Research article

UDC 519.872, 519.217
PACS 07.05.Tp, 02.60.Pn, 02.70.Bf
DOI: 10.22363/2658-4670-2021-29-1-36-52

A simulator for analyzing a network slicing policy with SLA-based performance isolation of slices

Nikita A. Polyakov[1],
Natalia V. Yarkina[1], **Konstantin E. Samouylov**[1, 2]

[1] *Peoples' Friendship University of Russia (RUDN University)*
6, Miklukho-Maklaya St., Moscow, 117198, Russian Federation
[2] *Federal Research Center "Computer Science and Control"*
of the Russian Academy of Sciences (FRC CSC RAS)
44-2, Vavilov St., Moscow, 119333, Russian Federation

(received: February 24, 2021; accepted: March 12, 2021)

Efficient allocation of radio access network (RAN) resources remains an important challenge with the introduction of 5G networks. RAN virtualization and division into logical subnetworks – slices – puts this task into a new perspective. In the paper we present a software tool based on the OMNeT++ platform and developed for performance analysis of a network slicing policy with SLA-based slice performance isolation. The tool is designed using the object-oriented approach, which provides flexibility and extensibility of the simulation model. The paper briefly presents the slicing policy under study and focuses on the simulator's architecture and design. Numerical results are provided for illustration.

Key words and phrases: queuing system, resource allocation, network slicing, simulation, optimization

1. Introduction

Network slicing is a key next-generation networking technology that allows multiple virtual subnetworks to be built over a shared physical infrastructure. The virtual subnetworks are then configured to meet the specific needs of applications, services, devices, customers, or virtual network operators. This approach makes it possible to implement in practice flexible configuration and infrastructure management, which make part of the requirements for new generation networks [1]. This concept allows the infrastructure provider to lease network slices to tenants. These relationships are governed by the Service Level Agreements (SLA). Efficient use of network bandwidth and adherence to the terms of these agreements provides economic benefits to all parties. Guaranteeing slice isolation when allocating RAN radio resources makes the problem of efficient resource allocation even more challenging.

The emerging fifth generation (5G) telecommunication networks are envisioned to offer a large number of end-to-end network services for various applications. These stem not only from traditional mobile services, but also from vertical market segments such as automatic driving, unmanned aerial vehicles, telemedicine, massive Internet of Things (mIoT), etc. To provide services with so different requirements for the quality of service (QoS), it is crucial to be able to implement specific virtual subnetworks by using network slicing, since fourth generation (4G) networks with their one-fits-all paradigm are no longer fitted for the task [2], [3].

In this paper, we propose a simulation model as a reusable, versatile tool for evaluating slicing policies for next-generation network resource sharing. The rest of the article is structured as follows. Section 2 presents the system model. In Section 3 we briefly present the slicing policy under study, which was initially proposed by the authors in [4]. Further, it is considered in terms of queuing theory in Section 4. Section 5 explains the architecture of the simulator. The experimental results are discussed in Section 6. Finally, in Section 7, conclusions are drawn and future work is outlined.

2. System model and notation

Following [4], [5], we consider the downlink transmission of a 5G base station (BS) with a virtualized RAN and network slicing. We assume that there are S instantiated slices at the BS and denote their set by $\mathscr{S}$, $|\mathscr{S}| = S$. Let $C_{s[Gbps]} \geqslant 0$ denote the capacity of slice $s \in \mathscr{S}$, so that

$$\sum_{s \in \mathscr{S}} C_s \leqslant C, \tag{1}$$

where $C_{[Gbps]}$ is the total BS capacity. Let N_s denote the number of users in slice $s \in \mathscr{S}$, and let $\mathbf{N} = (N_s)_{s \in \mathscr{S}}$. We assume that each slice is intended for one type of services (e.g., for video streaming, video conferencing, gaming, file transfer, web browsing), and hence the traffic in each slice is homogeneous in terms of characteristics and QoS requirements. Let $R_{s[Gbps]}$ denote the average user data rate in slice s, i.e.,

$$R_s = \frac{C_s}{N_s}, \ s \in \mathscr{S}. \tag{2}$$

The column vector of data rates is denoted by $\mathbf{R}_{[S \times 1]} = (R_s)_{s \in \mathscr{S}}$.

It is assumed that the infrastructure provider (InP) leases parts of its infrastructure in the form of slices to tenants. A Service Level Agreement (SLA) between the InP and the tenant includes the following slice characteristics:

— a minimum average user data rate $0 < R_s^{\min} \leqslant R_s$,
— a maximum average user data rate $R_s \leqslant R_s^{\max} \leqslant C$,
— a guaranteed capacity share γ_s or contracted number of users N_s^{cont}.

We assume that performance isolation of slice s is provided as long as

$$N_s \leqslant N_s^{\text{cont}}, \text{ or equivalently, } \frac{N_s R_s^{\min}}{C} \leqslant \gamma_s, \quad 0 \leqslant \gamma_s \leqslant 1. \tag{3}$$

By performance isolation we understand that traffic fluctuation in one slice does not negatively affect performance in other slices.

3. Slicing scheme

The calculation of slice capacities is performed according to the slicing scheme with SLA-based isolation [4].

