Performance Evaluation of Computer Systems and Networks

Project 11 : Queuer and Servers

Jacopo Carlon 587412

Omnet++ modelling :

(Spawner) : generates jobs following a customizable time distribution, sends them to Queuer

Queuer : FCFS infinite queue, manages queueing logic :

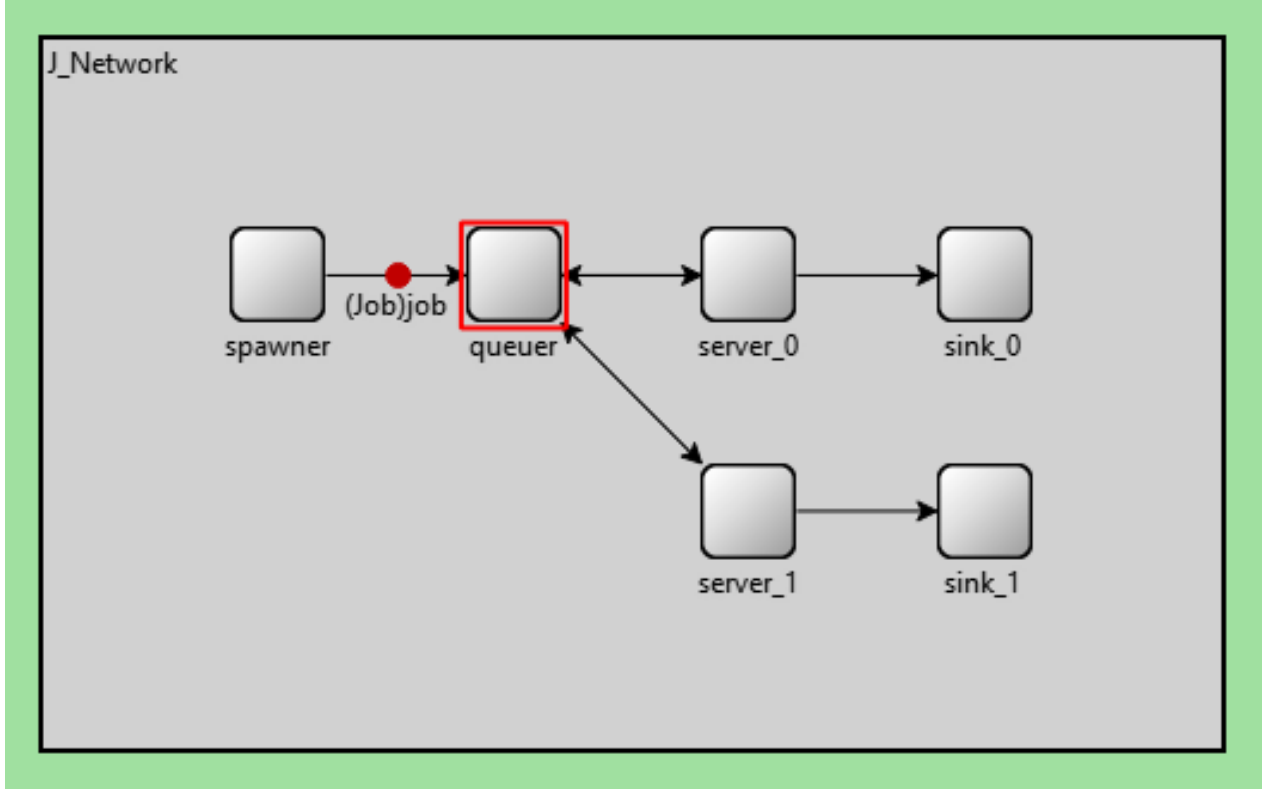
- When new_Job arrives from server, put it in queue, then :
- - If both servers occupied, do nothing ;
- - If both servers empty, assign to Server0 with probability *p*, else to Server1 ;
- - If only one server empty, assign job to that server
- When receive “free” msg from server, then try again to send job (if there is a job, it goes to that server, because had the other server been free it would have already gone there)

(keep free_server_bits value to know which server is free when, update when sending job or receiving “free”)

Server : elaborates jobs following a customizable time distribution :

- Receive job from queue, begins “working”
- After “serviceTime” :
- - send job to Sink
- - send “I am free” message to Queuer

(Sink) : receives jobs and destroys them



System Description : ports

- 1 infinite FCFS queue : Queuer
  - input jobIn : receive jobs from *Spawner*
  - output jobOut[] : send jobs to Server0 or Server1
  - Input ctrlIn : receive control-message from servers, so as to know that a server is free
- 2 servers : Server0 and Server1
  - input : jobIn : receive assigned job from Queuer
  - output : msgDone : send “Free” message to Queuer
  - output : jobOut : send job to *Sink*

Spawner and *Sink* are used to mimic the outside real world, which can generate (with a customable distribution) jobs, which are queued in my J_Network, and after some elaboration time are received back.

