An On-line Access Selection Algorithm for ABC Networks Supporting Elastic Services

Abstract—The problem of access selection (AS) for multi-access networks has for long been addressed by both the standardization and research communities. As a result, a number of papers have proposed efficient AS algorithms that can take into account radio resource efficiency, overall capacity and quality of service (QoS) requirements in a multi-service environment. However, only a few works have developed on-line AS algorithms that do not require a priori knowledge of the traffic mix when delay sensitive (e.g. voice) and best effort (data) applications are supported. In this work, we present an online AS algorithm that performs well in a multi-access network supporting two service classes and specifically takes into account the elastic nature of data applications. We model AS as a bin-packing problem and realize that the problem is NP-complete. Therefore, we develop a heuristic algorithm called LessDamage that calculates a damage parameter and uses it as a metric for the allocation strategy. Simulation results show that LessDamage performs better in terms of blocking probability and elastic data throughput than available online binpacking heuristics, independently of the number of the available access technologies.

I. INTRODUCTION

Today it is widely recognized that efficient access discovery, selection and traffic steering mechanisms are needed to maximize the overall capacity of multi-access networks. From a research perspective, both the architectural and the algorithmic aspects of multi-access networks have been addressed by a number of recent papers. The notion of Always Best Connected (ABC) networks have been proposed by [11] and [9] and subsequently used as a platform for algorithm design in [15], [18] and most recently in [14] and [12]. Specifically the capacity of ABC networks is modeled and analyzed in [5], [6], [8], [13] and [14]. From these contributions it is clear that the overall radio resource utilization (and thereby the capacity) and the user perceived performance (and thereby the revenue) of multi-access networks are sensitive to AS algorithms. In [15] and [18] it has been argued that in real systems on-line algorithms are needed that allocate users to the available access technologies one-by-one without a-priori knowledge of the overall traffic mix. This problem is obviously more complex than off-line algorithms that assume that all users’ traffic and QoS characteristics are available at the same time and prior to access selection.

Along another line, a series of research contributions have shown that defining and determining the capacity of wireless networks in the presence of elastic traffic becomes complex. This is because elasticity increases the number of admissible application instances at the expense of increasing their residency time and so the net effect of allocating less resources to applications is not trivial [2], [4]. Specifically for wide band code division multiple access (WCDMA) networks, the Erlang capacity increases as traffic becomes increasingly elastic. However, the impact of elasticity in multi-access networks, according to our best knowledge, has not been studied previously.

From a standardization perspective, the 3rd Generation Partnership Project (3GPP) has developed a series of protocols that facilitate access discovery, selection and traffic steering mechanisms for multi-access networks - for a summary of recent developments refer to [1]. The supported access networks include legacy (GSM, GPRS) as well as newly developed technologies such as High Speed Packet Access (HSPA) and Long Term Evolution (LTE) systems.

These research results and standardization progress motivate us to consider a multi-access system that supports on-line access selection for multiple service classes. Specifically, the purpose of our work is to develop an on-line access selection algorithm that is applicable in a multi-access multi-service environment and can explicitly take into account the elastic nature of data applications.

To this end, we extend the on-line bin-packing algorithm such that we allow the size of objects to be specified as a range (rather than a fixed value) that corresponds to the minimum and maximum resource requirements of elastic applications. The contribution of this paper is a heuristic algorithm that has near-optimal performance in terms of blocking probability and elastic application throughput in an ABC environment supporting any number of access technologies and at least one elastic service class.

The next section develops our multi-access network model and relates the access selection problem to bin packing algorithms. Based on these considerations, Section III details the LessDamage heuristic. Section IV evaluates the performance of the LessDamage algorithm through simulation. Section V presents concluding remarks.
The classical bin packing problem is a well studied optimization problem [3, 17], [10], [16]: given \( n \) objects with sizes \( a_1, \ldots, a_n \in (0, 1] \), find a packing in unit-sized bins that minimizes the number of bins used. In the off-line version of this problem, it is possible to consider all the objects and choose the order of assignment. In the online version however, each object must be assigned in turn, without knowledge of the next objects. That is, given \( n - 1 \) already packed objects with sizes \( a_1, \ldots, a_{n-1} \in (0, 1] \), the new object \( n \) with size \( a_n \in (0, 1] \) must be packed in such a manner that the number of used bins is minimized. It is worth mentioning that the problem of finding an optimal packing is known to be NP-Hard [10]. We say that an online bin packing problem is bounded space if the number of available bins at any time is restricted to a predefined number [3]. In the variable-size version of the online bin packing problem the bins can have different capacities [17], and the goal now minimizing the sum of the capacities of the used bins.

