Verifying the sojourn time distribution in queueing systems with heavy tailed workloads

Boom, Erik F.J.

Award date:
2016

Verifying the sojourn time distribution in queueing systems with heavy tailed workloads

Erik Boom Student number: 0841779
e.f.j.boom@student.tue.nl

August 2, 2016

Contents

Abstract

Recently, Walraevens et al. [2] and Boxma et al. [3] have figured out how the sojourn time distribution behaves in different queueing systems, in a theoretical sense. In particular it has been proven that it asymptotically does not depend on the inter-arrival time distribution. But it is less known how much that translates to the real world. Using simulation and log-log plots, this report tries to capture and analyse the behaviour of the sojourn time distribution.

1 Introduction

Nowadays, optimization is a hot topic in the world. Everything needs to be faster, it needs to cost less resources, etc. Within queuing theory, it is easy to see why one would want to optimize the time a customer spends in the queue, since no one likes to wait.

Optimization in itself is quite a difficult problem, especially because you can optimize on different elements of the problem, rather than just one. But there is a large underlying problem here as well. To be able to optimize a system, one needs to know how it behaves exactly. Using queueing systems as an example, one wants to find out how the sojourn time (the time a customer resides in the system) behaves, given different policies. With this report I am trying to tackle that part of the problem.

The report is structured as follows. In Section 2, a literature study is conducted, where some queueing systems have been investigated and some findings have been made. In section 3, I try to verify these findings. Section 4 contains an extension of a described system, to see if that would make a substantial difference or not. The report ends with a small conclusion and discussion.

2 Literature Study

2.1 Introduction

In this section I will present the findings in the literature I have studied. The main point in the studied articles is that they stray away from the standard M/M/1 queue and are more general. First, Barabasi gives reasons why it is better to stray away from this particular system, by giving real world examples [1]. In Walraevens' article [2] this is further built on using asymptotic analysis, and an outcome for the sojourn time distribution in an M/G/1 queue is given. Finally, Boxma and Denisov [2] describe a system where it is possible to choose between different outcomes for the sojourn time distribution. These three articles will be summarised and reviewed in this section.

2.2 Summary of [1]

2.2.1 Introduction

In the early 2000s, it was widely believed that when a human engages in doing tasks, such as email-handling, the inter-event times (the time between the start of two consecutive activities,

denoted by T) are independent and exponentially distributed: $P(\tau) = \mathbb{P}(T > \tau) \approx e^{-\lambda\tau}$, with $1/\lambda$ the average inter-event time. Therefore, the whole process of a human engaging in tasks is well approximated by a Poisson process. But some empirical evidence has been found that the Poisson process is often not a good approximation, and that the inter-event times are then better approximated by a heavy tailed distribution: $P(\tau) \approx \tau^{-\alpha}$, for some α, usually close to 1. The independence of the tasks remains. An important difference between the two, though, is that long inter-event times occur much more often when the distribution is heavy tailed.

With this paper under review [1], Barabasi investigates the reason behind the findings. He believes that the decision making of humans is the cause: humans decide what task to do next based on some kind of priority. An example of this would be to pick the shortest task next. This means that tasks that take a long time to execute (in this example) could wait for a very long time, which is modelled better by a heavy tailed distribution. Barabasi backs this up with a mathematical model of a queue, where each task has a certain priority.

In Section 2.2.2, the main ideas of the mathematical model will be explained (for further elaboration, see [1]) and in Section 2.2.3 these ideas will be reviewed.

2.2.2 Model

The model consists of a queue with a single server, containing $L \geq 2$ tasks initially. Each task in the queue has its own priority x, chosen from a distribution $p(x)$. The following is also assumed:

- There are always exactly L tasks in the queue, waiting for service. This assumption indicates that a decision always has to be made which task to do next.
- The service times are i.i.d. Even stronger: Each task takes the exact same time to be executed.
- The task with the highest priority is chosen with probability p. In most of Barabasi's derivations, the limit $p \to 1$ is used, indicating that the task with highest priority is always chosen.
- The server is non-preemptive: Whenever a task is taken into service, the task is executed in its entirety.

The model is used to determine the distribution of the inter-event time of two consecutive tasks, $P(\tau)$(as defined before). An intermediate variable is used, namely the average waiting time $\tau(x)$ of a task with priority x (it follows from this that the inter-event time depends on the priority, which makes sense). This is believed to be approximately $\frac{1}{x^\gamma}$, with γ a parameter between 0 and ∞ (when $p \to 1$, $\gamma \to \infty$). Next, Barabasi uses that $p(x)dx = P(\tau)d\tau$ (How this is seen is left rather vague, as Barabasi does not elaborate on this fact.). Using this, it follows that in the deterministic case ($p \to 1$), $P(\tau) \approx \tau^{-1}$, indicating a heavy tailed distribution. This is checked against the empirical data, and the distribution fits (including the -1 in the exponent). Barabasi concludes that this must follow from the priority-based mechanism, using empirical arguments again.

2.2.3 Review

There are some problems I have with this article, mainly due to the assumptions made in the model:

- If there are always L tasks waiting in the queue, that would mean that whenever a task is finished, another task is immediately put in the queue. Not only does this completely stray away from a standard waiting queue, it strays away too much from reality. No server is busy all the time (not even when the average service time and the average inter-arrival time is the same). Barabasi also does refer to a model where the size of the queue is allowed to change in time though.
- In the model, all the service times are the same. For one, it is very interesting that the inter-event time is not the same then. Assuming that they are not, the service times might just be the reason for the heavy-tailed characteristics. As acknowledged, the size of emails follows a heavy tailed distribution. Only empirical arguments are used to say that the priorities are the main cause for the occurrence of heavy tails. I do not believe that is enough.

Even though I disagree with the assumptions made, I do support the conclusion that the engagement of humans in a number of tasks is not well approximated by a Poisson Process. Whether or not this is purely due to the priorities is a different matter, where I do not completely agree with Barabasi.

2.3 Summary of [2]

2.3.1 Introduction

Along with Barabasi, others studied how to model humans engaging in tasks, where each task has a certain priority associated with it. Problems that occured were that assumptions had to be made on the distribution of either the inter-arrival times or the service times. A more general model at this point in time was unknown.

With this paper under review [2], Walraevens et al. have found a way to accurately describe these human activities and mathematically analyse them, mainly through the use of generating functions and asymptotic analysis. Using this model, one can deduce all the characteristics of the total waiting time (or sojourn time, waiting time plus execution time).

In Section 2.3.2, the main ideas of the model will be explained, with some examples in Section 2.3.3 and the model will be reviewed in Section 2.3.4.

2.3.2 Model

Like with Barabasi's model, a standard queue with a single server and i.i.d. tasks is used. Each task also has a priority value x, chosen from a uniform distribution in Walraevens' case (although he mentions that a general distribution could be used). The main difference with [1] is that the distribution of inter-arrival times and execution times is general. Also note that time is modelled discretely.

The notion of pre-emptiveness is implicitly used in the model: Whenever a task with a higher priority arrives than the task currently executed, the higher priority task is executed and the lower priority task has to wait.

This model is used to determine the sojourn time of a task (as defined in Section 2.1). Now, let a task have priority x. Then its sojourn time is affected only by the tasks that have a priority higher than x, because it always gets taken into service before a task with lower priority has begun execution. Therefore, for each task, one can divide the continuous priority levels into two classes: Higher and lower priority. In an earlier paper, Walraevens et al. [4] found the generating function of the sojourn time of a task with priority x, making use of these two classes.

Now one can apply the following theorem from asymptotic analysis to the generating function (see [5] for more information):
Theorem 1: Let $Y(z)$ be the generating function of a random variable Y, with dominant singularity R_Y. Let $\beta \in \mathbb{R} \setminus \{0, 1, 2, ...\}$. If the following holds:

$$Y(z) \sim c_Y(1 - z/R_Y)^{\beta}, \qquad z \to R_Y,$$

then the distribution $y(n) = \mathbb{P}(y = n)$ satisfies

$$y(n) \sim \frac{c_Y n^{-\beta-1} R_Y^{-n}}{\Gamma(-\beta)}, \qquad n \to \infty.$$

Proof of this theorem can be found in Appendix A.

What this means is that when the distribution and its dominant singularity is known, a good approximation of the distribution function is known. Using this with the found generating function, one can find the asymptotic distribution of the sojourn time.

2.3.3 Examples and conclusions

Walraevens allowed the distributions of the inter-arrival and the execution times to be general. In [2], some examples of specific distributions are investigated:

1. Both the number of arrivals in a time step and the execution times are geometrically distributed (so-called "standard input"), with averages $\frac{1}{\lambda}$ and $\frac{1}{\mu}$ respectively. Applying the model yields the following distribution for the sojourn time for a task with priority x : $d_x(n) \sim cn^{-\alpha} R_Y^{-n}$, where R_Y is its dominant singularity (which is dependent on x), yet α is unknown. This is called a power law (power law and heavy tails have the same meaning) with an exponential cut-off. Because of the "standard" input, it is concluded that the distribution must follow from the priority selection. It is also noted that when $\lambda \to \mu$ (the (in)stability border), the exponential cut-off is gone and the power law remains, as was the case with Barabasi's article.

2. The execution times remain geometrically distributed, but the number of arrivals in one time step has a power law distribution. It is now argued that the sojourn time is also heavy tailed, but with an exponent 1 higher than the exponent of the arrivals' distribution (a "heavier" tail). As this seems to be a general outcome for power law

distributed arrivals, it is concluded that the heavy tail follows from the heavy tail of the inter-arrival time functions.

3. Now the arrivals are again geometrically distributed, but the execution times are heavy tailed. Basically the same outcome as with example 2; a heavy tail is encountered, and it follows from the heavy tail of the service function.

4. The execution times are again geometrically distributed and the inter-arrival times have a power law with exponential cut-off (see Example 1). Now the dominant singularity depends on the average inter arrival rate λ. If this average is relatively low, the singularity coming from the service time function is dominant. If this average is high, the priority selection produces the dominant singularity. In both cases, the sojourn time is distributed the same as in Example 1, but thus by different causes. The concluding remark is that for stable systems the service time function is always the dominant factor.

2.3.4 Review

I have some issues with this paper. The first one is about the pre-emptiveness of the model. It is not explicitly mentioned, although it plays an important role in the derivation of the generating function of the sojourn time [4]. A short mention would have been fine.
Second, the paper makes use of a general result that the heavy tail must come from the service or inter-arrival time distribution and that the tail is one degree "heavier". For a particular policy, Last Come First Served Pre-emptive Resume, this is not the case, as is shown in [6] . Here the tail has the same degree as the service time function, which contradicts this "general result" and actually harms the main conclusion drawn from it. I will run a short simulation on an M/G/1 queue with heavy-tailed service times and a First Come First Serve policy to try and verify the conclusion drawn from this article for the simple case also mentioned in this general result.