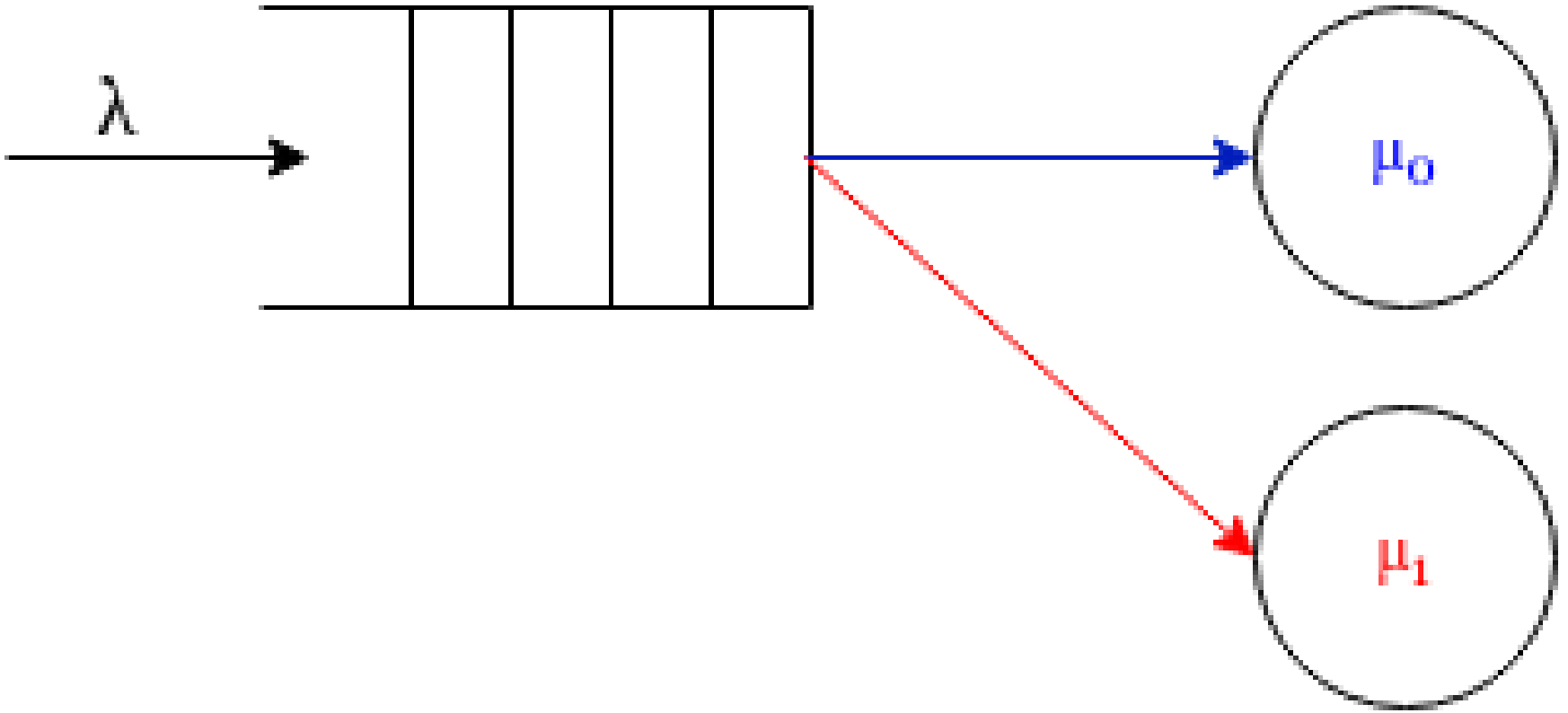
It is possible to create an equivalent system with only queuer and servers.

Queuing system modelling and main variables

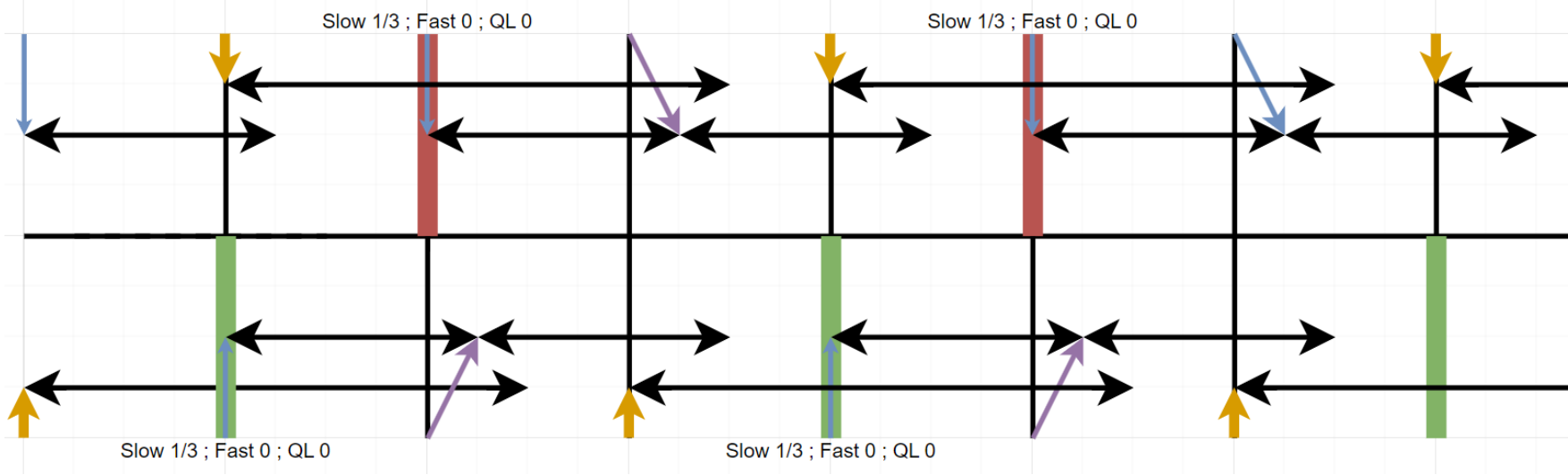
- Parameters:
  - t_n: interarrival time = $1/\lambda$
  - t_{so_0}: service time of Server0 = $1/\mu_0$
  - t_{so_1}: service time of Server1 = $1/\mu_1$
  - *p* : probability of choosing Server0 [0,1]

  (tso_1 = 2*tso_0)

- Studied:
  - Response time in both servers
  - QueueLen in queuer



Probability ..?



