PISA UNIVERSITY

PERFORMANCE EVALUATION OF COMPUTER SYSTEMS AND NETWORKS

FAIR-NETWORK
PROJECTS DOCUMENTATION

ACADEMIC YEAR 2019-2020

ANDREA TUBAK, FRANCESCO RONCHIERI, ALESSANDRO MADONNA

SUMMARY

INTRODUCTION

DESCRIPTION

A cellular network transmits its traffic to N users. Each user has its own FIFO queue in the transmitting antenna.

On each timeslot users report to the antenna a Channel Quality Indicator (CQI), i.e. a number from 1 to 15, which determines the number of bytes that the antenna can pack into a Resource Block (RB) according to the table below.

CQI	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bytes	3	3	6	11	15	20	25	36	39	50	63	72	80	93	93

Then the antenna composes a frame of 25 RBs by scheduling traffic from the users and sends the frame to the users. A packet that cannot be transmitted entirely will not be scheduled.

An RB can only carry traffic for one user, however, two or more packets for the same user can share the same RB. (e.g. packet 1 is 1.5 RBs and packet 2 is 1.3 RBs, hence ceiling (1.5+1.3) =3 RBs are required to transmit them).

The antenna serves its users using a least-served first policy: on each timeslot, users are served by increasing order of received data. When a user is considered for service, its queue is emptied, if the number of unallocated RBs is large enough.

Sorting takes place once per timeslot, before the schedule is decided.

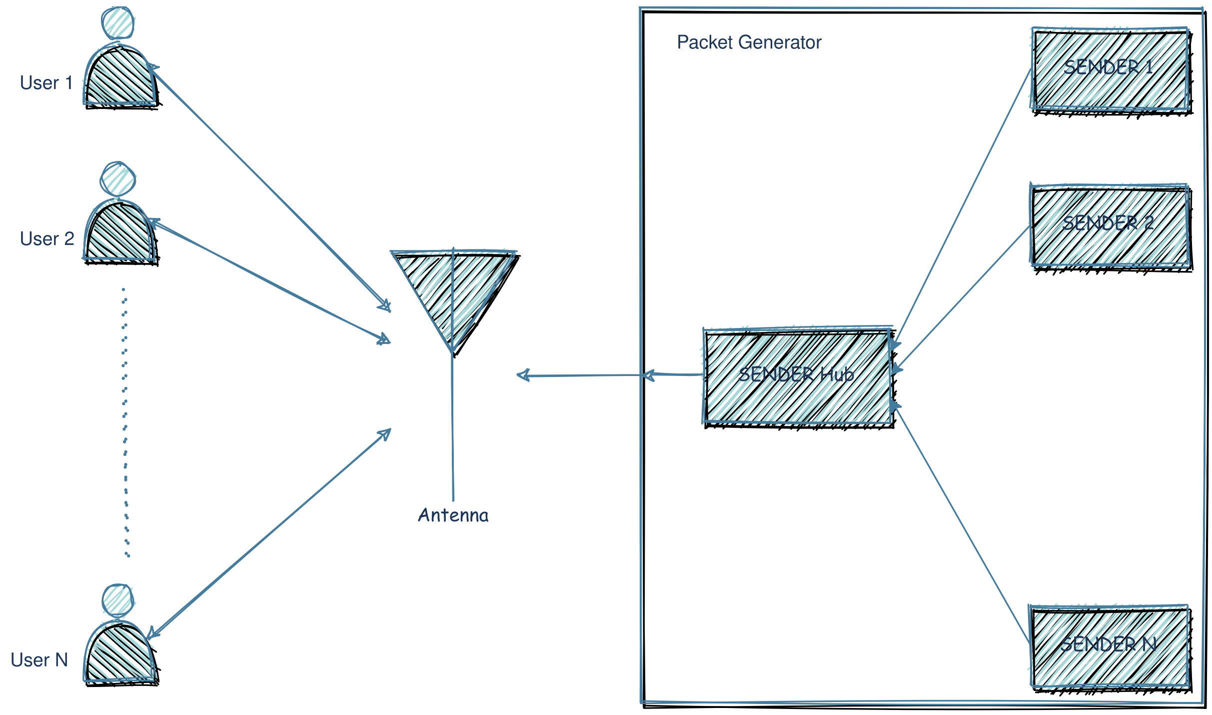
OBJECTIVE

The objective of this project is to study the throughput and the response time of the fair network system with a varying workload in two scenarios:

- ***Uniform*** *configuration*: exponential interarrivals, uniform service demands and uniform CQIs
- ***Binomial*** *configuration*: exponential interarrivals, uniform service demands and binomial CQIs, chosen so that the mean CQI of different users are sensibly different

SIMULATION LAYOUT

The overall architecture of the system that we developed in **OmNet** is the following:



ANTENNA

It is the main component of the system; it does the following:

- Every timeslot it receives a packet from every user containing the CQI for that specific user.
- It sends a frame in each timeslot composed by 25 RB; each RB is going to be allocated in accordance with the documentation. Every frame is sent to all users.
- It receives packets from the packetGenerator and places them in the appropriate user queue.

