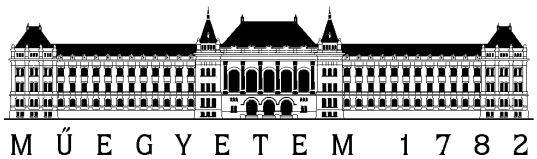
New Algorithm for Biased Load Balancing with Graceful Shutdown

Author:
István Bakos[1]
Loránd Nagy[1,2]

Consultants:
Gábor Járó, Ph.D.[2]
Dr. Tien Van Do[3]

[1] Institute of Mathematics, Faculty of Natural Sciences, Budapest University of Technology and Economics
[2] Nokia Hungary Ltd., Budapest
[3] Department of Networked Systems and Services, Budapest University of Technology and Economics

October 27, 2015



Contents

1 Introduction

In traditional telecommunication networks, network elements are permanently available independently of the part of the day and actual traffic. This means they constantly reserve hardware (HW) and consume energy. Furthermore, in most cases these (physical) elements can not be used for tasks other than their designated ones.
In several telecommunication networks, Round-Robin algorithm is used for 'resource management' and load distribution but – as we will see later – it is not so effective (especially for our purpose). First, it does not take into account the actual CPU load of the (internal) functional units (FU). Second, it assumes that every FU has the same capacity and performance in the system. That is why we have to find a solution that considers the foregoing ideas and is able to achieve the later-described horizontal scaling.
However, it is important to notice that Round-Robin, Weighted Fair Queueing and their variants have been used (for load balancing) for a long time in practice and of course, many results have already been achieved in dynamical load balancing [1], [2]. Furthermore, e.g. SIP (Session Initiation Protocol) based services are very well-investigated [3]. Applying these is common practice in distributed systems. So, the scalability in Cloud has been investigated almost since it's first appearrance.
Development in 'Telco' network solutions enables almost all core network elements – which can be seen in Figure 1 (light blue colored, like TAS, MSS, MME, HLR, HSS, etc.) – to run on Cloud-based environment [4]. At this point, horizontal scalability of distributed systems (such as core network) becomes a very important requirement, as it practically allows to add or remove resources (henceforth functional units) anytime to or from the system. Removal of a functional unit shall be 'graceful' in the sense that it cannot cause performance degradation and ongoing processes must not be interrupted.
Therefore an algorithm is needed which uses only certain (higher order) functional units continuously. The main idea of the new algorithm is that it just starts loading additional (lower order) FUs if the load of the higher order units reaches a pre-defined level (upper threshold). Thereby, in Cloud-based environment, traffic could be concentrated on a few functional units and further FUs are loaded only if it is necessary. So, this (kind of) algorithm enables a much more dynamic resource management in Cloud.
Our goal is to provide a method which loads the available (internal) functional units within a single network element (whether TAS or MSS) according to the above-mentioned logic. In this way, the operation of a network can be energy-efficient and cost efficient as not

used FUs can actually be shut down depending on the real-time traffic. It has to be highlighted that our solution is applicable in any 'Cloud-based' distributed telecommunication system, hence, it is independent from any protocol (i.e. SIP, H.248, M3UA). In addition, the unknown length of the running processes, which makes the seamless removal of functional units harder, is not a problem for this algorithm (typically, the length of calls has an unknown probability distribution). It is very crucial in call handling, as even very long calls must not be terminated by removal of that FU which controls them.
Henceforth, in section 2, we describe the problem in details and show some related examples from real networks. In section 3, we interprete our solution in both theoretical and practical point of view. In section 4, we compare Round-Robin and the new algorithm and in section 5, we summarize the results so far.

2 Problems and Real-life Examples

Before we describe the problem in details, we take a closer look at the core network which is in the scope of our investigation and introduce how it can be moved on to the Cloud.

2.1 The Core Network

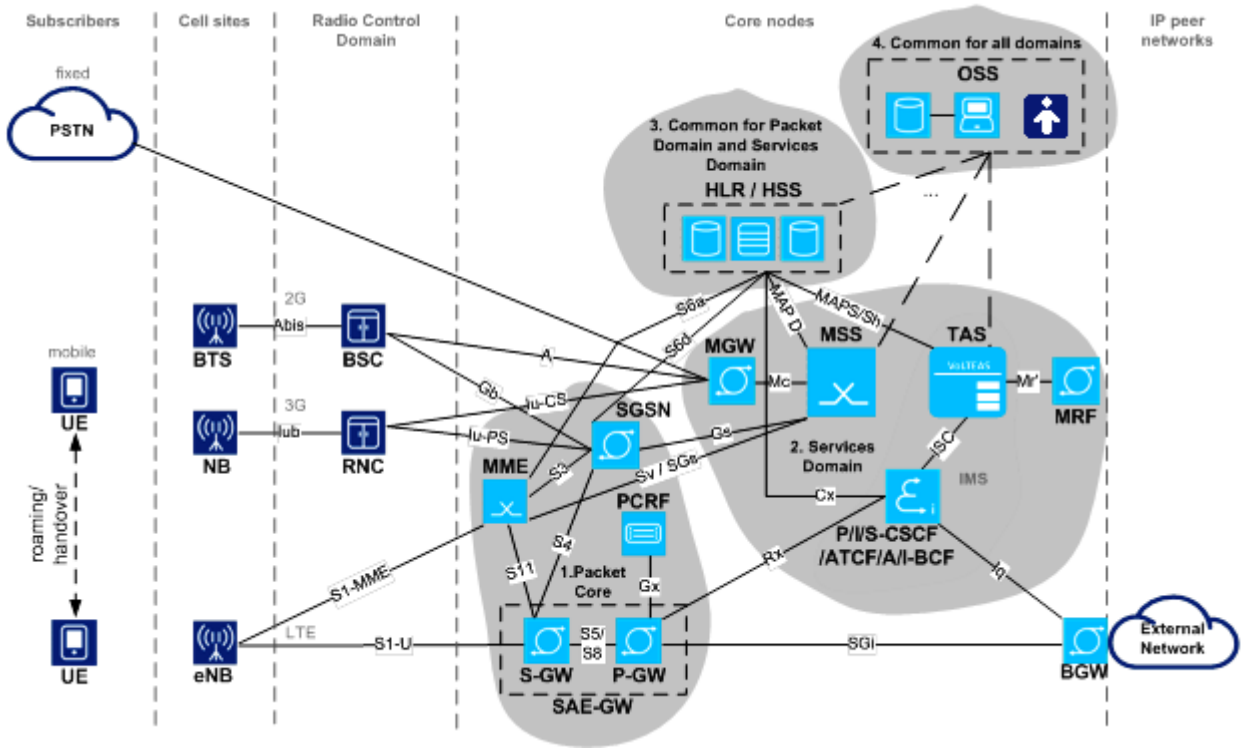


Figure 1 presents TAS and MSS in mobile (and IMS) network

The Core Network (Figure 1) has four main parts:

1. Packet Core;
2. Services Domain which includes e.g. MSS (Mobile Switching Center Server), TAS (Telecommunication Application Server) and the IMS (IP Multimedia Subsystem);
3. Registers (HLR or HSS) that are common for the Packet Core and the Services Domain;
4. OSS (Operations Support System) which oversees all of the network elements, including the radio network (2G, 3G, 4G/LTE) too.

The same resources (CPU, memory, storage, IP interfaces and networking) must be provided for (core) network elements in order to be able to operate in Cloud based environment (just as in traditional, e.g. ATCA, environment) [5].
On Cloud, storage capacity is given by SAN (Storage Area Network) and network connections are provided by SDN (Software-Defined Networking). As it can be seen in Figure 2, telecommunication applications (which are SW applications) are located on the top of Cloud. Similarly, OSS and CAM (Cloud Application Management which controls the horizontal scalability of certain network elements) can run on Cloud.