Let us partition $\Omega = \mathbb{N}^S$ as

$$\Omega = \Omega^{\max} \cup \Omega^{\text{opt}} \cup \Omega^{\text{cong}}. \tag{4}$$

Now, for $\mathbf{N} \in \Omega^{\max} \stackrel{\text{def}}{=} \{\mathbf{N} \in \Omega : \mathbf{N}\mathbf{R}^{\max} \leqslant C\}$ we set

$$R_s(\mathbf{N}) = R_s^{\max}, \ s \in \mathscr{S} \quad \Longrightarrow \quad C_s(\mathbf{N}) = N_s R_s^{\max}, \ s \in \mathscr{S}, \mathbf{N} \in \Omega^{\max}. \tag{5}$$

For $\mathbf{N} \in \Omega^{\text{opt}} \stackrel{\text{def}}{=} \{\mathbf{N} \in \Omega : \mathbf{N}\mathbf{R}^{\min} \leqslant C < \mathbf{N}\mathbf{R}^{\max}\}$ we determine the data rates as the solution to the convex programming problem

$$\text{maximize } U(\mathbf{R}) = \sum_{s \in \mathscr{S}} W_s(N_s) N_s \ln(R_s), \tag{6}$$

$$\text{subject to } \mathbf{N}\mathbf{R} = C, \tag{7}$$

$$\text{over } \mathbf{R} \in \mathbb{R}_+^S : R_s^{\min} \leqslant R_s \leqslant R_s^{\max}, \tag{8}$$

where $W_s(N_s)$ is given by

$$W_s(N_s) = \begin{cases} 1, & N_s \leqslant N_s^{\text{cont}} \\ N_s^{\text{cont}}/N_s, & N_s > N_s^{\text{cont}} \end{cases} \tag{9}$$

The objective function (6) is differentiable and strictly concave by assumption and the feasible region (7), (8) is compact and convex, there exists hence a unique maximum for the data rate vector R_s, which can be found by Lagrangian methods.

Now consider $\mathbf{N} \in \Omega^{\text{cong}} \stackrel{\text{def}}{=} \{\mathbf{N} \in \Omega : \mathbf{N}\mathbf{R}^{\min} > C\}$. Denote $N_s^{\min}(\mathbf{N}) \stackrel{\text{def}}{=} \min\{N_s, N_s^{\text{cont}}\}_{s \in \mathscr{S}}$. Thus $\mathbf{N}^{\min}\mathbf{R}^{\min}$ is a due capacity. If $\mathbf{N}^{\min}\mathbf{R}^{\min} \geqslant C$, we set

$$C_s(\mathbf{N}) = \frac{N_s^{\min} R_s^{\min}}{\mathbf{N}^{\min}\mathbf{R}^{\min}} C. \tag{10}$$

If, conversely, $\mathbf{N}^{\min}\mathbf{R}^{\min} < C$, then

$$C_s(\mathbf{N}) = N_s^{\min} R_s^{\min} + \frac{(N_s - N_s^{\min}) R_s^{\min}}{(\mathbf{N} - \mathbf{N}^{\min})\mathbf{R}^{\min}} (C - \mathbf{N}^{\min}\mathbf{R}^{\min}). \tag{11}$$

To solve the problem (6)–(8) numerically, we use the gradient projection method (Algorithm 1).

Algorithm 1: Numerical solution of (6)–(8) using the Gradient Projection Method

input : $C, S, \mathbf{N}, \mathbf{R}^{\min}, \mathbf{R}^{\max}, \mathbf{N}^{cont}$
output: $\mathbf{R}$

initialization
$\mathbf{W} := [W_1(N_1), ..., W_S(N_S)]$
$\mathbf{X^{stat}} := \mathbf{W}C(\mathbf{WN})^{-1}$ `// stationary point`
if $R_i^{\min} \leqslant X_i^{stat} \leqslant R_i^{\max}, i = \overline{1, S}$ **then**
  return $\mathbf{X^{stat}}$
$\mathbf{M}_{[1\times S]} := \mathbf{N}$
$\mathbf{P}_{[S\times S]} := \mathbf{I} - \mathbf{N}^T(\mathbf{NN}^T)^{-1}\mathbf{N}$
$\mathbf{X^0} := \mathbf{R}^{\min} + (C - \mathbf{NR}^{\min})(\mathbf{N}(\mathbf{R}^{\max} - \mathbf{R}^{\min}))^{-1}(\mathbf{R}^{\max} - \mathbf{R}^{\min})$
$\tau := ||\mathbf{X^0} - \mathbf{X^{stat}}||; \delta := 1$
while $\delta > 0.0001$ **do**
  $\mathbf{X^1} := \mathbf{X^0} + \tau\mathbf{P}$ `div(`$\mathbf{N}^T\mathbf{W}, \mathbf{X^0}$`)` `// div(A, B) is element-wise division of vector` A `by` B
  $t_{\text{bound}} := 2; t_{\text{coord}} := -1; \delta_+ = 0$
  for $i = \overline{1, S}$ **do**
    if $N_i > 0$ **then**
      if $X_i^1 < R_i^{\min}$ **then**
        if $t_{bound} > (R_i^{\min} - X_i^0)(X_i^1 - X_i^0)^{-1}$ **then**
          $t_{\text{bound}} := (R_i^{\min} - X_i^0)(X_i^1 - X_i^0)^{-1}; t_{\text{coord}} := i$
      if $X_i^1 > R_i^{\max}$ **then**
        if $t_{bound} > (R_i^{\max} - X_i^0)(X_i^1 - X_i^0)^{-1}$ **then**
          $t_{\text{bound}} := (R_i^{\max} - X_i^0)(X_i^1 - X_i^0)^{-1}; t_{\text{coord}} := i$
  if $t_{bound} < 2$ **then**
    $\mathbf{X^1} := \mathbf{X^0} + t_{\text{bound}}(\mathbf{X^1} - \mathbf{X^0})$
    if *Row number of* $\mathbf{M} < S - 1$ **then**
      Add empty row to $\mathbf{M}$
      $\delta_+ := 1$
    Last row of $\mathbf{M} := \mathbf{I}[t_{\text{coord}}]$
    if $||\mathbf{MM}^T|| > 0.0000001$ **then**
      $\mathbf{P} := \mathbf{I} - \mathbf{M}^T(\mathbf{MM}^T)^{-1}\mathbf{M}$
  $\delta := \delta_+ + ||\mathbf{X^0} - \mathbf{X^1}||; \mathbf{X^0} := \mathbf{X^1}$
return $\mathbf{X^0}$

The gradient projection method is a well-known algorithm for solving optimization problems with linear constraints. It is specified by a standard

iterative procedure [6]: $\mathbf{X^{k+1}} = \mathbf{X^k} + \tau \mathbf{d^k}$, where $\mathbf{X^k}$ is the point at which the algorithm arrived at the k-th iteration, τ — the stepsize, $\mathbf{d^k}$ — the increment vector, which is found as the projection of the target function gradient on the constraints: $\mathbf{d^k} = P\nabla U(\mathbf{X^k})$, where the projection matrix is initially given by $P = \mathbf{I} - \mathbf{N}^T(\mathbf{N}\mathbf{N}^T)^{-1}\mathbf{N}$.

