Riga Technical University

Faculty of Electronics and Telecommunications

Department of Transport Electronics and Telematics

Romans Jerjomins

PhD Program "Computer Control, Information and Electronic Systems of Transport"

Research of Wireless Local Area Network in Non-stationary Mode

Doctoral diploma thesis

Supervisor

Dr. hab. ing. sc., professor

Ernests Petersons

Riga 2012

Abstract

Research of various network traffic types proves that it is self-similar. Therefore, recently many works and fundamental monograph is devoted to its research. Self-similar traffic influence on communication devices, such as network switchboards, concentrators and network servers, lead to sharp decrease in their functioning quality, comparing to operating mode in conditions of traditional streams of packets and queries. It reveals in throughput reduction of network nodes, insufficiency of buffer memory and results in increased of denial of service. Usually all researches of self-similar traffic influence were made in stationary operating mode of switched devices.

Author of the given work observed indicators of self-similar traffic at inclusion of communication devices in work, when load swing arise and certain time – relaxation time – was required for device to enter into a normal stationary operating mode. Therefore in the given work the attention is given to relaxation time.

As a result of research data about transient process durations, utilization of buffer memory depending on server load have been obtained.

Contents

Abstract **i**

1 Introduction **1**
1.1 Wireless networks . . . 1
1.1.1 IEEE 802.11 a/b/g/n networks . . . 1
1.1.2 Mobile clients . . . 2
1.2 Self-similarity and Hurst Parameter . . . 3
1.2.1 Self-similar processes . . . 5
1.2.2 Properties of self-similar processes . . . 6
1.2.3 Self-similarity of web traffic . . . 8
1.2.4 Hurst parameter . . . 9
1.3 Problem statement . . . 12

2 Wireless network traffic **13**
2.1 Experiment . . . 13
2.1.1 Passive packet capture . . . 14
2.1.2 Device statistics monitoring . . . 15
2.1.3 SNMP . . . 15
2.1.4 Data acuision tool . . . 18
2.2 Experimental data . . . 19
2.3 Data processing methods and tools . . . 22
2.3.1 Methods of Hurst parameter evaluation . . . 22
2.3.2 Analysis. Algorithms and tools . . . 24
2.4 Traffic analysis results . . . 27
2.5 Analysis summary . . . 31

3 Transient process research methods **32**
3.1 Analytical method . . . 32
3.2 Diffusion approximation . . . 34
3.2.1 Mathematical model . . . 34

3.2.2 Modeling results 37
3.3 Simulation 40

4 Transient process simulation **41**
4.1 GPSS 41
4.2 Transient process simulation 44
4.2.1 The model 44
4.2.2 Simulation tool 45
4.3 Simulation modeling results 52
4.3.1 Simulation vs. analytics 52
4.3.2 $M/M/1/K$ vs. $P/M/1/K$ 55
4.3.3 $P/M/1/K$ 58
4.4 Analysis of obtained results 63
4.4.1 Relaxation time 63
4.4.2 Self-similar traffic generation 67

5 Conclusions **70**
5.1 Research summary and contributions 70
5.2 Possible practical use 71

A Code listings **72**

B Wireless traffic **104**

C Diffusion approximation results **109**

D Simulation vs. analytics modeling **113**

E $M/M/1/K$ vs. $P/M/1/K$ modeling **115**

F $P/M/1/K$ modeling results **117**

Acronyms **137**

List of Symbols **139**

1 Introduction

1.1 Wireless networks

1.1.1 IEEE 802.11 a/b/g/n networks

The global trend of Internet connection methods is that more and more clients go wireless. And this is not only our PC's and laptops - most of the handhelds and mobile phones produced nowadays are wi-fi enabled. Even most e-book readers [2] have wi-fi chips not to mention the growing tablet market.

The major advantages of wi-fi over GPRS, 3G and 4G for a customer are:

- Connection speed is up to 600 Mbps using 802.11n
- Longer battery life - 3G/4G devices are usually very power hungry comparing to wi-fi
- Low price - a wireless router in retail could be found for as little as fifteen euros not to mention the volume price and manufacturing cost

The last one is an advantage for Internet service providers along with the fact that 802.11 standard devices can be seamlessly integrated into currently winning Ethernet infrastructure. The biggest disadvantage for now is the range and complex authorization comparing to GSM/3G. Users have to enter username and password and sometimes setup security manually. And this is a

burden if we are talking about public networks. Nevertheless there is IEEE initiative called 802.11u [9] which was brought to resolve this issue.

Combination of high throughput 802.11n with seamless authorization 802.11u, mesh networking 802.11s, fast BSS transition 802.11r and seamless handover 802.21 would make a very competitive technology at lower price.

1.1.2 Mobile clients

Due to rapid advances in wireless technology, the Internet becomes more mobile. Not only smart phones become more affordable and ubiquitous, also car manufacturers are looking into leveraging Internet connectivity in order to provide advanced applications on car maintenance - such as monitoring and diagnosis, on road assistance - such as providing route navigation, weather maps and automated toll payments, as well as on passenger entertainment, including various types of Internet applications. Most of today's network connected cars still rely on systems with low-bandwidth connectivity (e.g. GSM or satellite links), which don't correspond to needs of emerging new applications. It is expected that such situation will change quickly. Several car manufacturers are offering Internet connectivity for handful models of cars via 3G network and other manufacturers also consider offering Internet-enabled car applications or linking smart phone applications to cars. The current trend suggests that ten millions of cars will go on-line in next several years. This will emerge innovative car-based Internet applications and services, which can have major impact on both manufacturers and passenger experiences.

In the last few years we have witnessed increasing number of cars connected to the Internet. All indicators suggest that this trend will continue and drive-through vehicles will become soon first class citizens on the Internet.

Recent research shows [52] that it is possible to use 802.11n based network for drive-through Internet services at the speed of mobile clients up to 100 kilometers per hour.

The problems we face in usual wired network will become worse in drive-through network as all it's clients will be mobile and constantly switching from one access point to another. Specifically the problem of packet congestions when many clients try to get Internet access at the same time what causes bursts of traffic and packet buffering problems [51].

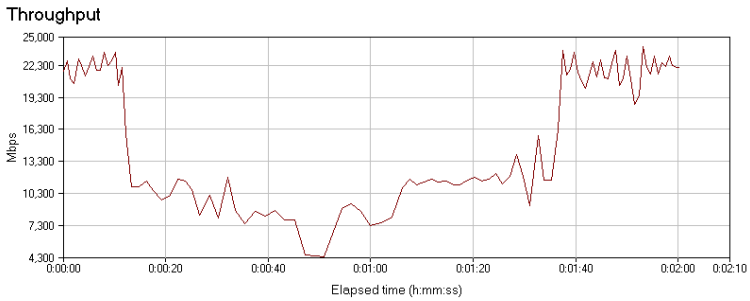


Figure 1.1: Transient processes in wireless networks

Another problem the author faced even with stationary wireless clients is transient processes when a new client connects to access point. Researching transient processes in wireless network became the primary interest of the author.

1.2 Self-similarity and Hurst Parameter

Self-Similarity and fractal are notions pioneered by Benoit B. Mandelbrot, who describe the phenomenon where a certain property of an object, such as a natural image, is preserved with respect to scaling in space or in time. The important property that almost all fractals have is the property of self-similarity (scale invariance). It seems the fractal can be divided into small parts so that each part appears simply a reduction of the whole. In other words, if we look at the fractal in a microscope, we shall see the same picture, as without a microscope. In fact, most things in existence are not circles, squares or lines. Instead, they are fractals, and the creation of these fractals is usually determined by the equations of chaos.

Self-similarity represents the concept uniting fractals, chaos and power laws. Self-similarity, or invariance, concerning changes of scale or the size represents distinctive feature of many laws of a nature and the uncountable phenomena in our world. Self-similarity is actually one of main symmetry, forming our universe and influencing on our attempts to understand it.

The phenomenon having property of self-similarity, looks equally or equally behaves by its consideration with a different degree of ”magnitude” or in different scale. Scaled size can be space (length, width) or time.

There is a well known fractal called Cantor set which resembles network traffic. It can be found practically in any book devoted to a theme of chaos, fractals and nonlinear dynamics. Although Cantor himself defined the set in a general, abstract way, the most common modern construction is the Cantor

ternary set, built by removing the middle thirds of a line segment. In essence, it is recursive process which can be defined as follows more precisely.

Let S_i represents Cantor set after i levels of recursion. Then

$$S_0 = [0, 1];$$
$$S_1 = [0, 1/3] \cup [2/3, 1];$$
$$S_2 = [0, 1/9] \cup [2/9, 1/3] \cup [2/3, 7/9] \cup [8/9, 1];$$

If we consider Cantor line as a time scale then on each subsequent step the scale of a time scale is tripled. Pay attention that on each step left (and right) part of set is an exact copy of full set on the previous step. Cantor set has two properties, which can be noticed in all phenomena of self-similarity.

1. It has the structure which is found out at any as much as small scale. If you increase a part of set repeatedly, we will continue to see a complex pattern of the points divided by intervals of the different size. And this process seems infinite. It differs radically from a usual continuous curve which at magnitude becomes smooth.

2. The structure repeats. The self-similar structure contains the reduced copies of the all levels of scaling. For example, on each step left (and right) Cantor sets is an exact copy of full set on the previous step.

These properties can not be carried out indefinitely for real objects. At a level of the magnitude the structure and similarity collapses. But many phenomena show property of self-similarity on a plenty of levels of the magnitude.

Though this example is too simple, we can understand some features of the self-similar traffic better when studying it. Probably, the most remarkable feature of the self-similar traffic in a context of network performance is stability of clusterization. In Poisson traffic clusterization is observed in short-term scale, but smooths out in long-term. We can design system from servers and queues with buffers in calculation on similar long-term smoothing. In result there will be enough buffers of the moderate size. The queue can be formed in short-term prospect, but for the long period of time buffers will be cleared. However if non-uniform behavior of the traffic itself is non-uniform - that is clusters are clustering - then queues of greater length can be formed, than it is possible to expect from a Poisson stream. In result there is, that a traditional analysis of queues in which basis lays the assumption about a Poisson stream, can not predict system performance in conditions of the self-similar traffic precisely. We will see, that there are acknowledgments of that.

1.2.1 Self-similar processes

Application of the mechanism of the theory of nonlinear dynamics (the theory of chaos) for research of the self-similar teletraffic also seems to be a most perspective direction and reasonable development of the ideas of fractal traffic. It is worth mentioning that the term of chaos means the word collocation deterministic chaos, however, in non formal conversation the word deterministic is frequently omitted. In this respect, the principle of determinacy can have a potential of playing a significant role not only for prediction of the network traffic and many similar processes apparent random at first sight. Unlike deterministic fractals, stochastic fractal objects (processes) are described as a rule by scale invariance (self-similarity) of statistical characteristics of the second order (the property of invariance of a correlation coefficient at scaling). These are such stochastic fractals that we will come across while studying the characteristics of the network traffic. In this connection in the literature the notions of fractal and self-similar teletraffic are often used as synonyms.

A self-similar time series has the property that when aggregated, i.e. leading to a shorter time series in which each point is the sum of multiple original points, the new series has the same autocorrelation function as the original.

