Project 6: Aerocom System 1

Alberto Cristallo, Luca Ciampi, Valerio Tanferna

***Abstract*— In this paper we evaluate the performance in terms of response time of a system composed by a queue and L time-varying capacity servers, used one at a time and chosen by an algorithm, that serves jobs according to the FIFO policy. We test the system under various scenarios, changing quantitative and qualitative factors.**
We use Omnet++ Simulator for the implementation of the system and Pyhton language for the data analysis.

I. INTRODUCTION

The aim of this project is to evaluate the performance in terms of response time of a system composed by a queue and L servers that serves jobs according to the FIFO (First In First Out) policy.
Only one server at a time can process a job, while other L-1 remain in stand-by. The serving server is chosen according to a particular algorithm.

The case studies that have been explored are an M/G/1 system and a G/G/1 system, the first with an exponential distribution for the inter-arrivals time while the second (from here called L/G/1 system) with a log-normal distribution for the inter-arrival times.

Service time is affected basically by two factors: the length of the job to be processed expressed in bits and the processing capacity of the server expressed in bps (bits per second). In particular the capacity is time-varying: every t seconds a server select a new target capacity (values are taken from a uniform distribution) and the capacity will linearly change from the current one to the target one, which is reached after t seconds; t is a random variable with an exponential distribution in the first scenario and a log-normal distribution in the second.

In addition to the service time, the response time is affected by another parameter: the time S needed to perform a selection of a new server, expressed in seconds.

The selection of the serving server is performed when the job has to be sent to the server in the following way: if the capacity between service i-2 and i-1 decreased by more than X% the system will perform a server selection, choosing the one with the highest capacity at the current time, otherwise the serving server remains the same.

We have measured the response time for various values of X, S and the parameters that characterized the distribution of t (*quantitative factors*), fixing the length of the packet, the intervals of the set of possible values of the target capacity and the parameters characterizing the distribution of the inter-arrival time, and we have compared the results. As a further meter of comparison we have exploited two additional algorithms for the serving server selection (*qualitative factors*): a random algorithm that chooses the new server in a random manner every time, and a single-channel algorithm that maintains always the same server for all the services.

The paper is structured as follows. Section II reviews an higher-level description of the system (the model and the implementation of it). Section III is about performance analysis and shows the simulation scenarios. Finally in Section IV we summarize the results obtained.

II. SYSTEM DESCRIPTION

A. System Modeling

The real system from which we started has been an aero-system composed by N AirCrafts (ACs), each of which transmits a packet to a Control Tower (CT) through a DataLink (DL) characterized by a transmission capacity, chosen from a pool of five DLs available.

Following the simulation work-flow, we have built a model of this system, abstracting away some of the details and including some sub-models. In particular these assumptions should be taken into account:

- CT can receive N packets at the same time, i.e. packets are never discarded
- the time required for a packet to cover the distance between the aircraft and the tower is considered negligible, i.e. aircrafts may be at a different distance from the CT
- no transmission errors may occur
- ACs can store an infinite number of packets waiting for the end of the transmission of the packet currently in processing

Basically ACs are modeled with a system composed by a source that generates packets, a queue (with infinite size) that eventually stores packets and five servers that simulate the transmission time of a link processing the packet, while the CT is modeled just as a sink.

Fig. 1 and Fig. 2 show, respectively, the high-level model of the whole system and the model of a single AC.

As mentioned before, the inter-arrival times are exponentially distributed (i.e. the Source block in the figure dispatches jobs according to a Poisson Process) in the first case study, while are log-normally distributed in the second case.

The service is performed by a single server whose service rate is independent from the state of the queue and the arrival rate of jobs, but it depends instead by the value of its target capacity, by the moment of its choice and by the length of the job itself.

Furthermore, global performance are eventually affected also by the switch-server operation.

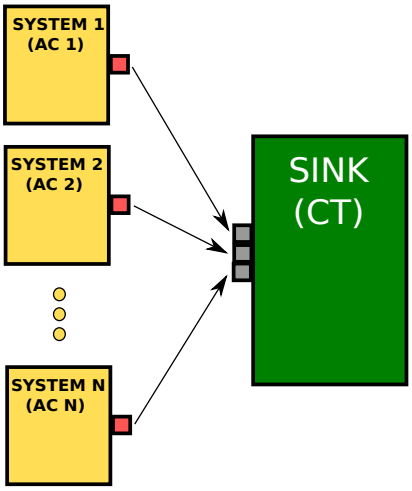

Fig. 1. High-level model of the whole system

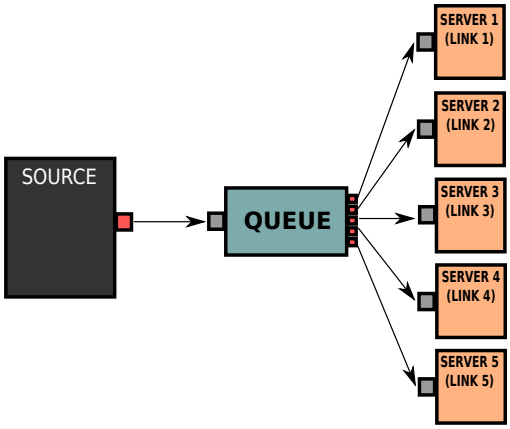

Fig. 2. Model of a single AC

The metrics on which we focused for this study are *in primis* the response time, and *in secundis* the queue length and the number of switches carried out.

B. System Implementation

The implementation of the above model has been achieved through the use of the Omnet++ Simulator that allows the developer to define the behavior of the network nodes using the C++ language.