2.4 Summary of [3]

2.4.1 Introduction

At this point in time it was known that the sojourn time of the customers in a GI/GI/1 queue (the distributions are general, but all customers are independent of each other), with some sort of priority scheduling policy, has a power law distribution, given that the execution time of the customers is power law distributed as well. What remains to be investigated was the exact exponent of the distribution: Let B_i and W_i be the execution time and the sojourn time respectively. If $\mathbb{P}(W_i > x) \sim cx^{-\alpha}$, α needs to be determined, given that $\mathbb{P}(B_i > x) \sim cx^{-\gamma}$.
A known result for this problem is $\alpha = \gamma$ for the Last Come First Served Pre-emptive Resume policy, and $\alpha = \gamma - 1$ for the First Come First Served policy. Both of these policies are thoroughly investigated in [6]. Boxma and Denisov investigate in [3] if other results are possible as well. They do this by straying away from so-called "static" priorities: Once they are determined, they cannot change any more. This opens up a lot more possibilities for service policies.
In Section 2.4.2 the particular service policy will be elaborated on, with the main result being presented. In Section 2.4.3 the policy and the result will be reviewed.

2.4.2 Service policy and main results

The service policy works as follows: The inter-arrival times are i.i.d. random variables with distribution A, and the execution times are i.i.d. random variables with distribution B. To ensure stability, let $a := \mathbb{E}[A - B] > 0$. At first all customers have high priority and are serviced in First Come First Served order. Now, let $\beta \in (0, 1)$. Consider customer i, with execution time B_i. Let M_i be the maximum execution time of all customers that arrived earlier than customer i, but belong in the same busy period as customer i (a continuous period in which the server has work). If $B_i \leq M^{1-\beta}$, then the customer will keep the high priority status until it is fully served. Otherwise the customer only receives service for $B_i^{1-\beta}$ time, returns to the end of the queue and receives low priority. This not only implies that high priority customers receive service before this customer, but the customer's service can also be pre-empted by a high priority customer. Low priority customers are served in First Come First Served order with respect to other low priority customers.

To present the main result, the notion of regularly varying distribution needs to be introduced: A distribution $F(\cdot)$ on $[0, \infty)$ is called regularly varying of index α, if $1 - F(x) = x^{-\alpha}L(x)$, with $\lim_{x\to\infty} \frac{L(x)}{L(\eta x)} = 1$ for all $\eta > 1$. For example, the power law distribution is regularly varying of index α if the index of the power law is α.

Now, the main result of the paper is the following theorem:

Theorem 2: Consider a GI/GI/1 queue with above described service policy. Assume that $\alpha > 1$ and $\beta \in (0, 1/(\alpha + 1))$. If the distribution of the execution time of the customers is regularly varying of index α, then the following asymptotics hold:

$$\mathbb{P}(W_i > x) \sim \frac{1}{a} \int_x^\infty \mathbb{P}(B_i > y^{1/(1-\beta)})dy, x \to \infty.$$

In particular, the distribution of W_i is regularly varying with index $\frac{\alpha}{1-\beta} - 1$.

The proof of the theorem consists of constructing a lower and an upper bound for the distribution and showing that these are equal. But due to the length of the proof I refer the reader to [3]. The consequences of this theorem are more important, as $\beta \in (0, 1/(\alpha + 1))$ means that $\frac{\alpha}{1-\beta} - 1 \in [\alpha - 1, \alpha]$, meaning that the sojourn time of customers in a GI/GI/1 queue can be regularly varying with any index between $\alpha - 1$ and α. Not only that, but apparently the sojourn time has the same tail behaviour, independent of how the inter-arrival times are distributed (as long as stability is ensured).

2.4.3 Review

The one issue I have with the service policy is that it is incomplete. The step of calculating M_i might not be possible, as there could be no predecessors of customer i in the same busy period. Does M_i become 0 then, or does the customer have high priority until it is fully served? Arguments could be made for both options. The proof of the theorem is not affected by this though, which means that the main result is unharmed, yet for simulation purposes this choice is important.

What I find interesting is the reason behind only choosing 2 priority classes. Was it just for simplicity, or is there something else? Also, this service is more interesting to simulate, mainly due to the number of variables. Both α and β are free to choose, along with both the inter-arrival and execution time distributions, alongside the possible choice for multiple priority classes. This is the reason I choose to simulate this policy more extensively than the "regular" M/G/1 queue.

3 Simulation study

3.1 Introduction

In Section 2.3, a policy has been explained with some interesting properties. The main goal of this project is to simulate a queue with this specific policy, to see if those properties indeed hold. For reference purposes, a "simple" M/G/1 queue (with the First Come First Serve policy) is also simulated. Last, a possible extension to the policy in Section 2.3 is explained and analysed.

The simulation is done in the Java language, as I am the most experienced in this particular language, even though it is not the most obvious choice to simulate queues. This comes with some limitations, for instance not being able to plot graphs so easily. Therefore I implemented the parts where Java is limited in the R language.

In Section 3.2 I will explain the choices that needed to be made with regards to parameters, and in Section 3.3 I will present the results with a small discussion.

3.2 Parameter choices

As described before, the particular queue I would like to simulate brings a lot of variables with it. These variables have to have certain properties, or be within certain ranges, in order for the model to correctly predict the behaviour. In this section I will describe these variables and their concrete values.

Execution times

In both [2] and [3] a lot of thought is spent on the distribution of the execution times (or service times), as apparently it is the main cause for the heavy tailed behaviour of the sojourn times.

In order for the sojourn time to be heavy tailed (or regularly varying), the distribution of the execution times needs to be heavy tailed (or regularly varying) as well. A relatively simple example of such a distribution is the Pareto distribution:

$$\mathbb{P}(B > x) = (x_m/x)^\alpha,$$

with $x_m > 0$ the minimum value that B can take, and α the shape parameter. Clearly, this distribution is heavy tailed. It is also clear that it is regularly varying of index $-\alpha$.

With regards to the value of x_m, I chose x_m to be 1, as that seemed to be the simplest.

As for α, Walraevens suggests a value in the interval $(2, 3)$ is suitable [2]. I chose the middle ground and went with a value of 2.5. This means that the mean of the distribution is $2.5/1.5 \approx 1.67$, which will be important for stability purposes. An interesting case is also when $1 < \alpha < 2$, as the variance will be infinite in this case. Therefore I also simulated with a value of 1.5. In this case the mean will be equal to 3.

In the policy described in [3], a value for β is needed as well, where $\beta \in (0, \frac{1}{\alpha+1}) = (0, \frac{1}{3.5})$, if $\alpha = 2.5$ or $\beta \in (0, \frac{1}{2.5})$ if $\alpha = 1.5$. With this in mind, I chose $\beta = \frac{1}{4}$.

Inter-arrival times

Since the simulation is about an M/G/1 queue, there is not as much choice here, knowing that the distribution for the inter-arrival times must be the exponential distribution.

The only variable here is the average inter-arrival time $1/\lambda$. This must be higher than the average execution time, in order for the system to remain stable. But the exact average does not matter (much) for the asymptotical distribution, as shown in earlier sections. Therefore I should be able to choose any value for the average, as long as the stability of the system is preserved. With this in mind, I chose multiple variants. Different values will most likely have an effect on the accuracy of the simulation, and therefore this is worth to investigate.

Number of customers

Due to a simulation being a simulation, one cannot have an infinite number of customers. Therefore I might not even be able to reach the asymptotic behaviour I would like to investigate, as the accuracy of the simulation might be too low.

Not only that, but there is a trade-off between simulation time and accuracy. The higher accuracy you want to achieve, the longer it takes. As an example: the simulation I am running takes approximately 3 hours, where I generate 1 million customers. The cause for this is mainly due to inefficient programming on my side, but due to lack of time the structure has not been changed. So, when I increase the number of generated customers, the time increases by a lot (definitely not linear in the number of customers).

Due to me having limited time/resources, the results are to be taken with a grain of salt. This also works in conjunction with an underlying problem, described in Section 3.3. When these results have been presented, I will argue about the accuracy of the simulation, to see if the results make sense.

Simulation code is available in Appendix B.

3.3 Results

Now that the required data has been generated, samples of the sojourn time distribution are present (given the chosen parameters in Section 3.2), and are capable to be analysed. However, there is a large issue, concerning simulations with heavy tailed service time distributions (or workloads), explained by Crovella and Lipsky [7]. They argue that simulations that work with heavy tailed distributed workloads converge very slowly to a so-called steady state. The main reason is because the extreme events (a very large service time) are needed in order

to obtain a decent accuracy. But with a limited number of customers, these extreme events are still very rare. So when simulating with heavy tailed services, a very large number of customers is needed.

But even when this state is reached, the performance of the system is highly variable. The main reason are again these extreme events that need to happen to reach this steady state. Now, the result of this is that simulations can be very unstable, and this can more often than not lead to serious accuracy problems. Whether or not that is the case here is the question. With this in mind, there are two questions I will try to answer:

1. Is the sojourn time distribution regularly varying with index $-\alpha$ (for known $-\alpha$)?
2. If yes to the first question, is this true for all λ, with λ the inter-arrival time distribution parameter (given that the system is stable)?

I also split the cases where the variance of the execution time distribution is finite or not, as I suspect that the behaviour will be quite different, also keeping the instability in mind.

The main idea to tackle these questions is also described by Crovella and Lipsky [7], and works as follows: Let S denote the sojourn time random variable, with distribution F. If $\bar{F}(x) = \mathbb{P}(S > x) \sim cx^{-\alpha}$, then $\lim_{x\to\infty} \frac{d \log \bar{F}(x)}{d \log x} = -\alpha$. This implies that for large values of x, the tail of the distribution should follow a straight line in a log-log plot, if the distribution is regularly varying. Moreover, an approximation of the coefficient is known.