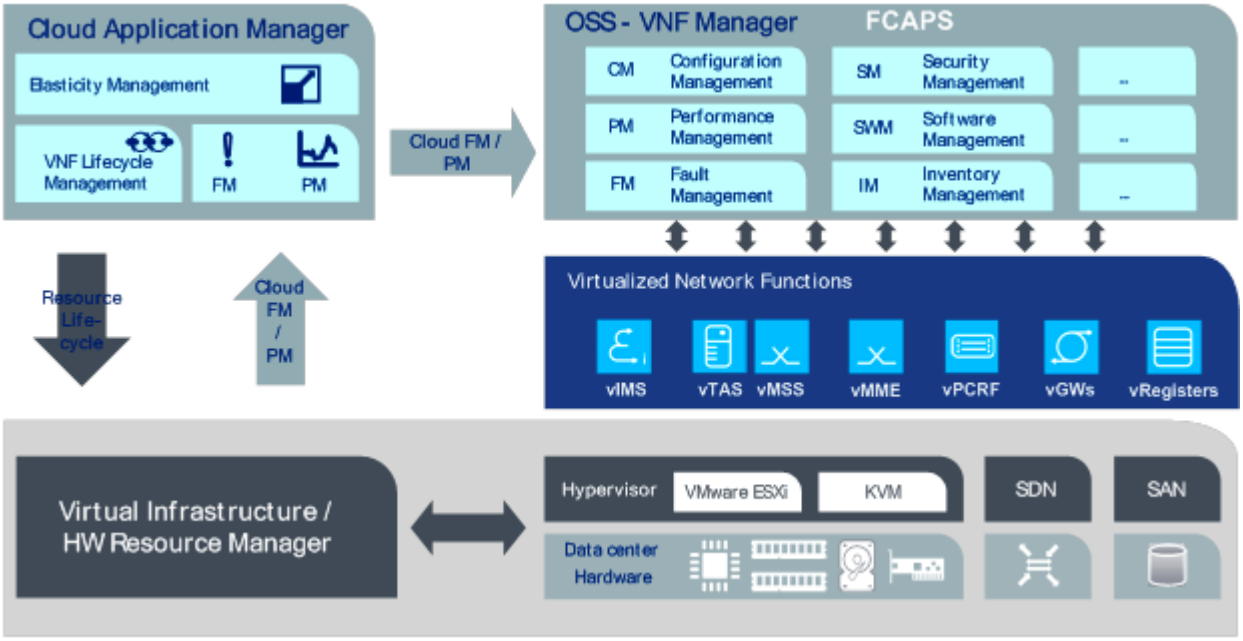


Figure 2 presents the core network elements in Cloud based environment [6]

Based on ETSI NFV architecture [5], MSS and TAS are virtualized network functions (VNFs) [6], [7]. The function of MSS and TAS in mobile and IMS network is defined in detail by the following 3GPP technical standars: TS 23.002 [8], TS23.218 [9] and TS 23.228 [10].

2.2 Load from a real network

We collected data from several mobile network operators[1]. First of all, it can be observed that the load of the functional units fluctuate significantly both daily and weekly basis. It

[1]We do not name specific operators for confidentiality reasons, so from now on, we indicate every operator as unknown.

can be seen in Figure 3 how steeply the load changes between night hours and (daytime) busy hours. It is also clearly shown in the same figure that people make less phone call on weekend (red circle) compared to weekdays. Moreover, sudden big load can occur anytime, even during busy hours (green circle).

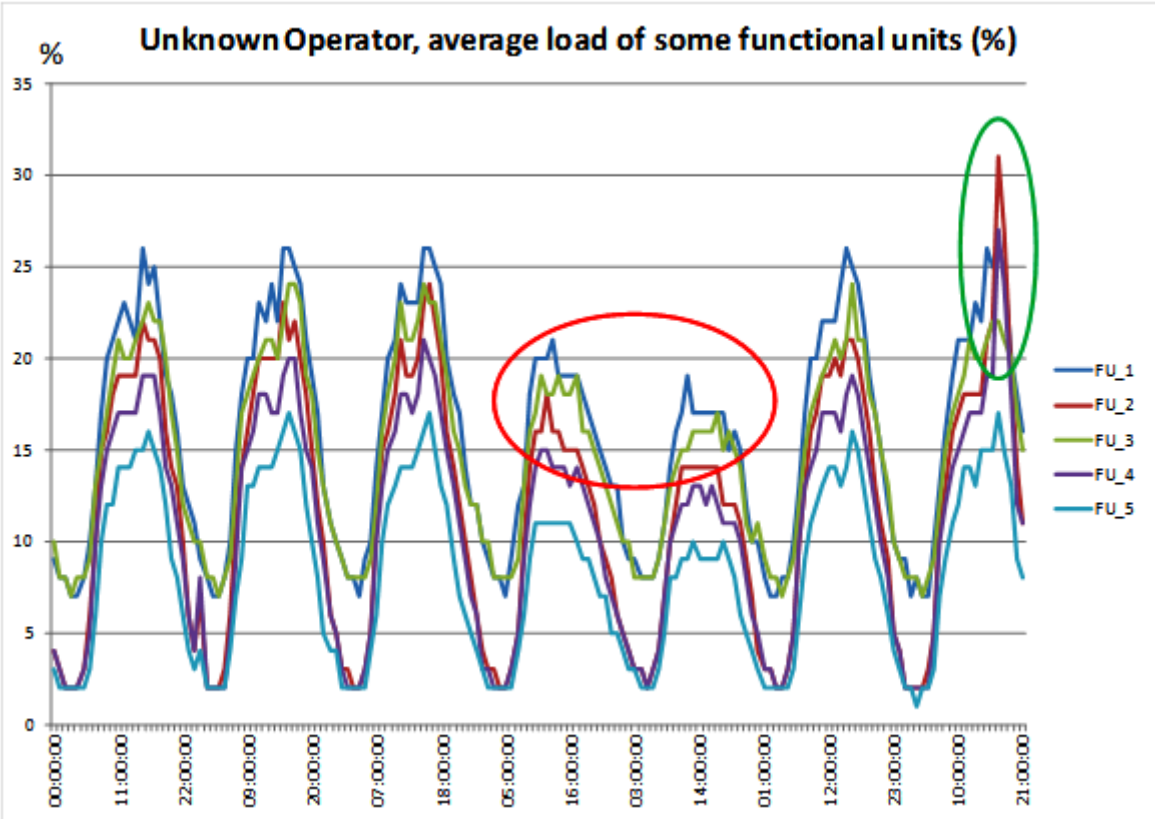


Figure 3 presents the average load of some FUs in the same network element (Operator 2), one week measurement, and one hour resolution

Of course, traffic also fluctuates at the level of the entire network (Figure 4) which foresees the possibility of switching functional units off. For example, if we take a closure look at daytime and nighttime traffic in the network, then fluctuations can be measured up to 60%. This means less functional units would be needed in nighttime than during the busy hours. (Currently, all the resources are used continuously.)

These analysis show that the utilization of the resources does not often reach even the 35% level which is far from the recommended 60 or 70% level. So, better utilization is one of the aims because it leads to an optimized resource management and helps to decreasing the energy consumption of the system.

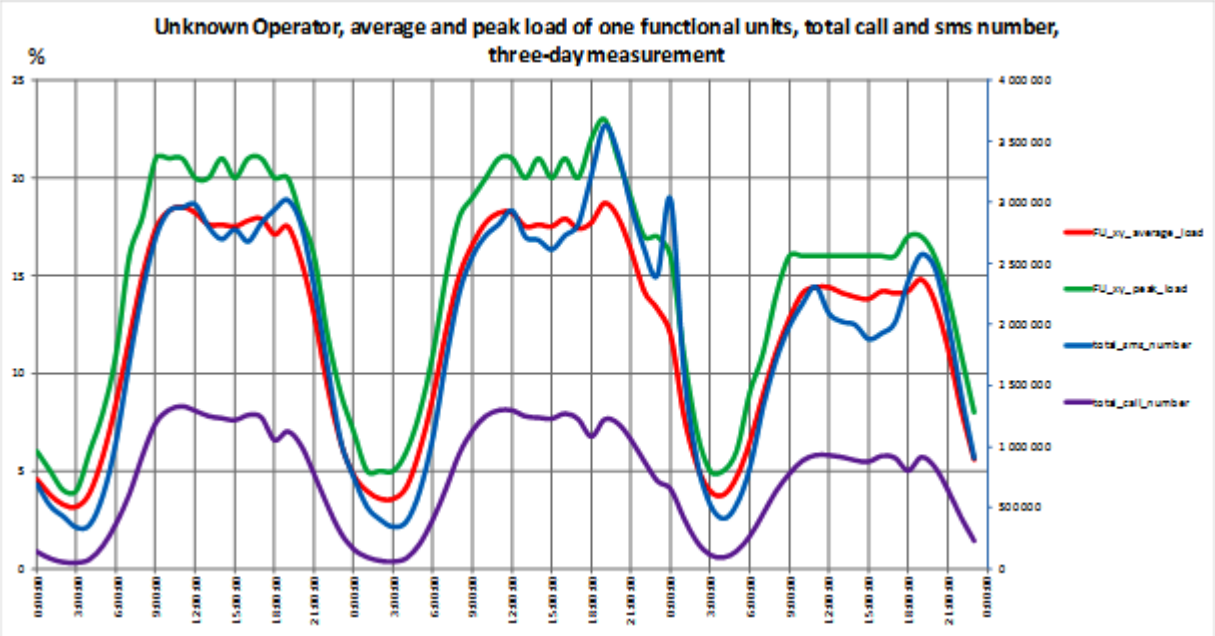


Figure 4 presents the average (red) and peak (green) load of a single FU compared to the total call (purple) and sms (blue) number, three days measurement

2.3 Problems with Round-Robin

As we mentioned earlier, Round-Robin (scheduling) is used widely for call-handling (or call distribution) in telecommunication networks. Let's see the following simple example to this algorithm.