Let's define a self-similar (fractal) process. Let $X = (X_t, t = 0, 1, 2 \dots N)$ be a covariance stationary (sometimes called wide-sense stationary) stochastic processes of discrete argument – time, with mean M_X, variance (or dispersion) $D_x = \sum_t X_t - M_X$ and autocorrelation function $r(k) = \sum_t (X_t - M_X) \cdot (X_{t+k} - M_X)$ which depends only on k. In particular we assume that X has an autocorrelation function of the form:

$$r(k) \sim k^{-\beta} L_1(k) \qquad , \text{as } k \to \infty \tag{1.1}$$

where $\beta \in (0, 1)$ and L_1 slowly varying at infinity. For simplicity we assume that L_1 is asymptotically constant. For each $m = 1, 2, 3 \dots$ (time scale) let $X^{(m)}$ denote the corresponding aggregated sequence with level of aggregation m, obtained by dividing the original series X into non-overlapping blocks of size m and averaging over each block, that is for each m $X^{(m)}$ is given by

$$X_k^{(m)} = \frac{1}{m}(X_{(k-1)m} + \dots + X_{km-1}) \qquad , k \geq 1 \tag{1.2}$$

Definition: a process X is called (exactly) second-order self-similar with self-similarity parameter $H = 1 - \beta/2$, $H \in (0.5, 1)$ if the corresponding aggregated processes $X_{(m)}$ have the same correlation structure as X, i.e.

$$r^{(m)}(k) = r(k) \qquad , \forall m \tag{1.3}$$

where $r^{(m)}$ denotes the autocorrelation function of $X_{(m)}$. In other words X is exactly self-similar if the aggregated processes $X_{(m)}$ are indistinguishable from X at least with respect to their second order properties.

Definition: a process X is called (asymptotically) second-order self-similar with self-similarity parameter $H = 1 - \beta/2$, $H \in (0.5, 1)$ if for all k large enough

$$r^{(m)}(k) \to r(k) \qquad , \text{as } m \to \infty \tag{1.4}$$

In other words X is asymptotically second-order self-similar if the corresponding aggregated processes $X_{(m)}$ are the same as X or become indistinguishable from X at least with respect to their autocorrelation functions.

1.2.2 Properties of self-similar processes

Long-range dependence and the Hurst effect

A stochastic process satisfying relation 1.1 is said to exhibit long-range dependence [75]. Thus, processes with long-range dependence are characterized by an autocorrelation function that decays hyperbolically as the lag increases. Moreover, it is easy to see that 1.1 implies $\sum_k r(k) = \infty$.

This non-summability of the correlations captures the intuition behind long range dependence, namely that while high-lag correlations are all individually small, their cumulative effect is of importance and gives rise to features which are drastically different from those of the more conventional, i.e., short-range dependent processes. The latter are characterized by an exponential decay of the autocorrelations, i.e. $r(k) \sim \rho^k$, as $k \to \infty$ and $\rho \in (0, 1)$ resulting in a summable autocorrelation function $0 < \sum_k r(k) < \infty$.

When working in the frequency domain, long-range dependence manifests itself in a spectral density that obeys a power-law behavior near the origin. In fact, equivalently to 1.1 (under weak regularity conditions on the slowly varying function L_1), there is long-range dependence in X if

$$f(\lambda) \sim \lambda^{-\gamma} L_2(\lambda) \qquad , \text{as } \lambda \to 0 \tag{1.5}$$

where $0 < \gamma < 1$, L_2 is slowly varying at 0 and $f(\lambda) = \sum_k r(k) e^{ik\lambda}$ denotes the spectral density function. Thus, from the point of view of spectral analy-

sis, long-range dependence implies that $f(0) = \sum_k r(k) = \infty$ that is, it requires a spectral density which tends to $+\infty$ as the frequency λ approaches 0 ("$1/f$-noise"). On the other hand, short-range dependence is characterized by a spectral density function $f(\lambda)$ which is positive and finite for $\lambda = 0$.

Heuristically, long-range dependence manifests itself in the presence of cycles of all frequencies and orders of magnitude, displays features suggestive of non-stationarity, and has been found to be relevant in economics, in hydrology and geology and in telecommunication [75].

Historically, the importance of self-similar processes lies in the fact that they provide an elegant explanation and interpretation of an empirical law that is commonly referred to as Hurst's law or the Hurst effect. For a given set of observations $(X_k, k = 1, 2, \ldots, n)$ with sample means $\overline{X}(n)$ and sample variance $S^2(n)$ the rescaled adjusted range or the R/S statistic is given by

$$R(n)/S(n) = \frac{1}{S(n)}\left[max(0, W_1, W_2, \ldots, W_n) - min(0, W_1, W_2, \ldots, W_n)\right] \qquad (1.6)$$

with $W_k = (X_1 + X_2 + \cdots + X_k) - k\overline{X}(n)$, $k = 1, 2, \ldots, n$.

Hurst found that many naturally occurring time series appear to be well represented by the relation

$$E[R(n)/S(n)] \sim cn^{0.5} \qquad \text{, as } m \to \infty \qquad (1.7)$$

with Hurst parameter H typically about 0.73 [75], and c a finite positive constant that does not dependent on n. On the other hand, if the observations X_k come from a short-range dependent model, then Mandelbrot and Van Ness in 1968 showed that

$$E[R(n)/S(n)] \sim dn^{0.5} \qquad \text{, as } m \to \infty \qquad (1.8)$$

where d a finite positive constant, independent of n. The discrepancy between 1.7 and 1.8 is generally referred to as the Hurst effect or Hurst phenomenon.

Slowly decaying variances

From a statistical point of view, the most salient feature of self-similar processes is that the variance of the arithmetic mean decreases more slowly than the reciprocal of the sample size; that is, it behaves like $n^{-\beta}$ for some $\beta \in$

$(0,1)$, instead of like n^{-1} for the processes whose aggregated series converge to second-order pure noise. For our discussion below, we assume for simplicity that the slowly varying functions L_1 and L_2 in 1.1 and 1.5, respectively, are asymptotically constant. It was shown [75] that a specification of the autocorrelation function satisfying 1.1 (or equivalently, of the spectral density function satisfying 1.5) is the same as a specification of the sequence $(var(X^{(m)}) : m \geq 1)$ with the property

$$var(X^{(m)}) \sim am^{-\beta} \qquad \text{, as } m \to \infty \tag{1.9}$$

where a is a finite positive constant independent of m, and $\beta in(0,1)$; in fact, the parameter β is the same as in 1.1 and is related to the parameter γ in 1.5 by $\beta = 1 - \gamma$. On the other hand, for covariance stationary processes whose aggregated series $X^{(m)}$ tend to second-order pure noise, it is easy to see that the sequence $(var(X^{(m)}) : m \geq 1)$ satisfies

$$var(X^{(m)}) \sim bm^{-1} \qquad \text{, as } m \to \infty \tag{1.10}$$

where b is a finite positive constant independent of m.

The consequences of the slow decaying variances $var(X^{(m)})$ for classical statistical tests and confidence or prediction intervals can be disastrous [75], since usual standard errors (derived for conventional models) are wrong by a factor that tends to infinity as the sample size increases.

One of the attractive features of using self-similar models for time series, when appropriate, is that the degree of self-similarity of a series is expressed using only a single parameter. The parameter expresses the speed of decay of the series' autocorrelation function. For historical reasons, the parameter used is called the Hurst parameter.

1.2.3 Self-similarity of web traffic

One of the most important aspects of self-similar traffic is that there is no characteristic size of a traffic burst; as a result, the aggregation or superimposition of many such sources does not result in a smoother traffic pattern [25,75]. One way to assess this effect is by visually inspecting time series plots of traffic demands.

In Fig.1.2 we show four time series plots of the WWW traffic induced by our reference traces. The plots are produced by aggregating byte traffic into discrete bins of 1, 10, 100, or 1000 seconds.

The upper left plot is a complete presentation of the entire traffic time se-

ries using 1000 second (16.6 minute) bins. The diurnal cycle of network demand is clearly evident, and day to day activity shows noticeable bursts. However, even within the active portion of a single day there is significant burstiness; this is shown in the upper right plot, which uses a 100 second timescale and is taken from a typical day in the middle of the dataset. Finally, the lower left plot shows a portion of the 100 second plot, expanded to 10 second detail; and the lower right plot shows a portion of the lower left expanded to 1 second detail. These plots show significant bursts occurring at the second-to-second level [25].

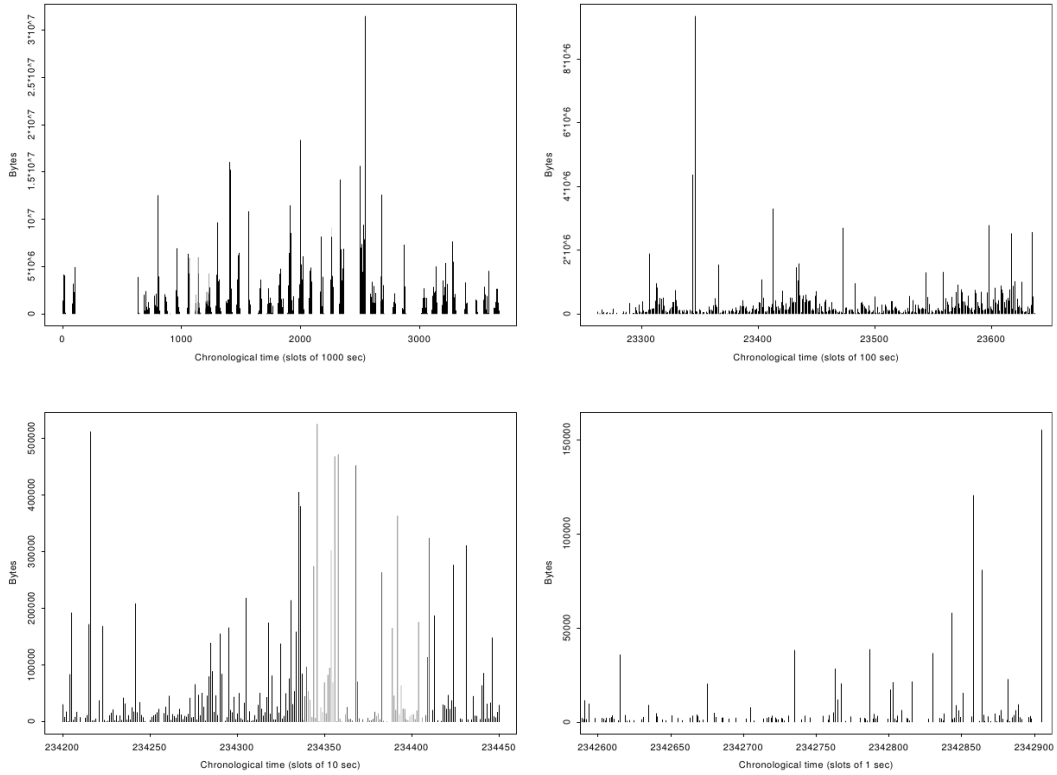


Figure 1.2: Self-similar WEB traffic

1.2.4 Hurst parameter

Hurst parameter is a convenient unit of measurements of self-similarity of stochastic process. This parameter was named by name of H.E.Hurst, devoted the life to studying of Nile and other rivers, and also problems of storage of water. Among other questions Hurst has found out, that the water level in Nile for the 800-years period shows attributes of self-similarity. This section acquaints with a history of creation of parameter H.

The hydrologist by education, Hurst investigated long-term characteristics of streams of water of the various rivers. He has noticed, that some rivers (for example, Rein) have rather soft fluctuations. Other rivers, such as Nile, show completely other picture. Hurst has noticed, that long-term and short-term characteristics are similar. In short-term prospect, as one would expect, the water level in Nile changed from year to year. Also it was possible to expect, that these fluctuations will tend to averaging so for longer period the level of Nile should remain in rather narrow range with the good and bad periods, frequently changing each other. However actually Nile behaves differently. In long-term prospect the long periods of a drought follow the long periods of high water, in which flooding occurred almost each year. Hurst has dealt with a problem of designing of the ideal tank for regulation of a stream of Nile on the basis of available record of supervision over a level of the river. The ideal tank should provide the constant stream equal to an average entrance stream which is never overflown and never runs low. For the decision of the given problem it is required to obtain the data on variability of a stream of water. Assuming that the water level is measured once a year we can define the following variables:

X_j - an entrance stream for one year j, $(I \leq j \leq N)$; these are investigated time sequences;

$M(N)$ - the constant annual proceeding stream based on supervision during N years;

L_j - a water level in the tank by the end of year j, $(1 \leq j \leq N)$;

N - number of years of supervision.

These sizes are illustrated with Fig.1.3. Being based on record of N years, we would like to receive values of the minimal and maximal water levels in the tank $L_{min}(N)$, $L_{max}(N)$, and also range $R(N) = L_{max}(N) - L_{min}(N)$.