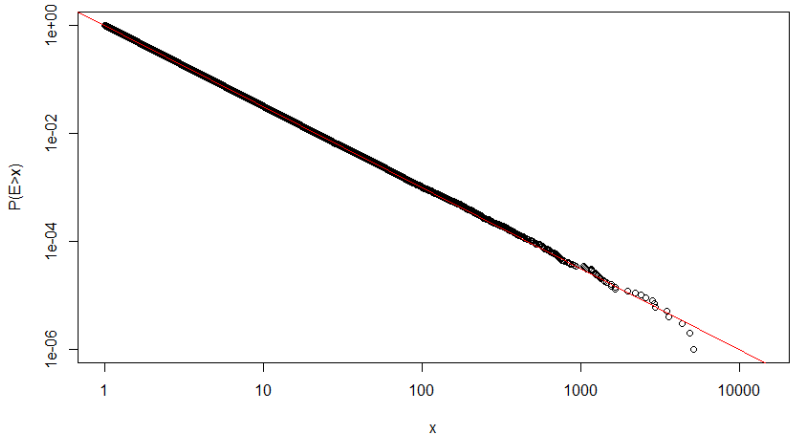

Figure 1: **Random samples of a Pareto distrbution with shape parameter 1.5, on a log log scale. The red line is a straight line with slope -1.5.**

An example of this is shown in Figure 1. This figure shows 1 million random samples of a Pareto distribution with $\alpha = 1.5$ (taken from the execution times of each customer in the simulation). As expected, the distribution follows the red line (which has slope -1.5) almost exactly. The deviation at the end comes from the stability issues described before, due to the limited number of samples, and that a probability is always greater than 0 (whereas the line can cross 0 and become negative).

One can see an arising problem: If the convergence to the steady state is slow, and there might still be instability at the steady state, how reliable will the outcome be, using this method? To this, I do not have a good answer. Crovella and Lipsky give estimates on the accuracy, but I question if it is applicable to this problem. However, this method is by far the easiest and the most understandable I could find. Alongside it, it has the property that if the tail function follows a straight line in the log-log plot, then it is regularly varying: $\lim_{x\to\infty} \frac{d\log \bar{F}(x)}{d\log x} = -\alpha \implies \bar{F}(x) \sim x^{-\alpha}$. $x^{-\alpha}$ is the most 'trivial' regularly varying function (take $L(x) = 1$ in the definition in Section 2.4.2). Because of this fact, I will use this method to try and answer the given questions.

3.3.1 FCFS policy

As described in Section 2.3.3, Walraevens used a so-called general result, regarding service policies, to draw conclusions about sojourn times of the single server queue. In particular, this conclusion is drawn using the FCFS policy [2]. My intention is to validate this conclusion, only using this policy.

As described in Section 3.2, I only leave one parameter to be variable, the average inter-arrival time. The role of this quantity will also be analysed, as pointed out.

To introduce some notation, Let A be the arrival time distribution (exponential with parameter λ, left free) and B be the service time distribution (Pareto with parameter α, 2.5 or 1.5, depending on whether I want the variance to be finite or not).

Finite variance

The first question I wanted to answer was if the sojourn time can be regularly varying of index $-\alpha$, in this case -1.5 (for a certain average inter-arrival time). Because the average inter-arrival time should not matter, I chose it such that the quantity $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$ is close to the middle. This seemed the most intuitive and it does not violate any conditions.

Using a bit of trial and error, a good value for this quantity turned out to be 0.4. As shown in Figure 2, using these parameters turned out to give a good result: for larger values of x, the graph starts to follow the red line (which indicates a regularly varying function of index -1.5) more and more, until the point that there are not enough customers any more (since this is still a simulation). This gives me enough evidence to believe that the first question can be answered as 'yes'.

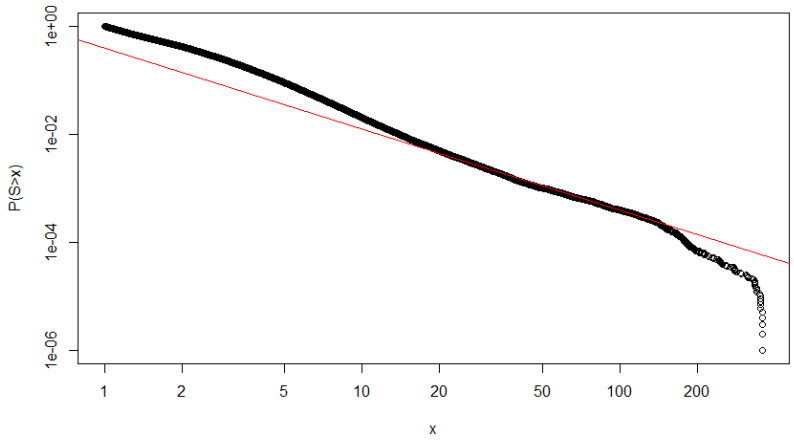

Figure 2: **Sojourn time samples, using the FCFS policy on a log log scale, with $\alpha = 2.5$ and $(\mathbb{E}[B])/(\mathbb{E}[A]) = 0.4$. The red line is a straight line with slope -1.5.**

The second question, whether it is regularly varying for all λ is a lot more difficult to answer, since it asks for a general approach, which is hard to do with a simulation. In this case, the claim will become incorrect sooner or later: Large deviations in the quantity $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$ will have the effect that the asymptotic behaviour will only occur when the number of customers increases greatly. Since I cannot do that without a huge performance loss (see Section 3.2), the effect will be that the sojourn time will behave quite differently than expected (see Figures 3 to 6).

Some behaviour is easily explainable. When the quotient $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$ becomes smaller and smaller, the average waiting time of a customer becomes smaller and smaller (the less customers there are to serve, the less time one has to wait). Therefore the sojourn time will have approximately the same distribution as the service time distribution. This behaviour is visible in Figure 4.

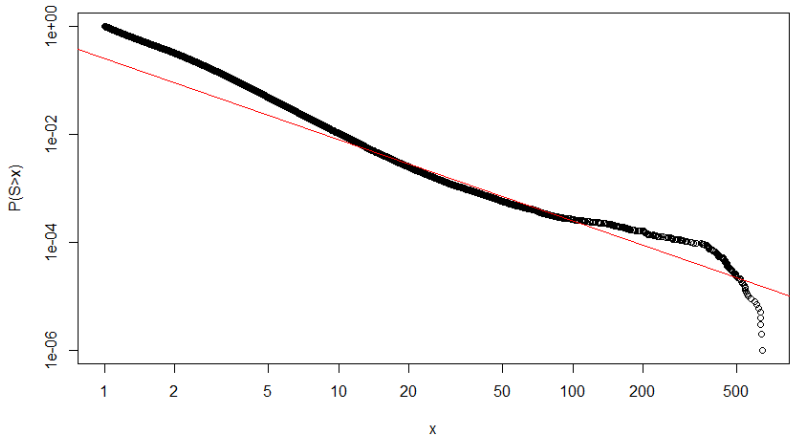

Figure 3: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.25$. The red line is a straight line with slope -1.5.**

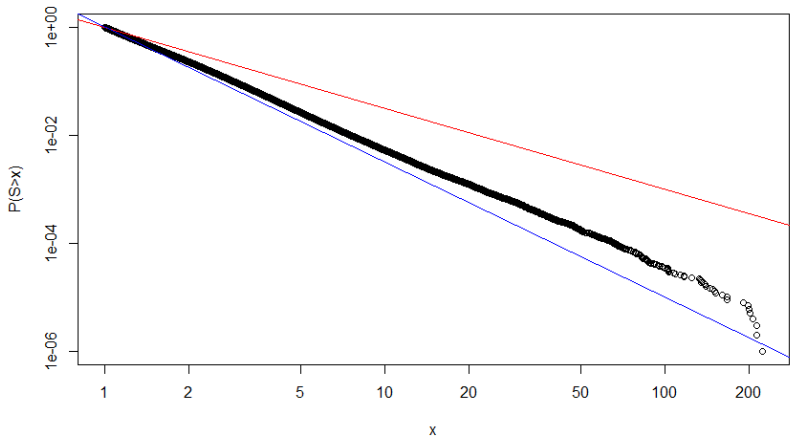

Figure 4: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.1$. The red line is a straight line with slope -1.5. The blue line is a straight line with slope -2.5.**

What is more interesting is what happens when the quotient becomes larger, in the sense that I do not have a clue of what happens. Intuition says that the waiting time becomes more dominant, causing the sojourn time distribution to become 'larger' (in the sense that the average sojourn time is larger), but this is not evident from the graph (see Figures 5 and 6).

A small conclusion is that the first question is answered 'yes', but the second is still unanswered, due to the (in)accuracy of the simulation.

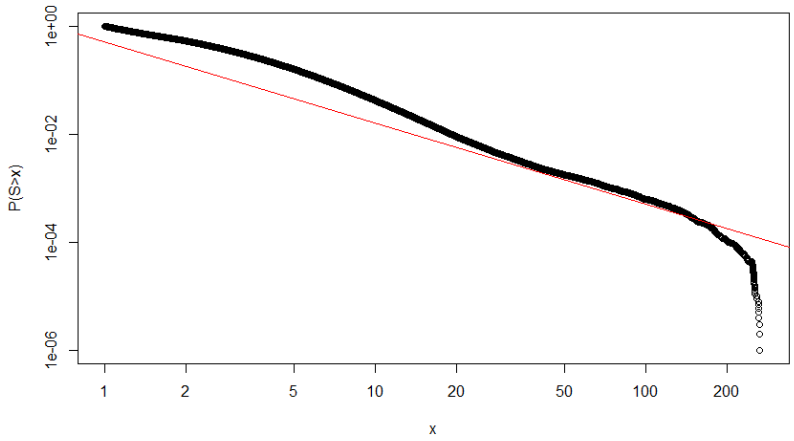

Figure 5: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.55$. The red line is a straight line with slope -1.5.**

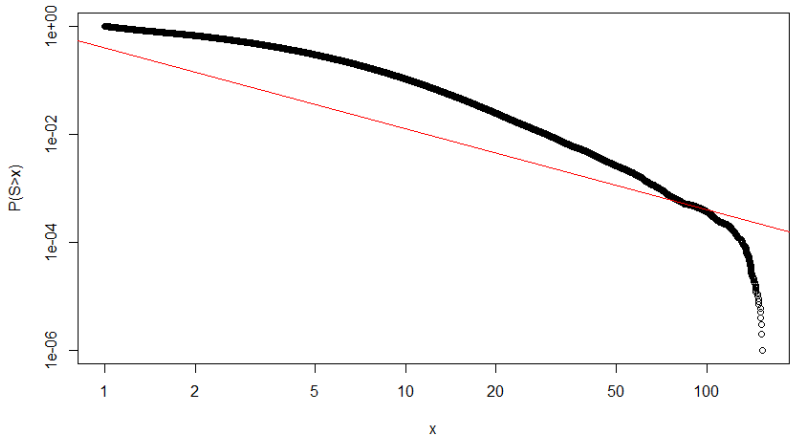

Figure 6: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.7$. The red line is a straight line with slope -1.5.**

Infinite variance For the infinite variance case, the accuracy problems really start to show. See Figures 7 to 11. Nevertheless, the answer to the first question is still 'yes' (see Figure 8) .