4. Queuing system model

We use queuing theory to model the system described in the Section 2. Each slice is modeled as a separate queuing systems (QS). The types of QS must be selected in such a way as to adequately reflect the nature of the service provided. Jobs in QSs correspond to user sessions in slices. Since in the system model the slices are part of a single network of a total capacity C, the S queuing systems share a total resource (capacity) C, which is partitioned so that the resource share available to QS s equals C_s.

At the moment, we have implemented a slice of a Best Effort (BE) type without admission control and with maximum user data rate, which we denote by $BE^{\max}$. It is represented by a QS with the EPS (egalitarian processor sharing) service discipline. The job service rate R_s of all jobs is equal and inversely proportional to their number N_s, but cannot exceed $R_s^{\max}$. Serving jobs in such a QS can be interpreted as downloading files.

Network slicing from this perspective corresponds to a repeated redistribution (re-slicing) of the capacity C among otherwise independent QSs. The considered model is shown in figure 1, where $A_s(x)$ is the distribution law of the interarrival times, $B_s(x)$ is the distribution law of the job lengths (service time on one resource unit) for $s \in \mathscr{S}$.

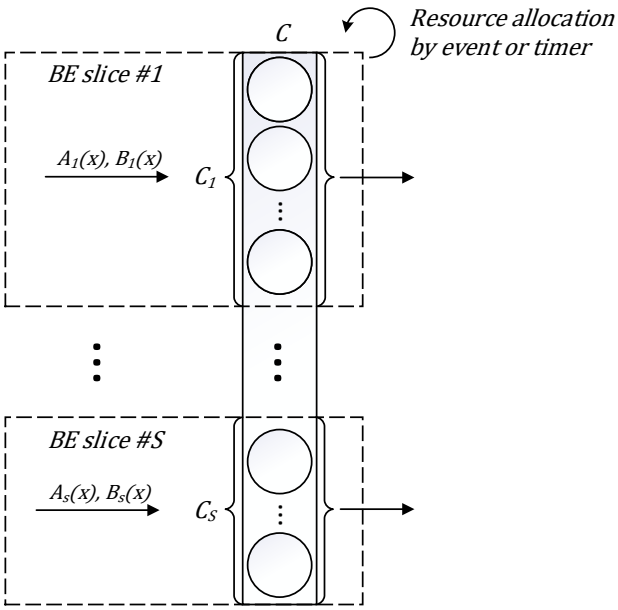

Figure 1. A system with S slices of type $BE^{\max}$

It should be noted that in our model admission control and resource allocation within a slice are individual characteristics for each type of slice. For the $BE^{\max}$ considered in this work, we assume the same service rate for all users (jobs) and unlimited admission (any number of jobs in service). Since this type of slice lacks admission control and queue, it makes sense to introduce a service level degradation threshold ($0 \leqslant R_s^d \leqslant C$) to assess the efficiency of the slicing scheme. This parameter sets the threshold for job service rate in the slice, below which degradation of service occurs, the service is provided poorly. Slice degradation can occur as a result of user arrival and/or redistribution of capacity.

For simplicity, in what follows, the terms slice and queuing system will be used interchangeably.

5. Simulator architecture

5.1. Modules

The discrete event simulator is written on the OMNeT++ platform using the queuinglib standard library. The implementation of the algorithm for solving the optimization problem for the slicing scheme required the inclusion of Boost library for operations with matrices. The construction of a simulation model in OMNeT++ assumes a modular structure, and also allows the use of both standard and modified modules (figure 2).

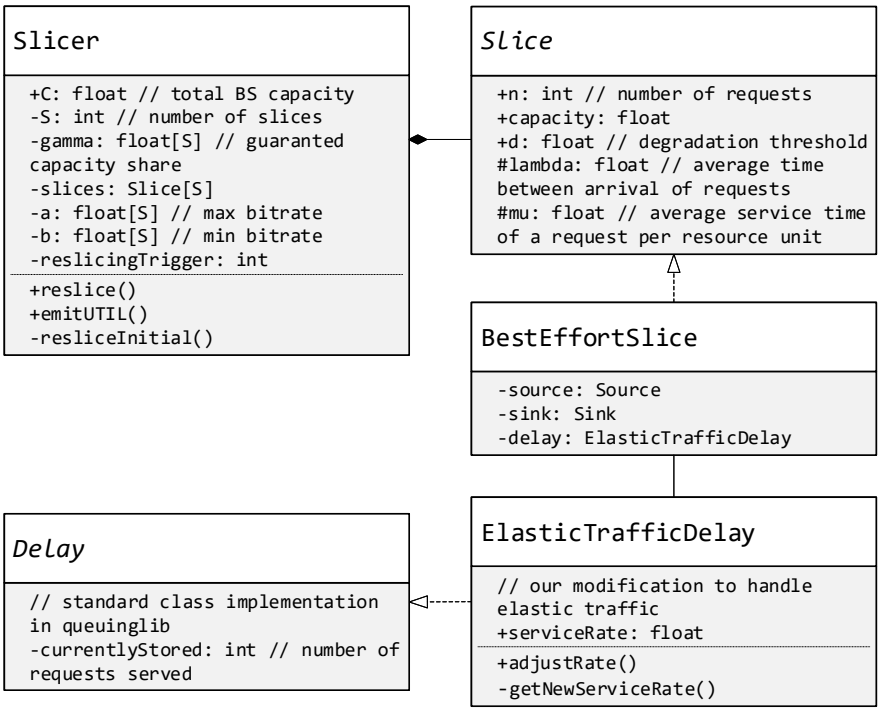

Figure 2. UML diagram of classes developed for the simulator based on the queuinglib standard library