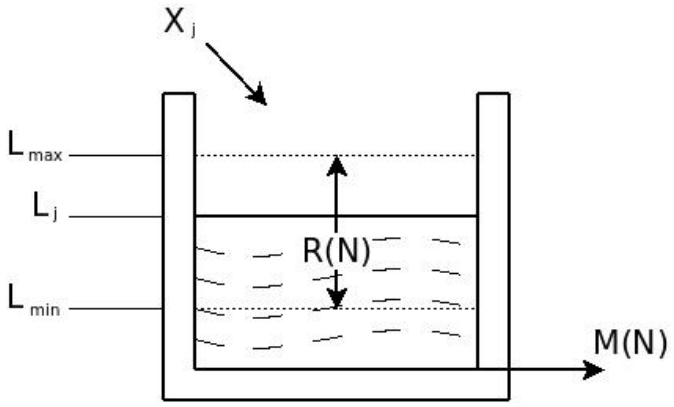


Figure 1.3: An illustration of parameters for Hurst analysis

Taking into account the data on an entrance stream for the period, parameters wanted are easy for calculating:

$$M(N) = \frac{1}{N}\sum_{j=1}^{N} X_j \tag{1.11}$$

$$L_j = \sum_{j=1}^{N} X_j - jM(N) \tag{1.12}$$

$$R(N) = \max_{1\le j\le N} L_j - \min_{1\le j\le N} L_j \tag{1.13}$$

Thus, $M(N)$ represents an average entrance stream for N years, L_j a total entrance stream for the first j years a minus a total target stream for the same years, $R(N)$ is a difference between maximal and minimal L_j values for these years. Obviously, the range depends on an interval of time N and represents not decreasing function N. Pay attention that parameter R is not equal to a range of time sequence X_j which has the following form:

$$Range(X, N) = \max_{1\le j\le N} X_j - \min_{1\le j\le N} X_j \tag{1.14}$$

Instead, we can consider L_j as accrued size on which the time sequence deviates average value in time j. Thus, R represents size which, in the certain sense, characterizes variability of casual variable X better.

Hurst investigated a number of the phenomena and has developed the normalized dimensionless size R/S describing variability where S represents selective average:

$$S = \sqrt{\frac{1}{N}\sum_{k=1}^{N}[X_k - M(N)]^2} \tag{1.15}$$

In the certain sense both R, and S measure variability of the data. Size R linearly depends on the data whereas S takes into account squares of reference values. Hurst named this attitude the rescaled range. He has found out, that for many natural phenomena, including the charge of water in the rivers and annual rings of trees, R/S attitude as function N is well described by the following empirical formula for big values of N:

$$R/S \sim (N/2)^H \qquad \text{, when } H > 0.5 \tag{1.16}$$

It is simple to be convinced of it visually if to construct the diagram of R/S

dependence from N in logarithmic scale. Hurst has found out, that for many data sets points are laying on a straight line and the inclination of this line represents parameter H from the previous formula.

It is possible to show, that for any short-term process relation R/S becomes asymptotically proportional $N/2$, that is $H = 0.5$. However Hurst has found a set of the phenomena with values H varying in a range from 0.7 up to 0.9. Such big values of H parameter assume the big degree of variability of the data.

The Hurst parameter will be used as a main parameter to define if a processes or time series are self-similar throughout this work.

1.3 Problem statement

Taking into account the said above the following problems may arise in a mobile wireless system:

- System overload, causing:
- High latency and
- Packet loss
- Broken resource management (CPU, memory, channel throughput)

To help solving these problems we should research how the serving nodes (access points) process the requests - how the queue behaves, how long is the relaxation time of the system. Knowing that we have limited resources and the buffer is not unlimited, knowing the current traffic parameters and when we can accept the next client request we could develop MBAC mechanisms for serving node resource control.

Current research is covering the problem of finding queue sizes and system relaxation times under different realistic traffic parameters. But to start the transient mode research we first have to research the wireless traffic parameters and it's nature. The next chapter is devoted to wireless traffic research.

2
Wireless network traffic

2.1 Experiment

The statistical characteristics of network traffic has been of interest to researchers for many years, not least to obtain a better understanding of the factors that effect the performance and scalability of large systems such as the Internet. Early studies of Internet traffic proved particularly interesting as they showed self-similar characteristics that were not previously commonplace. That means that similar statistical patterns can be seen over different time scales varying from milliseconds to hours. And that for instance means that models which assume traffic to be Poisson (for example) distributed and short-range dependent can not be used for modeling networks on practice.

Unlike many previous studies analyzing traffic in wired networks (mostly Ethernet) in the given work local wireless intranet traffic of a company was collected and analyzed.

It is shown that the traffic in wireless computer networks is also self-similar and long-range dependent. So hardware manufacturers, software development companies and wireless ISPs should take this into account and should not rely on classical traffic models.

Experiment took place in a hybrid network consisting as from the big segments of a wireless network and middle size (some floors of the main building,

offices of clients) segments of a 100 Mbit Ethernet network. Practically all network is constructed on the basis of RouterBoard [13] routers with MikroTik RouterOS software installed on them. Exception are some switchboards on floors and networks of clients.

In the following sections we will consider some methods of network information statistics acquisition and why author choose to build it's own tool to collect needed information.

2.1.1 Passive packet capture

Many network monitoring tools are based on passive packet capture. The principle is the following: the tool attaches itself to the network socket in the kernel and makes a copy of each network packet that arrives at the network interface - passively captures packets flowing on the network and analyzes them in order to compute traffic statistics and reports including network protocols being used, communication problems, network security and bandwidth usage. Most network tools that need to perform packet capture (tcpdump, ethereal, snort) are based on a popular programming library called libpcap that provides a high level interface to packet capture. The main library features are: ability to capture from various network media such as ethernet, serial lines, virtual interfaces; same programming interface on every platform; advanced packet filtering capabilities based on BPF (Berkeley Packet Filtering), implemented into the OS kernel for better performance. Depending on the operating system, libpcap implements a virtual device from which captured packets are read from userspace applications. Despite different platforms provide the very same API, the libpcap performance varies significantly according to the platform being used. On low traffic conditions there is no big difference among the various platforms as all the packets are captured, whereas at high speed the situation changes significantly.

It was shown [29] that capturing packets in 100 Mbit network using a usual PC, the simplest packet capture application is not able to capture everything (i.e. there is packet loss) and Linux, a very popular OS used for running network appliances, performs very poorly with respect to other OSs used in the same test. Linux (this is interesting for us, as MikroTik RouterOS is based on Linux kernel), with standard library, showed extremely low results – it was able to capture only 0.2% of packets. However with optimized library it was able to capture up to 76% of traffic on small (64 bytes) packets and up to 93% on large (1500 bytes) packets. So average error is about 15% i.e. we can capture

85% of traffic.

This method is very popular in network analysis and very convenient when we need to analyze network applications errors or protocol usage in the network. It allows to collect statistics with very high resolution (up to milliseconds) but packet loss is still very significant even with optimized libraries. This was the course of giving up this method for our measurements.

2.1.2 Device statistics monitoring

The Linux operating system keeps track of the number of packets and bytes sent and received in a virtual file called `/proc/net/dev` using counters. The counters get reset at machine reboot.

A simple PERL (or other language) script can be written to query this file at regular intervals of one second and record the value of the counters of interface, we are interested in, and a time stamp to a file. Several network interface monitoring programs like `ifconfig` or `netstat` use `/proc/net/dev` to show statistics on bytes and packets.

This method is believed to be very accurate (and it really is) only if the system is able to handle all traffic e.g. CPU is not high loaded and network interface corresponds to all requirements (in most cases this is speed). If the system is not able to handle the traffic `/proc/net/dev` will show values which are much lower than the real traffic. This method requires at least user access to the system shell.

2.1.3 SNMP

SNMP stands for Simple Network Management Protocol [14]. It's initial aim was to integrate the management of different types of networks with a simple design that caused very little stress on the network. SNMP operates at the application level using TCP/IP transport-level protocols so it can ignore the underlying network hardware. This means the management software uses IP, and so can control devices on any connected network, not just those attached to its physical network. This also has disadvantages: if the IP routing is not working correctly between two devices, it's impossible to reach the target to monitor or reconfigure it. There are two main elements in the SNMP architecture: the agent and the manager. It's a client-server architecture, where the agent is the server and the manager is the client.

The agent is a program running in each of the monitored or managed nodes of the network. It provides an interface to all the items of their configuration.

These items are stored in a data structure called a management information base (MIB).

The manager is the software that runs in the monitoring station of the network, and its role is contacting the different agents running in the network to poll for values of its internal data. It's the client side of the communication.

The extensibility of the protocol is directly related to the capability of the MIB to store new items. If a manufacturer wants to add some new commands to a device such as a router, he must add the appropriate variables to its database (MIB).

Almost all manufacturers implement versions of SNMP agents in their devices: routers, hubs, operating systems, and so on. Linux is not an exception to this, and publicly available SNMP agents for Linux can be found on the Internet.

SNMP provides very little support for authentication schemes. It supports only a two-password scheme. The public allows managers to request the values of variables, and the private allows these values to be set. These passwords in SNMP are called communities. Every device connected to an SNMP-managed network must have these two communities configured. It is very common to have the public community set to "public" and the private community to "private", but it's very important to change these values to reflect the security policy of your organization. Note: MikroTik RouterOS supports only GET (read) requests, so there will be only public by default.

SNMP defines a separate standard for the data managed by the protocol. This standard defines the data maintained by a device in the network and what operations are allowed on it. The data is structured in a tree form, and there is a unique path to reach each variable. This structured tree is called the Management Information Base (MIB) and is documented in several RFCs.

The current version of the TCP/IP MIB is MIB-II and is defined in RFC-1213. It divides the information a TCP/IP device should maintain into several categories and each variable included in this information must fall in one of them.

The MIB definition of a particular item also specifies the data type it can contain. Usually, items of an MIB can store single integers, but they can also contain strings or more complex structures, like tables. Items in an MIB are called objects. Objects are the leaf nodes of the MIB tree, but an object can have more than one instance e.g. a table object. To refer to the value contained in an object, you must add the number of the instance. When only one instance exists for an object, this is the "0" instance. For example, the object `ifNumber` from category "interfaces" contains an integer with the number of

interfaces present in this device, but the object `ipRoutingTable` from category "ip" contains the routing table of the device. Remember to use the number of the instance to retrieve the value for an object. In this case, the number of interfaces present in a router can be viewed with the instance `ifNumber.0`. It's important to notice that most of the software needs the leading dot (root) to locate the object in the MIB. So if we want to know how many interfaces a router has the query will look like this: `.interfaces.ifNumber.0` snmpget example request to MikroTik router:

```
$ snmpget -v 1 -c public x.x.x.105 .interfaces.ifNumber.0 ifNumber.0 4
```

So the router with `x.x.x.105` IP address has 4 interfaces. A new specification called SNMPv2 is being actively developed. It addresses the lack of security of the actual protocol with mechanisms that focus on privacy, authentication and access control. It also allows more complex specification of variables and has some additional commands. The problem with SNMPv2 is it still is not a commonly accepted standard, unlike SNMPv1 (as it's shown in example command MikroTik routers support only SNMPv1). It is not easy to find SNMPv2 versions of the agents and software to take advantage of the new commands.

The Multi Router Traffic Grapher (MRTG) is a widely used SNMP tool to monitor the traffic load on network links [11]. MRTG generates HTML pages containing graphical images which provide a live visual representation of this traffic.

MRTG consists of a Perl script which uses SNMP to read the traffic counters of your routers and a fast C program which logs the traffic data and creates pretty graphs representing the traffic on the monitored network connection. These graphs are embedded into web pages which can be viewed from any modern Web-browser. In addition to a detailed daily view, MRTG also creates visual representations of the traffic seen during the last seven days, the last five weeks and the last twelve months. This is possible because MRTG keeps a log of all the data it has pulled from the router. This log is automatically consolidated so that it does not grow over time, but still contains all the relevant data for all the traffic seen over the last two years. This is all performed in an efficient manner. Therefore you can monitor 200 or more network links from any halfway decent UNIX box.