This image is meaningful only for TN<TS<tn*3/2

In case of <u>Constant</u> TN and TS :

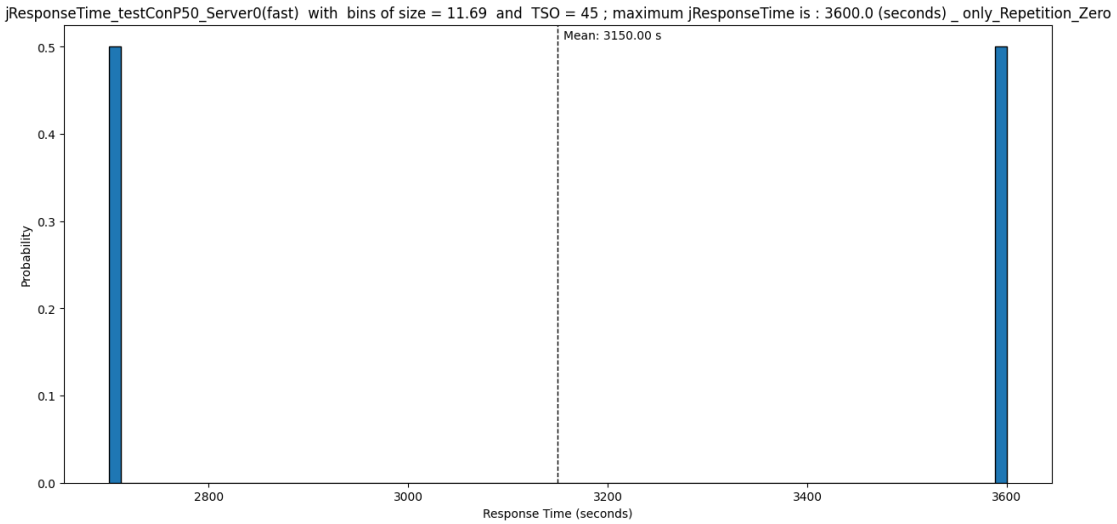
- If p==0 AND TS<TN/2 :
  - then we always choose server1, and it is always fast enough to serve the inputs
  - -> QueueLenght == 0 and ResponseTime = S1_RT == 2*TS
- If p==1 AND TS<TN :
  - then we always choose server0, and it is always fast enough to serve the inputs
  - -> QueueLenght == 0 and ResponseTime = S0_RT == TS
- In general, in non trivial cases i.e. TS > TN, then probability is basically non-influential on average RT :
  - If QL==0 and both servers are free, when either server is chosen, the following job will go to the other server
  - The servers will then begin alternating jobs since a single server would be unable to cover input rate
  - Also sperimentally we can observe that response time is not influenced by probability
- If TS > TN*3/2, then the system is oversaturated and p is also non-influential

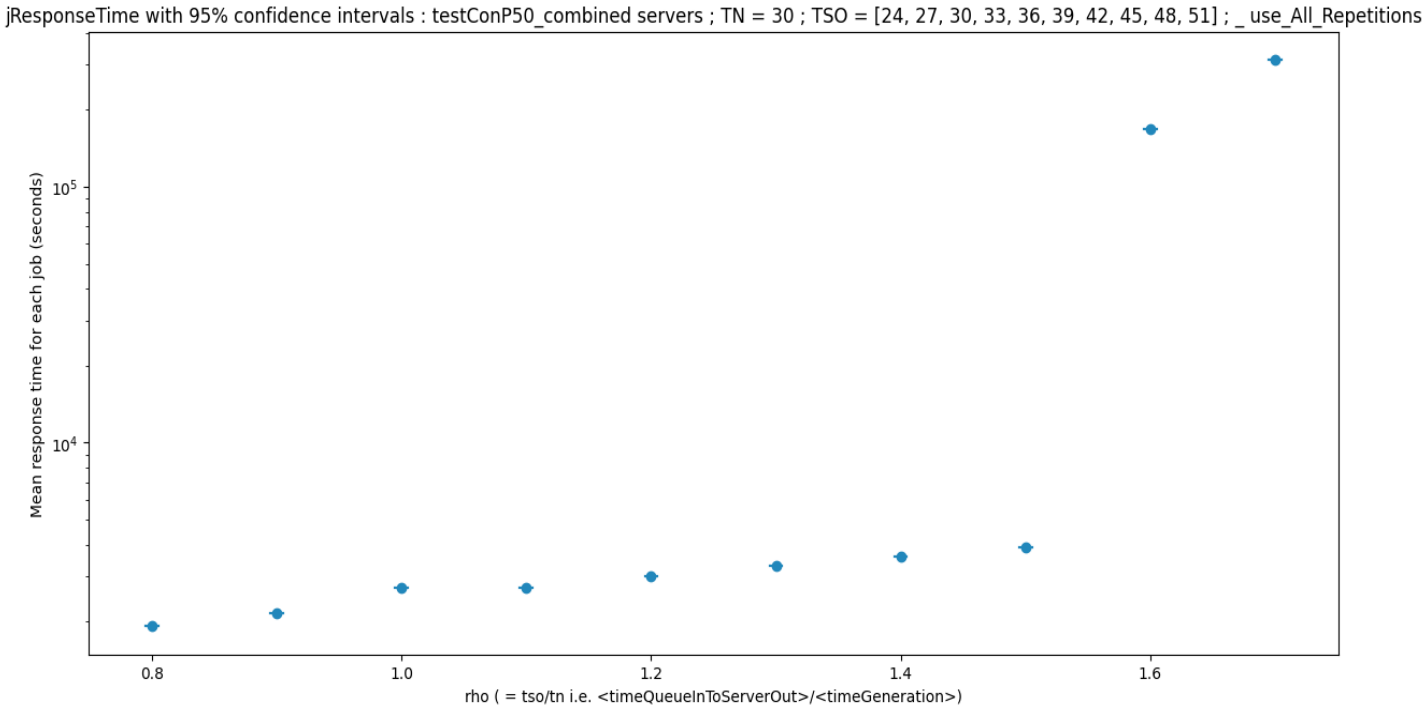
Deterministic system : constantTime TN and TS