In this work we consider a multi-access, multi-service system \( N \) (henceforth mentioned network) consisting one or more subsystems. Just like in [5], each subsystem \( s \in N \) is associated with an access technology \( T \) supporting a set \( K \) of bearer services (or application classes). The subsystems are assumed to have the same coverage area, and the terminal capabilities are assumed to be such that any multimode terminal can connect to any subsystem. A number of application instances \( A = \{a_1, a_2, \ldots, a_n\} \), each one associated with an application class \( k \in K \), arrive to be allocated in \( N \). We denote \( Cap(s) > 0 \) as the capacity of the subsystem \( s \), measured in bits per second (b/s), which is determined by the access technology associated with \( s \) (e.g. GSM/EDGE, WCDMA, Bluetooth, WiMAX, ...). Hence, it is possible to have \( Cap(i) \neq Cap(j) \) for different subsystems \( i \) and \( j \).

Applications arrive one after the other, and there is no a priori knowledge on the next arriving applications. We denote \( Class(a) \) as the application class associated with the application instance \( a \in A \) and \( Size(a) \) as the bandwidth requirement of \( a \), measured in b/s. It is important to note that for a given application instance \( a \), \( Size(a) \) is determined by each subsystem, in such a way that \( Size(a, s) \) is used to denote the bandwidth resource that \( a \) would take from \( s \) if \( s \) was selected to hold \( a \). Assuming \( A(s) \) as the set of application instances already allocated to subsystem \( s \), we define \( Free(s) = Cap(s) - \sum_{a \in A(s)} Size(a, s) \) as the available resources in \( s \). Thus, an application \( a \) can be allocated to a selected subsystem \( s \) only if \( Size(a, s) \leq Free(s) \) and otherwise rejected (blocked). If successfully allocated to subsystem \( s \), application \( a \) consumes \( Size(a, s) \) resource units from \( Cap(s) \) under its residency time and leaves the subsystem, freeing the resources.

Access selection is formulated as the problem of finding the best way of allocating applications in subsystems in order to minimize the number of rejected applications, i.e., the blocking probability, and maximize system capacity. Given the dynamic nature of this problem, where user applications arrive for immediate utilization of the system, it is clear that only online algorithms are appropriate to deal with it. Thus, we map the problem of access selection onto the bounded space variable-size online bin packing problem where objects are applications arriving and bins are subsystems where these should be packed.

Unlike in classical bin packing where the object size is known before packing is performed, in our problem the size of a given application can not be determined a priori, since it depends on the subsystem that will hold it. Hence, without loss of generality, we adopt existing algorithms to consider the actual size of the objects as the one would have if packed to a specific target bin. That is, the algorithms always consider \( Size(a, s) \) to know the amount of space object \( a \) would take from bin \( s \). A second adaptation made refers to working in a dynamic environment, that is, applications arrive, hold part of network resources for a while and leave the network, freeing the resources. Therefore, any access selection decision must consider the network state at the time of making a decision.

### III. THE LessDamage ALGORITHM

The LessDamage algorithm for access selection is a service-based heuristic that draws its inspiration from the mathematical foundations laid out by [7], which defines the optimal offline strategy that can allocate applications belonging to a supported service class to subsystems, in such a way that the amount of applications served is maximized. The LessDamage algorithm is an evolution from the LessVoice algorithm proposed by the authors in [15] and unlike it, is able to consider more than two supported technologies.

LessDamage captures this behavior by defining a damage index \( \delta_{a,s} \) for a application \( a \) and a subsystem \( s \) and using it to select the best subsystem for a given application. \( \delta_{a,s} \) can be understood as the amount of resources that an application associated to a service class consumes from a specific subsystem, relative to all other supported service classes.