USER

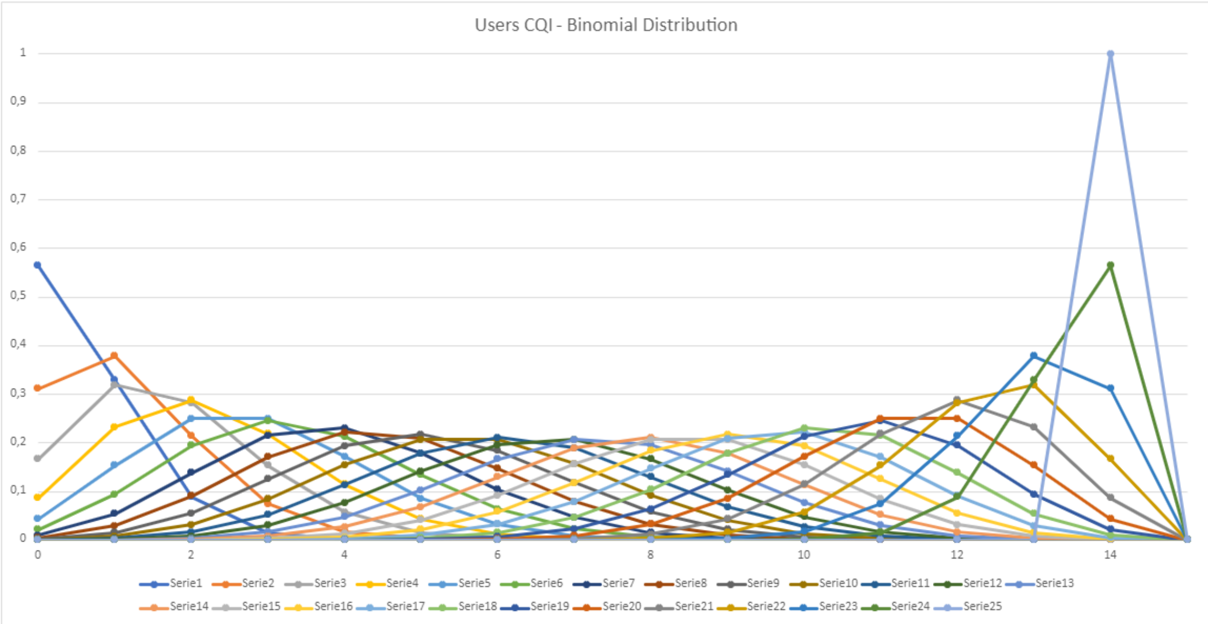
Each user receives a frame per timeslot from the antenna, he will then discard all packets not directed to him. Additionally, he sends a CQI message to the antenna.

The CQI's are generated in different ways, based on the type of the simulations. In the case of binomial simulations, for example, we must have users with mean of the CQI significantly different from each other, to do this we used the following formula:

$$Mean_i = \frac{15}{N_U} * i$$

Where i is the id of the i-th user, N_U is the total number of users in the simulation and $Mean_i$ is the mean of the binomial distribution used to extract the CQI of the i-th user. In this way, the user with low id are the ones with bad channel quality, while those with high id are the ones with good coverage.

Following there is a graph that for N_U equal to 25, show the distribution of the CQI for each user, with the mean chosen in the way explained above.



SENDER

It is a simple component that generates packets. There is one sender for every user. Each sender generates a packet with an interarrival time picked from an exponential distribution with a rate λ - *UserLambda* (more details about it will be given in the User Lambda section). The packet dimension is also random; it is chosen with a uniform distribution ranging from [1, 75] Bytes. The value 75 comes from the documentation in which we can find the following indication: "*The largest packet dimension is such that it fits a frame at the minimum CQI*", hence 25*3 B = 75 B.

SENDERHUB

This module receives the packets from all the senders and then sends them to the antenna.

PACKET GENERATOR

It is a module composed by both the Senderhub module and all the senders of the simulation.

ASSUMPTIONS

Let us go through the choices we made for each relevant component of the simulation:

TIMESLOT

For choosing a reasonable timeslot we took inspiration from 4g networks in which different timeslots are spaced 1ms from one another. This value is constant throughout all simulations.

GENERATION OF PACKETS (USER_LAMBDA)

The *UserLambda* is one of the factors in our simulation. This factor indicates the rate with which packets are generated for a certain user. In order to find a reasonable range of values to study we followed these two ideas:

- For the lower bound we chose a value such that, even with the maximum number of users (see section below), it allows the system to be in a balanced state; not saturated, but not even completely discharged. It ended up being 25.
- For choosing the upper bound we looked for a value such that we had at least one non-saturated configuration among all NUM_USER chosen (see below for more information). The upper bound ended up being 200.