What is so interesting is that the behaviour is very erratic, in the sense that there is no dependence on the value of $\mathbb{E}[B]/\mathbb{E}[A]$, or I cannot see it at least. A reason for this might be because of the infinite variance, which means that 1 run for each value is not enough. Even though this might be the case, the system should behave along the lines of the theoretical findings, which it clearly does not in all cases. Therefore I am leaning towards 'no' as an answer to the second question.

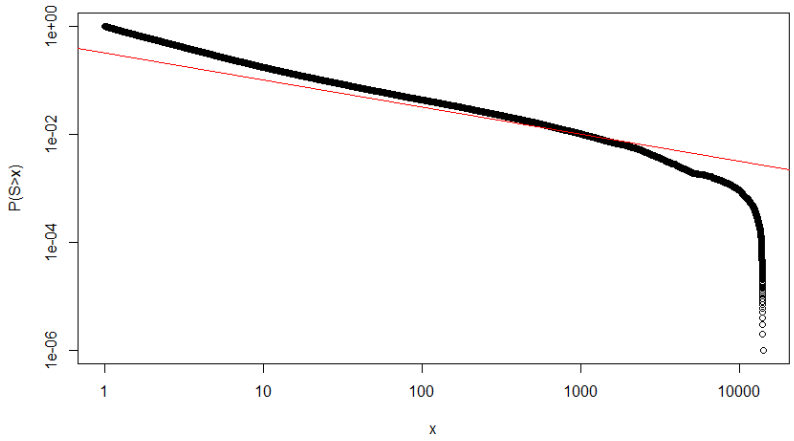

Figure 7: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.4$. The red line is a straight line with slope -0.5.**

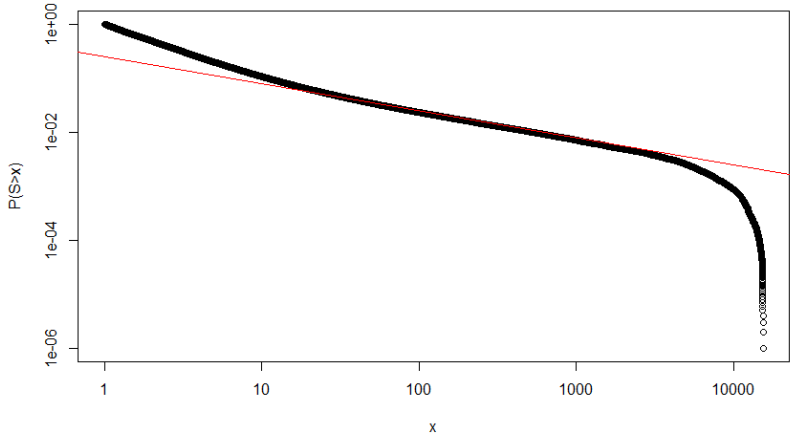

Figure 8: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.25$. The red line is a straight line with slope -0.5.**

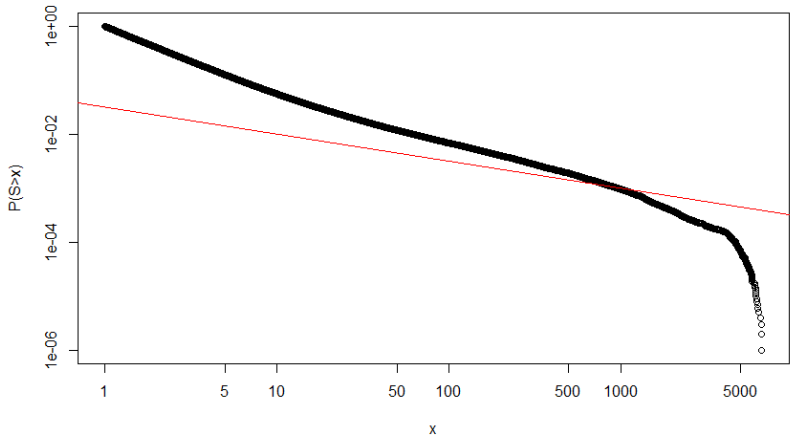

Figure 9: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.1$. The red line is a straight line with slope -0.5.**

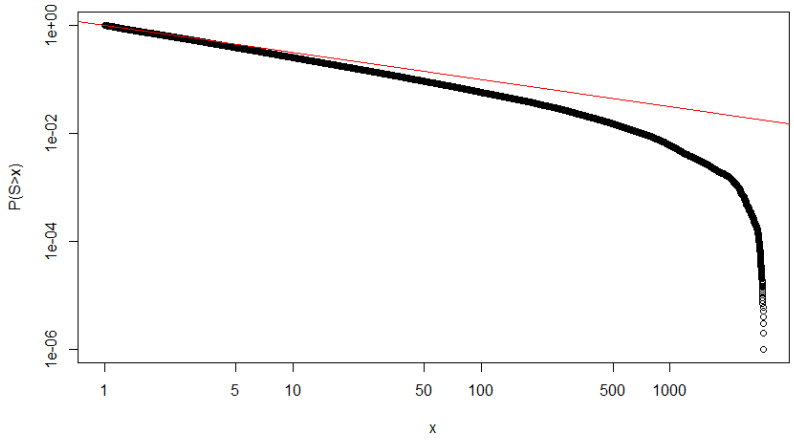

Figure 10: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.55$. The red line is a straight line with slope -0.5.**

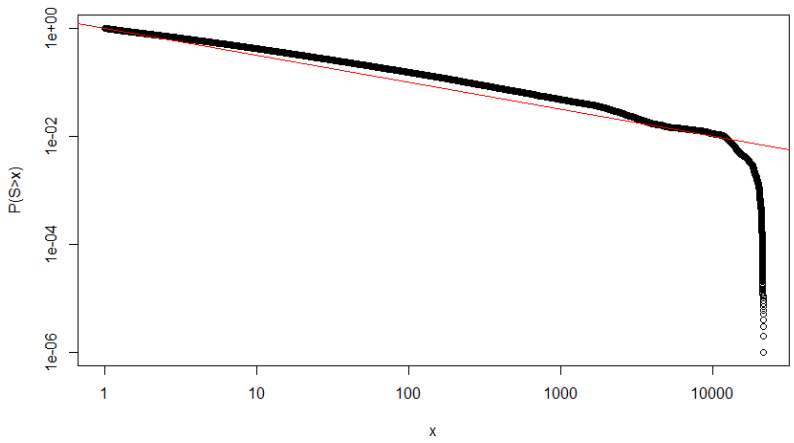

Figure 11: **Sojourn time samples, using the FCFS policy, on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.7$. The red line is a straight line with slope -0.5.**

The second question remains unanswered, although I lean stronger towards 'no' in this case, judging from Figure 7.

3.3.2 Policy in [3]

As described in Section 2.4.2, it is exactly clear how the sojourn time should behave asymptotically, given the needed variables. The main goal will be the same as before; can these findings be validated?
I apply the same notation from Section 3.3.1. I also, again, leave only the average inter-arrival time to be variable, to simplify analysis. This quantity is encapsulated again in the quotient $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$.

Finite variance See Figures 12 to 16. There are two things that are interesting to me:

- The 'best' result (closest to the line) occurs when $\frac{\mathbb{E}[B]}{\mathbb{E}[A]} = 0.4$. This is the exact same for the FCFS policy. Is this just a coincidence, or is there a specific reason? It could be interesting to investigate this in more detail in other researches.
- The effect of changing $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$ from 0.4 is quite different from the FCFS policy. The sojourn time asymptotically is larger. While this is expected when increasing $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$ (which is the case for Figures 15 and 16), it is not expected when decreasing it (see Figures 13 and 14). Clearly, the more times a customer is pre-empted (which is more often the case when $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$ is higher), the larger the sojourn time will be. Then lowering $\frac{\mathbb{E}[B]}{\mathbb{E}[A]}$ must decrease the sojourn time, but this is not always the case. It does not make much sense to me.

To answer both questions again, I am leaning towards 'yes' to both, given what I have seen with the FCFS policy. The first question seems an obvious 'yes' to me, while for the second

question there are more cases where the curve follows the red line than with the FCFS policy. It is still unknown though, as not all possible parameter combinations have been tried.

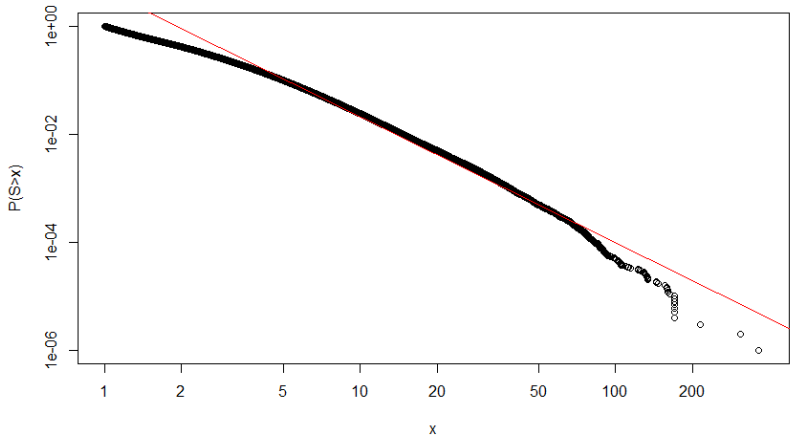

Figure 12: **Sojourn time samples, using the policy in [3] on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.4$. The red line is a straight line with slope -2.33.**

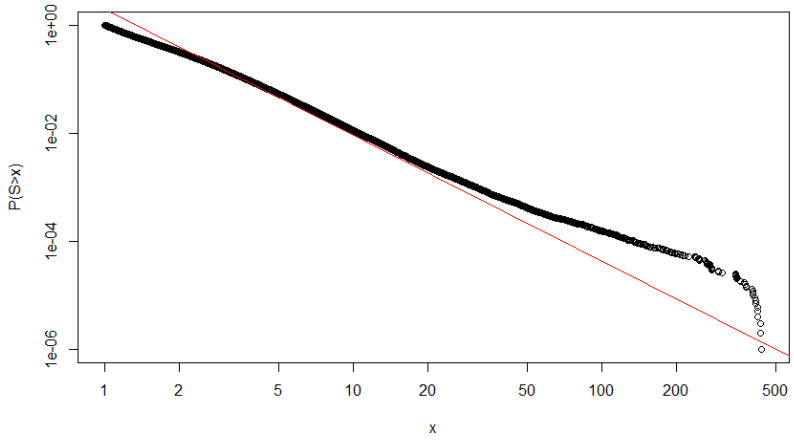

Figure 13: **Sojourn time samples, using the policy in [3] on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.25$. The red line is a straight line with slope -2.33.**