To achieve the required level of abstraction, we have developed two modules:

— is a container consisting of simple modules inside that form a QS. For different types of slices, the way of servicing jobs (users), and as a consequence, the internal structure is not strictly defined and can vary greatly. However, all types of slices are inherited from a common ancestor, which defines the required external parameters that are used to receive the initial data of the model (table 1), and the characteristics that are passed to the slicer as re-slicing parameters, which collected to vectors: $\mathbf{N}, \mathbf{R}^{\min}, \mathbf{R}^{\max}$.

— is a simple module that handles requests for capacity re-slicing from slices. The slicer also performs initial re-slicing of the capacity by formula (12). Slices and slicers communicate via channels — standard OMNeT++ technology.

Let us take a closer look at the table 1. First, the structural characteristics of the model are determined, such as the number of slices and the type of each one. Further, the distribution laws for the arrival of requests and their service time are established, the parameters of the slices are selected, etc.

Table 1

Input data structure

Slicer	
Total capacity C	**float** > 0
S	**int** > 0
Re-slicing trigger	{ All events, Arrivals, Degradation, Timer, Static}
Timer interval t_{timer}	**float** > 0
Slice $i, i = \overline{1, S}$	
$R_i^{\min}$	$0 \leqslant$ **float** $\leqslant R_i^{\max}$
$R_i^{\max}$	$R_i^{\min} \leqslant$ **float** $\leqslant C$
R_i^d	$0 \leqslant$ **float** $\leqslant C$
γ_i	$0 \leqslant$ **float** $\leqslant 1$
Distribution $A_i(x)$	$\{U(a,b), Exp(\lambda), N(a,\sigma^2), \Gamma(\alpha,\beta),$
Distribution $B_i(x)$	$W(k,\lambda), Beta(\alpha,\beta), Cauchy(\theta), Pareto(\alpha), ...\}$

One of the parameters of the initial data is the way of invoking the re-slicing — this is an event or message that occurs periodically during the simulation, which is a condition for invoking the capacity re-allocation algorithm. We consider re-slicing triggered by

— events:
  – all events, i.e., job arrivals and departures (in our case this corresponds to optimal real-time slicing),
  – arrivals only,
  – degradation in any slice;

— timer (every t_{timer} s);
— static slicing (no re-slicing, corresponds to complete partitioning), where the capacity of slice i equals

$$C_i = \frac{\gamma_i}{\sum_{j=1}^{S} \gamma_j} C, i \in \mathscr{S}. \tag{12}$$

Consider the implementation of $BE^{\max}$ slice type. Figure 3 shows a diagram of the correspondence of the QS elements with software modules in a slice, which include the Delay modification — **ElasticTrafficDelay**, and the **Source** and **Sink** modules from the standard set provided by the OMNeT++ and queuinglib bundle.

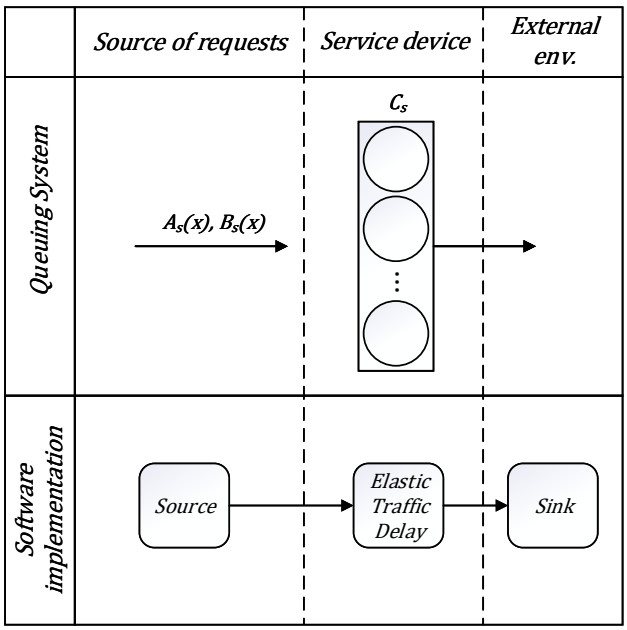

Figure 3. Scheme of logical correspondence of program classes with elements of the $BE^{\max}$ QS

Consider them:

— **Source** is a basic generator of requests that correspond to users' requests for the provision of a service, according to specified distributions.
— **Sink** is a module that receives serviced jobs and destroys them. The Sink collects all the primitive statistics on jobs, such as average, maximum, minimum time spent in the system, average time in queue, etc.
— **ElasticTrafficDelay** extension was written for the standard Delay module. This modification is intended to simulate the service of "elastic" traffic, as the name of the module implies. With the help of standard Delay, you can simulate the service of traffic on discrete devices: after the arrival, the job is in the system for a certain time, and then goes to the drain. ElasticTrafficDelay takes into account the presence of all jobs on the device and equally distributes the available resource between

them. Therefore, the standard module was extended with mechanisms for recalculating the service rate (13) and departure time (Algorithm 2):

$$R_i = \begin{cases} \min(\frac{C_i}{N_i}, R_i^{\max}), & N_i > 0, \\ 0, & N_i = 0, \end{cases} \quad i \in \mathscr{S}. \tag{13}$$

