemented the state-full testing approach where the test behaviour defines a sequence of requests and settings for controlling the state maintained between them.

The performance analysis and visualization are based on *offline* processing (after execution ends) of performance metrics gathered in the form of log files during the execution. All the log files from the test servers are collected and merged into one composite log file. Then, this composite log file is processed by the traffic analysis tool. The generated benchmark report

contains various graphs which visualize collected metrics in different views: dependency on time, delays between events, stochastic distribution of events, etc. Various statistics (max, min, average, variation, etc.) are also reported.

V. Experiments

The tested system in our experiments is based on the Open IMS Core [12], an open source implementation of the IMS architecture. The testbed network consists of Proxy-CSCF, Interrogating-CSCF and Serving-CSCF which are based on the SIP Express Router (SER). The UPSF functionality is implemented in CHeSS, a component developed also by Fraunhofer FOKUS. It is a simple stateless AAA (Authentication, Authorization, Accounting) component which uses MySQL for user data storage. The four components are installed on the same server (mem=4GB cpu=4x2.00GHz cache=512KB L2).

We run the benchmark for a user population of 20 000 users. The scope of the experiments is to determine the DOC of this IMS network. Our implementation supports currently five scenarios: 1. registration 2. re-registration, 3. de-registration, 4. voice-call set-up and 5. page-mode messaging. Any scenario can be instantiated by any user. For the voice call scenario we use an exponential distribution of the call-holding time with a mean of 120s. The load generator selects randomly users from the population and decides which scenario to run for them. The traffic set consists of a mix of these scenarios. The occurrence ratio of each scenario is presented in TABLE I.

TABLE I
Traffic Set Composition

Scenario type	Scenario ratio in the traffic set
registration	5%
re-registration	20%
de-registration	5%
voice-call	30%
page-mode messaging	40%

TABLE II
Benchmark Results per Load Step

Step Load	IHS rate	Observations
165	0.0	
170	0.0	
175	0.08	Some errors occurred in this step but the IHS is still below the threshold
180	0.0	
185	0.24	The DOC limit has been reached

TABLE III
CPU and MEM Consumption

Step Load	idle	user time	system time	free memory
[SApS]	[%]	[%]	[%]	[MB]
170	8.91	38.16	52.52	154.78
175	9.73	38.87	51.04	168.08
180	4.13	36.11	59.18	149.75
185	3.27	35.93	60.26	131.54

We run the benchmark for a number of steps, starting with 165 SApS and increasing the load per step with 5 SApS.

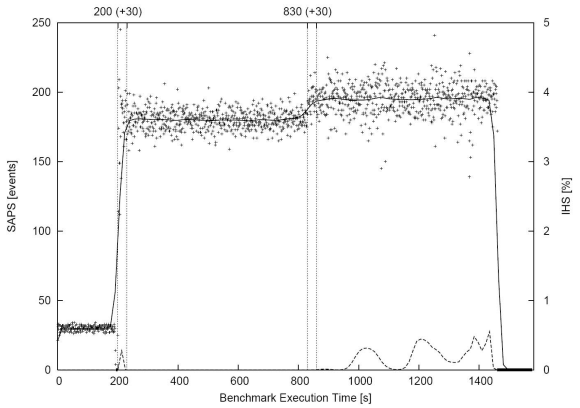

Fig. 3. Call Creation Rate and Error Rate

The step duration is 10 minutes, long enough to have a large enough number of transactions with the SUT. The results are captured in TABLE II. According to our benchmarking procedure, we consider that the DOC has been reached when the IHS rate exceeds 0.1% which is in our case 180. At this level the load is still handled correctly by the SUT, but increasing this load causes that IHS exceeds the threshold.

Fig. 3 shows the last two load steps of 180, and respectively 185 SApS per second. The first line indicates the average load sent to the SUT (additionally we see the points indicating the number of scenarios created in each second). The dashed line indicates the IHS as a percentage of fails out of the total number of calls. The figure reflects the results presented in TABLE II, where for the load of 185 we notice an IHS of 0.24.