To facilitate the discovery of the appropriate values for *UserLambda,* we carried out an analytical study to identify some rough estimates of the values for the stability of the system.

Once identified the lower and upper bound, we decided to split that range somewhat evenly. The result is the following set of values: {25, 50, 75, 100, 150, 200}

NUMBER OF USERS (NUM_USER)

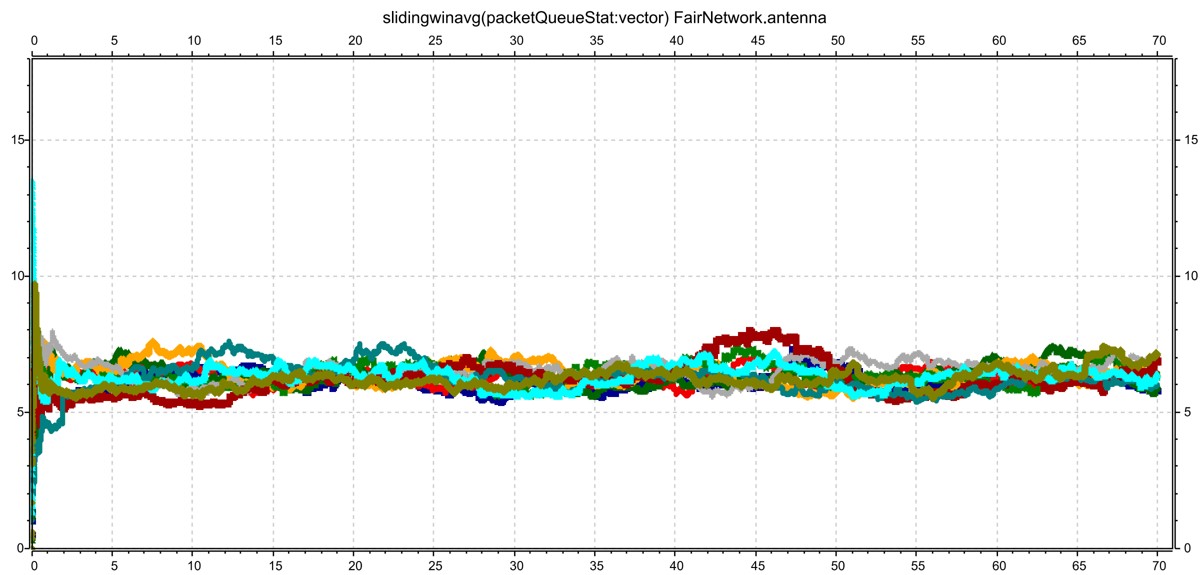
The number of users is the other factor of the simulation, as such we decided to study its effects by varying it across a range of possible values. The values we settled on are: {25, 50, 100, 150, 200}.

The rationale behind this choice is the following:

- For the lower bound we thought that values below 25 would be too small to allow a meaningful study of the fairness of our network. This is because the antenna, under reasonable workloads, can dispatch all the packets without too many problems, bringing the delay close to 0 and the throughput equal to the number of bytes in input.
- For the upper bound we chose a value which stays below an order of magnitude respect the min value.

WARM UP TIME

In our analysis we need to study the system in the steady state, to achieve this we need to identify the time needed to reach it. Shown below is a graph of 10 independent runs of the configuration NUM_USER = 100, USER_LAMBDA = 90. This configuration has a good balance of the workload on the antenna. This plot was made by applying the sliding window average, with window size 1000ms, over the total number of packets queued in the antenna.



As we can see from the graph above the system seems to reach stability around 5000ms, and that's exactly what we chose as the warmup time for all our simulations.

SIMULATON TIME

To select an appropriate simulation time, we studied the evolution of the bytes generated by the Sender modules. We wanted to make sure that any result on the user throughput was not influenced too much by input imbalances between different users. To do so we measured the load generated by each sender across different simulations with increasing amount of simulation time. The following table shows the mean and standard deviation between all senders of the same configuration. The configuration chosen is the one with N_USER = 200 and USER_LAMBDA = 25.

Time	Mean (B)	Std (B)
10	949,73	68,76
20	949,74	50,21
30	947,32	42,21
40	945,90	35,88
50	948,07	32,92
60	947,30	30,17
70	948,94	27,29
80	948,75	25,26
90	948,97	23,35
100	948,37	21,33

As we can see, increasing the simulation time on multiples of 10 seconds causes the standard deviation to decrease just by a couple of bytes each step, while the mean value stays pretty much stable. Those few bytes are not enough to introduce a significant perturbation on the user's throughput. Therefore, to balance the need to minimize input bias and the need for the simulations not to be too heavy, we chose to run our simulations for 70 seconds.