If urns represent the functional units and balls represent the calls the algorithm works in the following way: we throw the balls one-by-one into the urns consequently in arrival time and if we run out of empty urns, then we start again. This process is quite simple and easy to implement, but not appropriate for several reasons.

First of all, it cannot be used for priority-based or biased load balancing. It can be seen in Figure 5 that the average load of the functional units are not really balanced. This is due to the fact that we mentioned in the introduction: Round-Robin does not take into account the actual CPU load of the (internal) functional units and assumes that every FU has the same capacity and performance in the system.

The second problem is its deterministic behaivour. It is shown in Figure 6 that CPU usage of a single call is not continuous (control plane): there are a setup and an end phase. Based on the elapsed time between the two phases (T, distribution of T is unknown), we can

talk about short and long calls. Since, we do not know the real distribution of the call lengths, we cannot predict the end of a call. This causes the problem that there are FUs with higher load levels because of the many short calls. The others get more long calls, so they are less loaded.

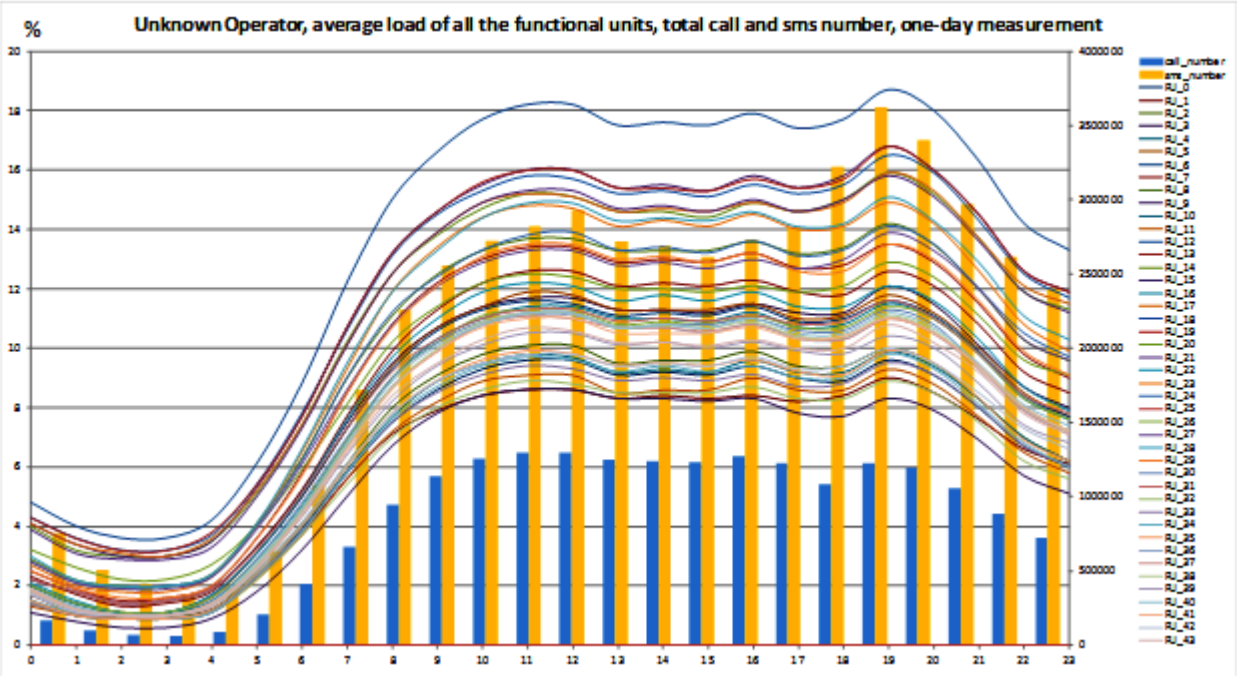


Figure 5 presents the effect of calls (blue columns) and messages (yellow columns) to the load of FUs

Let's assume we have twenty urns and blue ball represents the long call. We also assume that every twentieth call is long call. Then the twentieth urn contains all the blue balls (i.e. long calls).

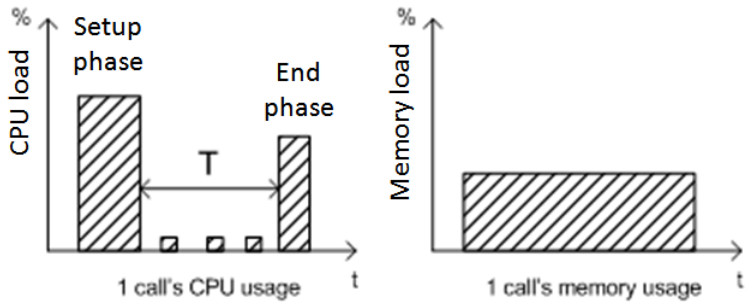


Figure 6 presents the CPU and memory usage of a call

Considering these observations, the analysis and the properties of Round-Robin, this kind of call-handling is really far from optimal. So, we have to handle these problems with a more sophisticated algorithm.

3 New algorithm for Biased Load Balancing with Graceful Shutdown

In our solution, one of the most important aspects is to use as many functional units as the current traffic needs at most. I.e. we do not endeavour to use all of the functional units at the same time or load them evenly. The other one is that we can add or remove FUs to or from the system based on the current traffic. (That is why there will be periods when the load of the units can be very different.)

3.1 Advantages of Load Balancers

Application of Load Balancers (LB) is inevitable in any distributed systems. With their help, incoming requests can be easily distributed among the internal functional units of the network element. Furthermore, the inner architecture, complexity and operation of the system can be hidden by Load Balancers, i.e. the outside world can communicate only the LBs (Figure 7).

3.2 Priority-based Horizontal Scaling Algorithm

From now on, we always assume that the system contains at least one LB. Due to redundancy and resiliency reasons, this means minimum 2 LBs. Our algorithm, called Horizontal Scaling Algorithm (HSA), is located in LBs.

a) The invention has four key parts. First of all, we organize FUs into groups or sets (S_i, $i = 1, \ldots, n$). Such a group can contain any number of FUs (at least one, at most as much as total available). On one hand, we can minimize the chance (using groups) that a newly activated FU is overloaded immediately after switching on. On the other hand, we can manage more CPU resource at once. This kind of configuration can be seen on Figure 7.