The implementation follows the model and it consists of some modules described in details in the following:

- **AirCraft**: it is an Omnet++ compound module containing various simple modules: a source module, a router module and five server modules.
  It has no input but it is connected to the CT module with five output gates (one for each server that contains) used to send packets.
  This module is also responsible to collect some useful statistics.
- **Control Tower**: it is an Omnet++ simple module that simply receives packets through input gates from one or more AC modules, deletes them and collects some useful statistics.
- **Source**: it is an Omnet++ simple module, part of the AC compound module, whose main task is to generate packets (represented as *Packet*, a subclass of *cMessage*) relying on a distribution (characterizing a RV k, time of creation of a new packet) specified in the configuration file. It also sets their length and sends them to the router module through an output gate.
  Note that Omnet++ automatically sets a time-stamp to each packet when they are created, so we are not worried about this.
- **Router**: it is an Omnet++ simple module, part of the AC compound module, whose main tasks are to (eventually) store packets in a queue, choose the right server and send a packet to it.
  In particular the module receives packets from the source module through an input gate and, if necessary (i.e. if a server is busy), pushes them in the queue. Whenever it receives a notification from a server (i.e. a server has completed its service and now the aircraft can transmit another packet), it pulls a packet from the queue (if available), it chooses a server and it sends the packet to it through an output gate.
  The selection of the server is done relying on a routing algorithm and using a routing table updated at each packet transmission.
  The queue is implemented as an abstract class, and the actual class used is chosen relying on a parameter which can be set in the configuration file. This allows the module to work flawlessly with different policies. The policy actually used is the FIFO, but also a priority-based policy is implemented and could be used for performance comparisons in a future work; other policies are still easily implementable without changing this module.
  Routing algorithms available and used as a meter of comparison are three: the first is the one described above that basically chooses a new server with the maximum capacity only if the last two transmissions are decreased by a certain percentage (set in the configuration file), the second chooses randomly a server at each transmission, while the third uses always the same server. The algorithm actually used can be set in the configuration file without changing this module.
  The routing table contains the indexes of the servers used in the last transmissions with their current capacities, and is maintained up-to-date asking to the chosen server its current capacity with a particular message (represented as *ServerCapacityMsg*, a subclass of *cMessage*).
  If the router wants to know the server having the maximum capacity, it sends a particular message (rep-

resented as *ChooseServerMsg*, a subclass of *cMessage*) to the first server: this message is then propagated to the other servers and finally comes back to the router within the requested information.
Finally note that if the server-switch operation is needed, the router must wait S seconds before sending the packet (where S is a parameter defined in the configuration file): this waiting time is implemented with an Omnet++ self-message.

- **Server**: it is an Omnet++ simple module, part of the AC compound module, whose main task is to process a packet arrived from the router module through an input gate (i.e. simulate the time needed for the transmission of the packet using an Omnet++ self-message) and then send it to the CT module through an output gate.
  This service time depends on the packet length and on the capacity of the server itself that, as mentioned before, is time-varying and rely on a distribution (characterizing a RV t, time of selection of a new target capacity) and on a set of target capacity specified in the configuration file.
  Whenever it receives a packet from the router, this module computes the service time and then, when the timer is expired, it sends a notification to the router and the packet to the CT.
  Furthermore, a second timer is needed: every t seconds the module must update its target capacity, selecting a new value of it from a set of possible value (this set belongs to a uniform distribution).
  Servers can also communicate each other because they must propagate particular messages coming from the router with the purpose of discovering the server with the current maximum capacity available.

Fig. 3 shows the network that has been considered, composed by N AC modules and one CT module, while Fig. 4 shows the internal structure of an AC module.

C. Configuration File

The configuration file *omnet.ini* allows us to control the configuration variables of the simulation.

In particular we have performed the simulation many times varying the *simulation parameters*, like the simulation and the warm-up time, the *quantitative factors* like the parameters characterizing the distribution of k and t, the parameters S and X, the packet size and the intervals of the uniform distribution characterizing the target capacity, and, finally, the *qualitative factors* like the routing algorithm.

Another important parameter in this file is N, which defines the number of aircrafts. We have considered the mean of the response times of an aircraft as a sample of an IID RV. Because we set N to 50, and we performed 10 repetitions of the same experiment, we had 500 samples available for each simulation that we have done.