FINAL RESULTS

We are now going to present the results of our simulations. To gather statistically significant data, we executed 10 independent repetitions, on both scenarios, of all combinations of the factors we identified. **OmNet** makes this easy since we just had to use a different random number generator every time we needed one.

UNIFORM

One of the first insight we can extract from the data is the state of the antenna. This value is summarized in one word that could be: UNLOADED, LOW_LOAD, NORMAL, HIGH_LOAD and SATURATED. To identify the different states of the antenna we analyzed the average number of resource blocks used per frame across 10 independent repetitions. The various labels have been associated to the following workloads: (values are expressed in Resource blocks)

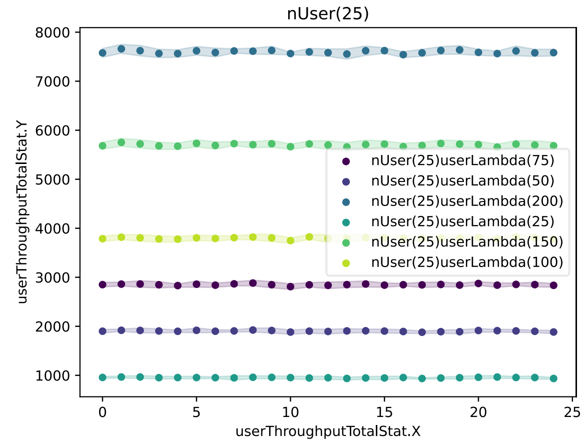
- LOW_LOAD → [3, 15]
- NORMAL → (15, 22]
- HIGH_LOAD → (22, 24.9]
- SATURATED → [25, 25]

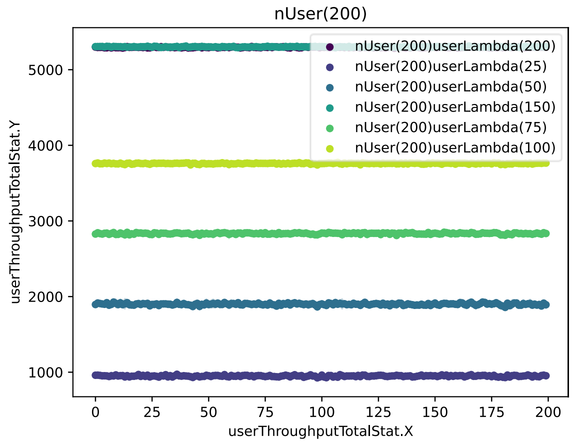
	nUser(25)	nUser(50)	nUser(100)	nUser(150)	nUser(200)
userLambda(25)	UNLOADED	LOW_LOAD	LOW_LOAD	LOW_LOAD	LOW_LOAD
userLambda(50)	LOW_LOAD	LOW_LOAD	LOW_LOAD	NORMAL	HIGH_LOAD
userLambda(75)	LOW_LOAD	LOW_LOAD	NORMAL	HIGH_LOAD	SATURATED
userLambda(100)	LOW_LOAD	LOW_LOAD	HIGH_LOAD	SATURATED	SATURATED
userLambda(150)	LOW_LOAD	NORMAL	SATURATED	SATURATED	SATURATED
userLambda(200)	LOW_LOAD	HIGH_LOAD	SATURATED	SATURATED	SATURATED

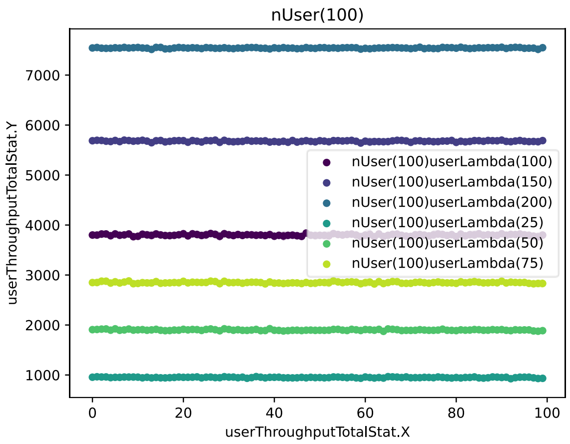
This is useful not only for the insight it provides, but also because we have to ignore the saturated configurations within the study of the delay between packets. The throughput, instead, can be studied no matter the status of the antenna.

THROUGHPUT

To declare this network as "fair" we expect the throughput for all users to be almost equal. To visualize better the results, we extracted the data from the simulations to compose a few graphs. We propose one graph for a couple of nUser values. Each graph shows on the Y axis the average total throughput of a user, while in the X axis it shows the id of the users in the simulation. All graphs like these ones also show the confidence intervals, even though in is so small to be almost impossible to see.







As we can see our expectations are correct, the average throughput across different users remains pretty much on the same level.