MRTG is the most popular software for monitoring network devices via SNMP protocol and it's perfect for understanding the overall picture on what is going on in your networks. But it's not suitable for getting detailed traffic logs as it aggregates statistics by 300 values (e.g. It shows average values for

5 minutes). For statistical analysis we need detailed logs (at least 1-5 seconds average), so this program was left behind.

After reviewing all possible methods for getting network traffic statistics it was decided to write own SNMP client to monitor needed routers.

2.1.4 Data acuision tool

For getting packet statistics from the router a Perl program was written. This is simple SNMPv1 client which queries a router at regular intervals of one second for number of packets a particular interface sent and received and then writes the difference between two last queries, e.g. it writes network interface speed in packets per second (one row for one second). MIBs used in program are:

- `.interfaces.ifTable.ifEntry.ifOutUcastPkts.` – number of packets received by an interface
- `.interfaces.ifTable.ifEntry.ifInUcastPkts.` – number of packets received by an interface

Because of Perl SNMP module specifics their number equivalents should be used:

- `.interfaces.ifTable.ifEntry.ifOutUcastPkts.` → `.1.3.6.1.2.1.2.2.1.17.`
- `.interfaces.ifTable.ifEntry.ifInUcastPkts.` → `.1.3.6.1.2.1.2.2.1.11.`

After getting number of packets and calculating the speed this program writes it to a file with sane name like “ipinoutdate.pkts”, where IP is an IP address of monitored router, in|out is direction (whether in or out) and date is date when log started in format daymonth-hourminute (e.g.: `10.10.10.10_in_1708_2203.pkts`). The program is called speed.pl. It works under Linux operating system (it will probably work under other OS but it was not checked) with PERL and Net::SNMP module installed. It should be run in command line like this:

```
$ ./speed.pl IP MIB COMMUNITY
```

where IP is an IP address of the router, MIB is the number of network interface you want to monitor which you have to know before you run the program and COMMUNITY is SNMP comuunity (this argument is optional, default is “mrtg”). An advantage of this program is that it opens an SNMP session and never closes it unless it’s stopped manually by Ctrl-c or simply killed by other

means. In this way it reduces the time needed to get wanted response but network influence can't be fully eliminated therefore not very big packet statistics loss take place. For example for week traffic 598742 records were made instead of 604800 (number of seconds in week) expected. So approximate statistics loss is about 1% which is much less than in case of passive packet capture method. Program `speed.pl` listing can be found in Appendix A, Listing A.1.

2.2 Experimental data

In this section only one out of eight data files will be considered. For others please see Appendix B. Fig.2.1 shows the incoming traffic captured from the wireless router during six days in packets per second.

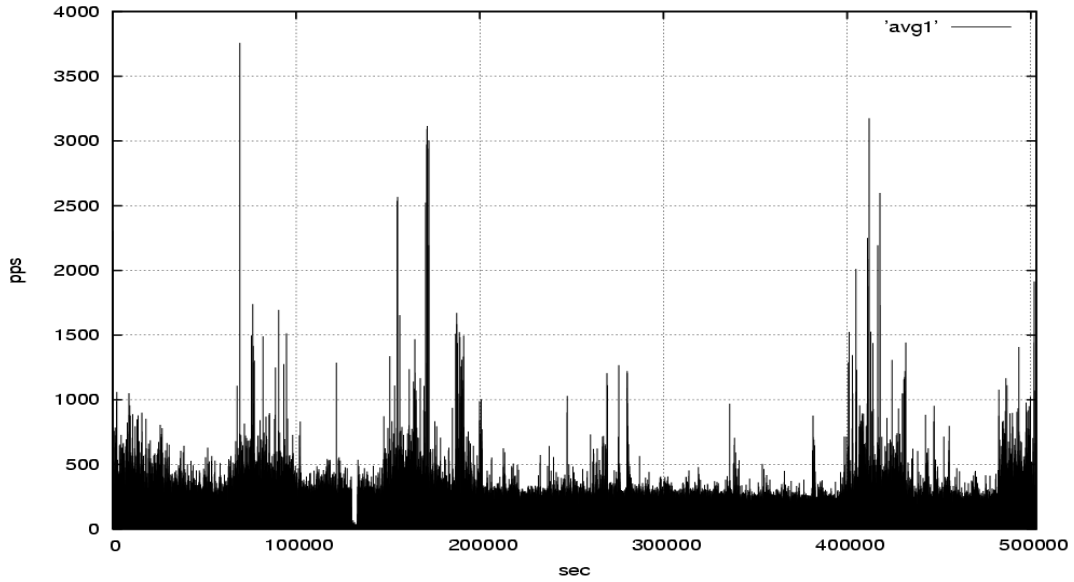


Figure 2.1: Wireless network traffic, six days period

Burstness of the traffic is clearly seen on Figure 2.1 what allows to suggest the the traffic could be self-similar. It's also seen that early in the Friday morning there was no data from the router for a particular amount of time due power line failure. Figures 2.2 through 2.9 show different magnification of the traffic which can be read from the x scale.

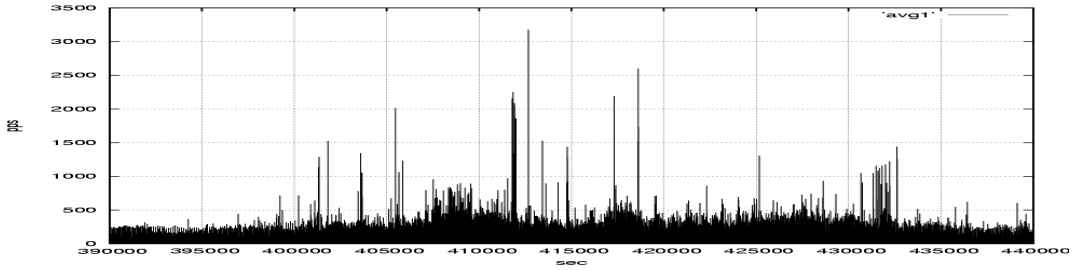


Figure 2.2: Wireless network traffic, one day period

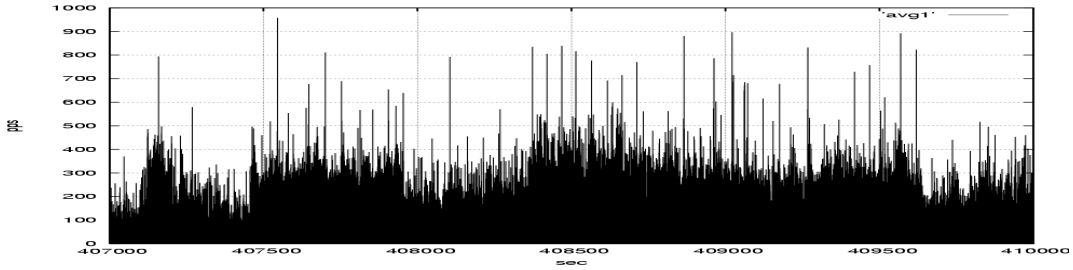


Figure 2.3: Wireless network traffic, one hour period

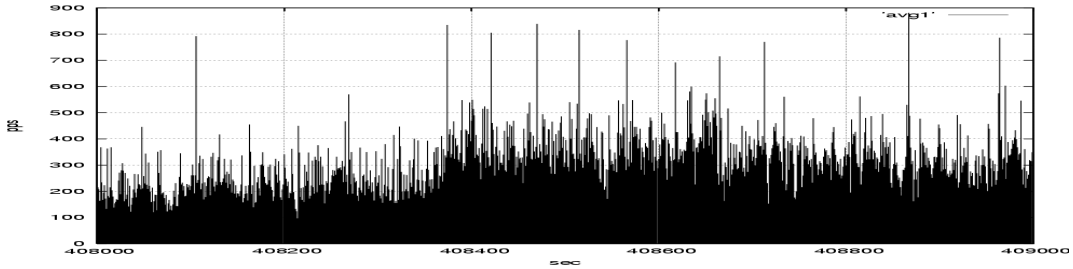


Figure 2.4: Wireless network traffic, 20 minutes period

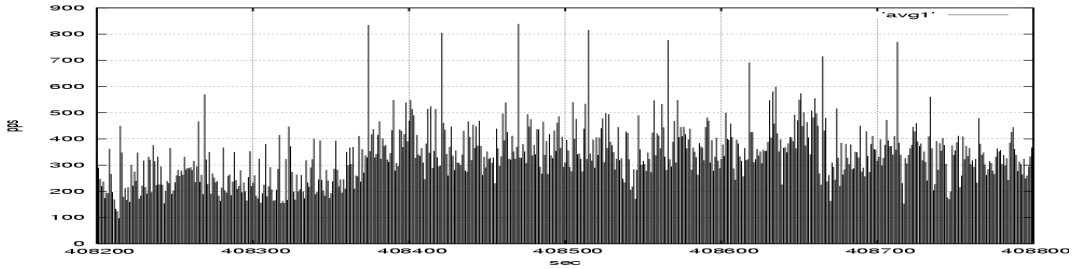


Figure 2.5: Wireless network traffic, 10 minutes period

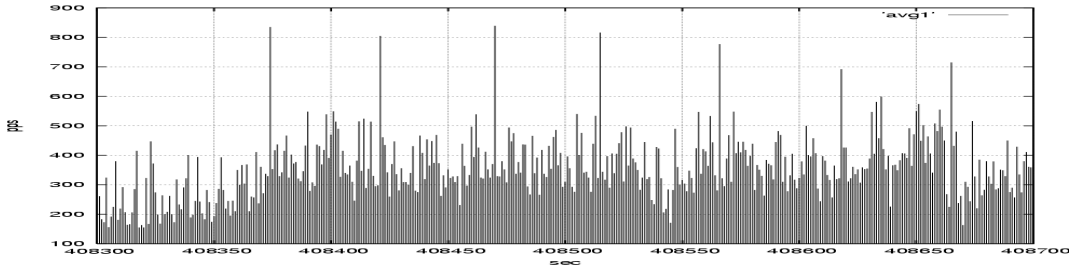


Figure 2.6: Wireless network traffic, 7 minutes period

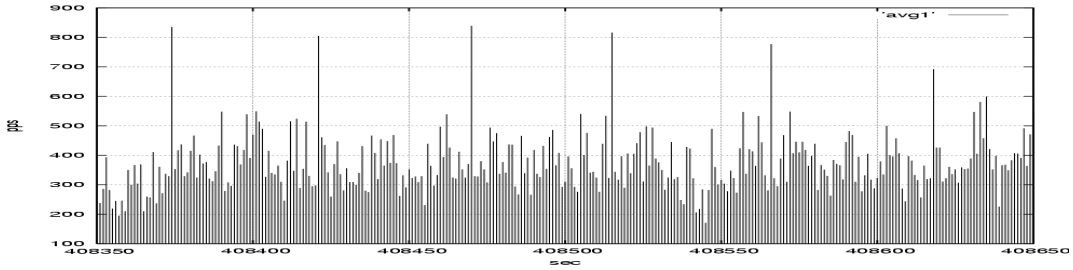


Figure 2.7: Wireless network traffic, 5 minutes period

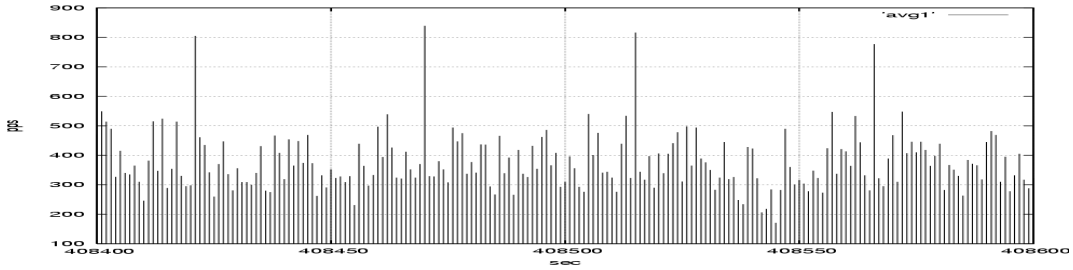


Figure 2.8: Wireless network traffic, 3 minutes period

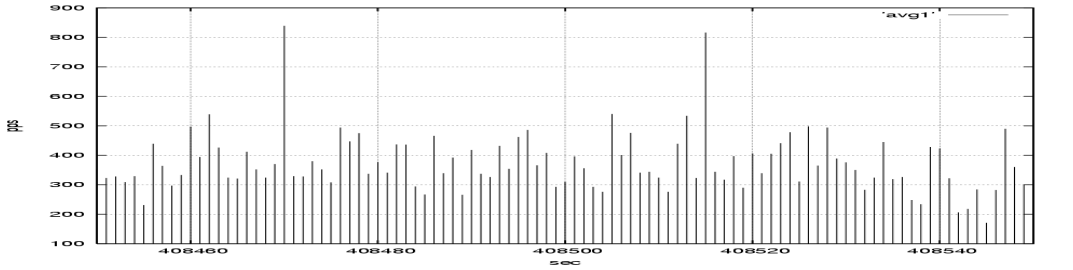


Figure 2.9: Wireless network traffic, 2 minutes period

The above figures show that the measured traffic is visually self-similar at different degree of magnitude. But to claim this we have to perform statistical computations over the dataset.