Algorithm 2: Service rate recalculation into $BE^{\max}$ slice i.

```
class Job {
    float t_arr   // is arrival time
    float t_dep   // is departure time
    ...
}
input: R_i, Job[N_i] jobs   // set of jobs in slice i
R_prev := R_i
R_i := getNewServiceRate()   // formula (13)
foreach job in jobs do
    Delete job from event queue
    // t_cur is model current time
    t_serv := |job.t_arr − t_cur|   // how much is already served
    t_serv^new := t_serv * R_prev / R_i
    job.t_dep := t_cur + t_serv^new   // set to job new service end time
    Add job in event queue
```

5.2. Simulation algorithm

Slices, in their essence, function independently of each other, however, as mentioned earlier, the simulator is built on a discrete-event basis, so there is a common queue of events. It contains all the events generated by the model and is executed in the occurrence.

Depending on their type, slices, can generate many different events, but all will be characterized by the following:

— arrival of a job in a slice;
— departure of a job from a slice;
— slice degradation;
— arrival of a job in a slice s with zero resource $C_s, s \in \mathscr{S}$.

Only the events of the model cause a change in the state of the system, which we designated as **N**. Therefore, re-slicing for all events is reduced to tracking the events of arrival and departure of jobs. In our system, the slices themselves notify the slicer of these events (figure 4). After capacity C allocation, the slicer notifies the slices that their available resource $C_s, s \in \mathscr{S}$ has changed. On these notifications, the slices adjust the end time of servicing their jobs in the event queue (Algorithm 2). If there are no jobs $N_s = 0$ in the slice s, then after re-slicing it can be assigned a zero resource value $C_s = 0$, which means that when the first request arrives, it will be necessary to activate the slice, in other words, call re-slicing again.

In the case when re-slicing is triggered by timer (figure 5), the slicer sends messages to itself with the required delay t_{timer} s. Since there is a chance that the slice can receive zero resource, it became necessary to enter the activation of the slice upon the arrival of the request in this case.

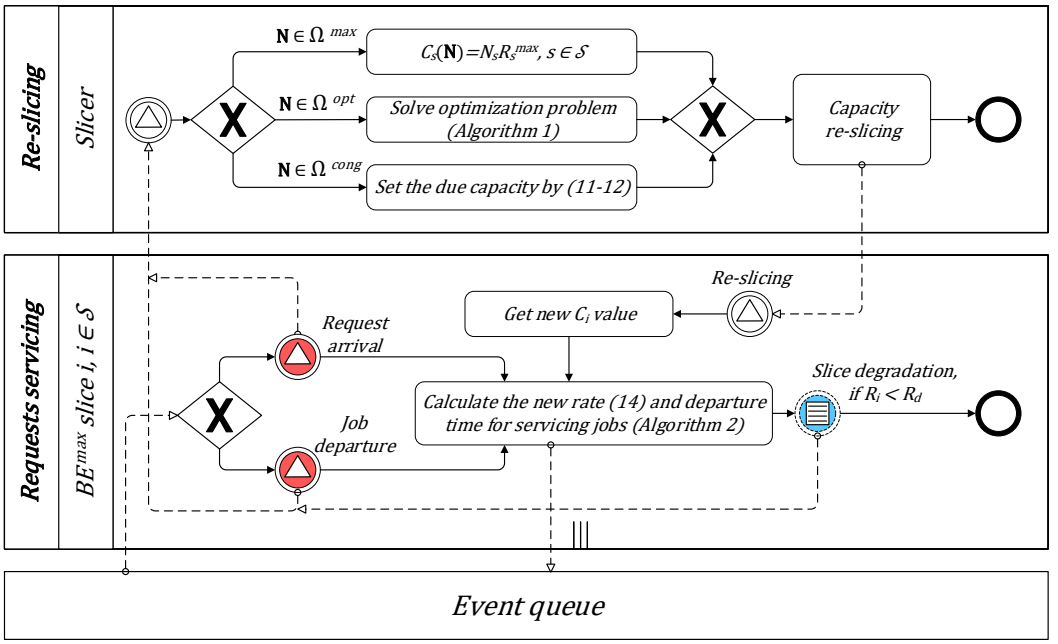

Figure 4. Interaction of slicer and slices when re-slicing triggered by all events (red) or degradation (blue). Re-slicing is called only by a group of events of the same color

With static slicing, the slices receive resource proportionally, in accordance with the values of γ by equation (12).

5.3. Metrics

The built simulator allows you to take indicators in various forms using the built-in OMNeT++ tools, and more specifically using signals and statistics. The signal (@signal) transmits information at the right moments in the form of values of primitive types: bool, int, float, etc., or more complex data objects [7]. Statistics (@statistic) is a signal processing mechanism that allows you to accumulate vectors of original data transmitted by signals and scalars calculated by these vectors: sum, quantity, average, time average, maximum, minimum, etc. Preset simulator settings allow you to take such indicators like:

— average time spent in each module of the constructed QS inside slices and in the network as a whole;
— average number of jobs in each module of the constructed QS within slices and in the network as a whole;
— average service rate in slice;
— average number of jobs in slice, etc.