From these graphs, however, we can see a strange behavior. For example, let's examine the graph with nUser = 100: we saw that from userLambda 150 and 200, the antenna is in saturation, but if we look at the throughput, it increases between the two configurations. This is because of the number of distinct users served within the same frame; a lower number of users served per frame entails fewer wasted bytes, whereas the opposite is true vice versa. In addition to this inefficiency there is also another phenomenon at play. Let us say that the runs with *UserLambda* 150 is saturated, but not as saturated as the ones with *UserLambda* 200, this means that, even though the antenna is still saturated (it always uses all 25 R.B.), the user's queues are not too filled up to cause the antenna to serve just one user. Consequently, if the user being served has extracted a "good" CQI it is very likely that his queue will be emptied easily. This situation can present itself for a few times, until either the frame is filled or a user with a "bad" CQI is found. Every time a user like the former is chosen by the antenna, the rest of the frame will be used in a non-optimal way, and this lowers the maximum throughput of the antenna from the average maximum we expected:

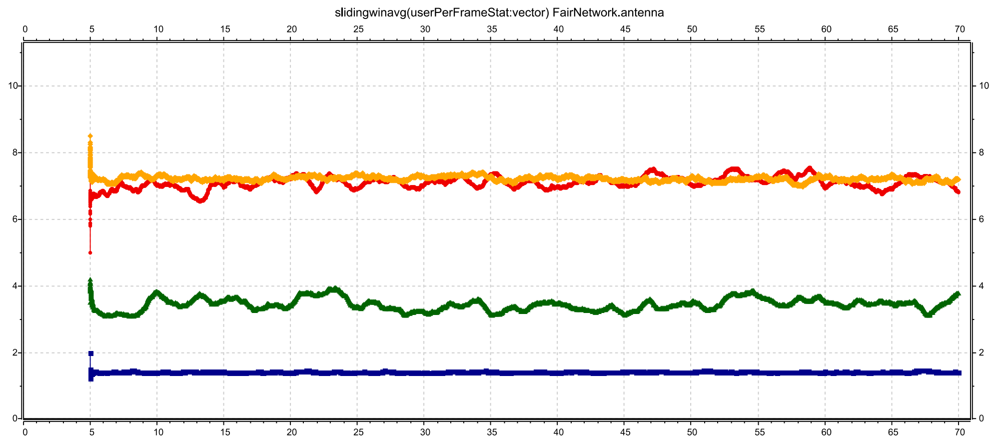
There is an average limit on the throughput of the antenna, that limit can be calculated as follows:

$$T_{tot} = A_{VG\_S\_RB} * \mathrm{N}_{RB_F} * N_{F_S} = 40{,}6 * 25 * 1000 = 1015000\ B/s$$

$$\mathrm{T_{u_i}} = \frac{\mathrm{T_{tot}}}{\mathrm{N_U}} = \frac{1015000}{N_U} = 10150\ B/s$$

Where T_{tot} is the average throughput on the antenna if a frame is fully used, $A_{VG\_S\_RB}$ is the average size of a resource block, N_{RB_F} is the number of resource blocks per frame and N_{F_S} is the number of frames sent each second. In the case of N_u equal to 100 we have T_{ui} equal to 10 150 B/s, and we can see that in the most saturated runs this value is approximately reached.

To support this, let's examine the following graph where different configurations are shown:

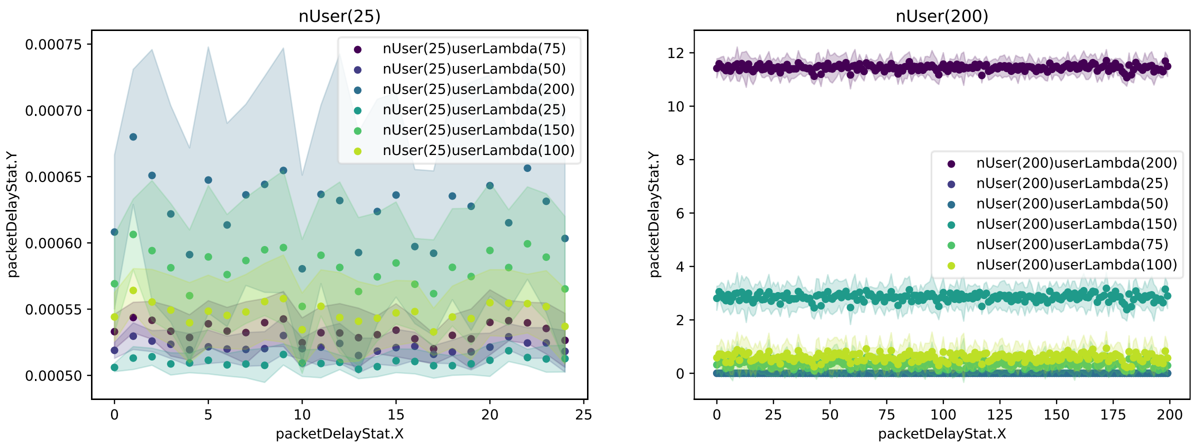


- Orange → UL = 50
- Red → UL = 100
- Green → UL = 150
- Blue → UL = 250

In this graph we plotted the sliding window average (window size 1000ms) of the number of different users served within each frame. The runs used to compose this graph are a single repetition of different configurations containing 100 users. Different values of user lambda are represented with different colors as reported beside the graph itself. As we can see the number of users per frame gradually declines, therefore increasing the efficiency of the average utilization of the frames.