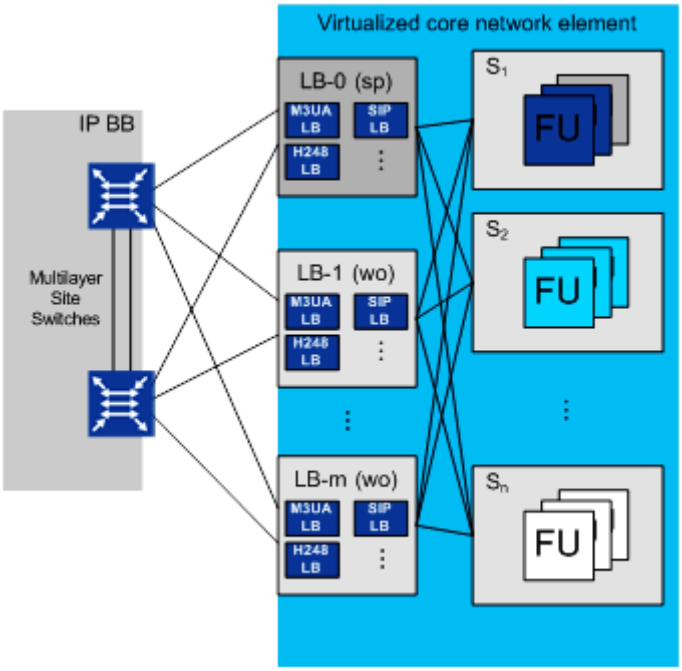


Figure 7 presents the planned network element configuration, where FUs are in sets

b) The second step is: we fix an upper threshold (e.g. 65%) for the average target load of the groups (or sets). What we want to achieve with this is that we load the groups as long as the average load of the groups reaches the given upper threshold. If the target load is exceeded, a new group is placed into operation (Figure 8). Hereby we force the system to activate a new set if and only if it is needed based on the current traffic.

   Similarly, we fix the same lower threshold (e.g. 20%) for the average load of the FUs within each group. This is useful, since when the last active group's average load is under the threshold we can deactivate that group.

c) In addition, each group gets a preference number for which the followings are true:

   - Each FU has the same preference number within its group;
   - If a FU is switched off (or in idle mode), its preference number is zero;
   - If a group's load reaches the upper threshold, its preference number is maximum (M);
   - If a new group is loaded, its preference number is dynamically changed between 0 and M (depending on traffic).

We use these to make resource handling and load distribution between groups adaptive in case of traffic changes.

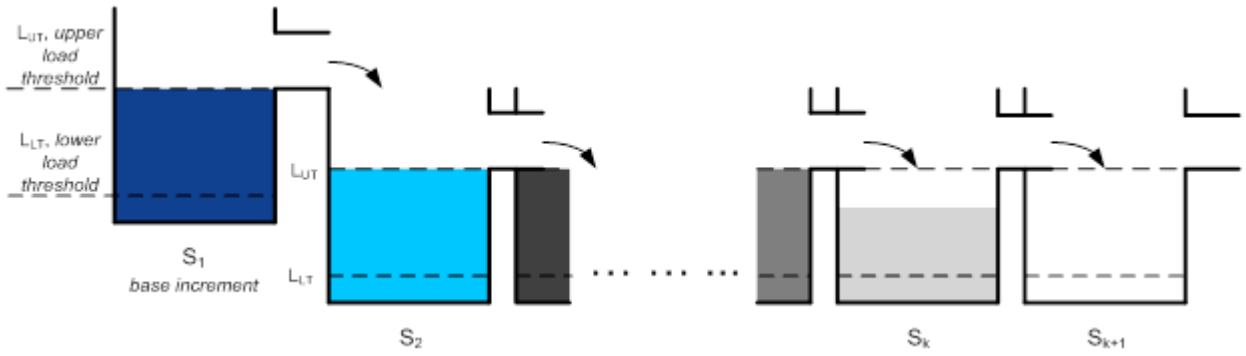


Figure 8 presents the bucket chain model

d) Finally, we randomize the call distribution. The process can be seen on Figure 9. The top line shows the incoming calls. The arrival of calls can be modeled with a Poisson process [11] (although some results yield that in fact this is not the most accurate model [12]). Such subdivision of $[0, 1]$ interval can be seen on the lower part of Figure 9 in which each subinterval belongs to different FU. The colour of the FUs indicates the group they belong to. The length of the subintervals is determined in real time by the following formula:

$$\text{length of subinterval} = \frac{\text{preference number of FU belonging to the subinterval}}{\text{the sum of all preference numbers}}$$

The bigger the FU's preference number is, the bigger the subinterval is which belongs to the FU. The call distribution is random, because every call gets a random number from $[0, 1]$. The call is sent to that FU, which belongs to the subinterval that contains the generated random number.
If we recall the urn model and the problems that appeared during the Round Robin it seems to be a logical choice to randomize call distribution.

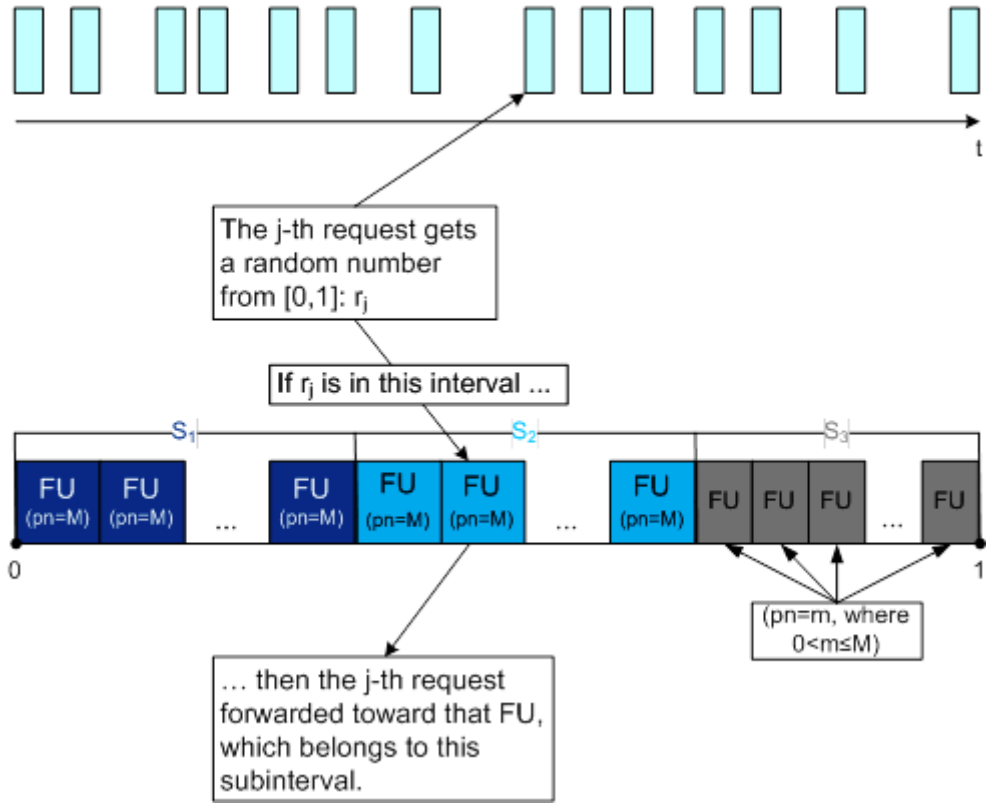


Figure 9 presents the conceptual illustration of the randomization

3.3 HSA in practice

As this research is motivated by telecommunication business needs, the first approach on HSA was to present a proof-of-concept simulation. Therefore we needed to create a computer program that can simulate calls and units to process them.
To create the simulation we created a simple c++ console application. The program used a simple representation of every element as a custom type:

call: an integer for id, and double precision floating point numbers (doubles) for start time, end time and length (in seconds)

functional unit: an integer for id and a vector of calls, with methods to measure load (the program setup defined the maximal number of half-calls[2] for a unit to represent load as percentage) at given time and accept calls (or reject in case of overload)

[2]Half-call: a call in setup or end phase.

group of FUs: boolean flags to indicate status of activation, graceful shutdown, an array of FU types and an integer storing the last used FU index (for Round-Robin simulation) with additional doubles to store relevant history data (past load, time of activation), methods to process arriving calls for both Round-Robin and HSA, to measure load and free capacity

With these element the simulation program used a file input to simulate call data, than used that data to first simulate the system's response first using the Round-Robin algorithm than the HSA. The Round-Robin was started with all groups activated, however the HSA just used one group at the start. In every case we used five groups of six FUs. Both methods stopped at every second to evaluate status but while the Round-Robin algorithm just removed unused data and wrote active call number and average load for FUs in each group, the HSA evaluated if the current state and intervened in case. The logic of the decision is described on a flowchart in the appendix section (this flowchart served as implementation strategy during development).
The reason for simulating calls instead of using measurement data is:

- key use-cases in proving the method's usability can be easily created;
- exact measurements containing call start and call end times with precision to the second are not available and even if such data existed the simulation would need to change the simulated processing units' specifications to create all needed scenarios.