2.3 Data processing methods and tools

2.3.1 Methods of Hurst parameter evaluation

Some methods of evaluation, whether the given time series given are self-similar, are developed and if yes what is self-similarity parameter H in this case. In this section some of the most widespread methods [96] are considered.

Variance-time plots

We have seen in Section 1.2.2 that for self-similar processes, the variances of the aggregated processes $X^{(m)}, m = 1, 2, 3, \ldots$ decrease linearly in log-log plots against m with slopes arbitrarily flatter than -1 (see 1.9). Actually when $m = 1$ the process is not aggregated, so calculation can be started from $m = 2$ without any statistics loss. On the other hand, none of the short-range dependent processes (e.g. MMPP, fluid models, ARMA models [75]) yield a power-law for the variances of the form 1.9; it can be approximated for some transient period of time by short-range dependent models with a large number of parameters, but the variance of $X^{(m)}$ will eventually decrease linearly in log-log plots against m with a slope equal to -1 (see 1.10). The so-called variance-time plots are obtained by plotting $log(var(X^{(m)}))$ against $log(m)$ ("time") and by fitting a simple least squares line through the resulting points. Values of the estimate β of the asymptotic slope between -1 and 0 suggest self-similarity, and an estimate for the degree of self-similarity is given by $H = 1 - \beta/2$.

Clearly, variance-time plots are not reliable for empirical records with small sample sizes [75]. However, with sample sizes large enough [75, 96, 100] variance-time plots become highly useful and give a rather accurate picture about the self-similar nature of the underlying time series and about the degree of self-similarity. An example of variance-time plot of wireless network traffic is shown on Figure 2.10.

R/S analysis

R/S analysis is based on a heuristic graphical approach that tries to exploit as fully as possible the information in a given record. The following graphical method has been used extensively in the past. Given a sample of N observations $X_k, k = 1, 2, 3, \ldots, N$, one subdivides the whole sample into K non-overlapping blocks and computes the rescaled adjusted range $R(t_i, n)/S(t_i, n)$ for each of the new "starting points" $t_1 = 1, t_2 = N/K + 1, t_3 = 2N/K + 1, \ldots$

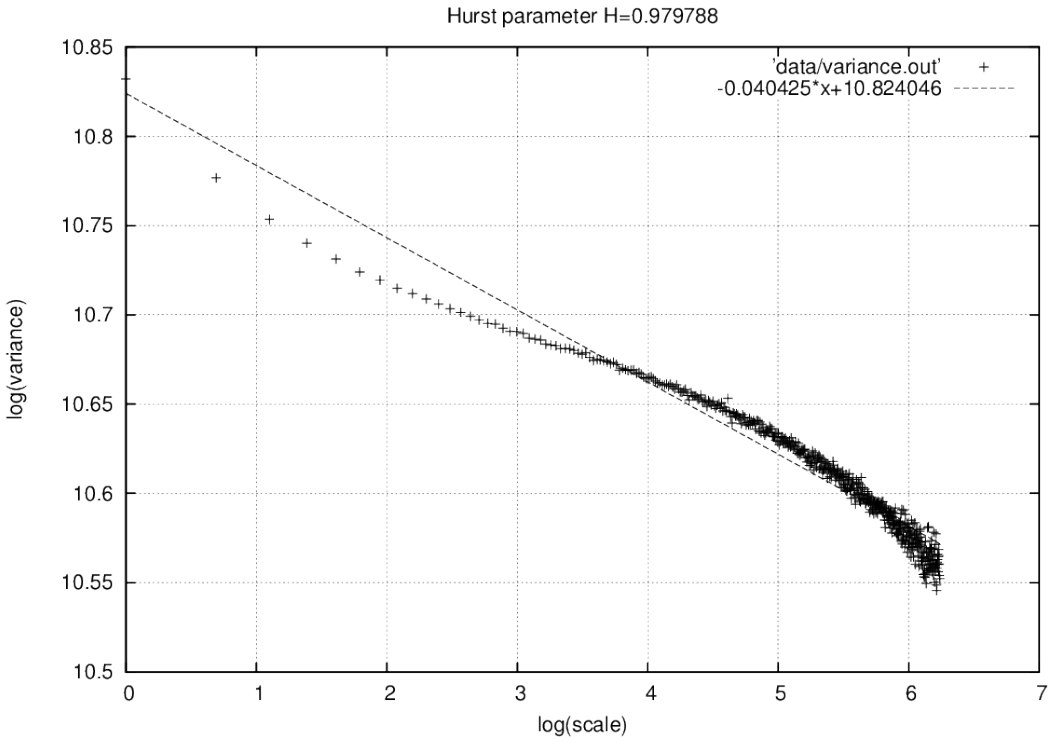


Figure 2.10: Example variance-time plot of wireless network traffic

which satisfy $(t_i - 1) + n \leq N$. Here, the R/S-statistic $R(t_i, n)/S(t_i, n)$ is defined as in 1.6 with W_k replaced by $W_{t_i+k} - W_{t_i}$ and $S^2(t_i, n)$ is the sample variance of $X_{t_i+1}, X_{t_i+2}, \ldots, X_{t_i+n}$. Thus, for a given value ("lag") of n, one obtains many samples of R/S, as many as K for small n and as few as one when n is close to the total sample size N. Next, one takes logarithmically spaced values of n, starting with $n \approx 10$. Plotting $log(R(t_i, n)/S(t_i, n))$ versus $log(n)$ results in the rescaled adjusted range plot (also called the pox diagram of R/S) [75]. When the parameter H in relation 1.7 is well defined, a typical rescaled adjusted range plot starts with a transient zone representing the nature of short-range dependence in the sample, but eventually settles down and fluctuates in a straight "street" of a certain slope. Graphical R/S analysis is used to determine whether such asymptotic behavior appears supported by the data. In the affirmative, an estimate $\hat{H}$ of the self-similarity parameter H is given by the street's asymptotic slope (typically obtained by a simple least squares fit) which can take any value between 0.5 and 1. For practical purposes, the most useful and attractive feature of the R/S analysis is it's relative robustness against changes of the marginal distribution. This feature allows for practically separate investigations of the self-similarity property of a given empirical record and of its dis-

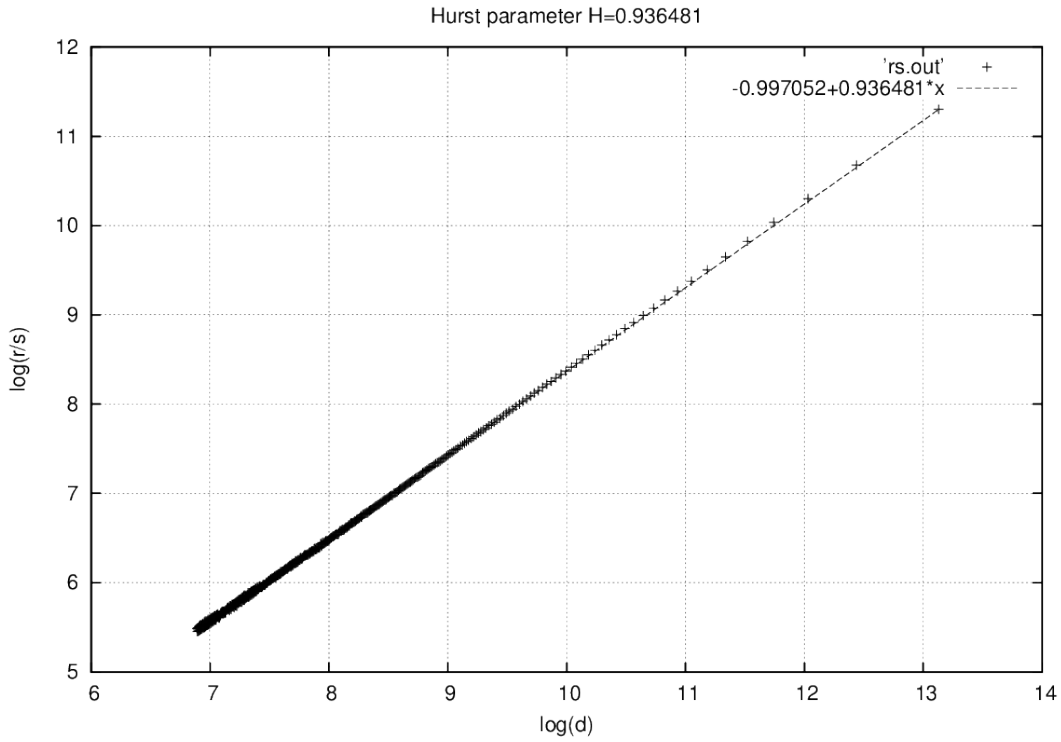


Figure 2.11: Example R/S statistics plot of the same data as on Fig.2.10

tributional characteristics. An example of R/S plot of wireless network traffic is shown on Fig.2.11.

Other methods

There are also other Hurst parameter estimation methods which were not used by the author due to their complexity but which we should mention for the sake of information completeness. Among them are wavelet methods, periodogram plots and periodogram-based maximum likelihood estimators [86, 96].

2.3.2 Analysis. Algorithms and tools

Statistical analysis of the collected data is reduced to a time sequence Hurst parameter evaluation i.e. an establishment of a degree of self-similarity of a sequence of packages transmitted on a wireless network and correlation analysis i.e. finding if the sequence is correlated with itself (autocorrelation doesn't drop to zero fast).

All tools were developed by the author and implemented in C programming language. Also few complex mathematical functions from GSL library [6] were

used. All programs were written under the GNU/Linux operating system and were not checked in other systems. For program compilation GCC compiler [4] was used. For program start and automation bash scripts [3] were written. Plots were made using gnuplot [7]. All the sources are available in Appendix.A. Data processing results are shown in the next section.

Data scaling

To estimate the Hurst parameter H of the traffic shown on Fig.2.1 we have to generate as many aggregated processes $X^{(m)}$ as possible from the original process X. The more aggregated processes we will have the more precise will be the evaluation of H. Hence the importance of original dataset X length. We generate $m = 2, 3, \ldots, 512$ aggregated processes as follows:

$$X_k^{(m)} = \frac{1}{m}(X_{(km+1)-m} + \cdots + X_{(km+1)-1}) \qquad , k \geq 1$$

It's obvious that calculations make sense only when $2 \leq m \leq N$, where N is the length of non aggregated sequence. At the same time the aggregated process should have enough values so that variance could be calculated with good precision too. Hence the limit of $m = 512$ processes. For the implementation see listings A.2 and A.3.