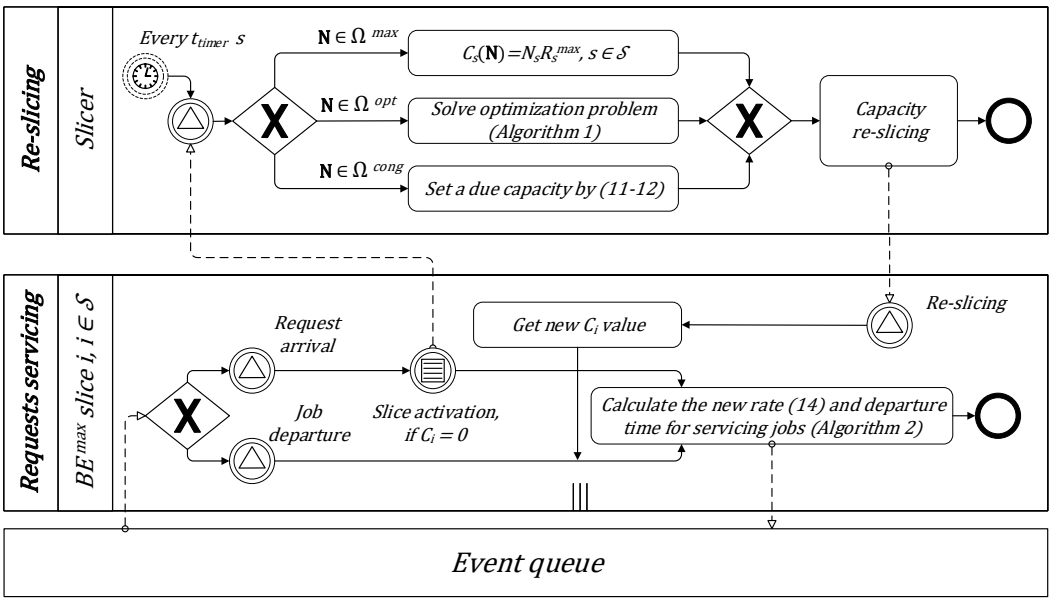

Figure 5. Interaction of slicer and slices when re-slicing triggered by timer

As part of assessing the effectiveness of slicing, the following additional indicators were taken:

— Slice degradation probability,

$$P_s^{\deg} = P\{R_s < R_s^d\} = \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{D_s(T)} (d_{s,i} - d_{s,i-1}) \mathcal{H}\{R_{s,i} < R_s^d\}, \qquad (14)$$

where $s \in \mathscr{S}$, T is model time, $D_s(T)$ — counter of slice s degradation threshold R_s^d crossing (in any direction), $R_{s,i}$ — time of the i-th rate change, $d_{s,i}$ — time of the i-th degradation threshold R_s^d crossing, and $\mathcal{H}$ is Heaviside step function.

— Average slice resource,

$$\overline{C_s} = \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{L_s(T)} (c_{s,i} - c_{s,i-1}) C_{s,i}, \quad s \in \mathscr{S}, \qquad (15)$$

where $L_s(T)$ — counter of slice s resource changes, $c_{s,i}$ — moment i of changing resource C_s.

— Average duration of slice degradation period,

$$\overline{t_s^{\deg}} = \lim_{T\to\infty} \frac{1}{D_s(T)+1} \sum_{i=1}^{D_s(T)} (d_{s,i} - d_{s,i-1}) \mathcal{H}\{R_{s,i} < R_s^d\}, \quad s \in \mathscr{S}. \qquad (16)$$

— Capacity utilization,

$$\text{UTIL} = \frac{1}{C}\sum_{s\in\mathscr{S}}\lim_{T\to\infty}\frac{1}{T}\sum_{i=1}^{Y_s(T)}(y_{s,i}-y_{s,i-1})N_{s,i}R_{s,i}, \quad s\in\mathscr{S}, \tag{17}$$

where $Y_s(T)$ — counter of slice s service rate R_s and number of jobs N_s changes, $y_{s,i}$ — moment i of changing R_s or N_s.

— Re-slicing frequency.

— Average duration of the re-slicing operation.

6. Numerical results

To illustrate the performance of the simulator, we consider five slices with the parameters given in the table 2.

Table 2

Parameters' values for the numerical example

Slicer					
Total capacity C	8000				
S	5				
Timer interval t_{timer}	100s				
Slice i	**1**	**2**	**3**	**4**	**5**
$R_s^{\min} = R_i^d$, Mbps	2	5	25	50	30
$R_s^{\max}$, Mbps	2.2	8	30	75	8000
γ_i	0.075	0.075	0.35	0.25	0.25
$A_i(x)$	$\exp(\lambda)$				
Request interarrival time λ^{-1}, s	1.65	7.25	16	19	5
$B_i(x)$	$\exp(\theta)$				
Mean file size θ^{-1}, GB	0.3	1.2	2.5	5	1

Scenario is intended to demonstrate a system with an increased workload in slices 1 and 2. The guarantees are selected in such a way, that slices 3 and 4 are the main donors of capacity.

Figure 6 illustrates how the degradation probability P_s^{deg} varies depending on the re-slicing triggers for the cases under study. Static re-slicing gives a high degradation probability in slice 1. For event triggers, we observe low degradation probability ($\sim 1\%$) for slices 1, 2, 4 and insignificant degradation probability in 5. When re-slicing is triggered by timer, the slicer reacts to the state of the system with a long delay, so there is an unacceptably high probability of degradation in donor slices 3 and 4.

The capacity utilization metric in figure 7 indicates that re-slicing upon all events and arrivals provide the highest resource utilization and the lowest waste of resources. This would be good if it were not for the fact that at a much lower system utilization, re-slicing upon degradation yields the same efficiency in terms of degradation probability.

Let us take a look at such an important indicator as the frequency of re-slicing calls. Figure 8 additionally confirms the efficiency of re-slicing upon degradation compared to re-slicing upon all events and arrivals, and even by timer. For all triggers, as expected, slicing takes roughly the same amount of time, averaging 0.04 ms.