However a later goal is to create simulations with real life measurement data to compare the simulated results to the measured values. To do this we needed to create a model for calls that can be defined through total call numbers during predefined time periods.
It is a commonly accepted model for independently arriving requests (like customers arriving in a shop, or calls started through a network etc.) to use Poisson distribution for the number of total requests [11]. The Poisson distribution has a great advantage, namely that we know exactly the distribution of the time between two events (arrivals or call starts) which is the exponential distribution. Therefore if we have the number of calls arriving in a period we can create a model that provides us with a total call number for that period as a random variable that follows Poisson distribution and its expected value is the given number of calls. More generally we can say that the total number of calls follows an inhomogeneous Poisson process and we are using a rate function that is piecewise constant.

If we accept the Poisson process we only need to create a function in the simulation that gives us random numbers with exponential distribution with the given parameter. Generating genuine random numbers is a popular topic among programmers and mathematicians as well. There are many algorithms to create numbers that follow certain distributions, but in fact most methods can not generate real randomness, that is usually achieved by seeding the random number generating algorithms with some random measurement data (with known distribution of course). The random numbers, our simulation used, were generated with functions provided by the program language's <random> extension. These functions needed some seeding which was provided as the sum of a number obtained from the system clock, another from the program's time variable and finally a number generated by another built-in random number generator. This allowed us to create pseudo-random numbers with the needed distributions. Pseudo-randomness means in our case that the probability density functions (PDF) of the generated numbers do not match exactly the desired distributions' PDFs. For example the uniform random number is not uniform, functions like the one the simulation uses has a small skewness, meaning that it favors one side of the interval. This could indeed cause a problem, but the effect of it can be dramatically reduced by using a small interval, where the skewness is virtually undetectable. For other distributions we can generate lots of numbers and than examine the difference between theoretical and empirical moments. In our simulations all generated distributions had error for the first two moments under 0.05%.

Our modeling of calls now has a method to create call starts at a defined rate (or following a rate function). To complete it we also need a model to terminate these calls. The first idea is to use some exponential distribution for it is now already implemented. In fact the first simulations that we did used this method. The reason why we accepted a "bad" model is that our simulation focuses on the load created by the start of a call, termination has much smaller effect and we did not consider ongoing calls as relevant load inducing parts (Figure 6).

Also simulating the length of calls is not as easy as the call starts. The reason is simple; while even complicated call number profiles can be simulated correctly with a well-chosen rate function, call length is much more random and there is no result accepted widely like the Poisson process in the other case. One solution could be that we suppose that like for call starts it is impossible that two calls start/end at the same time (technically if two calls were to start at the very same moment on the same unit one of those would be stalled for a short time) and we could use a queuing model. Imagine the calls as customers

at a post office. There are different rows for different kinds of transactions and the time required for a customer is a random variable with a distribution also depending on the desired action. This is an appealing model since it can be realized with a model for a multidimensional Poisson process. Another possibility is to say that a call can end during any time period. If we are measuring the load after every second for example, than we only need to know if calls were ended during that second. Suppose that there is a coin-toss at the beginning of every second of a call that decides if the call would end in that following second. Of course the coin ought to be really unfair otherwise calls would not last for long. The probability of ending can be a function that depends on the call length and the time (can be a call's start time or the current time as well). Since we only need this function evaluated at selected points no continuousness or monotonicity is required only that the function should take strictly positive values smaller than one (actually much smaller). For simplicity though we can say that the probability should be a continuous and monotone function of call time and for large call times the ending probability should converge to some constant (not even depending on the time). The good thing about this method is that after successfully creating a function we can adjust average times with only changing one multiplier and also coin-tosses are easily implemented. Another advantage is that we can create a simulation where calls behave like in the real world, after they are started there is no way to know when will they be terminated if ever.

Although these new models are tempting, unfortunately they ought to be validated before usage and since the creation of new call modeling is not among the main goals at this point therefore we used a simpler model where call length was defined as a log-normally distributed random variable. We chose this specific model although there is no intuitive explanation for the results in [12]. During the simulations, we used log-normal distribution with expected value: 90 (sec) and standard deviation: 50 (sec).

With these decisions for modeling we could start the implementation of the simulation. This article in fact describes the second phase of the simulation where phase one was the first implementation of the logic and concentrated all efforts on creating a code that can show the behavior of the logic in basic use-cases. For this first simulation we used the exponential call length model and data tailored for specific situation and all parameters were set alike this method.

After the first simulations we had results proving that further investigation is needed, mainly since the results only told us that the idea for the new algorithm is viable though it did not provide any information about the comparison between the new and the old

logic. Therefore the main goal was to create a better simulation that can simulate calls and provide this data set for the call distribution algorithm which should be able to run both based on the Round-Robin method and the HSA.
This sets the basic requirements for the program, these three features were needed.

4 Comparison of Round-Robin and the new algorithm

During the comparison we used data based on measurements from real operators. Initial testing ran with arbitrary data that we created. The real results came from two methods, where the first was to take data that represents some sort of periodicity and investigate the load results of that with 15 minutes sampling for the Poisson process' rate. The second approach was to get data from operator measurements and create a new set of data with a better – one minute – resolution. We used a standard spline[3] interpolation for the original points and evaluated the function at each minute. This method does not change the call numbers by which we define the rate of the Poisson process, but it reduces the time interval with the number should be normalized for one second, therefore if we want to keep the same rates we need to divide the received numbers.
In fact for some performance considerations and to keep program setting the same for most cases we divided the data numbers so that the result would be around one million call per hour.

4.1 Results for Round Robin based decision

In all scenarios the algorithm did exactly like one would expect (Figure 10), the load generated by the calls was equally distributed among groups of FUs (also referred to as sets). This result shows how this concept is working if there is just one loop of Round-Robin, however in real realizations there are concurring loops and that causes the load differences on the presented measurements. However the average load of the groups indicates that there is constantly huge free capacity for each group.

[3]Approximation for the joining curve with qubic polynomials on each interval with continuous first and second derivatives between intervals.

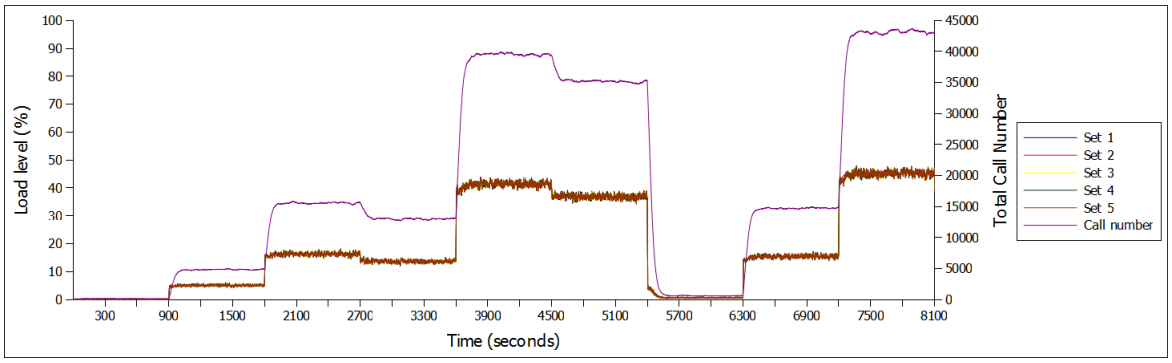


Figure 10 presents the simulation result of Round-Robin, 3 hours resolution

These results confirm the original assessment of the problem that even if the groups of functional units are in balance (as a result of circumstances or active load balancing) the efficiency of this setup is not ideal (Figure 11). For example one could suggest that decreasing the FU number would increase the average load and decrease resource consumption. However the resulting load shown above gives a perfect example why the problem cannot be solved only with this idea. The busy hours during the day generate significantly bigger load and operators should scale their networks to be able to handle these situations which gets us to the hours outside busy hours, where the system that was able to handle much bigger load can not save resources causing resource wasting.