DELAY

Let's now have a look at the delay:



Despite the graphs looking a bit scattered, the delay across different users is pretty much the same. (mind the scale of the Y axis)

QUANTILES

To support our discussions, we also calculated the 0.025 and 0.975 quantiles for each run:

(General)	THROUGHPUT			DELAY		
Runs	Q(0.025)	Q(0.975)	ratio	Q(0.025)	Q(0.975)	ratio
nUser(150)userLambda(75)	2826.841	2870.022	98.50%	0.002	0.156	1.58%
nUser(200)userLambda(50)	1874.411	1922.471	97.50%	0.001	0.005	18.85%
nUser(200)userLambda(75)	2816.585	2846.802	98.94%	0.117	0.588	19.87%
nUser(100)userLambda(150)	5654.117	5697.205	99.24%	0.1	0.42	23.75%
nUser(150)userLambda(100)	3765.932	3796.565	99.19%	0.132	0.491	26.84%
nUser(100)userLambda(100)	3768.349	3827.551	98.45%	0.001	0.003	32.33%
nUser(200)userLambda(100)	3745.391	3768.859	99.38%	0.33	0.85	38.79%
nUser(100)userLambda(200)	7521.796	7556.464	99.54%	0.246	0.628	39.13%
nUser(50)userLambda(200)	7535.526	7658.903	98.39%	0.001	0.002	41.41%
nUser(150)userLambda(150)	5626.001	5646.664	99.63%	0.422	0.837	50.45%
nUser(100)userLambda(75)	2825.164	2877.162	98.19%	0.001	0.001	66.87%
nUser(150)userLambda(50)	1878.121	1919.221	97.86%	0.001	0.001	66.89%
nUser(50)userLambda(150)	5649.154	5743.088	98.36%	0.001	0.001	72.12%
nUser(200)userLambda(150)	5294.771	5309.685	99.72%	2.506	3.081	81.34%
nUser(200)userLambda(25)	932.703	966.63	96.49%	0.001	0.001	84.76%
nUser(100)userLambda(50)	1880.719	1922.527	97.83%	0.001	0.001	85.68%
nUser(150)userLambda(200)	7052.251	7068.373	99.77%	2.591	3.01	86.08%
nUser(50)userLambda(100)	3773.61	3833.032	98.45%	0.001	0.001	86.25%
nUser(25)userLambda(200)	7549.609	7646.843	98.73%	0.001	0.001	88.14%
nUser(150)userLambda(25)	932.425	963.458	96.78%	0.001	0.001	90.98%
nUser(50)userLambda(75)	2822.403	2874.297	98.19%	0.001	0.001	91.09%
nUser(25)userLambda(150)	5657.429	5739.638	98.57%	0.001	0.001	92.49%
nUser(100)userLambda(25)	933.582	964.901	96.75%	0.001	0.001	94.35%
nUser(50)userLambda(50)	1882.084	1920.317	98.01%	0.001	0.001	94.59%
nUser(25)userLambda(100)	3766.618	3824.27	98.49%	0.001	0.001	95.26%
nUser(200)userLambda(200)	5287.04	5302.671	99.71%	11.177	11.628	96.12%
nUser(25)userLambda(75)	2822.286	2877.13	98.09%	0.001	0.001	96.29%
nUser(50)userLambda(25)	937.659	966.351	97.03%	0.001	0.001	97.09%
nUser(25)userLambda(50)	1881.042	1920.981	97.92%	0.001	0.001	97.31%
nUser(25)userLambda(25)	935.619	965.747	96.88%	0.001	0.001	97.78%

In the above table we reported all the different configurations we simulated. The label of the name reports the information on the status of the system already presented in a previous table. The table is sorted in increasing order on the ratio of the delay. The delay of the configurations painted red should not be considered.

As we can see the throughput is fair in all the configurations while the delay orders nicely the configurations from the least fair to the fairest configuration.