- Stability condition: input rate <= output rate : 1/TN <= 1/TS + 1/(2TS) : TS <= TN*3/2 .
- If TS = TN*3/2 then :
- - Both servers occupied all the time (job-receiving/sending and msg-sending have 0-time)
- - System evolves periodically with period = 2*TS==3TN : S0 does 2 jobs, S1 does 1 job
- - QueueLengt is 0 for 5/6 of the period, and 1 for 1/6 of the period (one job needs to wait TN/2 for S0 to be free before being serviced)
- - -> can calculate theoretical response time for S0 : [TS + (TS+0.5*TN)] / 2TS == TS*7/6
- - -> can calculate overall RT : TS(7/6 * 2/3 + 2* 1/3) =TS*13/9== TN*13/6 : experimentally confirmed !
- Since we have constant interarrival ratio (and since our system doesn't create nor destroy jobs), we can use Little's Law to estimate the mean number of jobs in queuer-servers system : E[N] = E[R]*TN = 13/6

Right : y : Average RT(s) LogScale ; x : ratio TS/TN-> RT = 65'==3900s

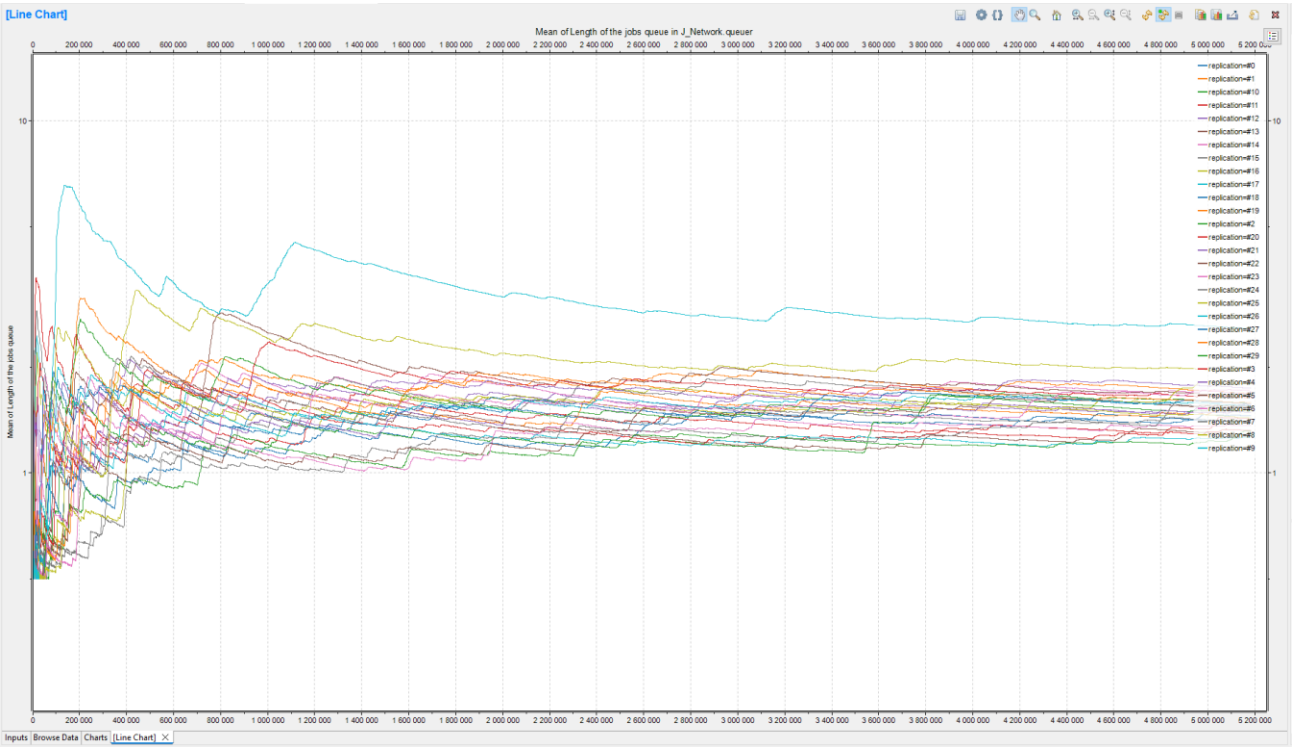
Below: y : probability ; x : Average RT(s)
(0.5,0.5) : (2700 (45*60), 3600(45*60+15*60))s