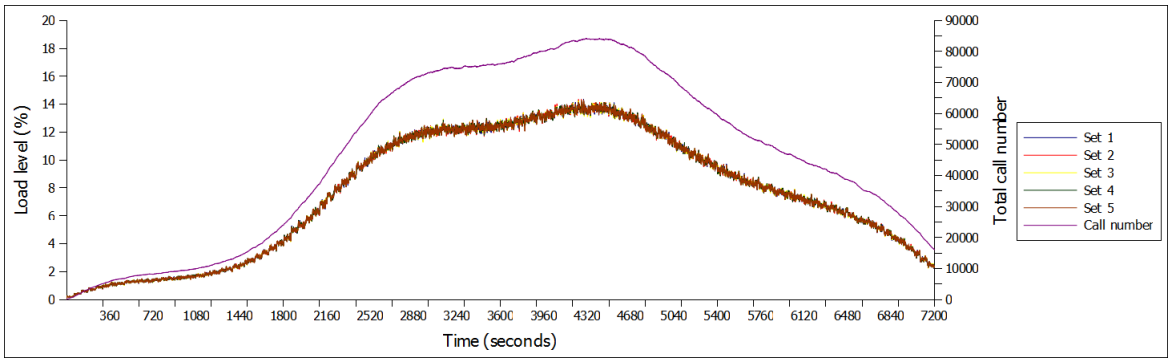


Figure 11 presents the simulation result of Round-Robin, 2 hours resolution

4.2 Results for HSA

In this case, we are able to see that the algorithm works as planned (Figure 12). In every situation, there are three states a set can assume:

1. inactive,

2. activated and working with load under the design level,

3. activated and working on the design level.

However the second state which is the less desired state for a set is always unique in the sense that at any time only one group of FUs can assume that state, the others are either inactive or working at the design limit.

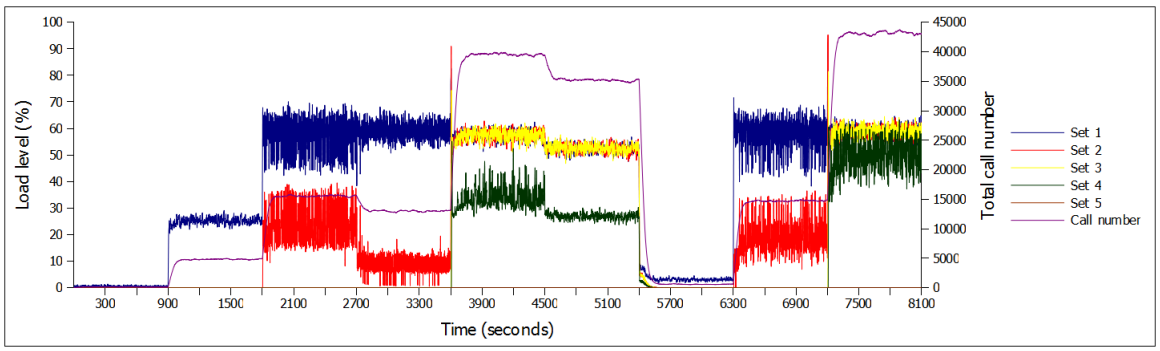


Figure 12 presents the simulation result of HSA, 3 hours resolution

This provides us with sets operating mostly around the design limit which was set to be 60%[4]. There are a few situations we should investigate each one associated with a typical change in load. This first result presents the quick responses the system gives since the data resolution was low thus the changes were sudden.

The first case we should take a closer look at is when the load of groups passes the upper threshold. Take the first occurrence for example when the load suddenly grows high, pushing the first group of FUs over the threshold which triggers the activation of a second set. After this we can see that both sets' load fluctuates with a higher amplitude. Knowing the algorithm this is caused by the fact that the lower set's load was around the lower threshold which makes the pair of sets to trade a part of their priority numbers back and forth. In fact this is not a problem though since the load of the first set is in close range to the design level and no increase in load goes beyond 70%.

In the next interval the load decreases causing the second set to operate under the lower threshold but it remains active and it even decreases the load fluctuations. This is again a consequence of the circumstances, basically the same happens as before with the difference that now the second set gets changed into inactive and back to active repeatedly.

[4]This is not a setting though it can be achieved by setting the upper threshold to 65%.

The other interesting case is when the load is growing or decreasing very suddenly (Figure 12). In the first case multiple sets are activated but the sudden increase creates a rapid overload in some of those. But the activation of sets and increasing the preference number solves the problem. This is a problem that did not appear with the Round-Robin algorithm. The second case where the load drops suddenly the logic turns all but one to inactive without any problem.
In the case of smaller load, only fewer sets are activated (Figure 13) which provides a better utilization of resources: Set 1 (blue line) is loaded until it reaches the upper threshold; Set 2 (red line) is activated only after Set 1 passes the threshold.

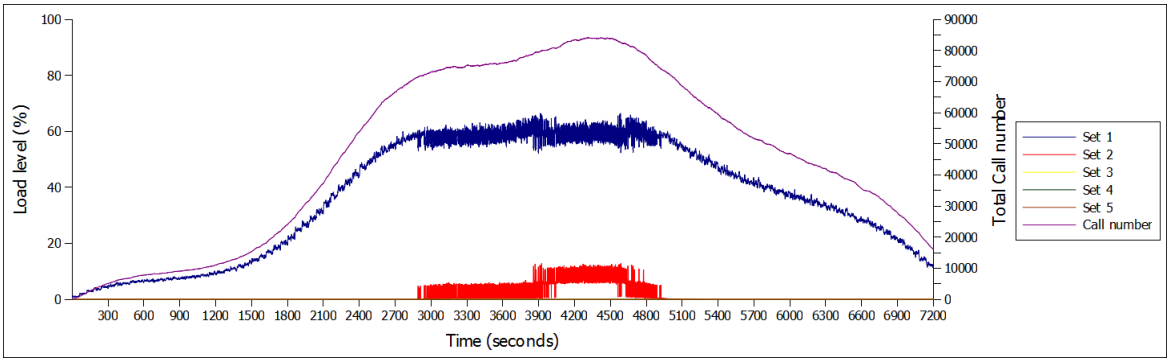


Figure 13 presents the simulation result of HSA, 2 hours resolution

4.3 Differences between the results

The results presented above show that the new algorithm has a real potential for allowing certain groups of FUs to become gracefully shut down. The main reason for that is the fact that without the need they will not even be started. The sets which are activated and later deactivated will not automatically reach the graceful state since they can contain ongoing calls and those can not be terminated. In the simulations calls of extreme length did not occur therefore in most cases inactive sets reached the graceful state.
As it was already explained, most of the results show that the new algorithm is more effective and it meets the criteria that we set as our goals at the beginning. The only questionable area is overload handling where the Round-Robin algorithm seems more capable but in fact that is the case only because in the simulations loads while Round-Robin distribution was applied were very low. In fact most of the time the groups of FUs did not even reach the other algorithm's lower threshold. So the question remains what could be the difference when the load problem's origin is the fact that the number of

requests is too high? If we have all of our available sets around the maximally approved load level[5] than the Round-Robin algorithm would choose target without considering the availability of the possibilities. However in this case the HSA algorithm has groups with identical preference numbers. To avoid the problem that appeared in the Round-Robin case we can add additional weighting to the preference numbers using the free capacity rate of the targets. With this the chance of being able to receive a call and the chance of getting a call request will be correlated. This means that in a constant big load situation the HSA algorithm would still perform better than the Round-Robin.

4.4 Suddenly growing load

```
Running simulation with Round-Robin algorithm...done:
        Total number of accepted calls: 56471
        Total number of rejected calls: 10284
        Reject percentage was:  15.4%
Running simulation with Priority based algorithm...done:
        Total number of accepted calls: 56366
        Total number of rejected calls: 10389
        Reject percentage was:  15.6%
```

Figure 14 presents data for overload situation – many call rejected in both cases

Above we present the results of an artificially created situation (FUs can only handle 30 half-calls at the same time), where suddenly growing load drives the system into severe overload. As Figure 14 shows, the number of rejected calls is higher for the HSA algorithm. This is not suprising though we expected that the overload handling needs further investigation. The problem here is that while the Round-Robin algorithm behaves like a system with groups of FUs that have the same preference number (Figure 15), for the HSA driven system (Figure 16) it takes some time to assume that state[6].

[5] In our setting it was the load level above which the FUs or groups of FUs had to refuse call requests. This value was set as 95%.

[6] Call number on the figures represents only the accepted calls (teal line).