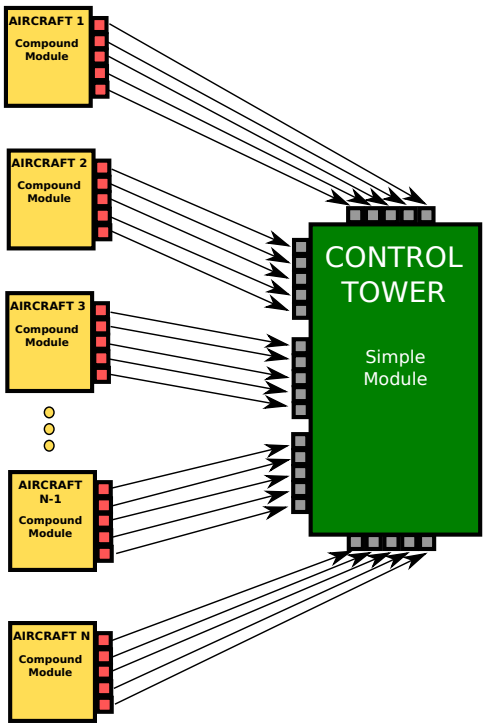

Fig. 3. Block diagram of the network

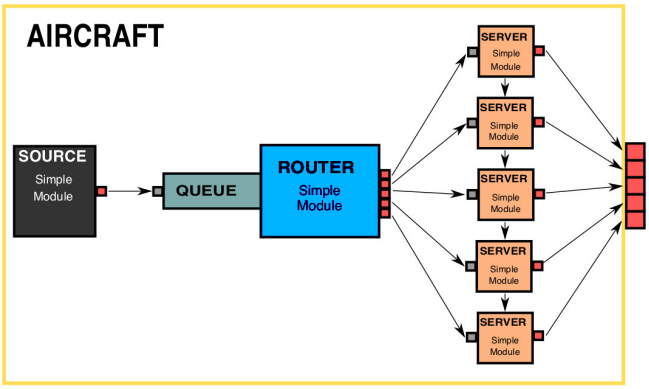

Fig. 4. Block diagram of an AC compound module

III. PERFORMANCE ANALYSIS

A. First Scenario: t and k with Exponential Distribution

In this scenario we have considered an exponential distribution for the inter-arrival times k, fixing the mean of the distribution to 2.5 seconds.
We have also fixed the length of the packet to 2048 bits and the intervals of the uniform distribution characterizing the set of the possible values of target capacity to 100 bps and 2000 bps.
So, considering only these parameters, service time is about 2 seconds; because the mean of the distribution of k is equals

to 2.5 seconds, assuming to have a single-server system (i.e. considering an M/D/1 system), at the steady-state the system will be stable.
However, our system is composed by 5 time-varying capacity servers, chosen in according to an algorithm: we want to check if, and under which conditions, we have an improving of the performance.
The implicit goal of the algorithm should be to optimize the service time, changing the server if the last two transmissions are decreased by a certain percentage; but we must also consider that the switch operation is not 'free', having a 'cost' of S seconds.
Furthermore, what about the time-varying capacities of the servers? In what way RV t affects these capacities and the performance of the whole system?

We have analyzed the simulation results tuning parameters S, X and the parameters characterizing the distribution of t. Fig. 5 shows the trend of the means of the response time fixing the mean of the exponential distribution of t to 0.2 seconds and using various values of S and X, considering a confidence interval of 99% (see the Appendix for the method used to compute confidence intervals).

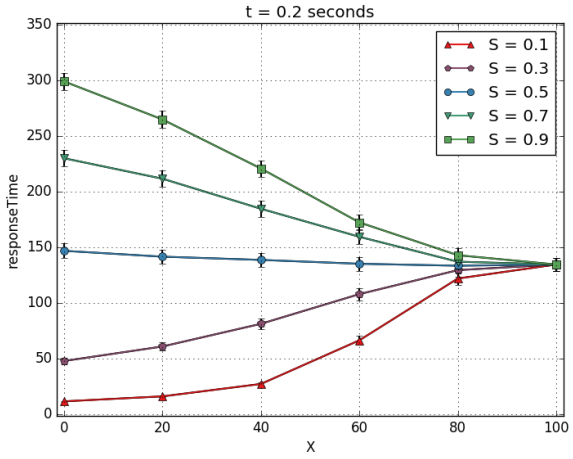

Fig. 5. Response time for mean_t=0.2s and various X and S

As we can see, if S is small enough, the curves increase with X (i.e. when the system performs switch-operations at a lower rate), otherwise they decrease.
It seems that the algorithm works well with small values of S: the more switch operations the system performs, the lower the response time is. In other words, the more the algorithm is able to choose the server with the maximum capacity, the lower the service time is and, obviously, the lower the response time is.
We can instead note a different behavior if S is greater than a certain threshold: in this case the cost of the switch operation heavily affected the performance of the system and it is more convenient to perform a lower number of switch operations. The benefits in terms of transmission time due to the algorithm are in some ways cloud by the value of S.
Finally note that for high values of X the curves tend to be more and more equal: this makes perfect sense since for high values of X the system performs a low number of switch operations, so t heavily affects the simulation while S not.

Fig. 6 shows the trend of the means of the response time fixing the mean of the exponential distribution of t to 2 seconds and using the same values of X and S as before.

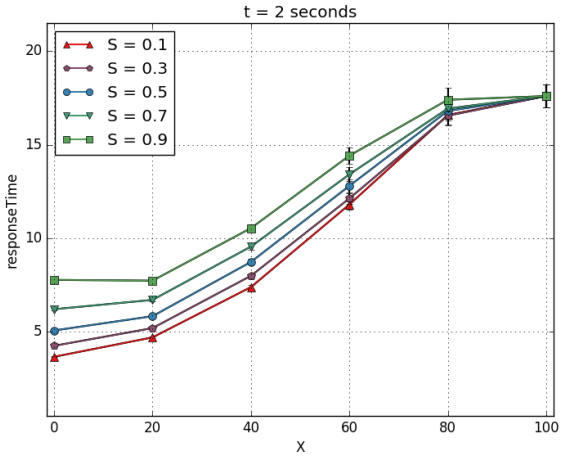

Fig. 6. Response time for mean_t=2s and various X and S

We note that now all the curves increase. It seems that t and the number of switch operations are inversely related: for high values of t the number of switch operations decreases. In fact now also the curves for S=0.7 and for S=0.9 seconds increase: it seems that now the benefits in terms of transmission time due to the algorithm are considerable also for high values of S.

