Applying fuzzy logic
to a radio system with location dependent errors

Nandor Csereoka, 4th Year - CTI ENG - FLA, 2020-2021

Problem statement

Consider a radio system consisting of K users (e.g. K=3) which transfer data.

Between the moments t1 and t2, one of the users has a very bad radio link, with so many errors that the scheduler will not allocate radio resources (radio channels) to that user.

Compared to an ideal system (i.e., without errors), the user affected by (location dependent) errors will be lagging, which means that the amount of data transfered by that user is with LAG smaller than the data transfered by its correspondent in the ideal system.

After the moment t2 the user will have again a good radio link, and its weight in the scheduling algorithm has to be increased, with the purpose to reduce the value of LAG to zero, or very close to zero.

LAG is defined as the difference between the amount of data transdered by a user in the ideal system, and the amount of data transfered by the same user, in the real system.

Like in the previous problem, you can start from the *fifo* model from *samples*. We can consider either a discrete scheduling algorithm (e.g. WRR - Weighted Round Robin), or you can consider that each user has a transfer rate equal with 1/K of the capacity C of the network (e.g. C= 1 Mega bits per second).

The fuzzy algorithm will adjust the weight of the user affected by errors (respectively its transfer rate) on the expense of the other users.

Process

We will use the existing components from our implementation: the WRR scheduler, the simulated user modules and the sink. We'll also keep in mind the previously stated information regarding the structure of the network and the different interactions between its modules.

But now, we also introduce the fuzzy logic controller. This will intelligently adjust the weight of the unlucky user which loses their connection.
In terms of OMNeT modules, this means the addition of a controller which is triggered by a generator. This starts sending messages at a predefined time and at predefined intervals. In each such interval, the controller recalculates the optimal weight for the user which is lagging behind using fuzzy inference.

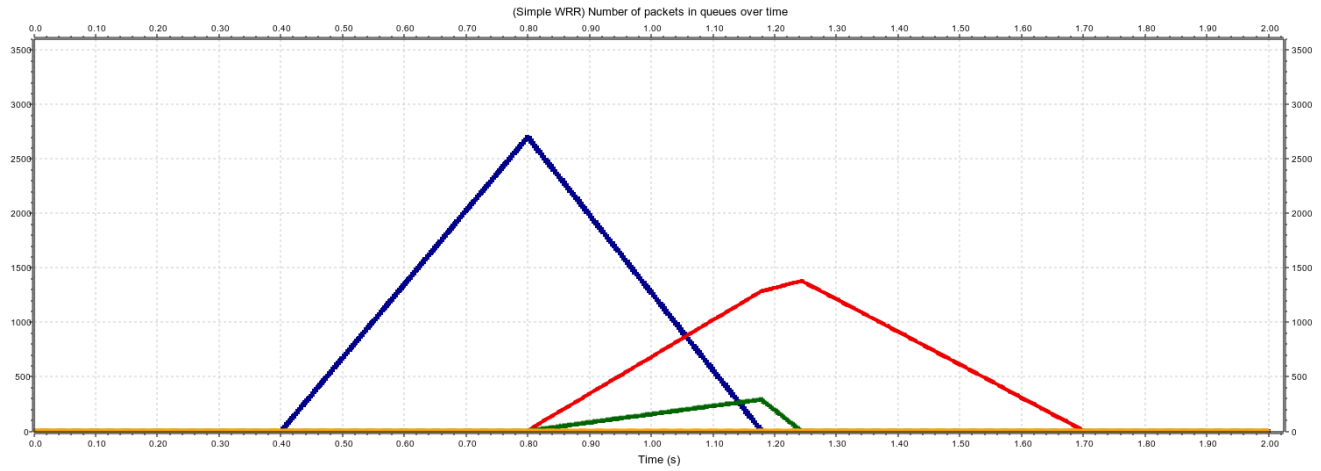
The generator module will be configured to start when the user regains their connection. After this, it will start triggering the controller at well-defined intervals.