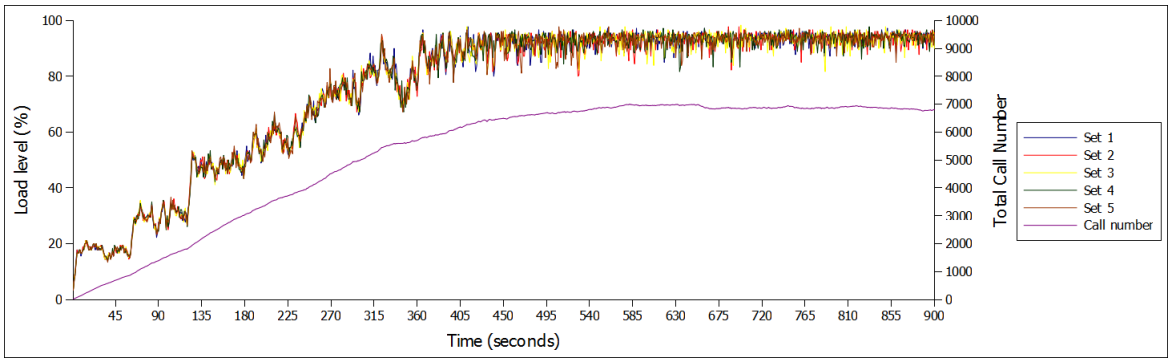


Figure 15 presents the simulation result of Round-Robin during overload

The solution could be that when a call is rejected and not all sets are activated or their preference number is not the same the decision algorithm should intervene to make the system more susceptible for the load.

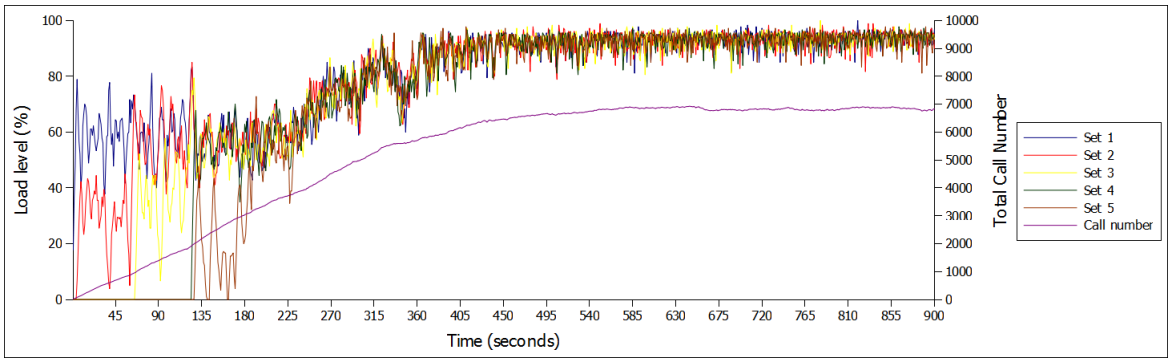


Figure 16 presents the simulation result of HSA during overload

Although the call rejection rate was higher for the HSA, it is also visible that in the overload section the two sytems behave very similarly.

5 Conclusion

The authors would like to emphasize that the newly developed Horizontal Scaling Algorithm (HSA) is planned to be implemented in all Nokia Application Server in the future.

5.1 Development history

The idea of Biased Load Balancing was presented at HTE Infokom 2014 (Kecskemét, Hungary, `http://www.hte.hu/web/infokom2014`) [13]. A patent was also filed by Nokia in November 2014 [14]. Than there was a proof-of-concept simulation in June 2015.

5.2 Improvement ideas for the logic

There are a few points where the HSA could be improved, the first one is the problem of changing active statuses and preference numbers back and forth for a longer time period. This is a performance issue for the possible application since no action needed is more efficient than any kind of action being executed. We believe that improvements are possible by changing the thresholds and the associated logic. The effect of this issue however is lower than the importance of the next ones.
The second problem is that extremely long calls could prevent groups reaching the gracefully shut down state. There are some ideas for the solution, where the most likely candidate is a two solutions in one good solution type. First of all the virtualization allows the handling of FUs independently therefore it would be natural to apply the same decision logic for the functional units in a group this combined with being able to transfer ongoing calls between FUs could also improve efficiency by helping more units to reach the graceful state.
The third problem is the overload caused by sudden load growth. The idea for this one is rather straightforward, there should be a selected watcher that could trigger the re-evaluation of the preference numbers on the right level (for FUs in the group or for groups depending on the level of the problem). The watcher could be waiting for call rejection, or a breach of a threshold.

5.3 Improvement ideas for the simulation

The simulation's main problems are performance concerns which did not receive our fullest attention since the main goal was to create a working simulation. The remaining tasks are linked to the investigations concerning the aforementioned problems. like obtaining even more data from each simulation (like graceful flag data) and creating simulations for interesting use-cases. The only big improvement remaining is the realization of the other call length models, however that should not change the basic behavior of the algorithms.

Acknowledgements

The authors would like to thank *Gábor Járó, Ph.D.* and *Dr. Tien Van Do* for their help and useful comments.
The authors also acknowledge employees of Nokia Hungary Ltd.: *Attila Hilt, Ph.D.*, *László Jánosi*, *Gergely Csatári* and *Gyula Bódog* for their kind help and useful comments. In addition, authors also thank *Attila Hegedűs* (also from Nokia) for his help with the compiling of the simulation's source code.

References

[1] Sagar Dhakal, Majeed M. Hayat, Jorge E. Pezoa, Cundong Yang, David A. Bader: "Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach", *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 18, NO. 4*, April 2007

[2] Parveen Jain, Daya Gupta: "An Algorithm for Dynamic Load Balancing in Distributed Systems with Multiple Supporting Nodes by Exploiting the Interrupt Service", *International Journal of Recent Trends in Engineering, Vol 1, No. 1*, May 2009

[3] Nico Janssens, Xueli An, Koen Daenen, Claudio Forlivesi: "Dynamic Scaling of Call-Stateful SIP Services in the Cloud", Lecture Notes in Computer Science Volume 7289, pp 175-189, 2012

[4] Gergely Csatári, Tímea László: "NSN Mobile Core Network Elements in Cloud, A proof of concept demo", *Proc. Of IEEE International Conference on Communications, Budapest, Hungary*, 9-13 June 2013

[5] ETSI GS NFV 002, Network Functions Virtualisation (NFV), Architectural Framework, *V1.1.1*, 10. 2013
`http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf`

[6] Nokia Solutions and Networks: "Open Telecom Application Server", *Product Description, DN09141866*, 11. 2013

[7] Nokia Solutions and Networks: "Telco Cloud Operation and Maintenance", *System Description, DN09157808*, 2014

[8] 3GPP, TS 23.002, 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, Network architecture, *Release 12, V12.5.0*, 06. 2014

[9] 3GPP, TS 23.218, 3rd Generation Partnership Project, Technical Specification Group Core Network and Terminals, IP Multimedia (IM) session handling, IM call model, Stage 2, *Release 12, V12.3.0*, 09. 2013

[10] 3GPP, TS 23.228, 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, IP Multimedia Subsystem (IMS), Stage 2, *Release 12, V12.5.0*, 06. 2014

[11] Athanasios Papoulis, S. Unnikrishna Pillai: "Probability, Random Variables and Stochastic Processes", *4th edition, McGraw Hill*, 2002

[12] Pedro O.S. Vaz de Melo, Leman Akoglu, Christos Faloutsos, Antonio A.F. Loureiro: "Surprising Patterns for the Call Duration Distribution of Mobile Phone Users", *Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science Volume 6323, pp 354-369*, 2010

[13] István Bakos, Gyula Bódog, Attila Hilt, László Jánosi, Gábor Járó: "Resource and call management optimization of TAS/MSS in Cloud environment (in Hungarian), Infokom'2014 Conference, Hungary, Oct. 2014

[14] István Bakos, Gyula Bódog, Attila Hilt, László Jánosi, Gábor Járó: "Optimized resource management in core network element on Cloud based environment", Patent PCT/EP2014/075539, Nov. 2014