Fig. 7 shows the relation between some values of the mean of the distribution characterizing t and the number of switch operations that the system performs, fixing X=50% and S=0.5 seconds.
The bar plot confirms our intuition: with higher values of t the system performs a lower number of switch operations.
An explanation for this behavior could be the following: for high values of t a server changes rarely the value of its target capacity and there is a good probability that its transmission capacity increases linearly many times (until it reaches the target capacity itself) without having switch operations. In other words the system is in some ways more stable, because it avoids useless switches.
With small values of t, instead, a server changes frequently the value of its target capacity, so the system is not very stable and performs many useless switches, i.e. it switches servers even though last transmissions were good.

Furthermore Fig. 6 shows another important thing: for all values of S we obtain better performance in terms of response time if we compare the results with the curves in Fig. 5.
This behavior is related with the previous: it seems that for high values of t the system is more stable, it performs

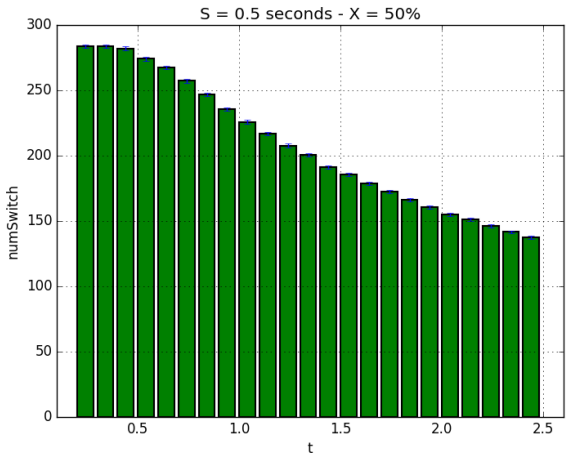

Fig. 7. Relation between mean_t and the number of switch operations

a lower number of useless switches and it exhibits better performance.

Summarizing, we can think of having two 'types' of switch operations.
The first one more related with the value of t: it represents in some ways useless switches and its frequency decreases with high values of t. The second one more related to X: fixed a value of t we would want as much as possible useful switches (i.e. we prefer small values of X) taking advantage of the algorithm.

We would want now to choose an optimal value of the mean of the distribution of t: in fact from Fig. 5 and Fig. 6 we have seen that mean=2 seconds is better than mean=0.2 seconds, but we want to evaluate how much high the mean would be for better performance.
Fig. 8 shows the trend of the means of the response time fixing X=100% and using various values of the mean of the distribution of t and S.

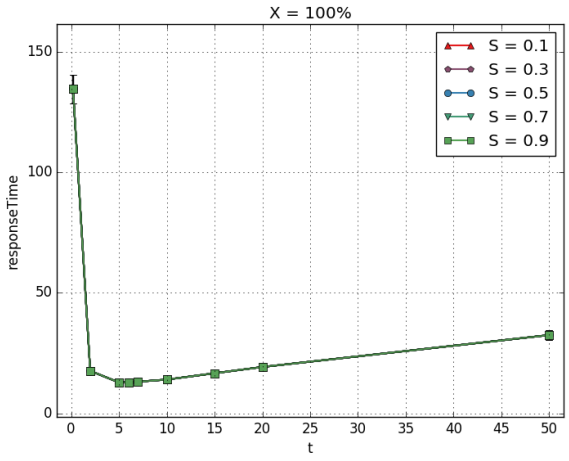

Fig. 8. Response time for X=100% and various mean_t and S

The choice of X=100% is not casual: with this value of X all curves are equal. This makes perfect sense since the system doesn't switch server anymore, so curves depend only by t. So we can study the behavior of the system relying only on the value on t.
It seems that performance increase until a value of t around 5 seconds, then the response time begins to increase.
This behavior can be explained noting that if we change rarely target capacity we can bump into a long series of good transmissions, but also into a long series of bad transmissions as well: in the last case it is more convenient to switch server as soon as possible. After a certain value of t (around 5 seconds) this behavior seems more likely to happen.

Fig. 9 shows the trend of the means of the response time fixing the mean of the exponential distribution of t to 5 seconds and using the same values of X as before and the extreme values of the previous Ss.

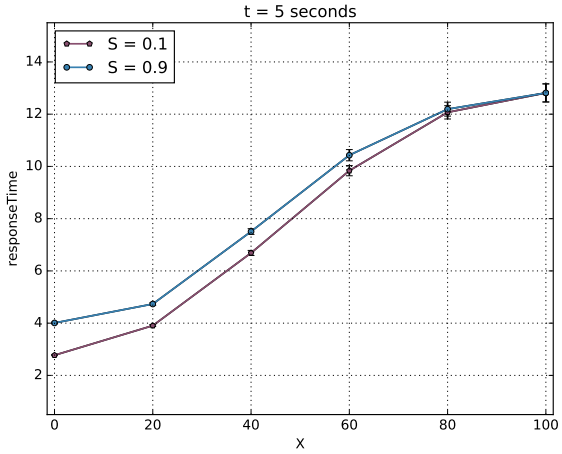

Fig. 9. Response time for mean_t=5s and various X and S

As we expected, the system exhibits better performance with respect to the system with the mean of the exponential distribution of t equal to 2 seconds.

Now that we have an idea about the influence of the parameters in the system we can choose them in order to build a system with good performance.
In particular we have chosen the mean of the distribution of t equal to 5 seconds because with this value the system promises the best performance (at least for the values of S tested). We have then chosen a small value of X (X=5%) because, again, the system ensures better performance with this value (in combination with mean of t equal to 5 and the values of S tested).
Finally we have compared the performance of the system tuning the value of S.