The general fuzzy logic controller needs to be configured for this particular scenario. The wanted delay should be considered, just like we defined in the problem statement. The actual delay or LAG will be measured as the number of packets that are sitting in the user's queue. The mapping and scaling will be done according to the number of available channels. At each recalculation, the starting weight of the unlucky user will be the one previously used, in order to apply the optimal weight.

After everything is set up, we let the fuzzy inference do its magic.

The basic scenario for 4 users

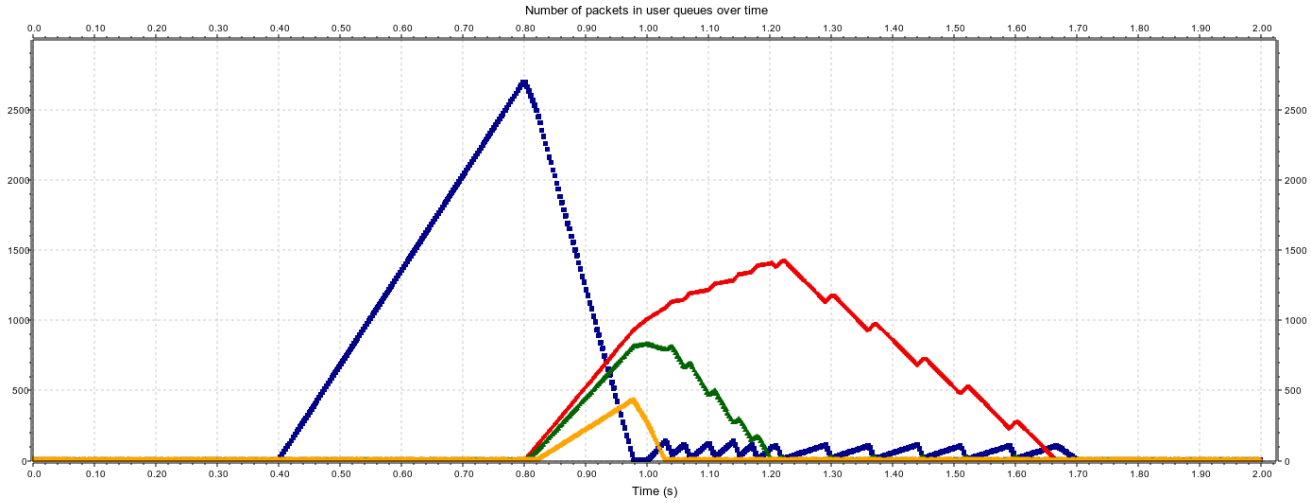
This is the number of packets for the WRR scheduler **without** fuzzy inference. After the user regains their connection, their weight becomes the same as the user's with the highest priority.



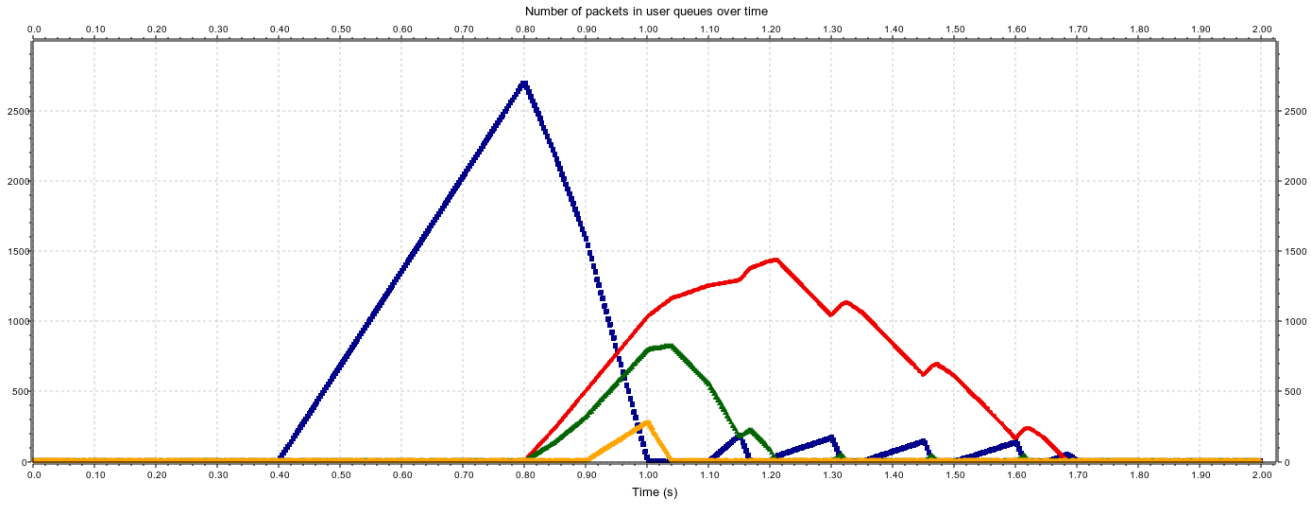
When the fuzzy logic controller is introduced, we can see the gradual adjustments which take place. We make several tweaks to the allowed delay and the fuzzy inference interval.