Fig. 4 shows the average latency of establishing a call through the IMS core network; we measure the round-trip time between sending the INVITE until receiving the ACK message (see Fig. 2). This graph indicates the dependency between the latency and the load level: during the second load step the latency is obviously higher. We also notice a spike right at the beginning of the benchmark which is caused by the SUT adaptation to the load.

In a further experiment we want to understand how the SUT behaves around the DOC in terms of resource consumption. Therefore, we started a resource monitor to observe the CPU and MEM consumption. The average values for these metrics is presented in TABLE III. We observe an increase of the CPU consumption (user time and system time). Also the free memory decreases from 154.78 MB at 170 SApS to 131.54 MB at 185 SApS; which is explicable since the SUT needs to process data for more open calls. We also notice some unusual

TABLE IV
Benchmark comparison among different configurations

Hardware Configuration	DOC
mem=4GB cpu=4x2.00GHz cache=512KB L2	180
mem=8GB cpu=4x2.00GHz cache=2MB L2	270
mem=8GB cpu=4x2.66GHz cache=4MB L2	450

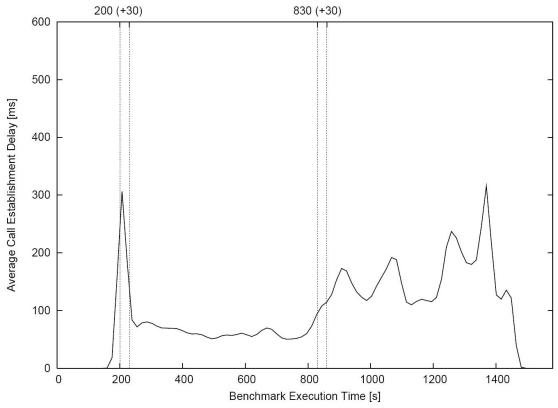

Fig. 4. Average duration of routing an INVITE message through IMS core network

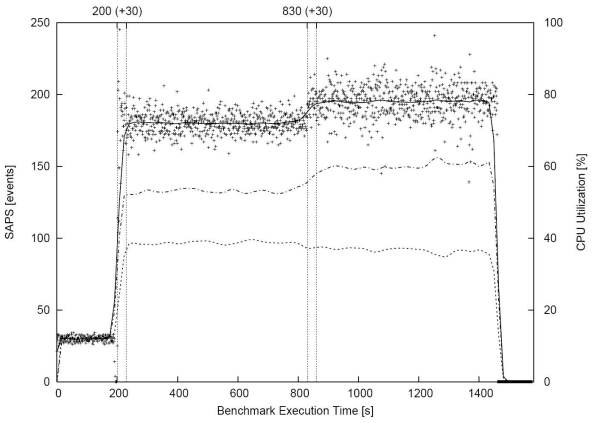

Fig. 5. CPU Consumption

values at 175 step i.e. the system time has a lower value than the 170 step. It also consumes less memory. That explains why during the 175 step we notice some inadequately handled scenarios.

Fig. 5 shows the dependency of the CPU system time (dashed-dotted line) and user time (dotted line) on load rate. The demand for this resource is higher for 185 SApS. In the second step the total CPU consumption is very close to 100%.

Fig. 6 indicates the memory demand (dashed-dotted line) along the benchmarking procedure. Also in this case the demand is higher in the second step.

TABLE IV presents a comparison of the DOC numbers for different servers. The SUT software has been installed with the same configuration on all servers. The DOC is obviously higher for the last two servers which have more memory. However, the cache seems to have the biggest impact since the last board has similar configuration to the second server but more cache.

VI. Conclusions

This paper describes a workload design methodology for future NGN/IMS networks and presents the first results of running the benchmark against an IMS core network.