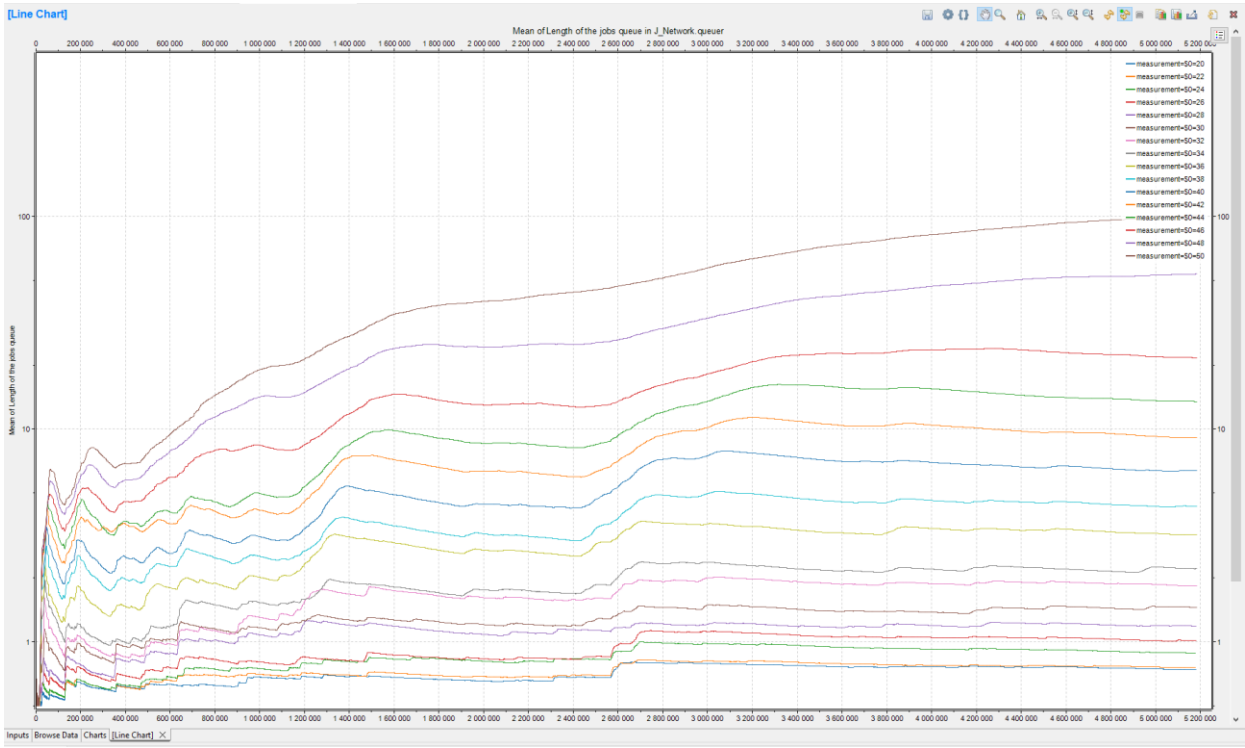




Preliminary analysis for the exponential case

- Warmup time: TN == TS : and varying in ratio [2/3..5/3 step0.3]