The FLC scenario for 4 users

Allowed delay: 100 - FLC interval: 0.01s

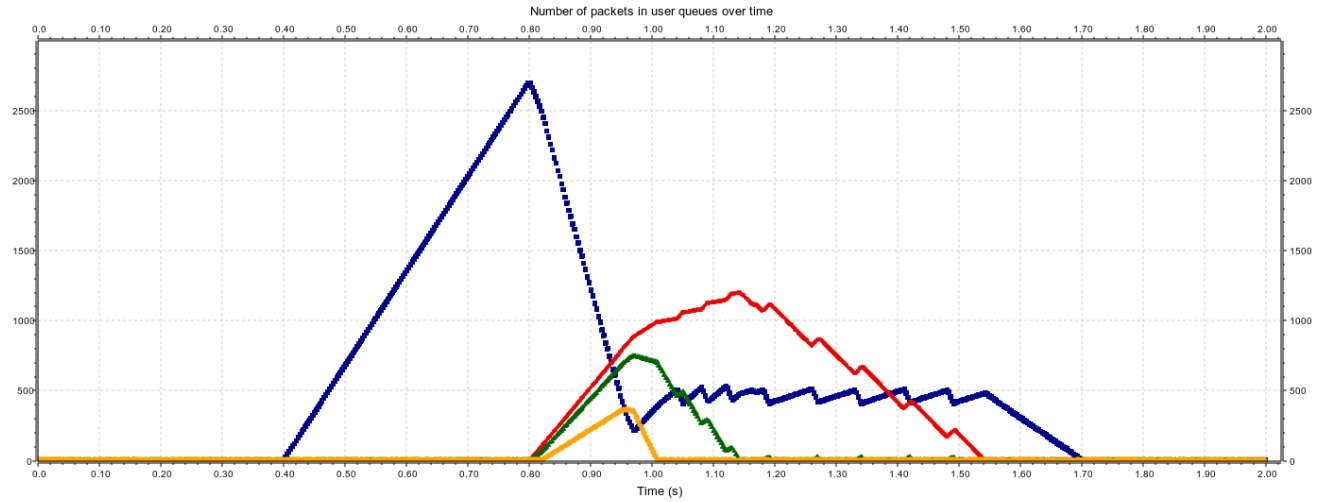


Allowed delay: 100 - FLC interval: 0.05s

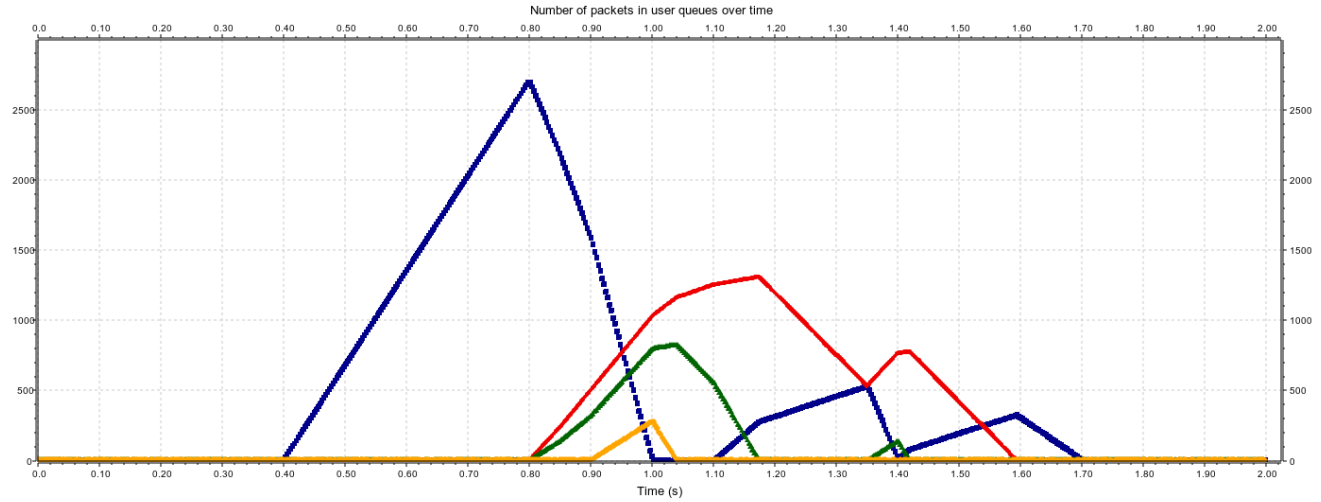


The FLC scenario is quite different: all other users are affected, as the weight of the unlucky user reaches a maximum. We can also see the fine adjustments which occur during the FLC's processing cycle.

Allowed delay: 500 - FLC interval: 0.01s



Allowed delay: 500 - FLC interval: 0.05s



When the unlucky user is allowed a greater delay, we can see bigger changes in terms of the delay of the other users.