Mathematically, \( \delta_{a,s} \) is defined as in equation (1), where \( a \) and \( s \) are, respectively, the application instance and the subsystem for which the damage index is being computed, \( K \) is the set of available service classes, and \( class(a) \) is the service class associated to the application instance \( a \). \( R(k, s) \) is defined as the ratio between the amount of resources the service class \( k \) consumes if it enters in subsystem \( s \) and the amount of resources available in \( s \). This means that \( \frac{1}{R(k, s)} \) determines how many applications of a given service class can be allocated into a given subsystem.

\[
\delta_{a,s} = \sum_{b \in K, b \neq class(a)} \frac{R(class(a), s)}{R(b, s)} \tag{1}
\]

After computing the damage that every service class will cause in each subsystem, the subsystems are then sorted in ascending order according to the damage index, defining an allocation preference: the smaller the damage, the smaller is
Algorithm 1 LessDamage (input: N, a)
1: \( S \leftarrow \emptyset \)
2: \textbf{for} all \( s \in \mathbb{N} \) \textbf{do}
3: \quad \textbf{if} \( \text{Size}(a, s) \leq \text{Free}(s) \) \textbf{then}
4: \quad \quad \( \delta_{a,s} \leftarrow \text{calculateDamage}(a, s) \)
5: \quad \quad \( S \leftarrow S \cup \{\delta_{a,s}, s\} \)
6: \quad \textbf{end if}
7: \textbf{end for}
8: \textbf{if} \( S \neq \emptyset \) \textbf{then}
9: \quad \( s \leftarrow \) the \( s \) from the \( \langle \delta_{a,s}, s \rangle \in S \) with smallest \( \delta_{a,s} \)
10: \quad \textbf{return} \( s \)
11: \quad \textbf{else}
12: \quad \textbf{return} \( \text{NULL} \)
13: \textbf{end if}

The impact on the system if the application is allocated in the respective subsystem and therefore it should be the best allocation option.

Algorithm 1 shows the pseudo-code for the LessDamage selection algorithm. For every supported subsystem, the algorithm checks if the application instance \( a \) can be allocated. If this is the case, it calculates the damage index \( \delta_{a,s} \) and adds a pair \( \langle \delta_{a,s}, s \rangle \) to \( S \), the set of candidate subsystems for \( a \). After checking all subsystems, the algorithm tests if there is at least one pair in the \( S \) set. If this is the case, it selects the pair that has the smallest damage index \( \delta_{a,s} \) and returns the associated subsystem.

The main reason why LessDamage has shown to be a good heuristic is that not only it attempts to allocate services into the subsystems they will consume less resources, but it also takes into consideration the impact of allocating a service instance to an access technology.

The computational cost of the algorithm is proportional to \( O(\|K\| \times \|N\|) \), where \( \|K\| \) is the number of supported service classes and \( \|N\| \) is the number of different subsystems. It is assumed that the computation of \( R(\cdot) \) is cheap and takes a constant time.

<p>| TABLE I | RESOURCE REQUIRED BY EACH APPLICATION AND ITS DAMAGE |
|------------------|-------------------|-------------------|</p>
<table>
<thead>
<tr>
<th>Application</th>
<th>Used Resource</th>
<th>Damage Index (( \delta_{a,s} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWW</td>
<td>0.01</td>
<td>0.015</td>
</tr>
<tr>
<td>VoIP</td>
<td>0.02</td>
<td>0.0000</td>
</tr>
<tr>
<td>Voice</td>
<td>0.02</td>
<td>0.010</td>
</tr>
</tbody>
</table>

To better understand how the algorithm works, an example will be given. Table I considers application classes \( K = \{WWW, VoIP, Voice\} \) and subsystems \( N = \{A, B, C\} \). The Used Resource column indicates the ratio between the resources that are consumed by a specific service that enters a specific subsystem and the total capacity of that subsystem. These values are based on [9] and [15] and would be the values returned by the \( R(\cdot) \) function in eq (1). For example, assuming that the system is empty, \( R(\text{Voice}, A) = 0.01 \), which means that a voice call (that is, one voice application) consumes 1% of the system A’s total capacity or, in other words, that 100 Voice applications can be allocated into A. After an application enters, \( R(\cdot) \) will be recomputed taking into consideration the resources already consumed. The Damage Index column was built by computing the damage index for every service class in every subsystem. An example of the damage index computation for VoIP in A follows:

\[
\delta_{\text{VoIP}, A} = \frac{R(\text{VoIP}, A)}{R(VoIP, A)} + \frac{R(\text{VoIP}, A)}{R(\text{Voice}, A)} = 0.02 + 0.02 = 3.0
\]

IV. PERFORMANCE EVALUATION

We study the performance of LessDamage and other selected algorithms by means of simulation in terms of the class-wise blocking probability and throughput. All curves shown in the graphics are based on the average of a set of 10,000 replications, and the adopted confidence level for the mean was 95%. Each replication represents a simulation of 12500 seconds of operation of the network, the necessary time to reach the steady state of the network for the adopted metrics.