Hurst parameter estimation

For the Hurst parameter analysis we first have to find variances for each aggregated process $X^{(m)}$ as follows:

$$var(X^{(m)}) = \frac{1}{N^{(m)}} \sum_{i=1}^{N^{(m)}} (X_i^{(m)} - \overline{X^{(m)}})^2$$

whereis $N^{(m)}$ is the length of the aggregated process. Then, plotting $X^{(m)}$ variances $var(X^{(m)})$ against scale m in logarithmic scale we will get so-called variance-time plot. But that's not all. To get the actual Hurst parameter value we have to carry out least squares fitting over the plotted points. This should give a straight line with negative inclination. Knowing the tangent of the line inclination we can calculate Hurst parameter as:

$$H = 1 - \beta/2$$

where β is tangent of approximating straight line inclination which can be easily found from the line equation $y = ax + b$. Taking into account the negative

inclination $\beta = -a$. See Listing A.5 for the implementation.

The second, R/S statistics, method was used for every dataset to exclude the possibility of erroneous Hurst parameter evaluations in case of using only one method. R/S statistics method is slightly different. Values plotted against $log(m)$ are not variances but R/S values calculated from variances. Also the approximating line slope is positive. See Section 2.3.1 and Listing A.6 for the implementation.

Correlation analysis

Correlation is a measure of the degree of linear relationship between two variables or processes. While in regression the emphasis is on predicting one variable from the other, in correlation the emphasis is on the degree to which a linear model may describe the relationship between two variables. In regression the interest is directional, one variable is predicted and the other is the predicto. In correlation the interest is non-directional, the relationship is the critical aspect.

If correlation is a coordination process between two different signals, then autocorrelation is a coordination of signal with it's own lagging version [99].

As soon as we have only one process and we are interested in finding out how much it is similar to itself we need to define autocorrelation.

Autocorrelation function $R_x(\tau)$ of an analog signal $x(t)$ can be defined as follows:

$$R_x(\tau) = \int_{-\infty}^{+\infty} x(t)x(t+\tau)\,\mathrm{d}t \qquad \text{, for } -\infty < \tau < +\infty \tag{2.1}$$

Autocorrelation function gives the measure of similarity of a signal with it's own copy lagged by τ units of time. Variable τ plays the role of scanning parameter. $R_x(\tau)$ is not a function of time, it's a function of time differences τ between the signal and it's lagged copy. Applying 2.1 to discretized signals we can define autocorrelation function $R(k)$ as follows:

$$R(k) = \frac{1}{N}\sum_{n=1}^{N}(X_n - \overline{X})(X_{k+n} - \overline{X}) \tag{2.2}$$

and autocorrelation coefficient $r(k)$ as

$$r(k) = \frac{1}{N}\frac{\sum_{n=1}^{N}(X_n - \overline{X})(X_{k+n} - \overline{X})}{D_X} \tag{2.3}$$

where N is sequence length, X is particular value from the sequence, $\overline{X}$ is

sequence statistical mean and D_X is variance. Correlation analysis implementation can be found in Listing.A.6.

2.4 Traffic analysis results

In this section we will take a glance on the analysis results of the same dataset as was shown in Section.2.2, leaving detailed information about other datasets in Appendix.B.

First let's consider aggregated processes. Figures 2.12 through 2.20 show aggregated traffic generated with the program described in Section.2.2 in packets per second. On every figure the scale is multiplied by two, i.e. first figure shows the traffic aggregated by two values, third by four and so on.

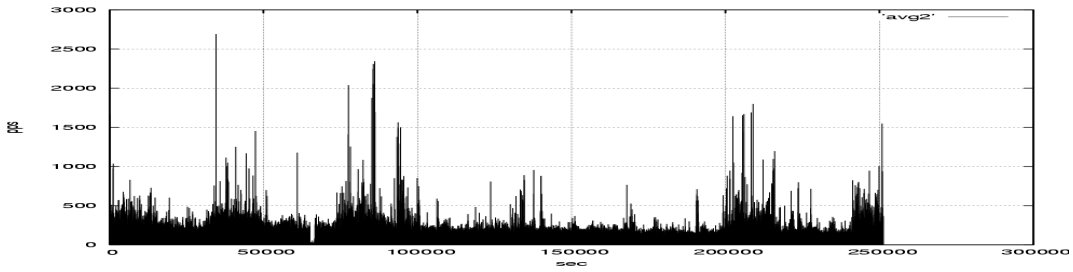


Figure 2.12: Wireless network traffic, aggregation ratio 2

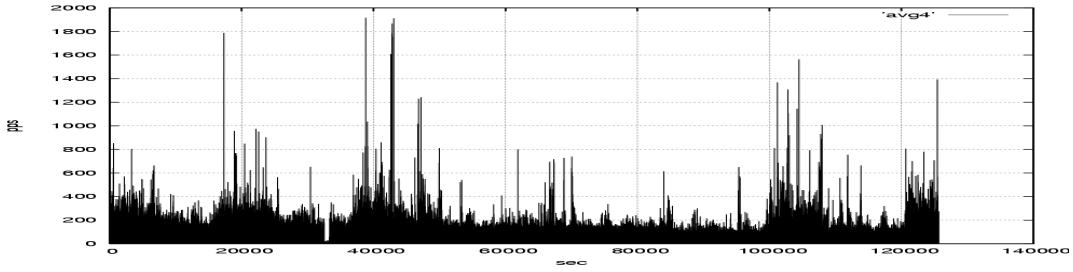


Figure 2.13: Wireless network traffic, aggregation ratio 4

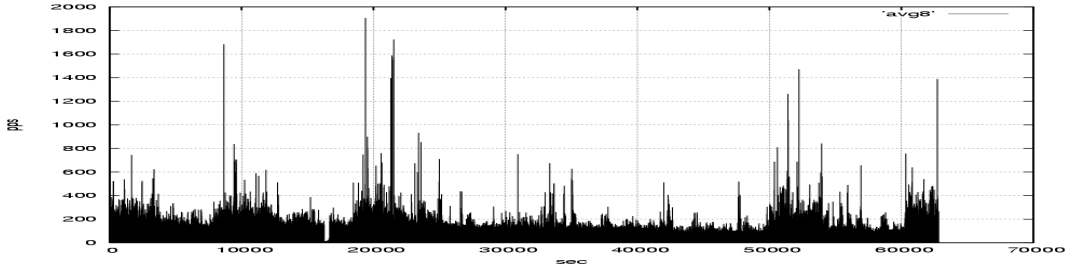


Figure 2.14: Wireless network traffic, aggregation ratio 8

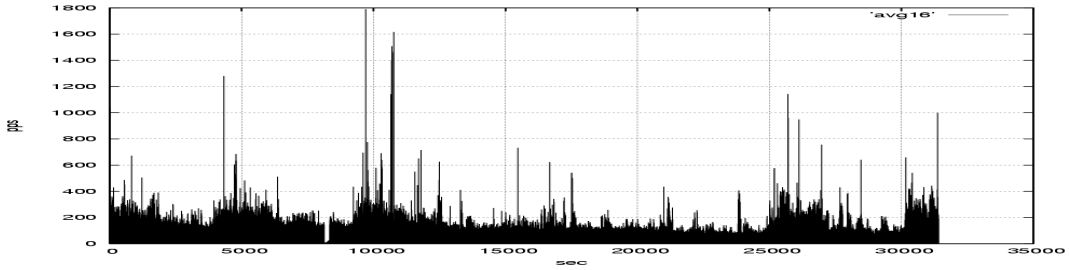


Figure 2.15: Wireless network traffic, aggregation ratio 16

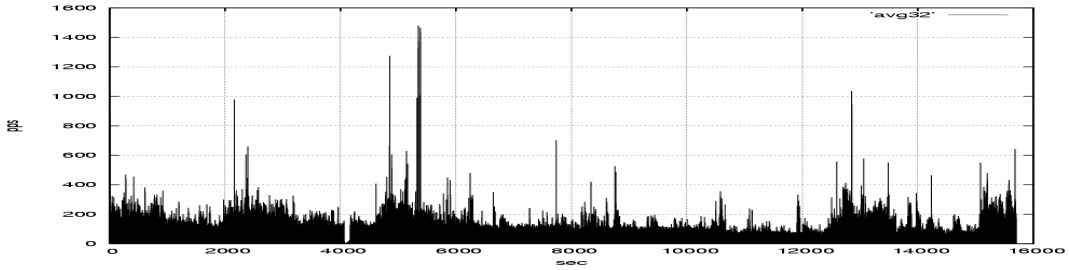


Figure 2.16: Wireless network traffic, aggregation ratio 32

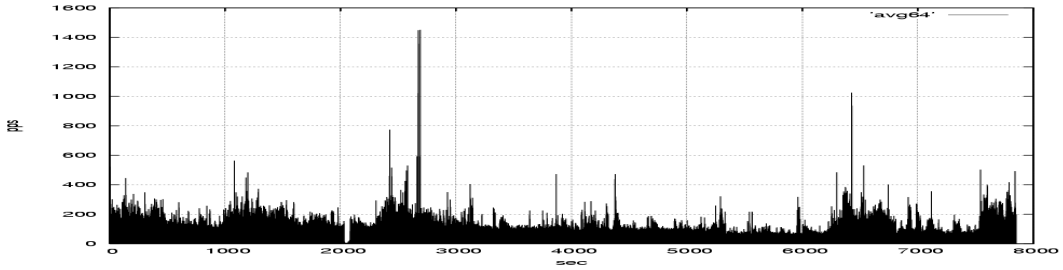


Figure 2.17: Wireless network traffic, aggregation ratio 64

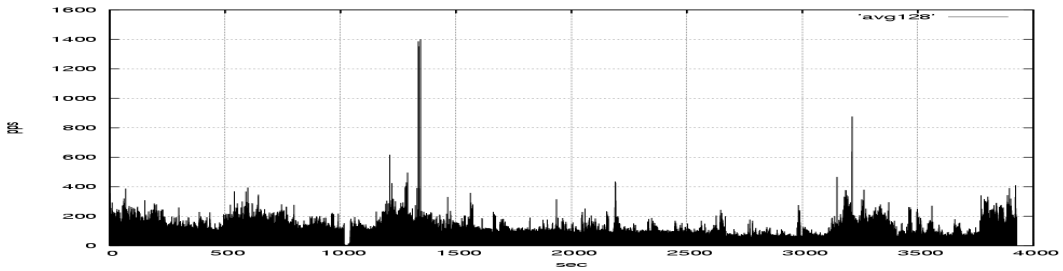


Figure 2.18: Wireless network traffic, aggregation ratio 128

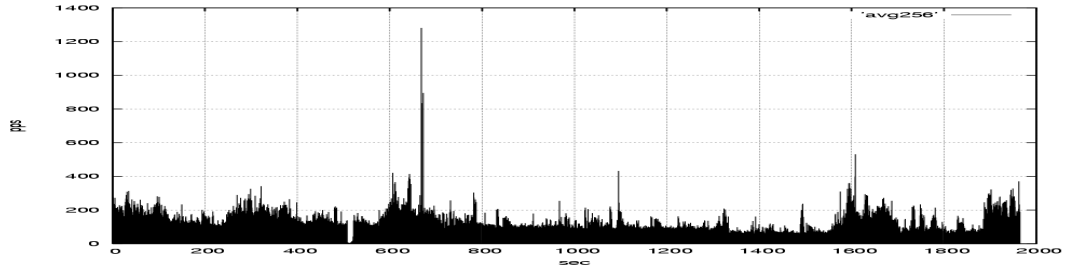


Figure 2.19: Wireless network traffic, aggregation ratio 256

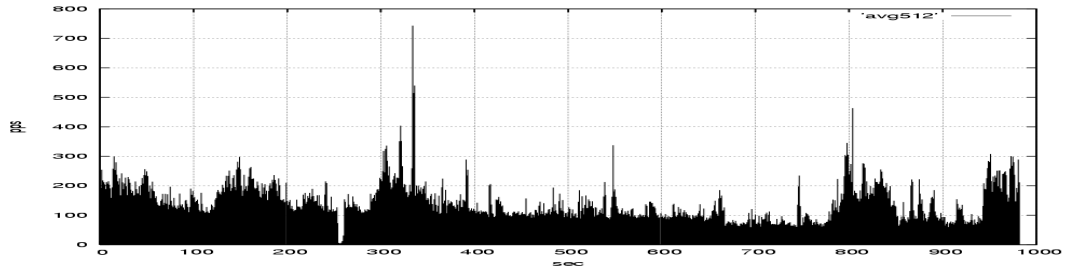


Figure 2.20: Wireless network traffic, aggregation ratio 512

Above plots show that the traffic doesn't lose it's burstness and stays visually almost the same even at high level of aggregation in contrast to non self-similar processes [100].