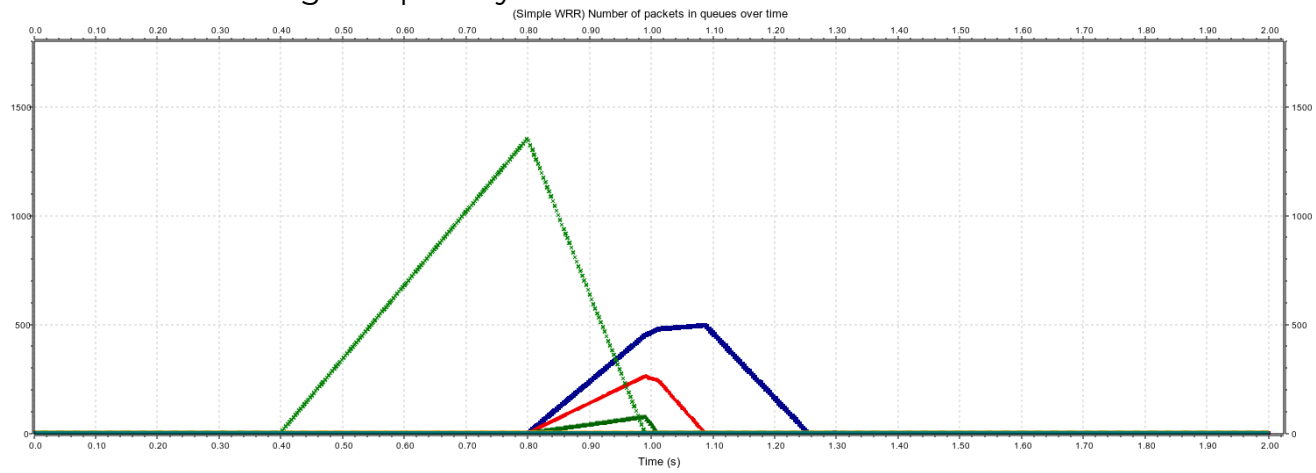
Comparison - 4 users

The weight of the other users doesn't change. This guarantees a certain behaviour. The maximum delay of the unlucky user reaches ~2700 packets just before regaining connection.
Let's denote with 'm_i' the maximum delay of the user with index **i** after the unlucky user regains their connection ($user_0$) and 't_i' the time it takes them to reach a state where no future periods with delays occur, starting from the connection regained event (0.8s).