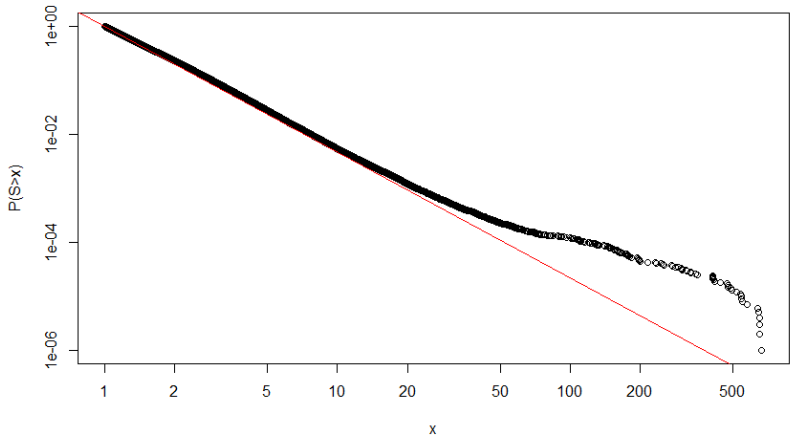

Figure 14: **Sojourn time samples, using the policy in [3] on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.1$. The red line is a straight line with slope -2.33.**

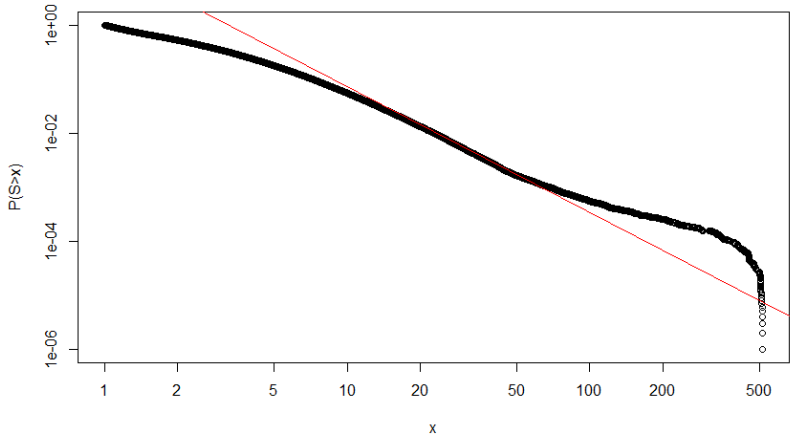

Figure 15: **Sojourn time samples, using the policy in [3] on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.55$. The red line is a straight line with slope -2.33.**

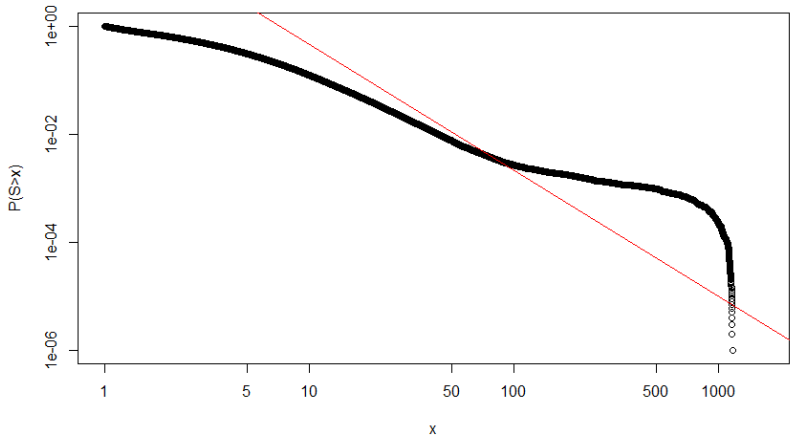

Figure 16: **Sojourn time samples, using the policy in [3], on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.7$. The red line is a straight line with slope -2.33.**

Infinite variance

In the case of the variance being infinite, accuracy problems might show again. Yet there are quite a lot of values for $\mathbb{E}[B]/\mathbb{E}[A]$ where the red line is followed nicely (see Figures 17 to 21). Note that I am looking for the required behaviour for larger values of x. The only case of really odd behaviour is when $\mathbb{E}[B]/\mathbb{E}[A] = 0.1$. Especially since the sojourn time distribution seems to have a much heavier tail, which is unexpected.

To answer both questions again, the first one is a definite 'yes'. For the second question, I am leaning towards 'yes' again, when comparing the inaccuracies here to the inaccuracies of FCFS. With this policy there is only one case of odd behaviour, which could just as well be a mistake on my end. Again, it is still unknown though.

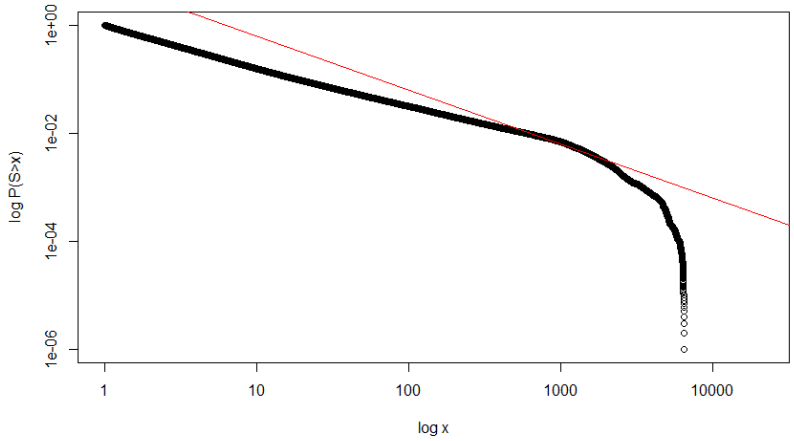

Figure 17: **Sojourn time samples, using the policy in [3], on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.4$. The red line is a straight line with slope -1.**

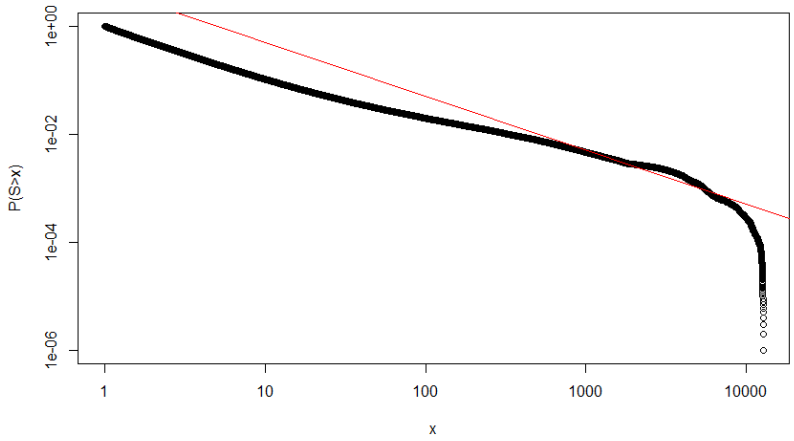

Figure 18: **Sojourn time samples, using the policy in [3], on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.25$. The red line is a straight line with slope -1.**

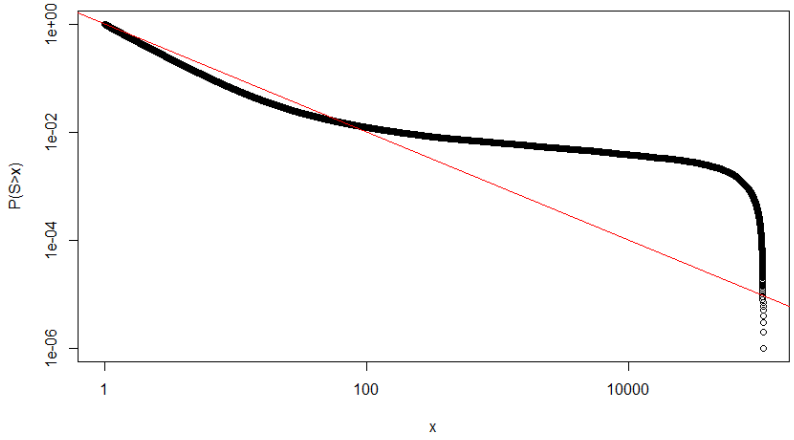

Figure 19: **Sojourn time samples, using the policy in [3], on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.1$. The red line is a straight line with slope -1.**

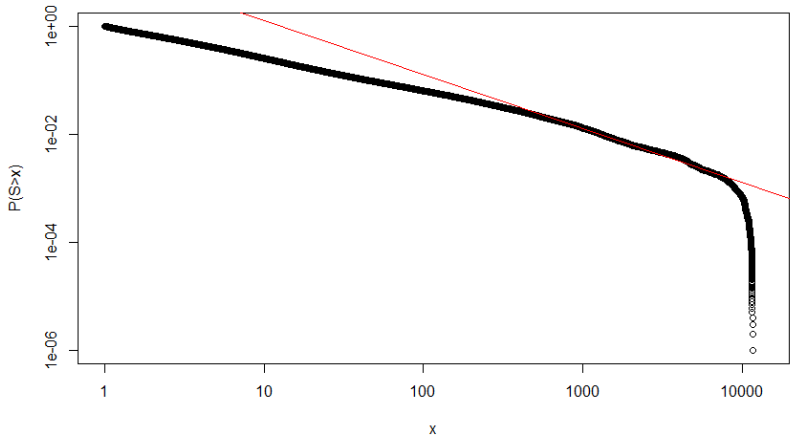

Figure 20: **Sojourn time samples, using the policy in [3], on a log log scale, with $\alpha = 1.5$ and$\mathbb{E}[B]/\mathbb{E}[A] = 0.55$. The red line is a straight line with slope -1.**

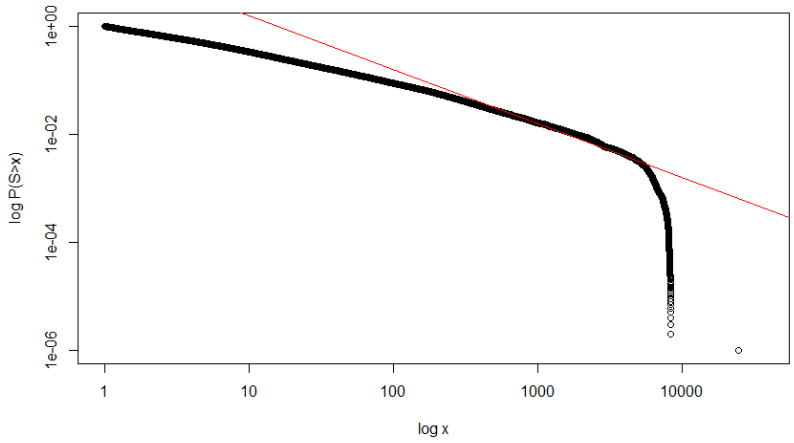

Figure 21: **Sojourn time samples, using the policy in [3], on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.7$. The red line is a straight line with slope -1.**

4 Possible extension

4.1 Introduction

In Section 2.4.3, I pointed out an interest in the reason behind choosing just 2 priority classes within the service policy, described in Section 2.4.2. In this section, I elaborate on a possible extension, using not 2, but more priority classes. The model behind the policy is described in Section 4.2, and simulation results using this policy are presented in Section 4.3.