A. Simulation Parameters

We consider three dynamic scenarios in which applications of two classes, voice call and elastic data applications, arrive to be allocated in the network. Table II shows the adopted applications and subsystems while Table III describes the resource requirements for each application and the system capacity for the selected access technologies. The numerical values used for the simulation parameters are in the right order of magnitude, but they are not an exact model. Nevertheless, the values can be used to expose the general properties of the proposed algorithm. It is assumed that, for every scenario, the users are within the coverage area of all available technologies and that the user terminals are capable of accessing any of the technologies. The applications are modeled as follows:

- **Voice Call**: Constant bit rate (CBR), remains inside the network until service time ends. Call duration is modeled as an exponential random variable (RV) with a mean value of 180s. When a subsystem does not support Voice (e.g. WLAN), it is converted into a VoIP application, which is modeled as CBR application with rate 50kb/s.
- **WWW Session**: Elastic sessions. Web page size is modeled as an uniform RV with a mean of 60 kB. Residency time depends on the time needed to transfer the page using the amount of resources allocated for the application in the selected subsystem.

For all scenarios, different input mixes and input loads were used. The input mixes define the proportion between.
the amount of Voice and WWW sessions that will request entrance in the network. The chosen mixes were 75 × 25, 60 × 40, 50 × 50, 40 × 60 and 25 × 75, where the numbers used to describe the mix represent the proportion, in percentile, of Voice sessions and WWW sessions, respectively. The applications arrive as a Poisson process with parameters \( \lambda_V \) for Voice and \( \lambda_W \) for WWW. For each selected mix \( m \), each application class \( k \in \{\text{Voice}, \text{WWW}\} \) and each scenario \( e \in \{1, 2, 3\} \), we define a set of ten different input loads \( t_{m,k,e} = \{l_0, l_1, \ldots, l_9\} \) in such a way that \( l_0 \) is the required \( \lambda \) for reaching a Blocking of 0.1% after simulating the scenario \( e \) with mix \( m \) using the LessDamage algorithm. The remaining loads are defined increasing the \( \lambda \) of previous load with 5% as in the recurring relation: \( l_n = l_{n-1} + 5\% \), \( 0 < n < 10 \). Table IV shows the Input Load \( l_0 \) for Voice and WWW for all five service mixes in all three scenarios.

Besides LessDamage, other algorithms were compared to each other: the classic bin packing heuristics FirstFit, BestFit and WorstFit, a random allocation algorithm - RandomFit - and a consumption based algorithm called LessResource [15].

The algorithms are described as follows:

- **FirstFit**: a subsystem \( s \) is randomly selected with equal probability among \( N \). The application instance \( a \) is allocated to \( s \) if there is enough space for \( a \) in \( s \), that is, if \( \text{Size}(a, s) \leq \text{Free}(s) \) holds. Otherwise, the next subsystem is selected in a round robin fashion until a new subsystem \( s \) in which \( a \) fits is found.

- **BestFit**: a subsystem \( s \) is selected if there is enough space available for the application \( a \) and if by allocating the application, there will be less free space left in the subsystem \( s \) compared to the others.

- **WorstFit**: a subsystem \( s \) is selected if there is enough space available for the application \( a \) and if by allocating the application, there will be more free space left in that subsystem compared to the others.

- **RandomFit**: a subsystem \( s \) is randomly chosen with equal probability for all available subsystems. If there is enough space for the application \( a \), it is allocated. Otherwise, it is rejected.

- **LessResource**: a subsystem \( s \) is selected if \( R(\cdot) \) returns the smallest value among the available subsystems. If there is more than one subsystem with minimum \( R(\cdot) \), then one of the subsystems with the smallest value is randomly chosen. LessResource is proposed in [15].

In order for the algorithms to take elastic applications into consideration, a “shrinkingley” must be employed to allocate an application when there is no more resources left in any subsystem. We adopt the same policy used in [15], which ensures that elastic applications reduce their throughput equally, leading to a fair share of the available resources. Further, when an application leaves the system, the released resources are redistributed among the remaining elastic applications.