List of Abbreviations

2G	2nd Generation Mobile Network
3G	3rd Generation Mobile Network
3GPP	3rd Generation Partnership Project
AS	3rd Application Server
ATCA	Advanced Telecommunication Computing Architecture
ATCF	Access Transfer Control Function
BCF	Border Control Function
BGW	Border Gateway
BSC	Base Station Controller, 2G Base Station Controller
BTS	Base Transceiver Station, 2G Base Station
CAM	Cloud Application Manager
CPU	Central Processing Unit
CSCF	Call Session Control Function
FU	Functional Unit
FCAPS	Network management model categorizing:
	Fault, Configuration, Accounting Performance and Security
GPRS	General Packet Radio Service
HLR	Home Location Register
HSA	Horizontal Scaling Algorithm
HSS	Home Subscriber Server
HW	Hardware
I-CSCF	Interrogating–CSCF
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IP BB	IP BackBone, IP network with high bandwidth
LB	Load Balancer
LBing	Load Balancing
LTE	Long Term Evolution, 4th Generation Mobile Network
MGW	Media Gateway
MME	Mobility Management Entity
MRF	Media Resource Function
MSC	Mobile Switching Center
MSS	MSC Server

NB	Node-B, 3G Base Station
NE	Network Element
OSS	Operations Support Systems
PCRF	Policy and Charging Rules Function
P-CSCF	Proxy–CSCF
PDF	Probability Density Function
P-GW	Packet Data Network Gateway
PSTN	Public Switched Telephone Network
RNC	Radio Network Controller, 3G Base Station Controller
SAE-GW	System Architecture Evolution Gateway
SAN	Storage Area Network
SDN	Software–Defined Networking
S-CSCF	Serving–CSCF
SGSN	Serving GPRS Support Node
S-GW	Signaling Gateway
SIP	Session Initiation Protocol
SMS	Short Message Service
SR	Site Router
SW	Software
TAS	Telecommunication (earlier Telephony) Application Server
UE	User Equipment
VoLTE	Voice over LTE

Appendix

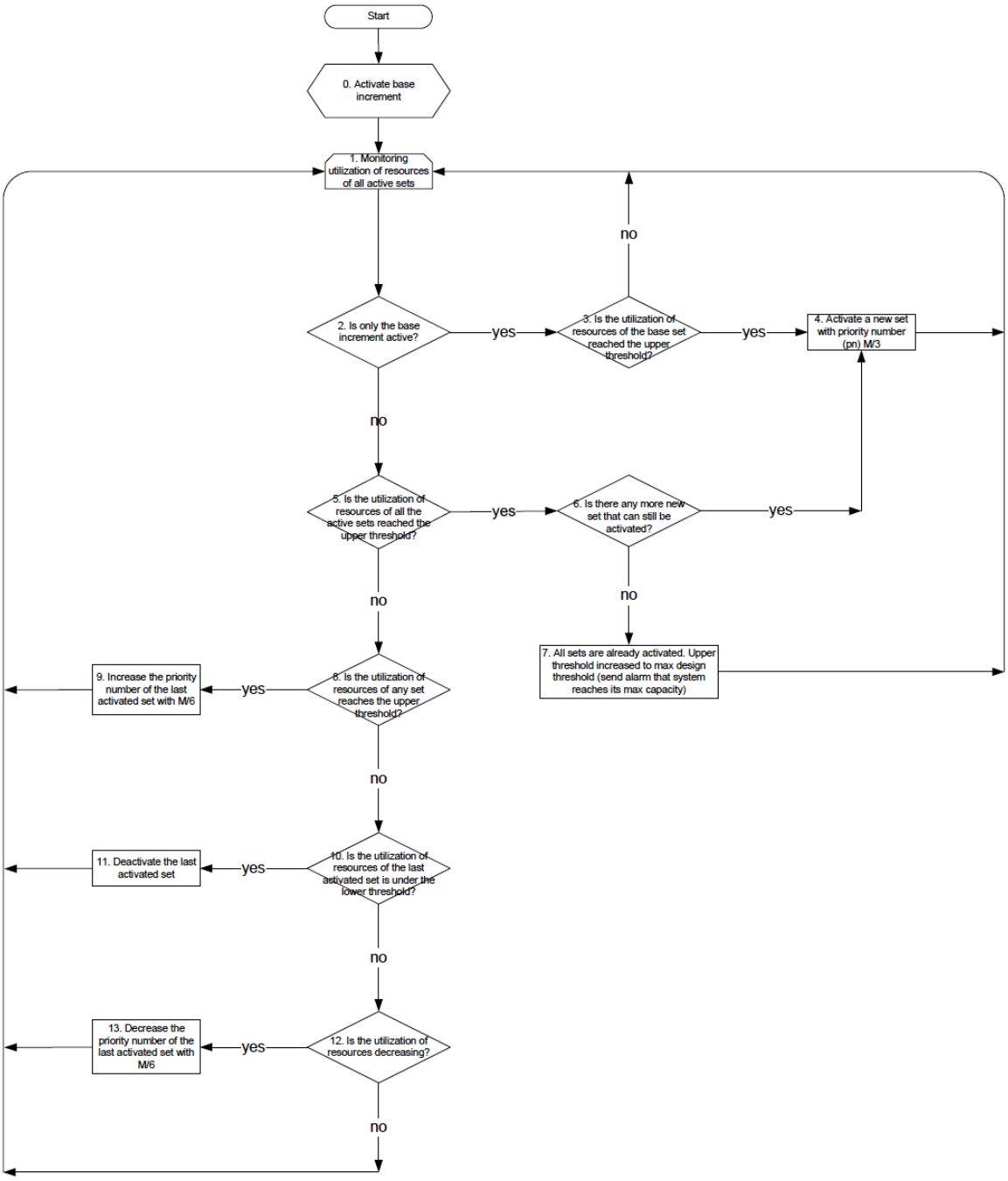


Figure 17 presents the flowchart of the algorithm for the scaling logic

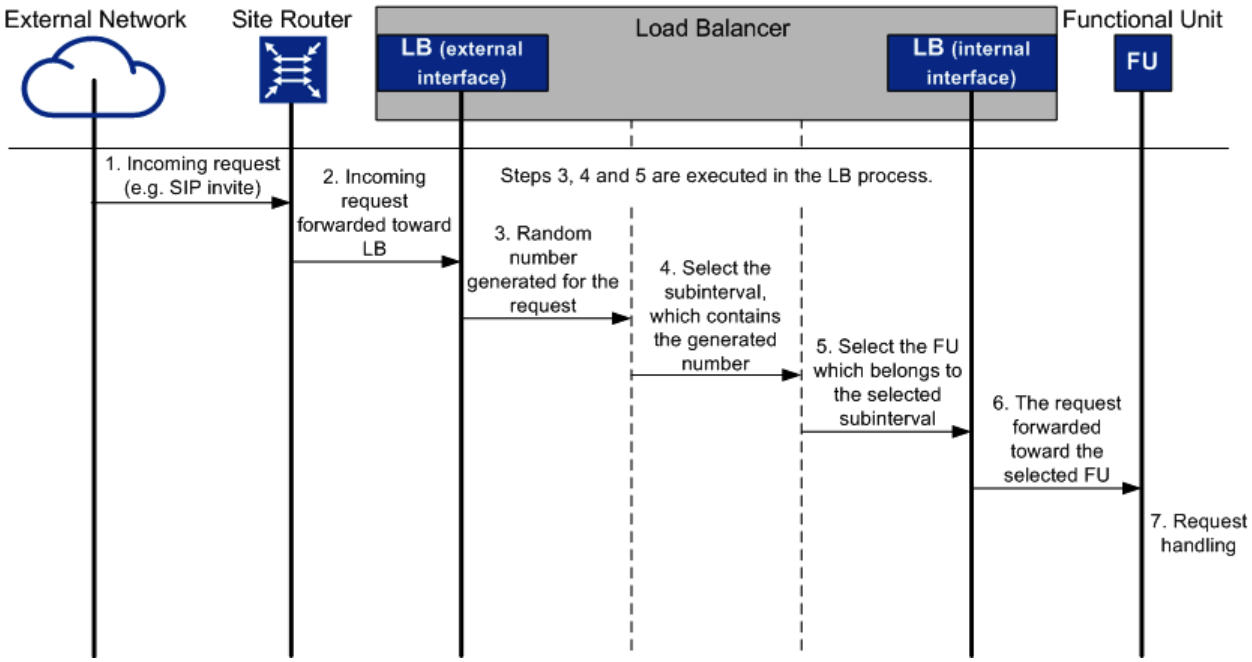


Figure 18 presents the theoretical flowchart of the algorithm for the scaling logic