4.2 Model description

The first few rules of the policy will be the exact same as described in Section 2.4.2; each customer starts in the highest priority class, M_i will be calculated and used in the exact same way as before. Each customer in a lower priority class can be pre-empted by a customer in the higher priority class as well. The one main extension is that after a pre-emption, the customer will fall in a lower priority class, if it exists. For example: Let customer X_i reside in the second priority class after some time. Whenever X_i is pre-empted by an other, arbitrary customer X_j, he shall be put in the third priority class, and so on.

This extra rule was for me the easiest to comprehend and implement into the already existing simulation, but it is not the only rule that could be used. One other way could be to use the busy period again and calculate M_i the same way as before, but with a minor adaptation; for instance, take the minimum of M_i and the time it takes for the next pre-emption as the time it is serviced. The problem lies in the implementation of the busy period: I could not find an understandable way to implement it. Because of this, I took the easier, but maybe less interesting option.

The question now is: Given that the service time distribution is regularly varying with index $-\alpha$, how does the tail of the sojourn time distribution behave? Intuitively, I can give a lower bound for it. Think of the policy described in Section 2.4.2. The asymptotic behaviour of the sojourn time distribution is well defined. Now the only difference is that there are three (or more) priority classes, which can only make the sojourn time grow (there are potentially more customers that can pre-empt a given customer), and as a larger sojourn for one conustomer cannot decrease the sojourn time for others, the sojourn time distribution from [3] can therefore be used as a lower bound. However, an upper bound is not so intuitive, as this depends on the exact parameter choices, which are unknown in this case. It is not really interesting either, since the lower bound tells that adding more classes can only make the sojourn time larger, which is something one would like to avoid. Still, it is interesting to investigate the behaviour.

4.3 Results

The policy described in the earlier section does not have a theoretical 'limit distribution function' for the sojourn time, in the sense that it is unknown to what the sojourn time converges, if at all. Therefore I cannot verify anything either, but I can investigate the behaviour.

For the sake of simplicity I use only three classes, even though it is allowed to have more. I will also use the same method and notation as before (see Section 3) to analyse the result.

Finite variance See Figures 22 to 26.
The one main interesting result is that in all cases (light or heavy traffic) the sojourn time distribution follows the same red line, as if I were to use two classes instead of three. It seems that it does not have an effect on the limit distribution at all, at least when using the same simulation set-up.

In contrast to using two classes, this method seems to be a better approximation (in the sense that the distribution seems to follow the red line better), and it approximates well in

both light and heavy traffic, whereas this was not true earlier (at least, not for all values of $\mathbb{E}[B]/\mathbb{E}[A]$). I am unsure if this is an actual advantage of using more than 2 classes or not though. I believe so, since one can then be sure what kind of distribution he/she is dealing with, in all cases.

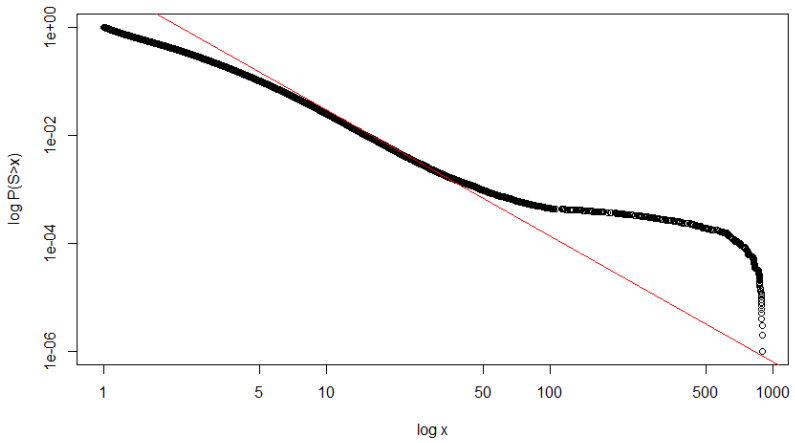

Figure 22: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A] = 0.4$. The red line is a straight line with slope -2.33.**

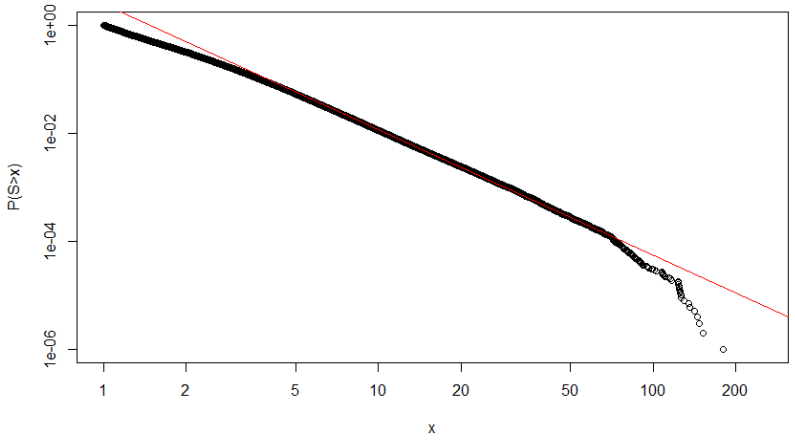

Figure 23: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.25$. The red line is a straight line with slope -2.33.**

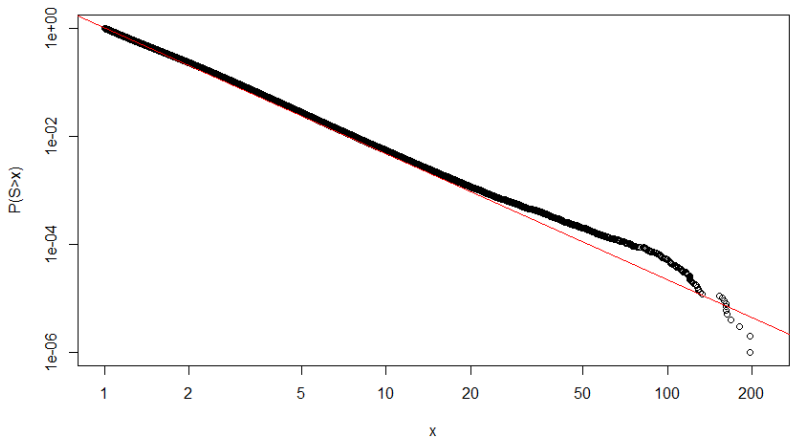

Figure 24: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.1$. The red line is a straight line with slope -2.33.**

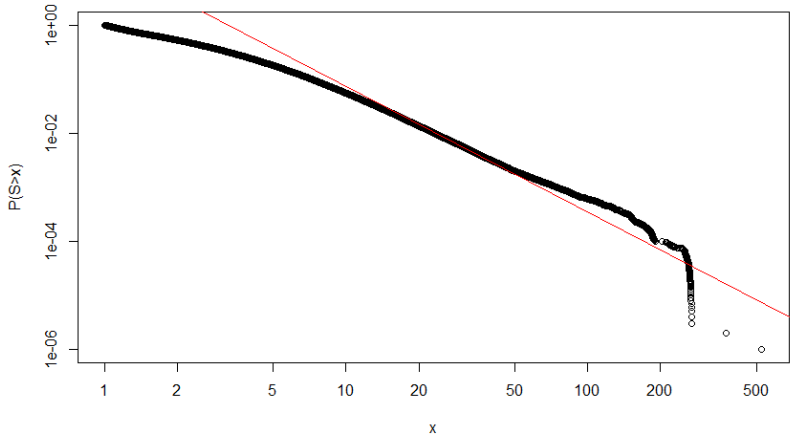

Figure 25: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.55$. The red line is a straight line with slope -2.33.**

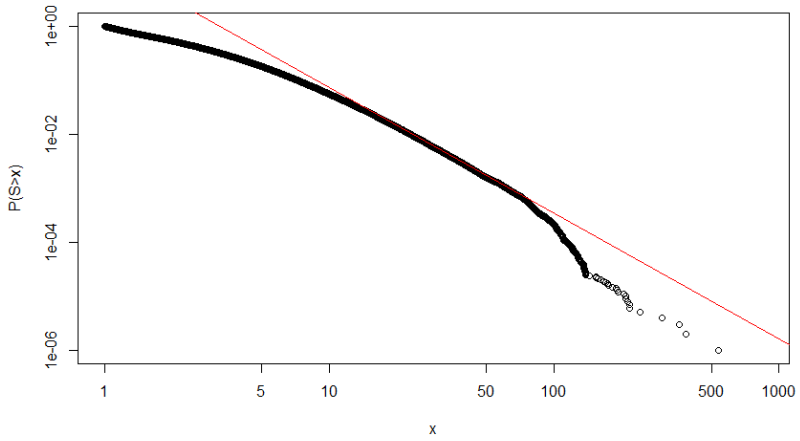

Figure 26: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 2.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.7$. The red line is a straight line with slope -2.33.**

Infinite variance See Figures 27 to 31.
The same thing as for the finite variance case can be said here. The sojourn time distribution seems to approximate the same line using the policy with two classes. And again, the approximation seems to be better in the same sense. But if this is even true for the case of infinite variance, I believe that there is a clear advantage to using more than two classes, as predictability might be useful in some cases.