BINOMIAL

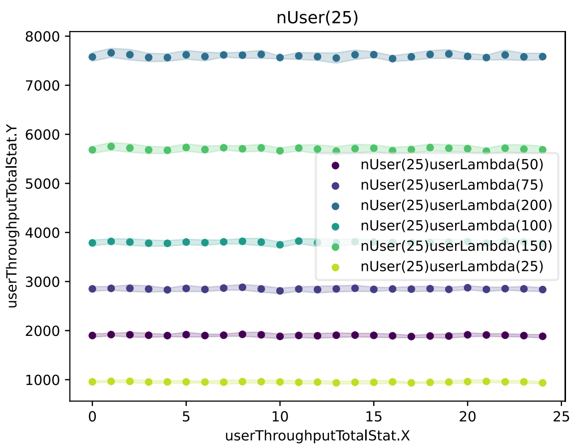
Firstly, we extract from the data the state of the antenna, in the same way we did for General.

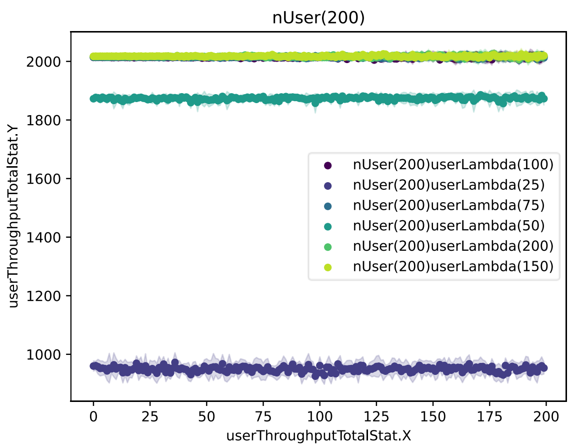
	nUser(25)	nUser(50)	nUser(100)	nUser(150)	nUser(200)
userLambda(25)	UNLOADED	LOW_LOAD	LOW_LOAD	LOW_LOAD	NORMAL
userLambda(50)	LOW_LOAD	LOW_LOAD	NORMAL	HIGH_LOAD	SATURATED
userLambda(75)	LOW_LOAD	LOW_LOAD	HIGH_LOAD	SATURATED	SATURATED
userLambda(100)	LOW_LOAD	NORMAL	SATURATED	SATURATED	SATURATED
userLambda(150)	LOW_LOAD	HIGH_LOAD	SATURATED	SATURATED	SATURATED
userLambda(200)	NORMAL	SATURATED	SATURATED	SATURATED	SATURATED

Already we can see that for Binomial the network goes in a *SATURATED* state before General

THROUGHPUT

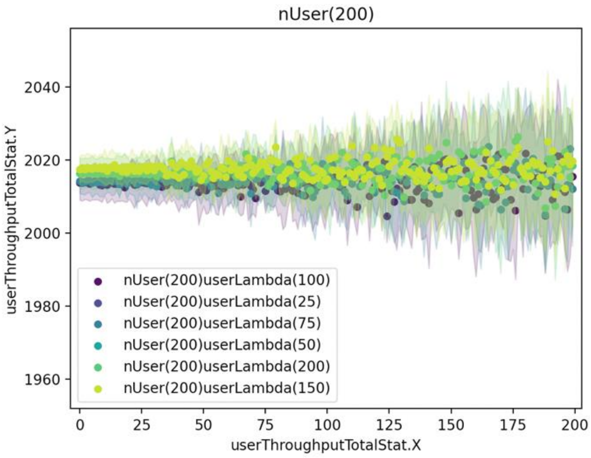
As before we expect the throughput to be almost the same for all the user, and indeed our expectation is confirmed by the following graph:





For the configurations that are NOT *SATURATED* (which are the same configurations for Uniform too) the behavior is exactly the same, this is due to the fact that the queues on the antenna are always almost empty and the throughput of the antenna is the same of the arrival packets for both configurations.

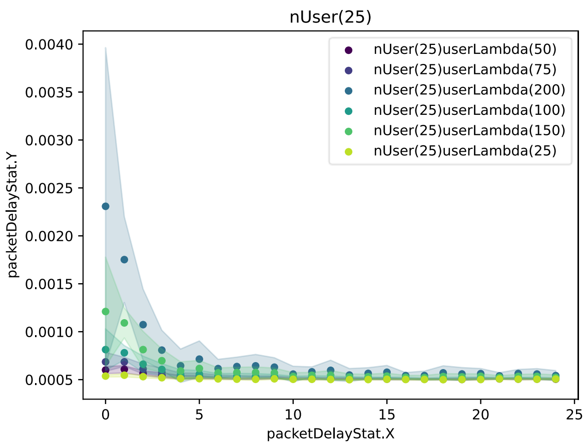
When the configurations are *SATURATED*, instead, the network is still fair, but the performances are worst. The degradation of performances can be explained by the algorithm itself; given the way we decided to associate different CQI to different users (check out the user paragraph), the users with lower ids will have decisively worse CQIs than users with higher ids. This creates an imbalance that causes users with lower ids to be chosen much more frequently than in the Uniform case. The result is that the average CQI used by the antenna is lower in the Binomial case, causing the performances to be inevitably worse.