Fig. 10 shows a comparison in terms of means of response time between the algorithm subject of this project and a second algorithm that uses a single server.

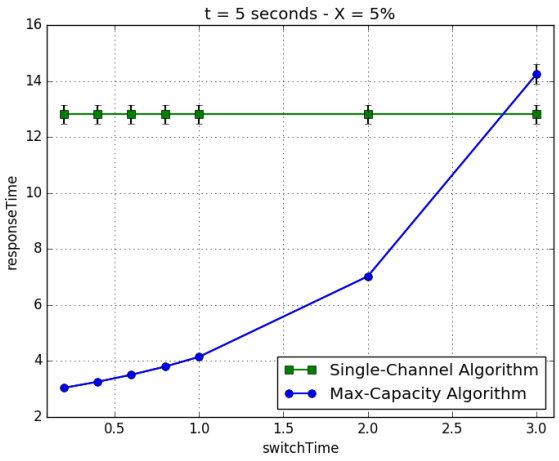

Fig. 10. Performance comparison between two algorithms

The trend of the single-channel algorithm is represented as an horizontal straight line: this makes perfect sense since in this case the system doesn't switch server and so it doesn't rely on the value of S.
As we can see the max-capacity algorithm ensures better performance until a certain value of S: if S is too high the best choice is instead the single-channel algorithm.
Anyway note that for high values of S, different combinations of X and t could be improve performance of the max-capacity algorithm.

While it is more useful to study a system at the steady state, we have checked if the systems considered before were stable. Fig: 11 shows the 500 trajectories of the queue length relative to the system with the single-channel algorithm.

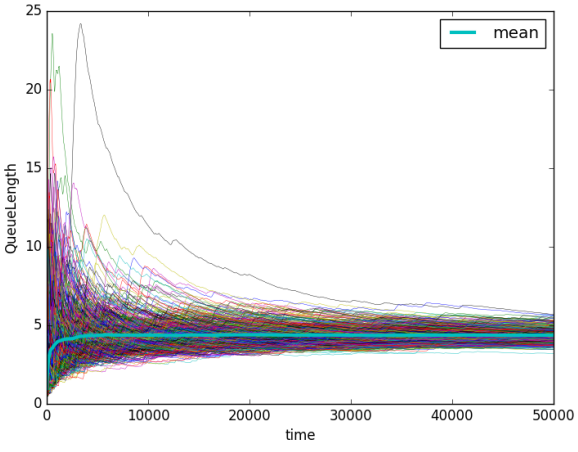

Fig. 11. Trajectories of the queue length using single-channel algorithm

As we can see the system is stable.
Note that for the graphs of the trajectories we used the cumulative moving average, because otherwise they would have been too jittery to understand if there was a convergence or not.

Considering again Fig:10 we can now assert that every system having values of S corresponding to curves below the curve of the single-channel algorithm are stable systems themselves.
We have checked this, plotting in Fig:12 the 500 trajectories of the queue length using a value of S of 1.

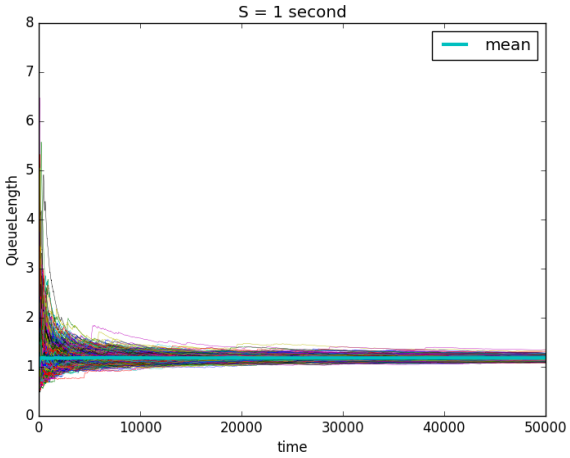

Fig. 12. Trajectories of the queue length using max-capacity algorithm

As we expected the system is stable. Some considerations about the simulation-time and the warm-up time can also be done: it seems that for S = 1 second we could set a warm-up time of about 2000 seconds and a simulation-time of about 20000 seconds.
A system having values of S corresponding to curves above the curve of the single-channel algorithm doesn't guarantee instead stability, as shown in Fig:13.

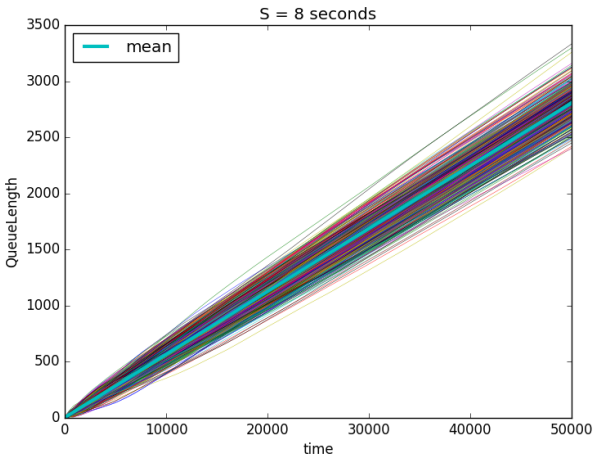

Fig. 13. Trajectories of the queue length using max-capacity algorithm

We have shown that, for some values of S, the utilization of more than one server ensures better performance. We want

now to show that it is not only important the number of servers of the system but also the algorithm that chooses them.
Fig. 14 shows a comparison in terms of means of response time between the algorithm subject of this project and a second algorithm that chooses randomly the server at each transmission (we have used very small values of S for better visualization).