D - allowed delay, *i* - FLC interval

	m_0	m_1	m_2	m_3	t_0	t_1	t_2	t_3
no FLC	↓	1400	300	-	0.38s	0.90s	0.44s	-
D = 100 i = 0.01s	100	1490	710	420	0.90s	0.86s	0.40s	0.23s
D = 100 i = 0.05s	100	1470	790	300	0.90s	0.88s	0.52s	0.24s
D = 500 i = 0.01s	500	1200	700	370	0.90s	0.74s	0.70s	0.19s
D = 500 i = 0.05s	500	1350	800	300	0.90s	0.79s	0.62s	0.24s

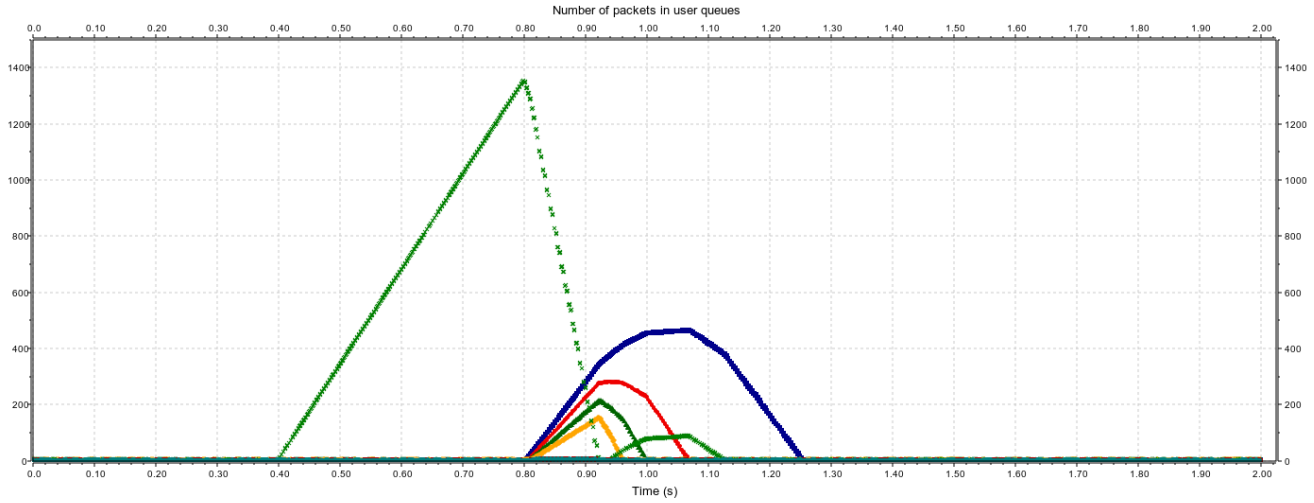
The basic scenario for 8 users

In the scenario without fuzzy logic, the user's weight is increased to the weight of the user with the highest priority.