Two metrics were used to analyze the performance of the algorithm, namely Throughput and Blocking. Throughput is defined as the percentage of the maximum throughput available for the elastic applications that is being effectively used. In other words, a Throughput of 100% means that the elastic applications are operating at their maximum. It is expected that the “smarter” the access selection algorithms is, the higher is the average throughput achieved by the data applications.

**Blocking** is defined as the ratio between the number of applications that were not allowed to enter the network due to resource exhaustion and the number of applications that requested entrance, measured in percentile.

### B. Numerical Results

As a general result, LessDamage was shown to always perform better than the other algorithms. The performance difference becomes more apparent with the increase in the number of supported technologies. We observed that LessDamage’s curves for both blocking and throughput under different mixes and loads kept roughly the same shape, changing only the values. However, this is not always true for the other algorithms, whose values oscillate in different mixes and loads.

Due to lack of space, we chose to show and discuss here only the figures with the most significant results.

Figures 1 and 2 show the blocking probability obtained by all algorithms in all mixes under a moderate load. Note that there is not a very significant difference in the performance obtained by LessDamage when the input mix is altered and this behavior was observed in all scenarios under every load. Also, as the number of WWW sessions increases, the blocking
probability decreases, since there is more bandwidth to be shared from these elastic applications.

From the aforementioned figures, it can be seen that in Scenario 1, BestFit performed similarly to LessDamage, whereas in scenario 3 this role is taken by LessResource, which was actually the worst algorithm in Scenario 1. This emphasizes that the performance of the other online algorithms will depend on the system configuration and on the input parameters, contrary to LessDamage which has a more consistent behavior.

The throughput achieved by data applications is shown in figures 3 and 4, using a relative scale: a throughput of 100 indicates that the elastic applications where operating using the maximum bandwidth associated with the class. As can be seen from the figures, the throughput obtained by LessDamage is always higher and the difference between throughput values grows with the number of supported technologies. As was the case with the blocking probability, the performance across mixes was also not very different.

It can also be observed that, although BestFit showed a similar blocking probability in scenario 1 compared to LessDamage (see Fig. 1), its throughput was much lower. This was not the case with LessResource in scenarios 2 (not shown) and 3 (Fig. 4), where both metrics were very similar but lower to the ones obtained by LessDamage. The difference between them became more apparent as the load increases.

Figures 5 and 6 show respectively the blocking probability and throughput of elastic applications, obtained by LessDamage in all three scenarios. As can be seen, the blocking probability was higher in Scenario 1 until mix 50% x 50%, when Scenario 3 has an increase in the blocking probability and the behavior becomes similar to Scenario 1. Regarding throughput, it increases with the number of supported technologies and there is practically no difference between the throughputs obtained in scenarios 2 and 3. It is evident the small difference between the metrics’ values in all mixes and all three scenarios and how this difference becomes even smaller with the increasing in the number of supported access technologies.

V. CONCLUSION

This work presented and evaluated an online access selection algorithm called LessDamage which can be used in ABC networks. The algorithm is modeled as a bin packing heuristic that took into consideration the resource consumption relations between the applications (objects) and the network technologies (bins).

The algorithm functions by calculating a damage index, which relates to the relative resource consumption by a service classes in an access technology. Informally, this index indicates how much resources a service $c_0$ takes away from service class $c_1$ in a specific access network technology, and the lower the damage, the more advantageous is to allocate the application in that subsystem.

Performance evaluation showed that regarding the application throughput and blocking probability, LessDamage behaved consistently well across scenarios and the increase in

![Fig. 1. Blocking achieved by the algorithms in scenario 1 under moderate load and different input mixes.](image1)

![Fig. 2. Blocking achieved by the algorithms in scenario 3 under moderate load and different input mixes.](image2)

![Fig. 3. Throughput achieved by the algorithms in scenario 1 under moderate load and different input mixes.](image3)
the number of supported subsystems did not alter consistently the behaviour of the algorithm. Further, the LessDamage heuristic performs well regardless of the number of supported technologies if used to perform access selection for any pair of service classes. The performance of LessDamage was much better in scenarios in which more than two subsystems are present, compared to the other online algorithms. This suggests that it is indeed a scalable access selection algorithm with regards to the number of supported technologies.

ACKNOWLEDGMENT

This work was supported by the Research and Development Centre, Ericsson Telecomunicações S.A., Brazil.

REFERENCES