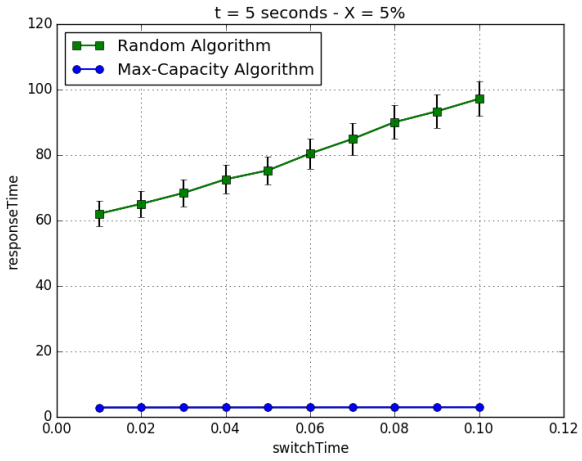

Fig. 14. Performance comparison between two algorithms

We can clearly see that the choice of the algorithm is a key point: with a random algorithm the performance of the system are always worse, even for small values of S. We can also assert that it is preferable to have just one server than to have more servers and a bad algorithm.

Finally, we have considered a system with a value of S equal to 0.2 seconds (which we consider plausible and that offers good performance as shown before).
To measure the variability of the response time of the system using the max-capacity algorithm compared to the other two systems (i.e. using the random algorithm and the single-channel algorithm) we used a Lorenz Curve, shows in Fig. 15:
From the plot we can see that the curves related to the random algorithm and to the single-channel algorithm are always below the one related to the max-capacity algorithm. This means that not only the variance is higher, but there are more packets in the extreme values (very low and very high response times).

Fig. 16 shows the Empirical CDF of the system using the max-capacity algorithm.

In order to evaluate the EPDF we have made a histogram using 13 bins. As shown in Fig. 17 mean and median are pretty the same, so we can assert that the distribution should be symmetrical: it seems to be a normal or a log-normal with a small positive skew.
We have checked this with two QQ-plots, shown in Fig. 18 and Fig. 19.