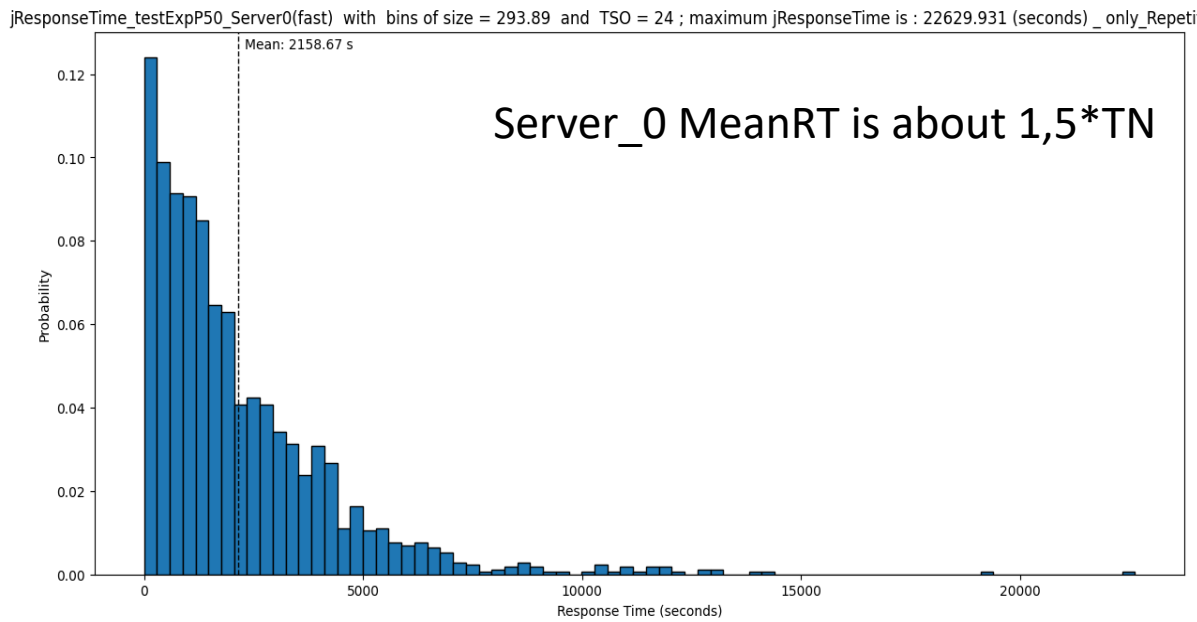


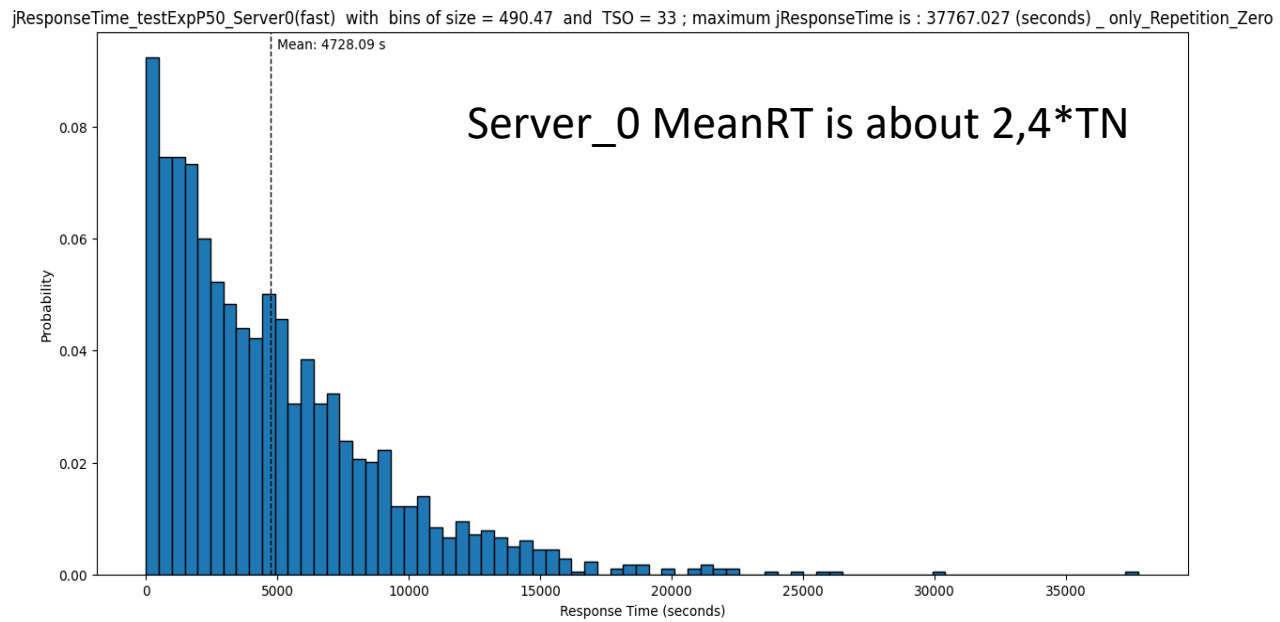


Plots above are over 60d with TN = 30’, Warmup time was chosen with conservative value of 20d

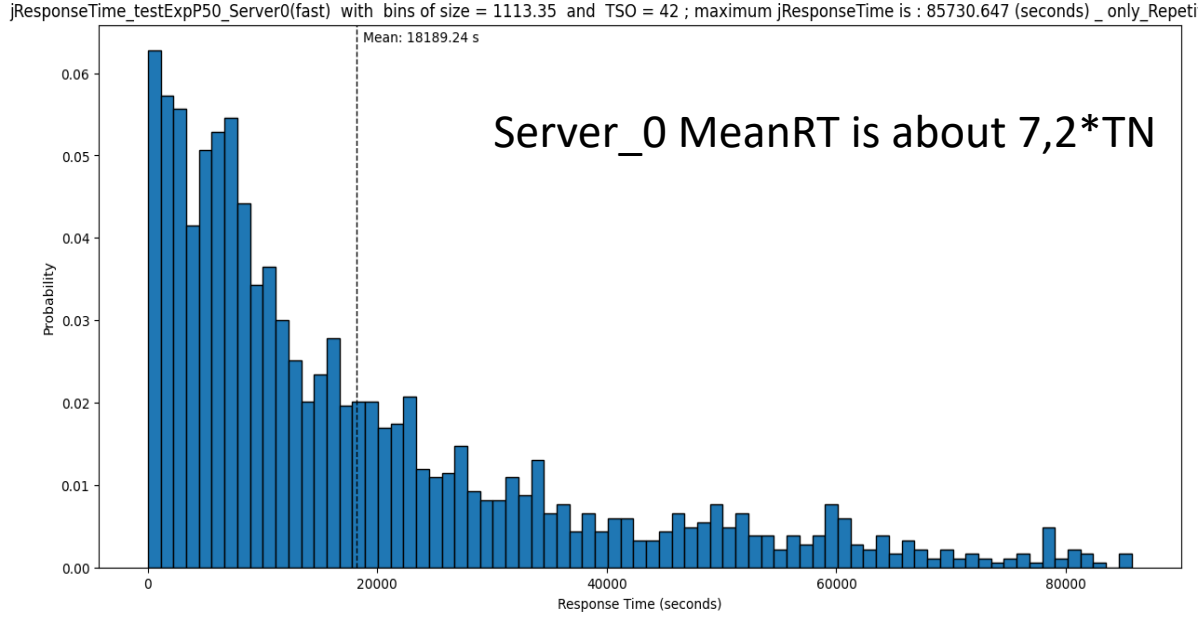
Even in the most simple systems, the saturation limit of constant distribution is the maximum.