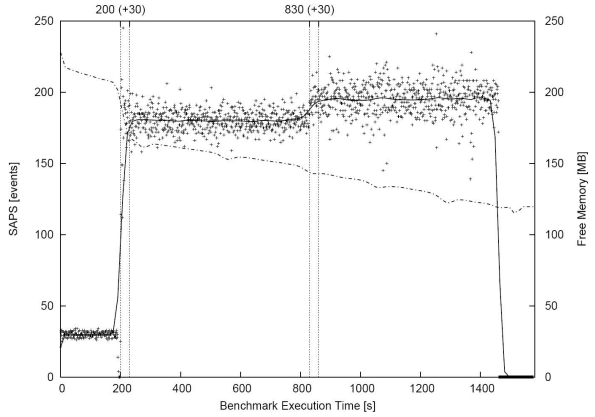

Fig. 6. Memory Consumption

The experimental work, proved the validity of the Design Objective Capacity concept, which serves as general performance indicator. Close to the DOC load, the SUT consumes a lot of resources (e.g. 95% of CPU). In our specific SUT, the CPU resource seems to be the cause of reaching the DOC.

The IHS metric shows that a system may fail calls also at lower loads (below DOC). This is explained by the random generation of the traffic which may cause some higher rates from time to time.

The reaction time of the SUT variation depends on the load. However, we detected seldom spikes while increasing the load from one step to another. The explanation for this phenomenon is that the SUT needs a short period of time to adapt to load changes. This is reflected by a longer response time for all calls in that period of time.

References

[1] 3GPP, "Technical Specification Group Services and System Aspects, IP Multimedia Subsystem (IMS), Stage 2, V5.15.0, TS 23.228," 2006.
[2] IETF, "RFC 3261, SIP: Session Initiation Protocol," 2005. [Online]. Available: tools.ietf.org/html/rfc3261
[3] Jerry Zayu Gao and Jacob Tsao and Ye Wu and Taso H.-S. Jacob, *Testing and Quality Assurance for Component-Based Software*. Norwood, MA, USA: Artech House, Inc., 2003, ISBN 1580534805.
[4] ETSI TISPAN, "IMS/NGN Performance Benchmark, Technical Standard (TS) 186 008," February 2007, Sophia-Antipolis, France.
[5] Fraunhofer FOKUS, "IMS Benchmarking Project," 2006-2007. [Online]. Available: www.fokus.fraunhofer.de/IMSBenchmarking?lang=en
[6] G. Din, "IMS Testing and Benchmarking Tutorial, The 2nd International FOKUS IMS Workshop," Nov. 2006. [Online]. Available: www.fokus.fraunhofer.de/event/ims_ws_06/details.php?lang=en
[7] George Din, "TTCN-3," in *Model-Based Testing of Reactive Systems*, ser. Lecture Notes in Computer Science, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., vol. 3472. Springer, 2004, pp. 465–496, ISBN 3-540-26278-4.
[8] Ina Schieferdecker, George Din, Dimitrios Apostolidis, "Distributed functional and load tests for Web services," *Int. J. Softw. Tools Technol. Transf.*, vol. 7, no. 4, pp. 351–360, 2005, ISSN 1433-2779.
[9] TestingTechnologies, "TTworkbench: an Eclipse based TTCN-3 IDE," 2006, Berlin. [Online]. Available: www.testingtech.de/products/ttwb_intro.php
[10] NIST/SEMATECH, "e-Handbook of Statistical Methods," 2006. [Online]. Available: www.itl.nist.gov/div898/handbook
[11] Mudumbai Ranganathan, Phelim O'Doherty, "JAIN SIP Tutorial," 2005. [Online]. Available: java.sun.com/products/jain/JAIN-SIP-Tutorial.pdf
[12] F. FOKUS, "FOKUS Open Source IMS Core," 2006. [Online]. Available: www.openimscore.org