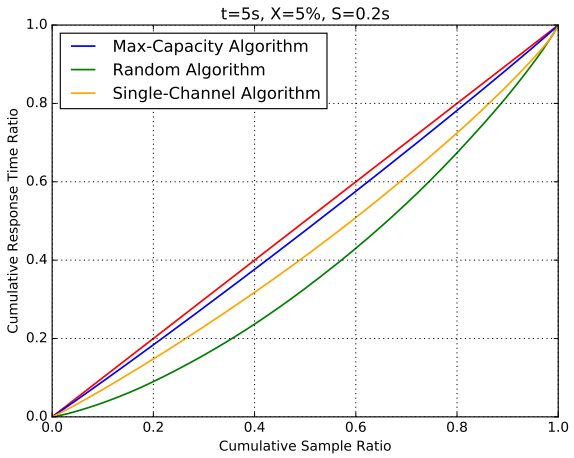

Fig. 15. Lorenz Curve

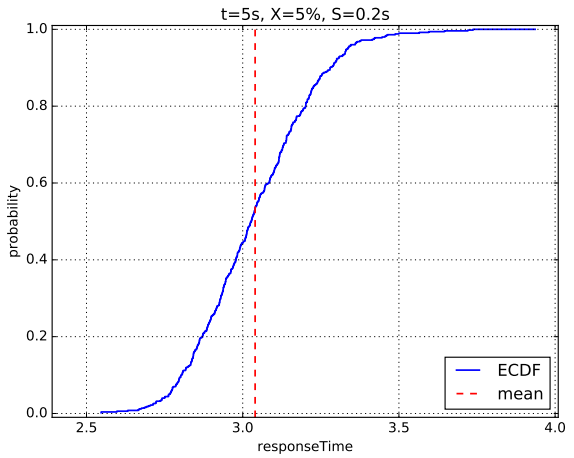

Fig. 16. ECDF

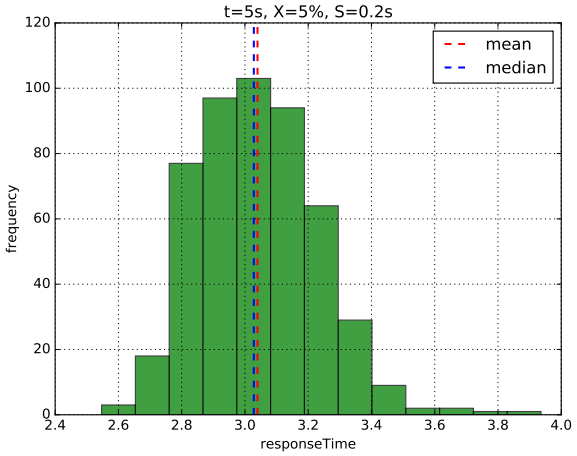

Fig. 17. EPDF Approximation

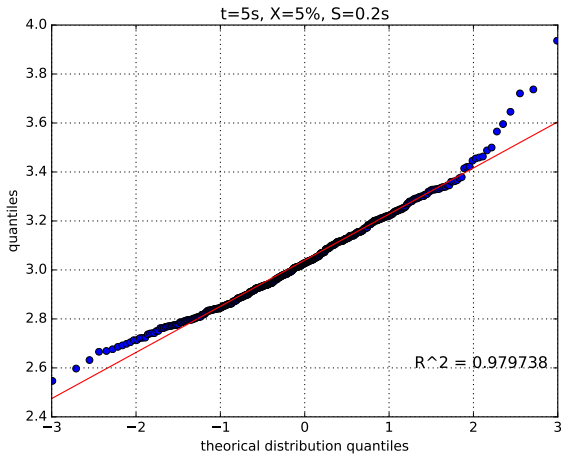

Fig. 18. QQ Plot - Normal Distribution

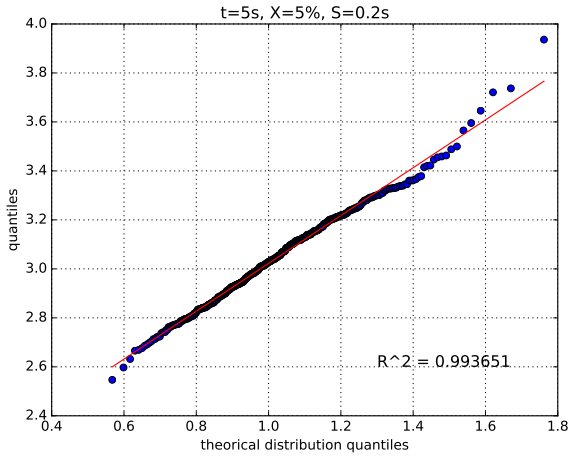

Fig. 19. QQ Plot - Log-Normal Distribution

In the second QQ-plot the value of R is greater than the first, so the log-normal distribution seems to fit slightly better than the normal one.

B. Second Scenario: t and k with Log-normal Distribution

In this scenario we have considered a log-normal distribution for the inter-arrival times and the selection of the target capacity (i.e. for the distribution of the RVs k and t), leaving the same means as before but using higher variances.
Then we have checked the behavior of the previous system using these new distributions.

Fig. 20 shows the trend of the means of the response time fixing the mean of the distribution of t to 5 seconds and using various values of S and X.
As we can see performance are considerably worse than before (see Fig. 9 for a comparison).
In fact, using a log-normal distribution for the inter-arrival times, packets arrive in bursts: as a consequence the system is busy for some times, then the queue becomes empty and so on. This phenomenon affects heavily the stability and the performance of the system.
Furthermore the log-normal distribution is worse also for RV t: it produces many consecutive small values of t, and we have seen before that as a consequence of this the system performs many useless switch operations.
Finally note that confidence intervals are quite large and curves overlap each other: we cannot precisely evaluate and compare the performance of these systems.

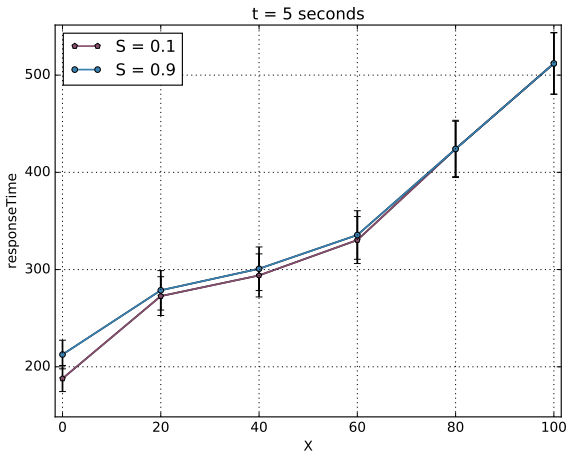

Fig. 20. Response time for mean_lognormal_t=5s and various X and S

Fig: 21 shows the 500 trajectories of the queue length relative to a system with S equal to 1 second and X = 5% (i.e. the parameters used in the first scenario).

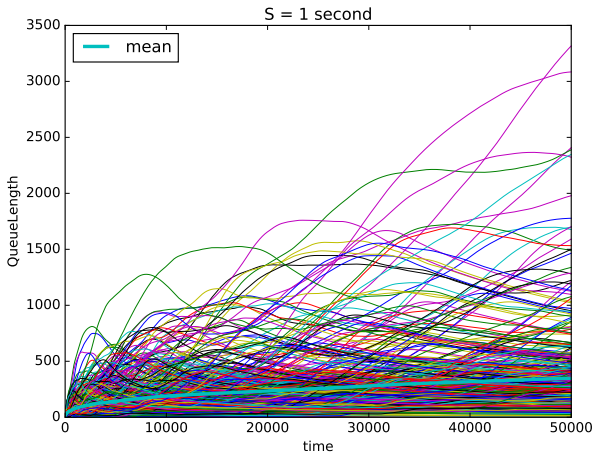

Fig. 21. Trajectories of the queue length

As we can see the average queue length increases slowly and the trajectories are very different among them: this justifies the large confidence intervals.

Fig. 22 shows a Lorenz Curve for the system of the first scenario and for the current system and it confirms that now the sample is more dispersed (i.e. it is more unfair).