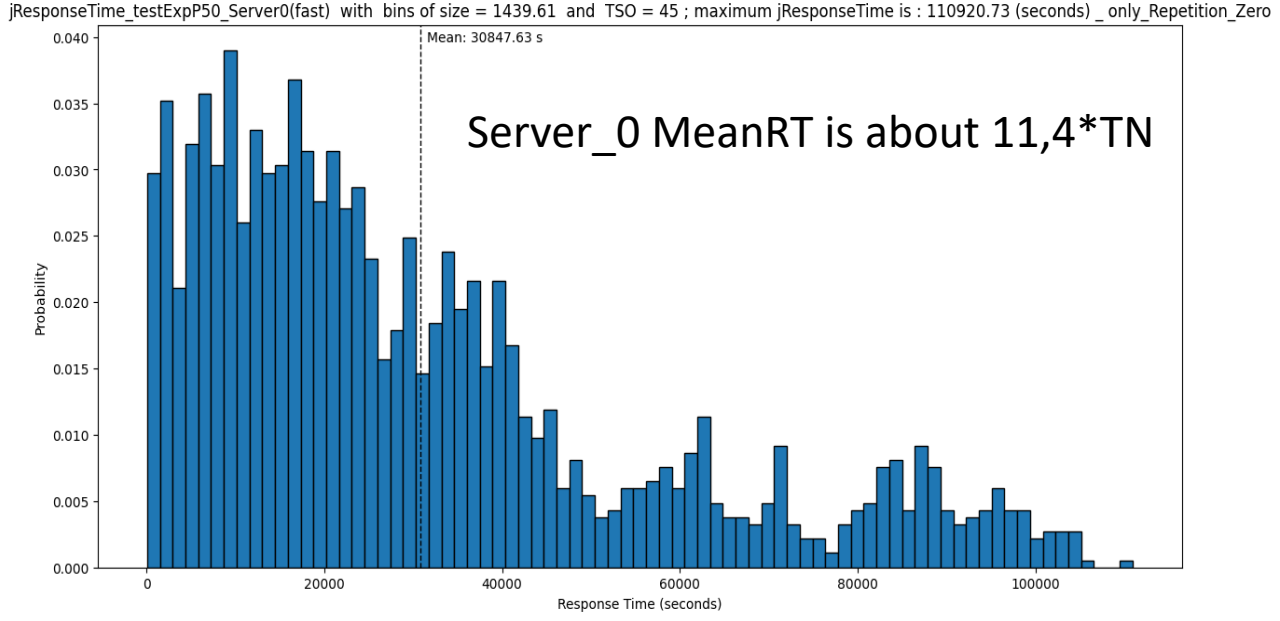
Since even few “unlucky” oscillation can sensibly increase queueLenght and thus responseTime, it is expected for the exponential case to be already oversaturated at TS = TN*3/2. This evaluation was numerically confirmed!





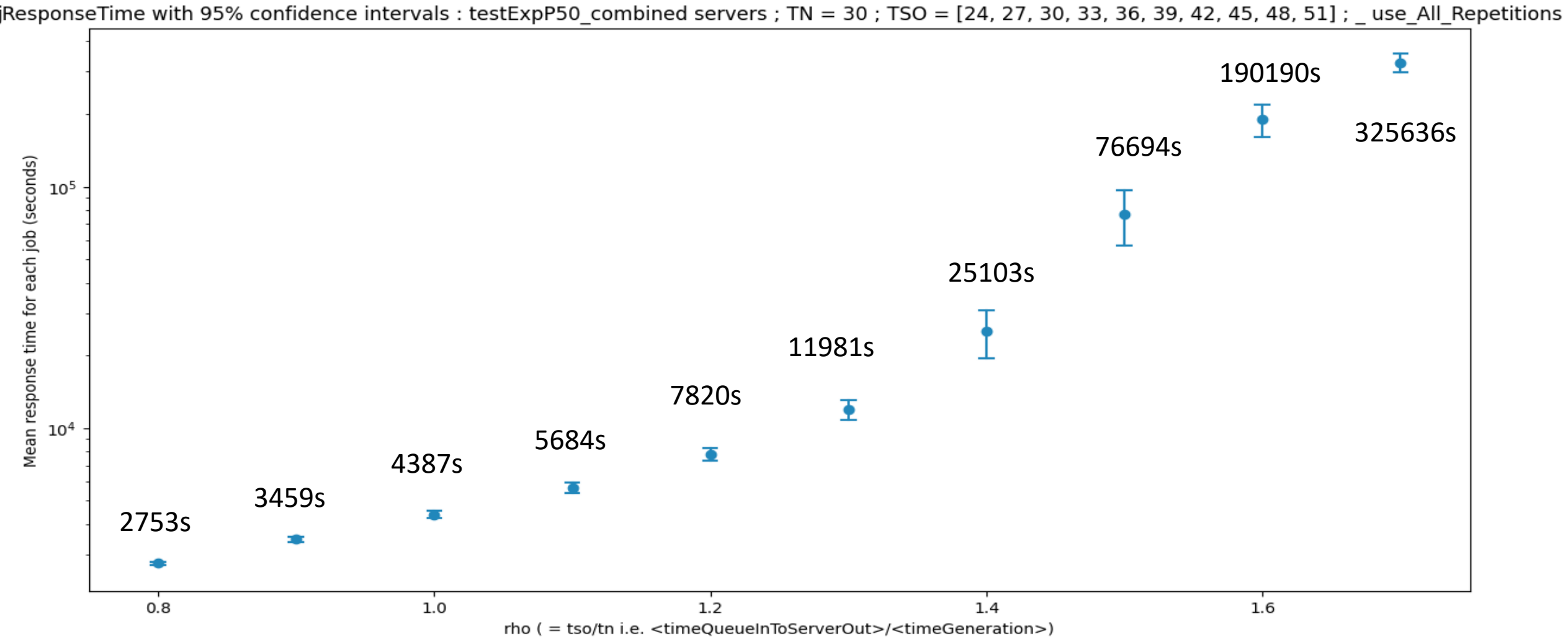
Exponential Case : ResponseTime at different TSOs, keeping TN=30’