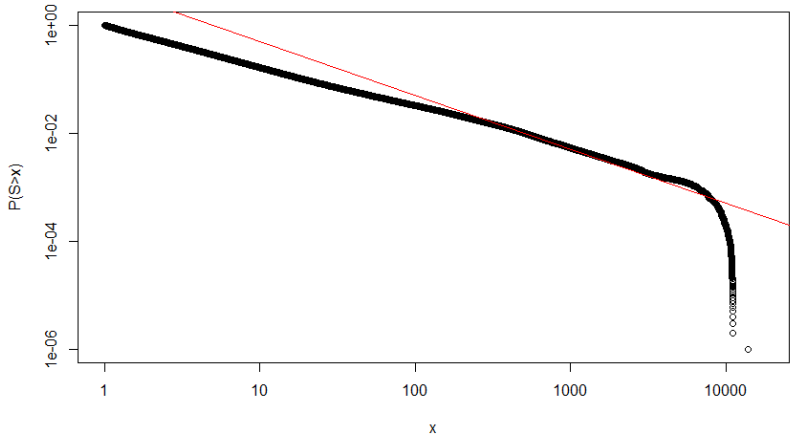

Figure 27: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.4$. The red line is a straight line with slope -1.**

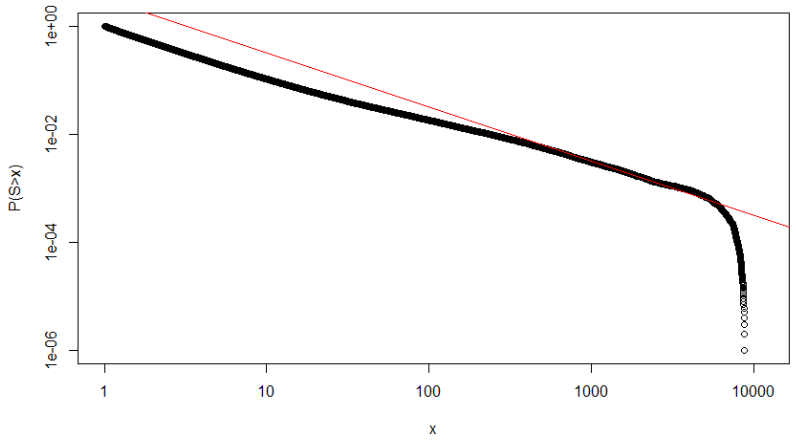

Figure 28: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.25$. The red line is a straight line with slope -1.**

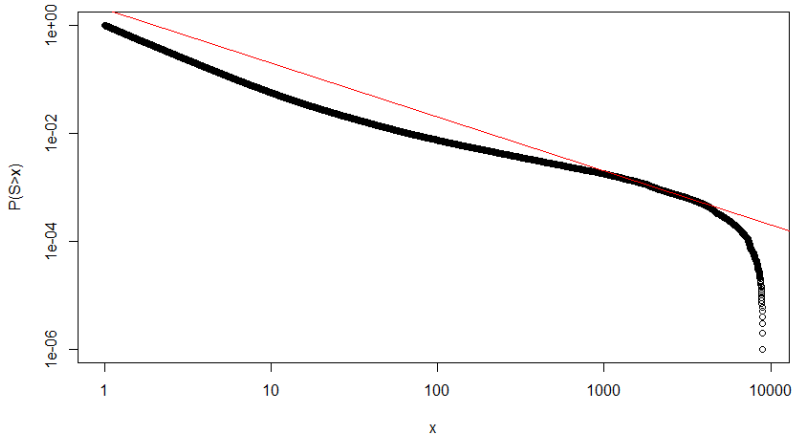

Figure 29: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.1$. The red line is a straight line with slope -1.**

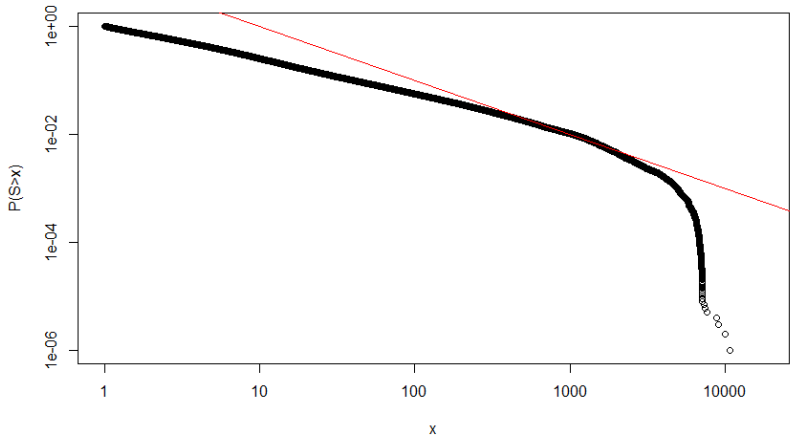

Figure 30: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.55$. The red line is a straight line with slope -1.**

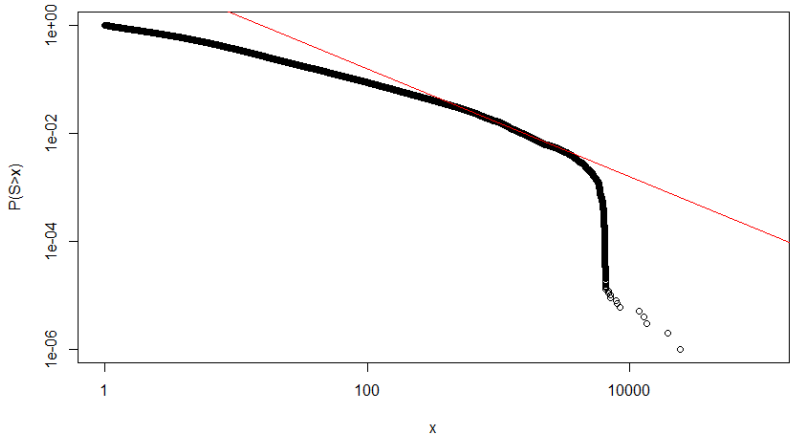

Figure 31: **Sojourn time samples, using the extended policy on a log log scale, with $\alpha = 1.5$ and $\mathbb{E}[B]/\mathbb{E}[A]) = 0.7$. The red line is a straight line with slope -1.**

5 Conclusion and Discussion

In Section 3, I said that I wanted to verify that the sojourn time distribution behaves such as described in the theoretical conclusions made in [2] and [3]. Using the method described in [7], I can say that part of it definitely is correct, in the sense that I can verify the theoretical conclusions for *some* inter-arrival time distribution. Unfortunately that is not all what was said. The method deemed not to be useful to verify that the theory holds for *all* inter-arrival time distributions. That question remains unanswered.

The most problems I had with simulating these systems, is the inaccuracy of the simulation, because of the underlying problem of heavy-tailed distributions. As described in [7], one would need a lot more customers than I used to gain some sort of accuracy. Surprisingly enough this inaccuracy had the largest effect when using the FCFS policy, even though there is more evidence that the findings are true than the other systems.

With that note, I do believe that the theoretical findings are indeed true. What I do not know, and what I also cannot verify with the simulation at hand, is *when* they start becoming true. I know the asymptotic distribution, but what is the actual distribution? For practical purposes, the behaviour on the 'earlier' could be more interesting, since that occurs much more often. For further research it would be interesting to investigate what happens in the sojourn time distribution for 'low' x.

6 Appendices

Appendix A: Proof of theorem 1

Knowing that $Y(z)$ is a generating function, we have:

$$Y(z) = \sum_{0}^{\infty} z^n y(n),$$

with $y(n)$ the distribution of Y. Using Taylor's theorem, we also have:

$$Y(z) \sim \sum_{n=0}^{\infty} z^n Y^{(n)}(0)\frac{1}{n!} = c_Y \sum_{0}^{\infty} z^n (-1)^n R_Y^{-n}\beta(\beta-1)...(\beta-n+1)\frac{1}{n!}.$$

It follows that

$$y(n) \sim c_Y (-1)^n R_Y^{-n}\beta(\beta-1)...(\beta-n+1)\frac{1}{n!}.$$

For $n \to \infty$, it is known that $(-1)^n\beta(\beta-1)...(\beta-n+1) \sim \frac{\Gamma(n-\beta)}{\Gamma(-\beta)}$. What is left to prove is as follows:

$$\frac{\Gamma(n-\beta)}{n!} \sim n^{-\beta-1},$$

which follows from the general result for the Gamma function:

$$\frac{\Gamma(n+a)}{\Gamma(n)} \sim n^a,$$

with $a = -\beta - 1$ substituted.