Now let's look at Hurst parameter estimation on Fig.2.21 and Fig.2.22

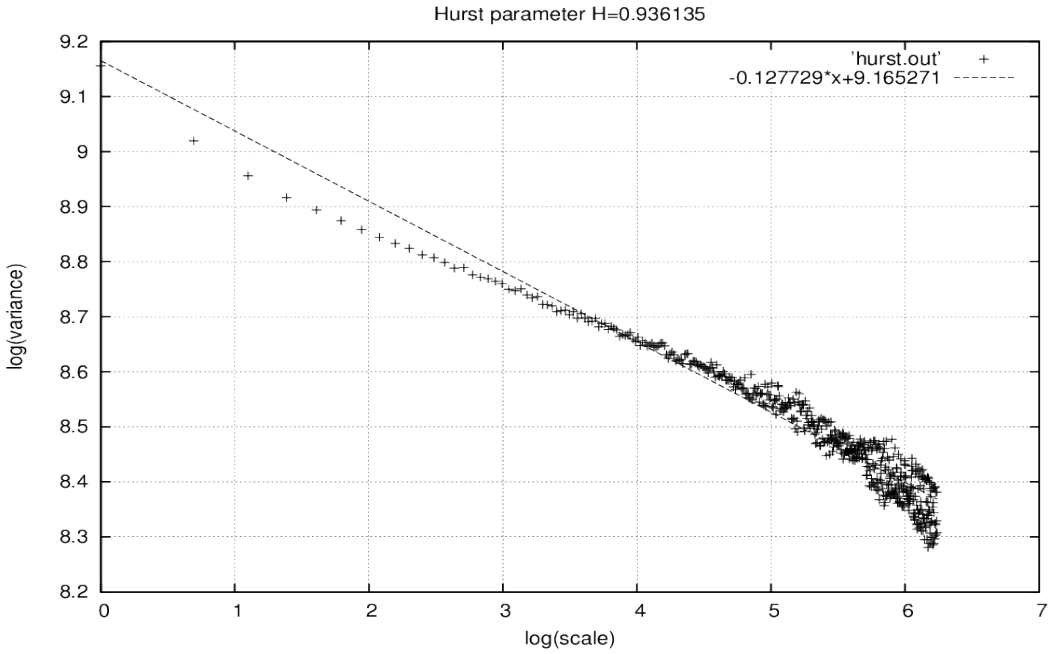


Figure 2.21: Variance-time plot of wireless network traffic

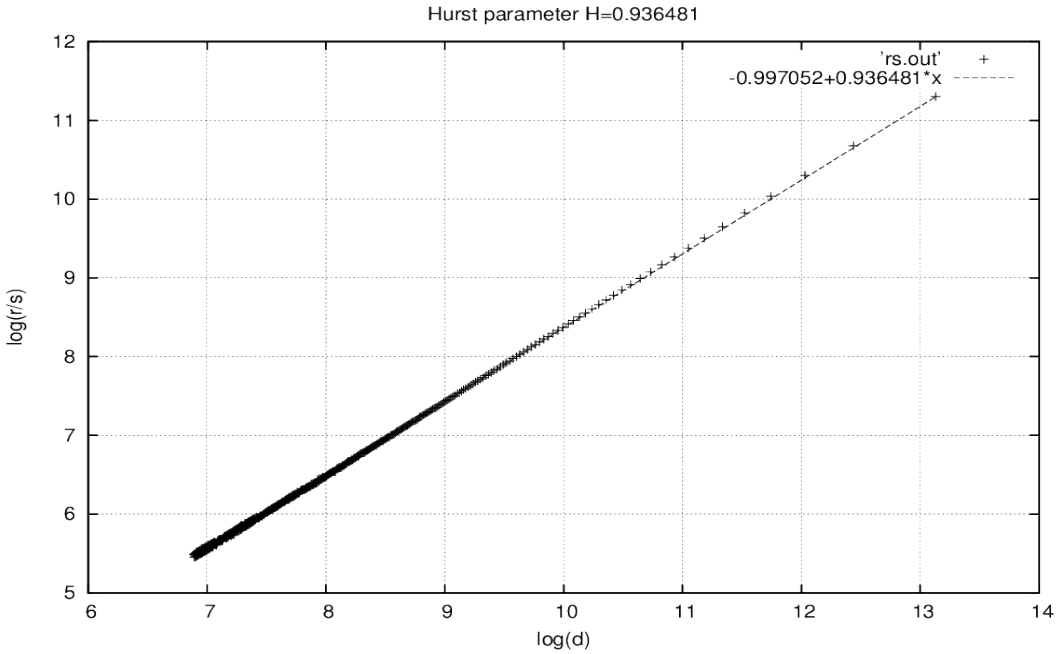


Figure 2.22: R/S statistics plot of wireless network traffic

Both methods show very close result - estimated Hurst parameter $H = 0.936$ indicates that the traffic which was analyzed is highly self-similar.

And last but not least - full range autocorrelation is shown on Figure 2.23 and first 10000 values on Figure 2.24.

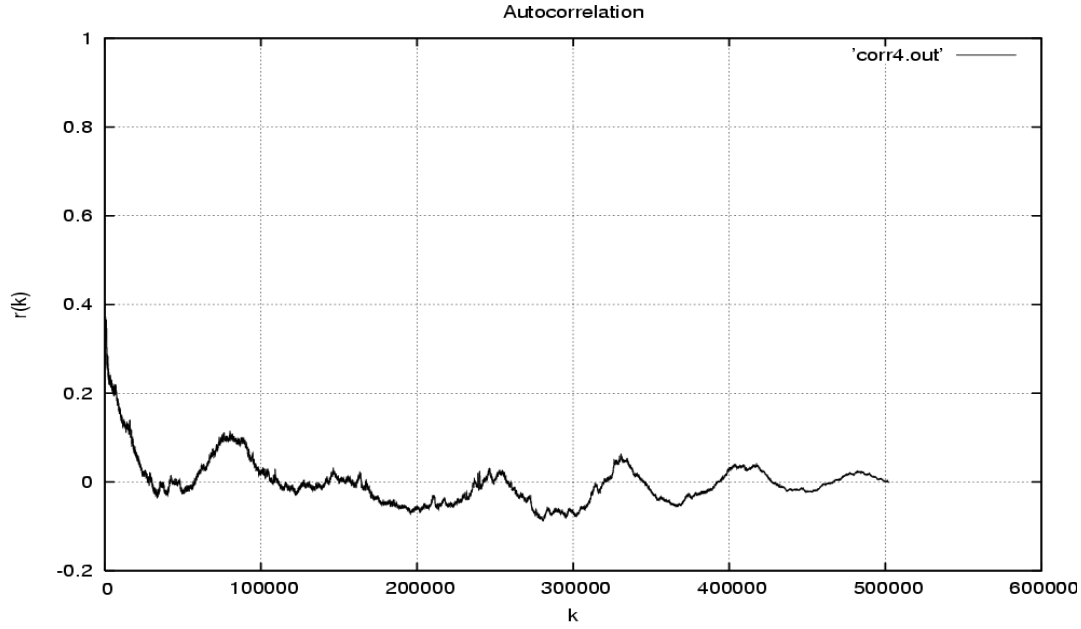


Figure 2.23: Autocorrelation of wireless network traffic, full range

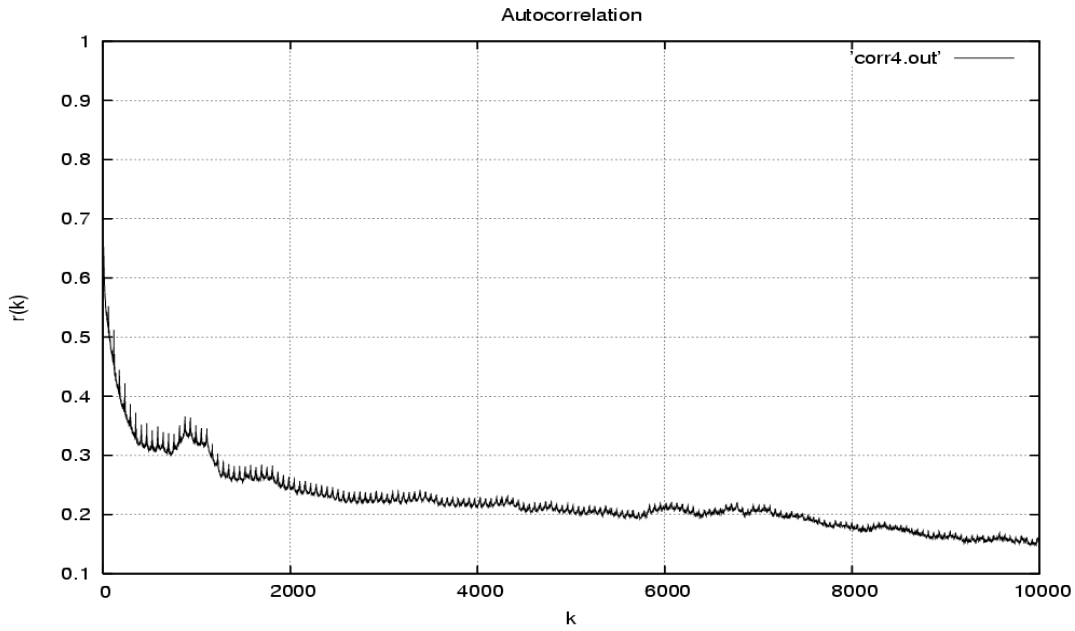


Figure 2.24: Autocorrelation of wireless network traffic, first 10000 values

It is clearly seen on the above figures that autocorrelation is not striving to zero at least for two day traffic. This means that process is long-range dependent and packets taken, for example, from sequence start and packets taken after a day are still statistically related.

2.5 Analysis summary

A number of studies (e.g. [25, 75, 96]) showed that traffic in wired networks is self-similar. In this chapter we have analyzed big amount of data collected from a wireless router and showed that the traffic in wireless network is also self-similar and long-range dependent. This study leads to the conclusion: traffic models which assume packets arrival process to be Poisson can not be applied for modeling wireless networks in practice. Instead, self-similar traffic model should be used for modeling to reflect the reality.

3

Transient process research methods

Unfortunately there are very few research methods for transient processes - most queuing theories suppose stationary system mode. The only one pure analytical method was developed by A.Kauffman an R.Cruon in [128] and allows to analyze system mean query number at any arbitrary moment of time for a system with exponentially distributed mean inter-arrival time. Another method is based on diffusion approximation and was developed by H.Kobayashi in [65,66]. Apart from queue size analysis it allows to model more than one client in the system. H.Kobayashi method was modified in order to be able to simulate self-similar query stream. But, as the author shows below, the only reliable method to research transient processes is simulation, especially when dealing with self-similar traffic where pure analytical research methods are not working or the ones which would work is not developed yet.