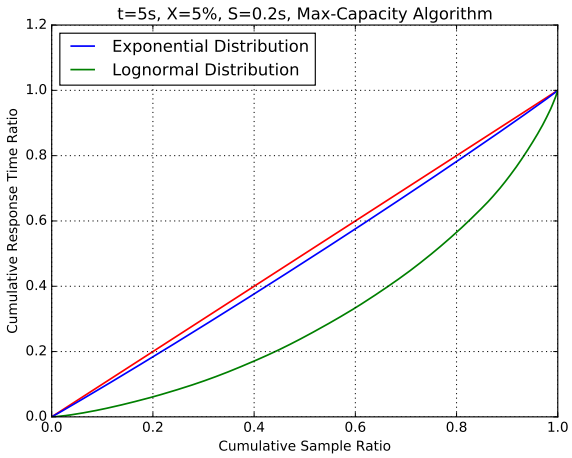

Fig. 22. Lorenz curve

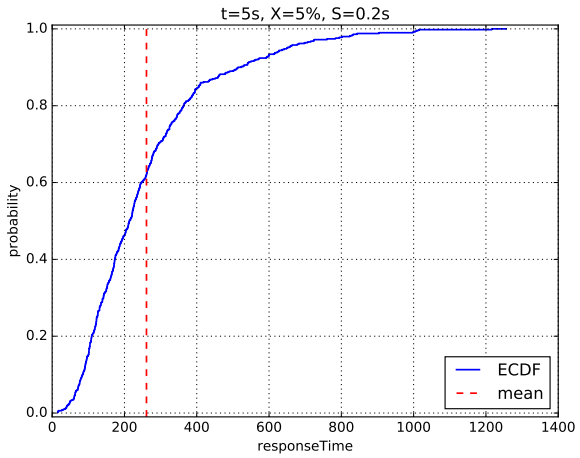

Fig. 24. ECDF

Fig. 23 shows a comparison in terms of means of response time between the algorithm subject of this project and the other two algorithms.

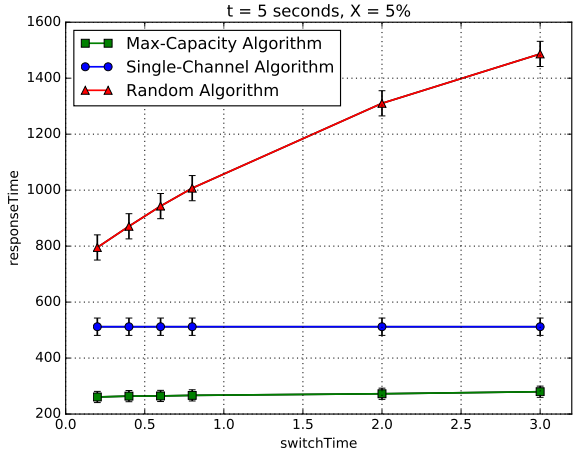

Fig. 23. Performance comparison between the three algorithms

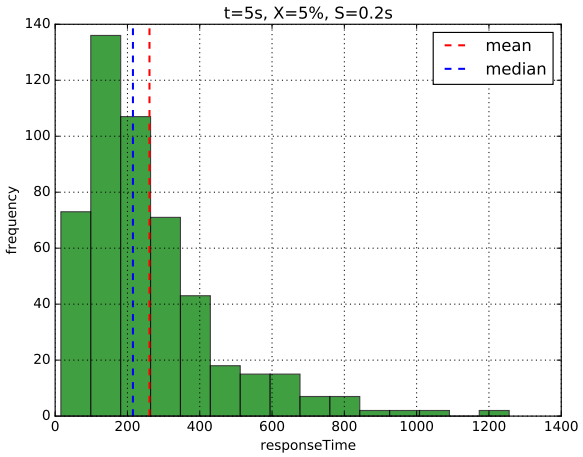

Fig. 25. EPDF Approximation

As we can see performance are worse with respect to the first scenario for all three algorithms: anyway, the algorithm subject of this project seems to be the best choice for the current scenario.

Finally, in order to characterize completely the distribution of the response time, we have plotted the ECDF in Fig. 24 and an approximation of the EPDF in Fig. 25 (using a histogram of 15 bins).

IV. CONCLUSIONS

The analysis of the simulation shows that, at least under some conditions, the use of more than one server results in a concrete improvement of the performance in terms of response time. In particular a key point is the algorithm used for the selection of the server to be used and the parameters that characterized it.

In the first scenario we have explored various combination of the parameters characterizing the algorithm and the whole system, then we have focused on a particular one of them, that guarantees better performance. Thus we have completely characterized the distribution of the response time resulting using this particular configuration. We have also shown that this particular system justifies the utilization of more than one server ensuring better performance only if an efficient selection-server algorithm is used (comparing the algorithm subject of this project with other two alternative algorithms).

In the second scenario basically we have tested the same configuration changing the distributions of the inter-arrival times and the target-capacity selection times: simulation results show that performance are heavily worse than the ones of the first scenario, nevertheless the utilization of more than one server in combination with an efficient selection-

server algorithm results in better performance with respect to a single-server system. Finally we again completely characterized the distribution of the response time resulting using this specific system.

Further analysis could be to find a more efficient combination of the parameters for the second scenario.
Another interesting work could be the analysis of the system using a different policy characterizing the queue: a priority-based policy is already implemented and ready to use.

APPENDIX

In this appendix we discuss the validity of the computation of the confidence intervals.
We have considered the mean of the response times of an aircraft as a sample of an IID RV. Because we have 50 aircrafts, and we performed 10 repetitions of the same experiment (using different seeds), the total of samples available for every set of parameters is 500.
Because the sample is big we can safely apply the following formula for the computation of the confidence intervals:

$$\left[\overline{X} - \frac{S}{\sqrt{n}} \cdot t_{\alpha/2,n-1}, \overline{X} + \frac{S}{\sqrt{n}} \cdot t_{\alpha/2,n-1}\right]$$

Furthermore note that, since the sample width is greater than 30, Student's T distribution overlaps a Standard Normal, hence we can compute percentiles of the Standard Normal instead of the Student's T distribution.