Appendix B: Java Simulation Code

```
/*
* The main class, where all the magic happens.
* First the customers are generated.
* Afterwards the whole queueing mechanism starts.
```

```
*/

import java.io.BufferedWriter;

import java.io.FileNotFoundException;

import java.io.FileWriter;
import java.io.IOException;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Objects;
import java.util.Random;

/**
*
* @author Erik Boom, 0841779
*/
public class Queue {

ArrayList<Client> Clients = new ArrayList<>(); // List of all customers waiting
ArrayList<Client> ServedClients = new ArrayList<>(); // List of all served cust
ArrayList<Client> TempClients = new ArrayList<>();
Double beta = 1 / 4.0; // Threshold of when to put a customer out of service

double time;

/**
* Generates customers for the queue
*/
private void generate() {
Random r = new Random(); // instantion of a random number generator

Double tempTime = 0.0; // Used to get good inter-arrival times(As these are exp
Double ArrivalTime = 0.0; //Used to insert the arrival times for one customer
Double ExecutionTime; //Used to insert the execution times for one customer
Double Lambda = 0.231; //Mean for the exponential distribution (Probably needs
Double alpha = 1.5; // Shape parameter for the Pareto distribution (Probably ne
Double xM = 1.0;// Scale parameter for the Pareto distribution
Double sum = 0.0;// Sum to determine busy period
int BusyPeriod = 0; // This value is to see which customer belongs to which bus
for (int i = 0; i < 1000000; i++) { //I now generate 1000000 customers, but thi
ExecutionTime = getRandomPareto(r, alpha, xM);


if (i == 0) { // Specific case when there is no predecessor
```

```
Client c = new Client(tempTime, ExecutionTime, 1, BusyPeriod);
Clients.add(c);
TempClients.add(c); // Needed to check for predecessors later
// it is assumed that the first customer is there at time 0, but this could be
} else {// Give the busy period the new element

sum = sum + (Clients.get(i - 1).ExecutionTime - ArrivalTime);

if (sum < 0.0) { // End the busyPeriod and initialize a new one
sum = 0.0;
BusyPeriod++;
Client c = new Client(tempTime, ExecutionTime, 1, BusyPeriod);
Clients.add(c);
TempClients.add(c); // Needed for later
} else { //Keep going with the busy period
Client c = new Client(tempTime, ExecutionTime, 1, BusyPeriod);
Clients.add(c);
TempClients.add(c);
}
}

ArrivalTime = getRandomExpo(r, Lambda);
tempTime += ArrivalTime;

}

}

private void exectuteFIFOQueue(ArrayList<Client> Clients) {
time = 0.0; // The time will keep going up until all customers have been served
while (!Clients.isEmpty()) { // Are there customers left?
if (time >= Clients.get(0).ArrivalTime) { // Are there customers in the queue?
Client ServedClient = Clients.get(0); // Handle the customer with the lowest ar
FinishClient(ServedClient);
System.out.println(ServedClients.size());

} else {
time = Clients.get(0).ArrivalTime;
}
}
}

/**
* Executes the queue system
*/
private void executeQueue(ArrayList<Client> Clients) {
time = 0.0; // The time will keep going up until all customers have been served
```

```
while (!Clients.isEmpty()) { // Are there customers left?
if (time >= Clients.get(0).getArrivalTime()) { // Are there customers in the qu

Client ServedClient = PreemptedClient(); // Which customer needs to be handled?
if (ServedClient.Priority == -1) {// Not served for 1st time, so can be preempt

Double timeInQueue = PreemptTime(ServedClient) - time; // Calculate the time th

if (ServedClient.ExecutionTime > timeInQueue) { // Not long enough in service t
time = time + timeInQueue; // Only have this element in queue for that time
ServeClient(ServedClient, timeInQueue); // Sort the list back to ascending orde
} else { // Enough time in service to finish
FinishClient(ServedClient);
}
//Whenever the 3-class model is needed, uncomment the follwing  lines (and chan
} else if (ServedClient.Priority == 0) {
Double timeInQueue = PreemptTime(ServedClient) - time;
if (ServedClient.ExecutionTime < timeInQueue) {
FinishClient(ServedClient);
} else {
ServeClient(ServedClient, timeInQueue);
ServedClient.setPriority(-1);
}
} else { // Know it is serviced for 1st time.
Double MI = CalculateMI(ServedClient);

if (ServedClient.ExecutionTime < MI) { // Treat the customer always as high pri
FinishClient(ServedClient);
} else {
// Treat the customer as high priority for only some time
Double timeInService = Math.pow(ServedClient.ExecutionTime, 1 - beta);
ServeClient(ServedClient,timeInService);
ServedClient.setPriority(0);
}
}

} else {
time = Clients.get(0).getArrivalTime();// Wait until a customer has arrived
}
}
}
/**
* Updates waiting time, whenever the service is finished
* @param c Client to finish
*/
public void FinishClient(Client c){
```

```
time = time + c.ExecutionTime;
c.setWaitingTime(time - c.InitialArrival); // Update sojourn time
Clients.remove(c); // Removing the customers make the loop stop.
ServedClients.add(c); //Add again, for printing purposes

}


/**
* Updates waiting time, whenever the service is served for a specific amount of
* @param c Client to serve
* @param timeInService Time to serve a customer
*/
public void ServeClient(Client c, double timeInService){
time = time + timeInService;
c.setExecutionTime(c.ExecutionTime - timeInService);// Set the new execution ti
c.setWaitingTime(time - c.InitialArrival);  // Update the sojourn time
c.setArrivalTime(time); // Set the new arrival in the queue
Collections.sort(Clients); // Sort again, such that the arrivals are in ascendi
}



/**
* Returns the time that an item has high priority for at the maximum
*
* @param c
* @return the time that an item has high priority for at the maximum
*/
public double CalculateMI(Client c) {
double MaxExecution = 0.0;
//Get the element in the customers list, that matches the customer c
for (int i = 0; i < Clients.size(); i++) {
if (Objects.equals(c.InitialArrival, TempClients.get(i).InitialArrival)) { // Y
break;
} else if (c.BusyPeriod == TempClients.get(i).BusyPeriod) { // Else look at the
//Get the maximum execution time of customers in the same busy period
if (TempClients.get(i).ExecutionTime > MaxExecution) {
MaxExecution = TempClients.get(i).ExecutionTime;

}
}
}

if (MaxExecution == 0.0) {
return Double.MAX_VALUE; // If only one customer is in a busy period, the custo
} else {
```

```
return Math.pow(MaxExecution, 1 - beta);
}
}

/**
* Returns the time that an item has service under low priority
*
* @param c
* @return the time that customer c has in service
*/
public double PreemptTime(Client c) {
double PreemptTime;
for (int k = 0; k < Clients.size(); k++) {
if (Clients.get(k).Priority > c.Priority) { // This item has a higher priority,
// I am only looking for the first one where this holds, as I know that will ha
PreemptTime = Clients.get(k).ArrivalTime;

if (time > PreemptTime) { // If this item already is in the queue
return time;
} else { // The item arrives at the queue at time PreemptTime and has higher pr
return PreemptTime;
}
}

}
// If no customer has a higher priority, there is no customer that can preempt
return Double.MAX_VALUE;
}

/**
* Returns the earliest customer(lowest ArrivalTime) with highest priority
*
* @return
*/
public Client PreemptedClient() {
for (int k = 0; k < Clients.size(); k++) {

if (Clients.get(k).ArrivalTime <= time) { // Client is in the queue

if (Clients.get(k).Priority == 1) {
return Clients.get(k); // Return the first customer you see with priority 1 in
}
} else { // Once there are no customers in the queue
break;
}

}
```

```
//Need to go through all customers with prio 1 first, then with prio 0.
for (int k = 0; k < Clients.size(); k++) {

if (Clients.get(k).ArrivalTime <= time) { // Client is in the queue

if (Clients.get(k).Priority == 0) {
return Clients.get(k); // Return the first customer you see with priority 0 in
}
} else { // Once there are no customers in the queue
break;
}

}
return Clients.get(0); // If there are no customers with priority 1 or 0
currently in the queue, return the first customer still in the list.
}

/**
* Returns a pseudo-random number from an exponential distribution
*
* @param r The random number generator
* @param Lambda The mean of the distribution
* @return A random number
*
*/
public static double getRandomExpo(Random r, double Lambda) {
return -(Math.log(r.nextDouble()) / Lambda);
}

/**
* Returns a pseudo-random number from a Pareto distribution
*
* @param r The random number generator
* @param alpha The shape parameter of the distribution.
* @param xM The scale parameter of the distribution.
* @return A random number
*/
public static double getRandomPareto(Random r, double alpha, double xM) {
double v = r.nextDouble();
while (v == 0) {
v = r.nextDouble();
}

return xM / Math.pow(v, 1.0 / alpha);
}

public void save(ArrayList<Client> customers) throws FileNotFoundException, IO
```

```
// Create file
FileWriter fstream = new FileWriter("outputBoxmaMore1.5_0.7.txt", true);
//writes data and blank line to file
try ( //create BufferedWriter to write data to the file
BufferedWriter out = new BufferedWriter(fstream)) {
//writes data and blank line to file

for (int i = 0; i < customers.size(); i++) {
out.write(customers.get(i).InitialArrival.toString() + ",");
out.write(" " + customers.get(i).InitialExecution.toString() + ",");
out.write(" " + customers.get(i).WaitingTime.toString());
out.newLine();
}
//Close the output stream
out.close();
}
}

/**
* @param args the command line arguments
*/
public static void main(String[] args) throws IOException {
Queue queue = new Queue();
boolean Fifo = false; // Change to apply the different system
queue.generate(); //generate customers
if (!Fifo) {
queue.executeQueue(queue.Clients); // apply the main system
} else {
queue.exectuteFIFOQueue(queue.Clients);// apply the fifo system
}
queue.save(queue.ServedClients);

}

}
/*
* This class holds all properties a customer in the queue can have.
* It also provides functions to get and set these.
*
*/

/**
*
* @author Erik Boom, 0841779
*/
public class Client implements Comparable<Client> {
public Double ArrivalTime; //The timing of arrival for this customer
```

```
public Double ExecutionTime; // The time it takes to serve this customer in its
public int Priority; // A priority level, indicating which task to do to first
public Double WaitingTime; // The to be computed time it takes to serve this cu
public Double InitialArrival; // Initial time of arrival, for plot purposes
public Double InitialExecution;// Initial execution time, for plot purposes
public int BusyPeriod; // This value is to see which customer belongs to which

public int getBusyPeriod() {
return BusyPeriod;
}

public void setBusyPeriod(int BusyPeriod) {
this.BusyPeriod = BusyPeriod;
}
/**
* Initializes the customer
* @param ArrivalTime The initial arrival time
* @param Executiontime The initial execution time
* @param Priority The initial priority
* @param BusyPeriod
*/
public Client(Double ArrivalTime, Double Executiontime, int Priority, int Busyl
this.ArrivalTime = ArrivalTime;
this.ExecutionTime = Executiontime;
this.Priority = Priority;
this.InitialArrival = ArrivalTime;
this.InitialExecution = Executiontime;
this.BusyPeriod = BusyPeriod;
}

public Double getInitialArrival() {
return InitialArrival;
}

public void setInitialArrival(Double InitialArrival) {
this.InitialArrival = InitialArrival;
}

public Double getInitialExecution() {
return InitialExecution;
}

public void setInitialExecution(Double InitialExecution) {
this.InitialExecution = InitialExecution;
}

/**
```

```
* Gives the ArrivalTime
* @return Arrivaltime
*/
public Double getArrivalTime() {
return this.ArrivalTime;
}
/**
* Sets the new arrival time
*
* @param ArrivalTime The new value to set it to
*/
public void setArrivalTime(Double ArrivalTime) {
this.ArrivalTime = ArrivalTime;
}
/**
* Gives the Execution time
* @return ExecutionTime
*/
public Double getExecutionTime() {
return this.ExecutionTime;
}

public void setExecutionTime(Double ExecutionTime) {
this.ExecutionTime = ExecutionTime;
}
/**
* Gives the priority
* @return Priority
*/
public int getPriority() {
return this.Priority;
}

public void setPriority(int Priority) {
this.Priority = Priority;
}
/**
* Gives the waiting time
* @return WaitingTime
*/
public Double getWaitingTime() {
return this.WaitingTime;
}

public void setWaitingTime(Double WaitingTime) {
this.WaitingTime = WaitingTime;
}
```

```
@Override
public int compareTo(Client c){
return Double.compare(this.ArrivalTime, c.ArrivalTime);
}
}
```

References

[1] A.-L. Barabási, "The origin of bursts and heavy tails in human dynamics," *Nature*, vol. 435, no. 7039, pp. 207–11, 2005.

[2] J. Walraevens, T. Demoor, T. Maertens, and H. Bruneel, "Stochastic queueing-theory approach to human dynamics," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 85, no. 2, pp. 1–7, 2012.

[3] O. Boxma and D. Denisov, "Sojourn time tails in the single server queue with heavy-tailed service times," *Queueing Systems*, pp. 1–16, 2010.

[4] J. Walraevens, B. Steyaert, and H. Bruneel, "Analysis of a discrete-time preemptive resume priority buffer," *European Journal of Operational Research*, vol. 186, pp. 182–201, apr 2008.

[5] P. Flajolet and R. Sedgewick, "Analytic Combinatorics," 2002.

[6] S. C. Borst, O. J. Boxma, R. Núñez-Queija, and A. P. Zwart, "The impact of the service discipline on delay asymptotics," *Performance Evaluation*, vol. 54, no. 2, pp. 175–206, 2003.

[7] M. Crovella and L. Lipsky, "Long-Lasting transient conditions in simulations with heavy-Tailed workloads," *Winter Simulation Conference*, pp. 1005–1012, 1997.