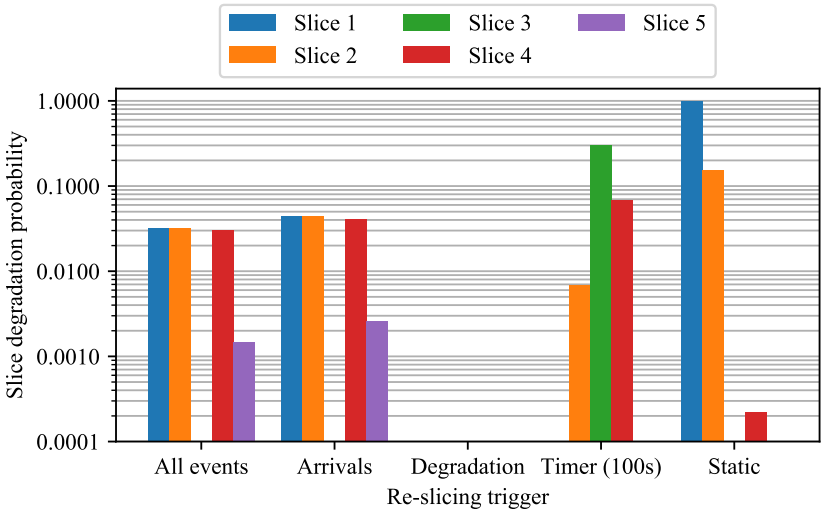

Figure 6. Slice degradation probability for different re-slicing triggers

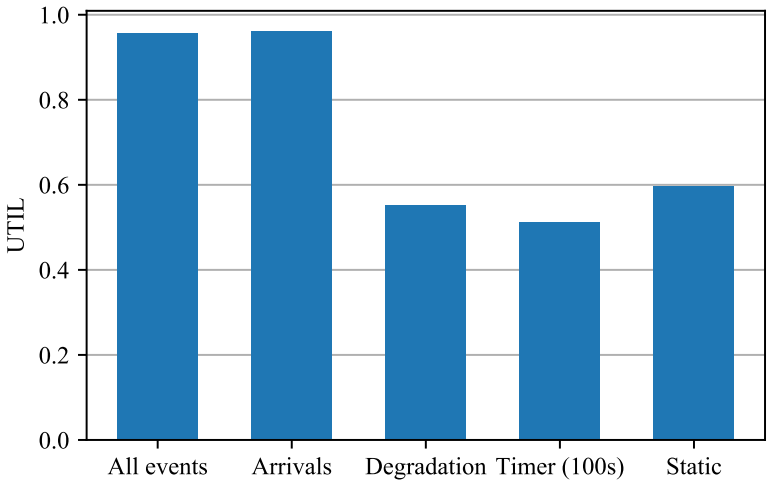

Figure 7. System utilization for different re-slicing triggers

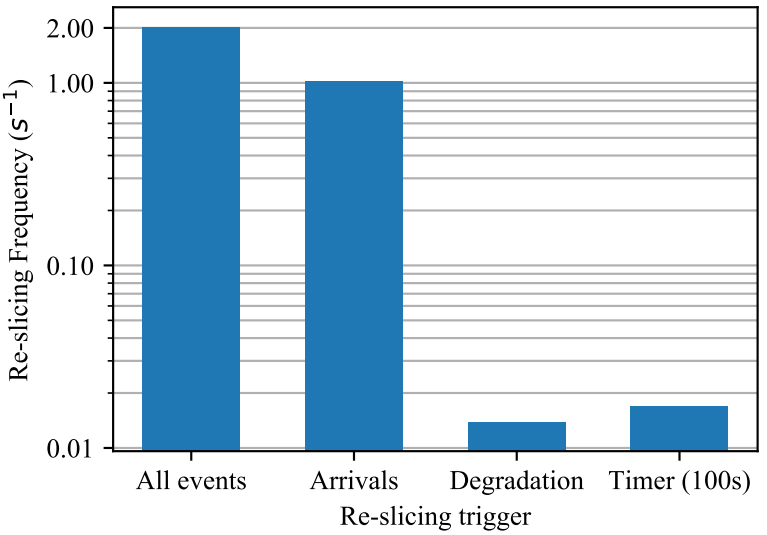

Figure 8. Re-slicing frequency for different re-slicing triggers

Let us consider the average share of capacity $\overline{C_s}$ allocated to each slice depending on the re-slicing trigger (figure 9). As we see, slices 1 and 2 receive significantly more capacity with frequent re-slicing than indicated in the SLA — the scheme allows this.

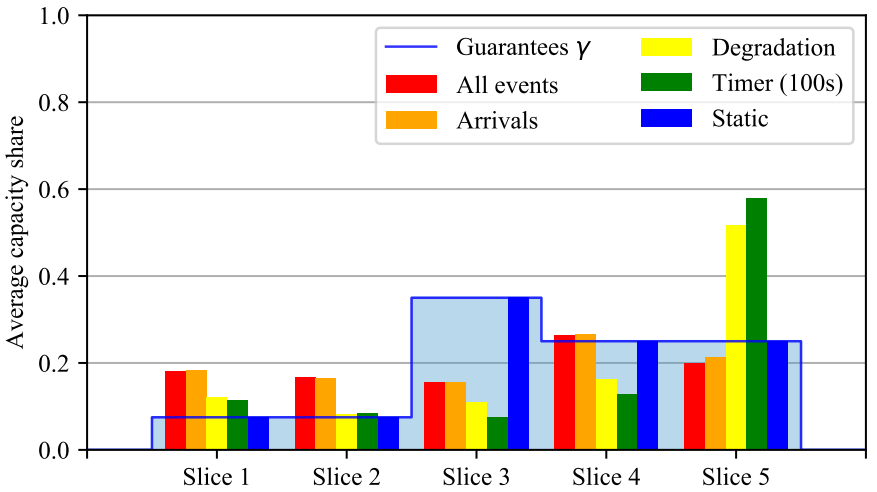

Figure 9. Average slice capacity share for different re-slicing triggers compared with the contracted share γ_s

7. Conclusion

A simulation model of network slicing with SLA-based isolation has been developed. By using the Object-Oriented Programming paradigm [8], as well as the built-in functionality of OMNeT++ and queuinglib, the following principles have been achieved:

— **Modularity of the system**: model elements (slicer, slice, queue, source of requests, delay, etc.) implemented as objects are logically separated, and the interaction among them occurs by transmitting global signals or messages through special channels.
— **Polymorphism, inheritance and encapsulation of slices**: all types of slices have a common ancestor which specifies all the mechanisms necessary for communicating with the slicer, so each descendant class describing a new slice type can replace their implementation with their own without breaking the interaction structure. In connection with the same principle, the QS describing the way of processing users (jobs) within a slice can take any form and be designed at the discretion of the developer. Thus, any slice is characterized only by its type and unified set of parameters.
— **Homogeneity of the structure of the input data**: an important characteristic for any simulator is the ease of use, in particular, the way of specifying the input data. In our implementation, based on the previous principle, the initial conditions for any slice are set in the same way using a configuration file.

Compliance with the indicated principles leads to scalability and extensibility of the simulation model.

Further research objectives:

— taking into account the state of the radio channel;
— adding and analyzing other re-slicing triggers;
— adding other types of slices;
— extensive numerical analysis.

Acknowledgments

This paper has been supported by the RUDN University Strategic Academic Leadership Program. The reported study was funded by RFBR, project number 19-07-00933, 20-07-01052.

References

[1] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: a survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018. DOI: 10.1109/COMST.2018.2815638.

[2] R. Su, D. Zhang, R. Venkatesan, Z. Gong, C. Li, F. Ding, F. Jiang, and Z. Zhu, "Resource allocation for network slicing in 5G telecommunication networks: a survey of principles and models," *IEEE Network*, vol. 33, no. 6, pp. 172–179, 2019. DOI: 10.1109/MNET.2019.1900024.

[3] H. Yu, F. Musumeci, J. Zhang, M. Tornatore, and Y. Ji, "Isolation-aware 5G RAN slice mapping over WDM metro-aggregation networks," *Journal of Lightwave Technology*, vol. 38, no. 6, pp. 1125–1137, 2020. DOI: 10.1109/JLT.2020.2973311.

[4] N. Yarkina, Y. Gaidamaka, L. M. Correia, and K. Samouylov, "An analytical model for 5G network resource sharing with flexible SLA-oriented slice isolation," *Mathematics*, vol. 8, 2020. DOI: 10.3390/math8071177.

[5] B. Rouzbehani, L. M. Correia, and L. Caeiro, "A service-oriented approach for radio resource management in virtual RANs," *Hindawi Wireless Communications and Mobile Computing*, 2018. DOI: 10.1155/2018/4163612.

[6] B. Gladkih, *Optimization techniques and operations research for computer science bachelors. Part 2. Non-linear and dynamic programming [Metody optimizacii i issledovanie operacij dlya bakalavrov informatiki. CH. 2. Nelinejnoe i dinamicheskoe programmirovanie].* Tomsk: Izd-vo NTL, 2011, in Russian.

[7] A. Virdis and M. Kirsche, *Recent Advances in Network Simulation.* 2019. DOI: 10.1007/978-3-030-12842-5.

[8] R. Lafore, *Object-Oriented Programming in C++, 4th Edition.* CourseSams Publishing, 2001.

For citation:

N. A. Polyakov, N. V. Yarkina, K. E. Samouylov, A simulator for analyzing a network slicing policy with SLA-based performance isolation of slices, Discrete and Continuous Models and Applied Computational Science 29 (1) (2021) 36–52. DOI: 10.22363/2658-4670-2021-29-1-36-52.

Information about the authors:

Polyakov, Nikita A. — Bachelor of Science, Master student (e-mail: goto97@mail.ru, phone: +7(916)5858743, ORCID: https://orcid.org/0000-0003-0152-9646, Scopus Author ID: 57211203793)

Yarkina, Natalia V. — Candidate of Sciences, ORCID: https://orcid.org/0000-0003-3197-2737

Samouylov, Konstantin E. — Doctor of Technical Sciences, Professor, applied Mathematics & Communications Technology Institute (e-mail: ksam@sci.pfu.edu.ru, ORCID: https://orcid.org/0000-0002-6368-9680, ResearcherID: E-9966-2014, Scopus Author ID: 14009785000)

УДК 519.872, 519.217
PACS 07.05.Tp, 02.60.Pn, 02.70.Bf
DOI: 10.22363/2658-4670-2021-29-1-36-52

Имитационное моделирование разделения ресурсов с изоляцией слайсов на базе SLA

Н. А. Поляков[1], **Н. В. Яркина**[1], **К. Е. Самуйлов**[1,2]

[1] *Российский университет дружбы народов*
ул. Миклухо-Маклая, д. 6, Москва, 117198, Россия
[2] *Федеральный исследовательский центр «Информатика и управление» РАН*
ул. Вавилова, д. 44, кор. 2, Москва, 119333, Россия

В настоящее время, несмотря на ввод в эксплуатацию сетей мобильной связи 5-го поколения, эффективное разделение ресурсов сети радиодоступа по-прежнему остаётся актуальной задачей. Свои коррективы в её постановку вносят технологии виртуализации и нарезки сети (network slicing), позволяющие разделять сеть доступа на логические подсети. В статье предложен инструмент имитационного моделирования, разработанный на платформе OMNeT++ для анализа эффективности схемы разделения ресурсов с изоляцией слайсов на базе соглашений об уровне обслуживания. Объектно-ориентированный подход к построению симулятора обеспечивает гибкость и расширяемость модели. В статье кратко изложена исследуемая схема слайсинга, подробно описана архитектура программного средства и особенности построения имитационой модели, приведены результаты численного анализа.

Ключевые слова: система массового обслуживания, разделение ресурсов, нарезка сети, имитационное моделирование, оптимизация