Exponential Case

Confidence Invervals over 30 repetitions,
Using TN = 30’ and probability = 0.5

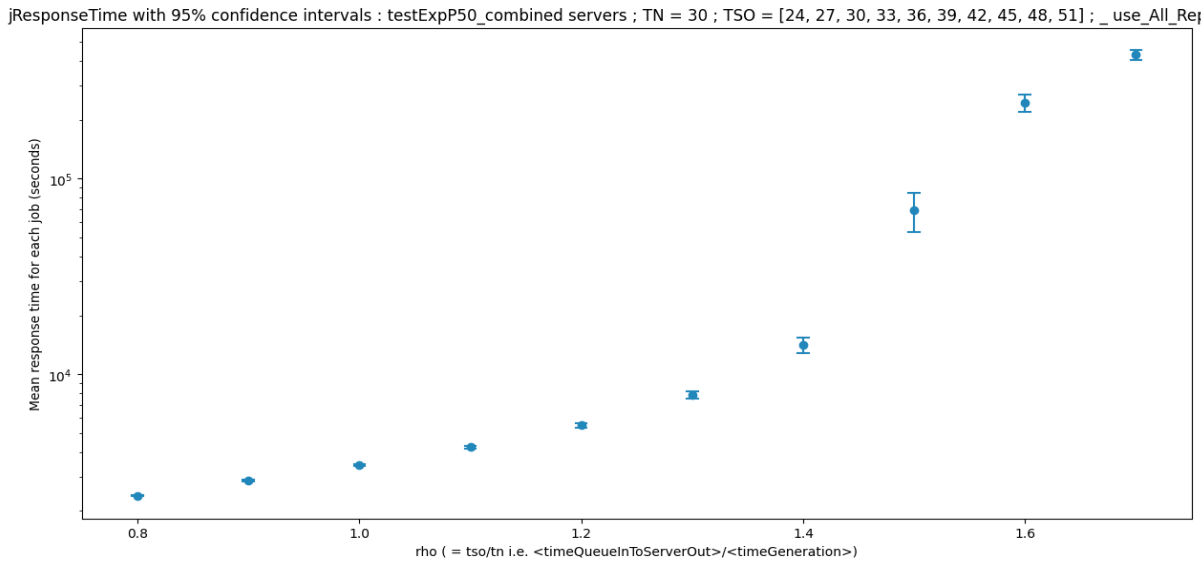


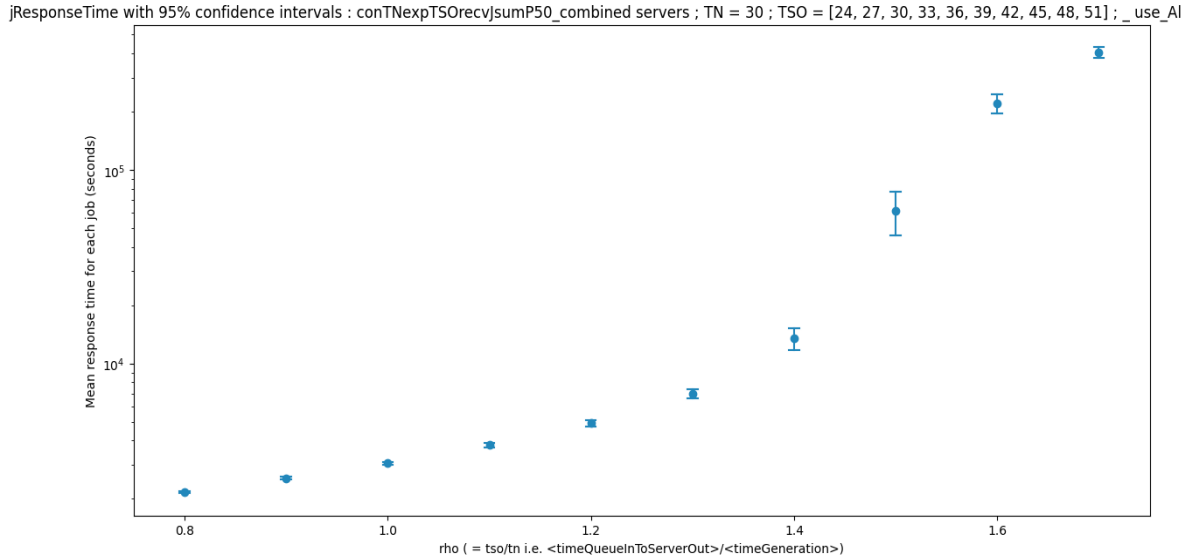
Exponential combinations :

exponentialTN; constantTS
ST=39m=2340s; mean : 7834s
ST=42m=2420s; mean : 14419s
ST=45m=2700s; mean : 68830s

exponentialTN; exponentialTS
ST=39m=2340s; mean : 11981s
ST=42m=2420s; mean : 25103s
ST=45m=2700s; mean : 76694s

constantTN; exponentialTS
ST=39m=2340s; mean : 6998s
ST=42m=2420s; mean : 13476s
ST=45m=2700s; mean : 61419s





Comparisons of combinations with exponentials :

- Ratio < 1.5 : expTNconTS is better than expTNExpTS
- (All Ratio <=1.5 conTNexpTS is the best of 3)
- All Ratio >1.5 expTNexpTS is the best of 3

➔ If generator is exponential, than better constant server (if the system is not saturated)
➔ If generator is constant, also better a constant server
➔ If system cannot manage input, than all is saturated, but exp-exp grows the least (among exponentials)

Conclusions

- Probability is non-influential for all any TS > TN; for lower values of TS, favoring the faster server helps reducing the mean Response Time (even down to TS if p==1) -> there is overall no reason to use any p different from 1.

- The system with constant TN and TS reaches saturation at TS = TN*3/2, having all servers working at all times. In this situation, meanRT is TN*13/6.

- Before saturation, exponential servers work better with constant input-distribution

- Before saturation, exponential inputs are better served by constant servers

- Having any exponential distribution noticeably increases RT, and lowers the saturation limit, and increases sensibly the worst-case-RT.

- The best performance of the system is (as one would expect) with only constant distribution.