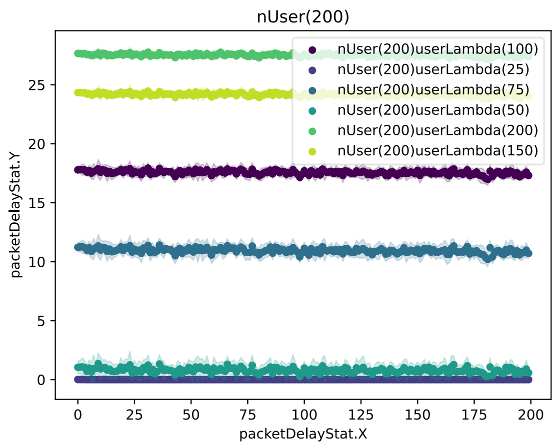


The only notable fact of this scenario is that the imbalance in choosing the CQIs manifest itself in the graph of the total throughput. As we can see the higher the id of the user, the higher the variability of the throughput. This phenomenon can be easily explained by the fact that every time a user with high id is chosen by the antenna the number of Bytes transferred is sensibly higher than all other users on the left side. On the contrary users on the left will be served more frequently but with smaller quantities of bytes per frame. The result is a graph in which users on the left approach the mean value closely while users on the right fluctuate below or above it more vigorously.

DELAY

For completeness sake we report the graphs of the delay as well as the table of the quartiles, even though there is not much more to add with respect to the Uniform case. Any imbalance we can see in the following data can be easily attributed to the phenomenon explained above.





QUANTILES

(Binomial)	THROUGHPUT			DELAY		
Runs	Q(0.025)	Q(0.975)	ratio	Q(0.025)	Q(0.975)	ratio
nUser(100)userLambda(75)	2825.139	2877.123	98.193%	0.001	0.098	0.693%
nUser(150)userLambda(50)	1878.184	1919.198	97.863%	0.001	0.092	0.785%
nUser(50)userLambda(150)	5649.359	5741.573	98.394%	0.001	0.060	1.168%
nUser(100)userLambda(50)	1880.742	1922.434	97.831%	0.001	0.004	14.464%
nUser(50)userLambda(100)	3773.610	3833.063	98.449%	0.001	0.003	17.773%
nUser(200)userLambda(25)	932.735	966.630	96.493%	0.001	0.002	25.400%
nUser(25)userLambda(200)	7549.609	7647.308	98.722%	0.001	0.002	27.383%
nUser(50)userLambda(200)	7533.441	7592.061	99.228%	0.118	0.413	28.678%
nUser(100)userLambda(100)	3753.391	3776.228	99.395%	0.251	0.821	30.530%
nUser(50)userLambda(75)	2822.403	2874.313	98.194%	0.001	0.002	32.121%
nUser(200)userLambda(50)	1863.266	1881.387	99.037%	0.407	1.233	33.021%
nUser(150)userLambda(25)	932.425	963.458	96.779%	0.001	0.001	38.100%
nUser(25)userLambda(150)	5657.384	5739.638	98.567%	0.001	0.001	45.885%
nUser(50)userLambda(50)	1882.174	1920.249	98.017%	0.001	0.001	53.892%
nUser(100)userLambda(25)	933.582	964.901	96.754%	0.001	0.001	56.905%
nUser(25)userLambda(100)	3766.618	3824.270	98.492%	0.001	0.001	64.135%
nUser(150)userLambda(75)	2689.447	2700.643	99.585%	1.922	2.649	72.572%
nUser(25)userLambda(75)	2822.286	2877.151	98.093%	0.001	0.001	73.602%
nUser(50)userLambda(25)	937.659	966.351	97.031%	0.001	0.001	80.775%
nUser(25)userLambda(50)	1881.042	1920.981	97.921%	0.001	0.001	83.339%
nUser(25)userLambda(25)	935.619	965.747	96.880%	0.000	0.001	92.372%
nUser(200)userLambda(75)	2008.356	2019.545	99.446%	10.557	11.335	93.139%
nUser(150)userLambda(100)	2691.259	2703.245	99.557%	10.530	11.124	94.662%
nUser(100)userLambda(150)	4076.012	4086.277	99.749%	10.322	10.899	94.706%
nUser(200)userLambda(100)	2006.468	2020.639	99.299%	17.218	17.858	96.418%
nUser(100)userLambda(200)	4079.817	4093.661	99.662%	17.080	17.515	97.514%
nUser(150)userLambda(150)	2694.939	2705.569	99.607%	19.483	19.926	97.774%
nUser(200)userLambda(150)	2012.832	2023.382	99.479%	23.926	24.392	98.088%
nUser(200)userLambda(200)	2010.934	2022.334	99.436%	27.294	27.720	98.464%
nUser(150)userLambda(200)	2695.310	2706.259	99.595%	23.949	24.309	98.520%