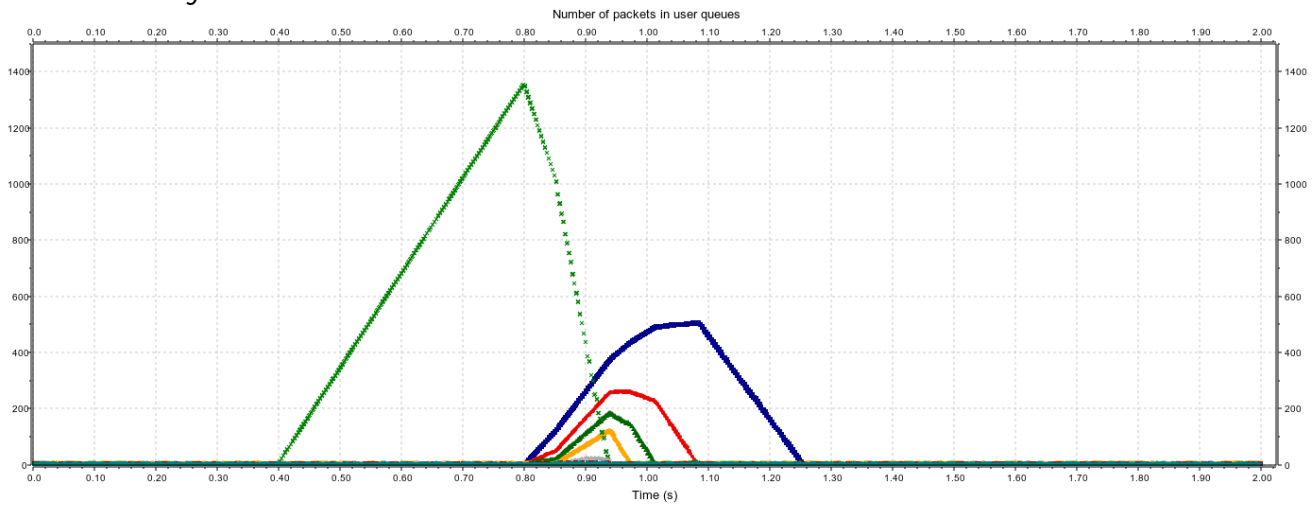


The FLC scenario for 8 users

Allowed delay: 100 - FLC time: 0.01s

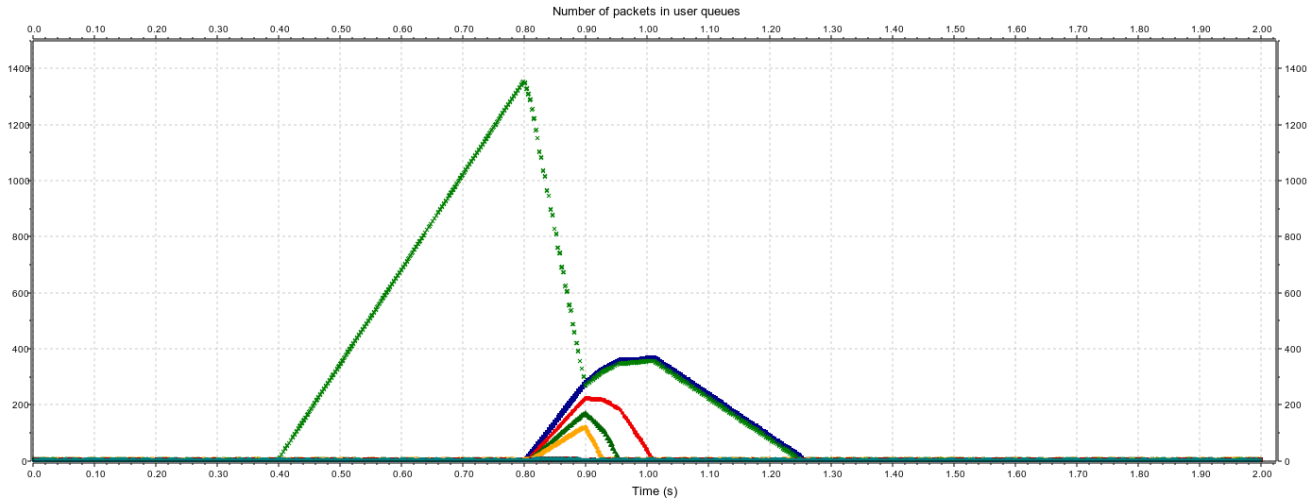


Allowed delay: 100 - FLC time: 0.05s

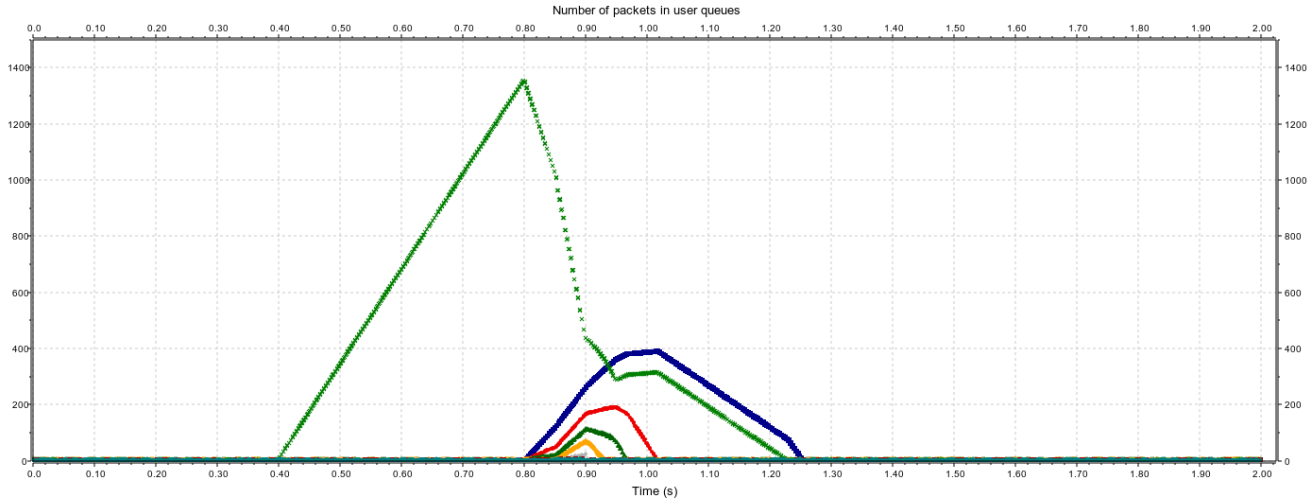


Here we can see that the more frequent FLC time causes the user to acquire a bit more LAG after reaching a minimal delay.

Allowed delay: 500 - FLC time: 0.01s



Allowed delay: 500 - FLC time: 0.05s



When the user is given more leeway in terms of delay, the other users are not as affected.

Comparison - 8 users

The weight of the other users doesn't change. This guarantees a certain behaviour. The maximum delay of the unlucky user reaches ~1375 packets just before regaining connection.
Let's denote with 'm_i' the maximum delay of the user with index **i** after the unlucky user regains their connection (user$_0$) and 't_i' the time it takes them to reach a state where no future periods with delays occur, starting from the connection regained event (0.8s).
D - allowed delay, *i* - FLC interval

	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
no FLC	↓	500	290	100	-	-	-	-
D = 100 i = 0.01s	100	470	300	200	170	-	-	-
D = 100 i = 0.05s	100	510	270	95	110	30	-	-
D = 500 i = 0.01s	370	370	210	150	110	-	-	-
D = 500 i = 0.05s	320	400	190	120	80	30	-	-

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
no FLC	0.19s	0.45s	0.28s	0.21s	-	-	-	-
D = 100 i = 0.01s	0.32s	0.45s	0.27s	0.20s	0.16s	-	-	-
D = 100 i = 0.05s	0.14s	0.45s	0.28s	0.21s	0.18s	0.14s	-	-
D = 500 i = 0.01s	0.45s	0.45s	0.21s	0.15s	0.13s	-	-	-
D = 500 i = 0.05s	0.42s	0.45s	0.22s	0.16s	0.12s	0.10s	-	-