3.1 Analytical method

By analytical method we mean there is an equation or at least a system of equations which gives a possibility to determine system mean query number at any

given moment of time. However a math apparatus is very complicated even for $M/M/1$ systems not to mention systems with non Poisson distributed arrivals. As far as we know there are no analytical methods developed for non Poisson distributed arrivals as of yet and the only one for Poisson distributed arrivals is developed in [128]. It's essence is presented in 3.1.

$$\begin{aligned}\overline{n}(t) =&(\lambda-\mu)t+\mu\int_0^t e^{-(\lambda+\mu)\tau}\cdot \\ &\cdot\left[I_0(2\tau\sqrt{\lambda\mu})+\frac{1}{\sqrt{\Psi}}I_1(2\tau\sqrt{\lambda\mu})+(1-\Psi)\sum_{k=2}^{\infty}\frac{1}{\sqrt{\Psi^k}}I_k(2\tau\sqrt{\lambda\mu})\right]d\tau\end{aligned} \tag{3.1}$$

where λ is the intensity of input, μ is the intensity of service, $\Psi = \lambda/\mu$ is load coefficient and I_x are first kind, order x, modified Bessel functions. Equation 3.1 found in [128] allows us to find system mean query number at any moment of time. But it can be used only for $M/M/1$ systems were incoming request intensity is Poisson distributed. Nevertheless let's see how relaxation time changes under different utilization factors. Figure 3.1 shows $M/M/1$ system mean query number $\overline{n}(t)$ behavior over time t.

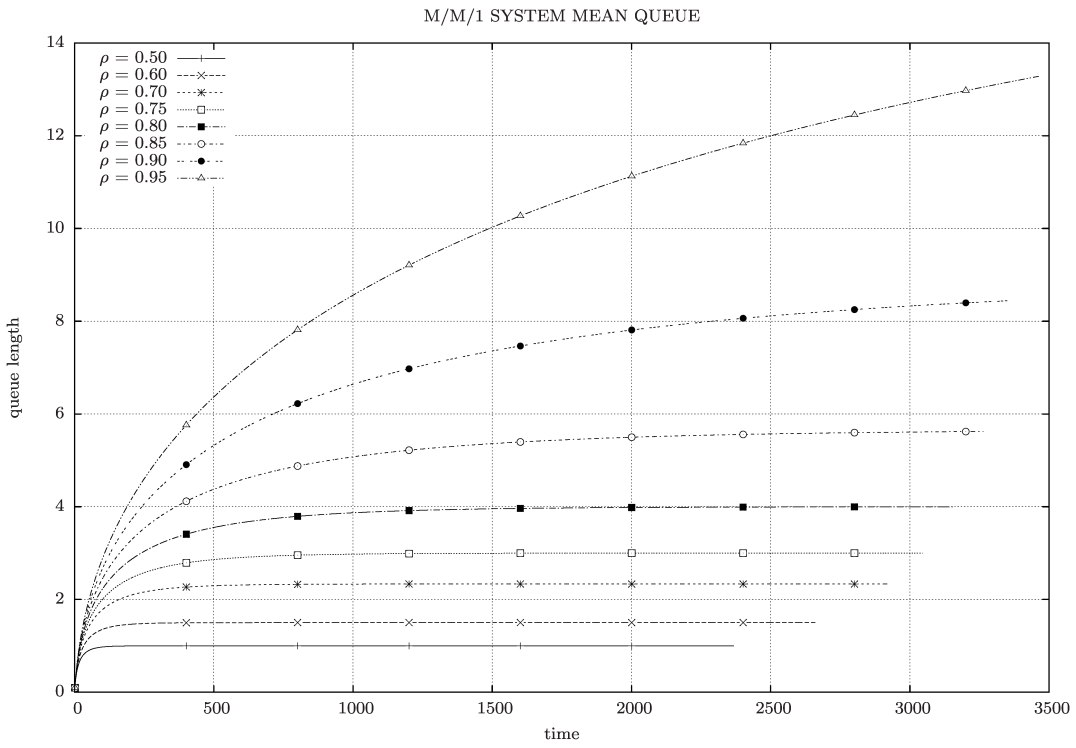


Figure 3.1: $M/M/1$ system mean query number behavior under different loads

This method was implemented in Octave [5] environment because it allows to save the data for later analysis and comparison with other methods. See Listing A.11 in Appendix A for the code.

Figure 3.1 gives very clear view - system relaxation time, i.e. the time needed for a system to enter stationary mode, increases along with utilization ρ. As the system enters stationary mode and stabilizes (i.e. $t \to \infty$) mean query number $\overline{n}(t)$ obeys 3.2 as per [63, 64]

$$\overline{n} = \frac{\rho}{1-\rho} \tag{3.2}$$

These results give us some clue on how relaxation time changes in $M/M/1$ queuing system. However we are more interested in a system with self-similar query stream, e.g. $P/M/1$ with Pareto distributed inter-arrival times. As we've already seen in Chapter 2 it would reflect the reality more closely. Thus we gradually move to the next chapter.

3.2 Diffusion approximation

For system behavior analysis in transitive non-stationary operating mode author decided to try using diffusion approximation method. In this case the model for research is cyclic closed model consisting of terminal system and a network server which common solution has been developed by H.Kobayashi [65, 66]. The model has been modified with the purpose of creation a stream of queries to a serving device which is coming nearer to self-similar.

The choice of this research method is caused by fact that imitation modeling of communication systems at self-similar traffic influence demands long time for modeling [43] and not always leads to unambiguously interpreted results. Therefore H.Kobayashi work has been taken as a basis for this research.

3.2.1 Mathematical model

Communication system model is presented in the form of a server, which has mean query service time equal to μ_1. Terminal subsystem sends queries to server input. Queries processed by server come back to the terminal system which processes them with mean time μ_2. Overall number of queries circulating in the system is equal to N. Thus, we have closed loop system. This model can be considered as an equivalent of server model with limited N size of queries buffer memory, which accepts a stream of queries on its input.

Query stream can be arbitrary, and intervals characteristics between queries are defined only by the first moment μ_1 and dispersion σ_2^2 of returned to the terminal system queries processing.

It's known, that one of the self-similar traffic models is the stream of queries, which arrival time intervals fits power-tail distributions. One of important characteristics of these distributions is dispersion tending to infinity and one of popular laws for such intervals description is Pareto distribution. Probability density of random variable X in this law can be defined by:

$$f(x) = \frac{\alpha}{k}\left(\frac{k}{x}\right)^{\alpha+1} \tag{3.3}$$

Here $x > k$ and $k > 0$. If $\alpha \leq 2$ then $\sigma^2 \to \infty$ but when $\alpha \leq 1$ both mean and dispersion are tending to infinity.

In the work used by the author it is impossible to strictly emulate Pareto law for query intervals distribution description. However, the author of this research proposes to increase the dispersion and consequently also variation coefficient

$$C = \frac{\sigma^2}{\mu^2} \tag{3.4}$$

where μ is mean of query interval times. In research it was assumed, that mean is limited, but dispersion increases, coming nearer to very great value as it takes place to be in self-similar traffic.

In the considered model work it is believed that in terminal system $\sigma_2^2 \to \infty$ and $\mu_2 = const$. The outgoing stream of such node, and, hence, the incoming stream of server, will be close to self-similar. Input of this system is a Poisson stream with mean service time μ_1 and terminal system load coefficient ψ.

Let's imagine a cyclic system where processing times on terminal i are subordinated to distribution law with mean μ_i and variation coefficient C_i, $i = 1, 2$. System is loop-closed, therefore N is total query quantity in the system, $N = const$. Lets define diffusion process which approximates system mean query number $n_1(t)$ through $x(t)$. Then corresponding diffusion equation will look like this:

$$(\partial/\partial t)\, p(x_0, x; t) = \frac{1}{2}\alpha\left(\partial^2/\partial x^2\right) p(x_0, x; t) - \beta\,(\partial/\partial x)\, p(x_0, x; t) \tag{3.5}$$

Where $\alpha = C_1/\mu_1 + C_2/\mu_2$ and $\beta = 1/\mu_2 - 1/\mu_1$. Solving this equation with boundary conditions $0 \leq x(t) \leq N + 1$ for all $t \geq 0$ use scaling transformation

$$
\begin{aligned}
y &= \frac{x}{|\alpha/\beta|} = \frac{x}{|(C_1 + C_2\rho)/(1-\rho)|} \\
\tau &= \frac{t}{|\alpha/\beta^2|} = \frac{t}{|\mu_1(C_1 + C_2\rho)/(1-\rho)^2|}
\end{aligned}
\tag{3.6}
$$

Where $\rho = \mu_1/\mu_2$. As a result we have coordinate-free diffusion equation:

$$
(\partial/\partial\tau)\, p(y_0, y; \tau) = \frac{1}{2}\left(\partial^2/\partial y^2\right) p(y_0, y; \tau) - \delta\,(\partial/\partial y)\, p(y_0, y; \tau) \tag{3.7}
$$

With two reflecting barriers $y = 0$ and $y = b$:

$$
\frac{1}{2}\,(\partial/\partial y)\, p(y_0, y; \tau) - \delta p(y_0, y; \tau) = 0 \qquad \text{at } y = 0 \text{ and } y = b \tag{3.8}
$$

Where

$$
\begin{aligned}
\delta &= \begin{cases} 1, & \rho < 1 \\ 0, & \rho = 1 \\ -1, & \rho > 1 \end{cases} \\
b &= \frac{N+1}{|(C_1 + C_2\rho)/(1-\rho)|}
\end{aligned}
\tag{3.9}
$$

Applying the method of "eigenfunction expansion", we obtain the following solution for 3.7:

$$
p(y_0, y; \tau) = \begin{cases} 2\delta e^{2\delta y}/(e^{2\delta b} - 1) + exp[\delta(y - y_0 - \delta\tau/2)]\cdot \\ \qquad\qquad \cdot \sum_{n=1}^{\infty} \phi_n(y)\phi_n(y_0) exp(-\lambda_n^2\tau/2), & \text{when } 0 \le y \le b \\ 0, \qquad \text{elsewhere} \end{cases} \tag{3.10}
$$

Where $\phi_n(y)$ and $\phi_n(y_0)$ are eigenfunctions associated with eigenvalues λ_n:

$$
\phi_n(y) = [2\lambda_n^2/b(\lambda_n^2 + 1)]^{\frac{1}{2}}\{\cos\lambda_n y + (\delta/\lambda_n)\sin\lambda_n y\} \tag{3.11}
$$

The first term of 3.10 represents the steady-state probability and the second term gives the transient part in terms of eigenfunction expansion. Note that 3.10 satisfies the initial condition $y = y_0$, i.e. $p(y_o, y; \tau) = \delta(y - y_0)$, since the delta function is expressed in terms of the eigenfunctions. The second term of 3.10 is an infinite series, but can be well approximated by finite terms, since

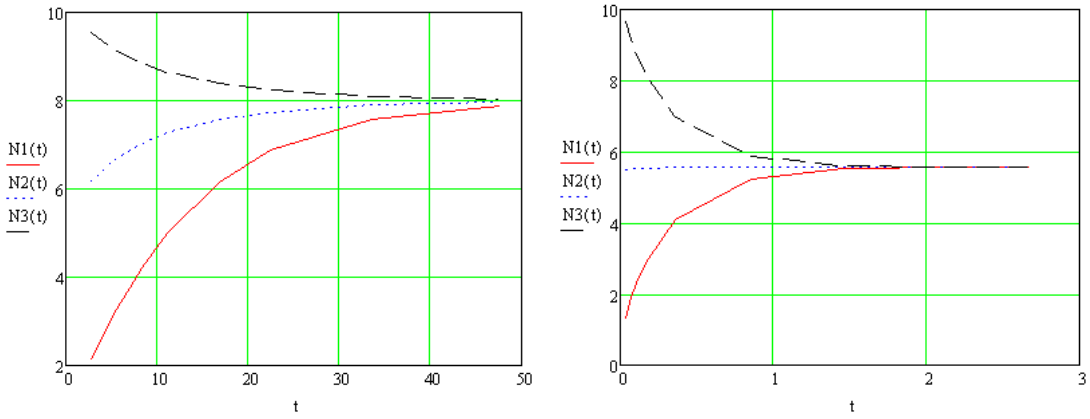
the factor $exp(-\frac{1}{2}\lambda_n^2\tau)$ approaches zero as n increases.

3.2.2 Modeling results

Approximation modeling described above, due to it's implementation complexity, was performed in Mathcad environment [12]. By increasing C_2 from one to one hundred we try to simulate self-similar traffic stream. At the same time we also change server utilization factor $\rho = 0.75, 0.95$ to see how the system behaves under different loads. Number of queries in the system N was set to 10 and remained constant.

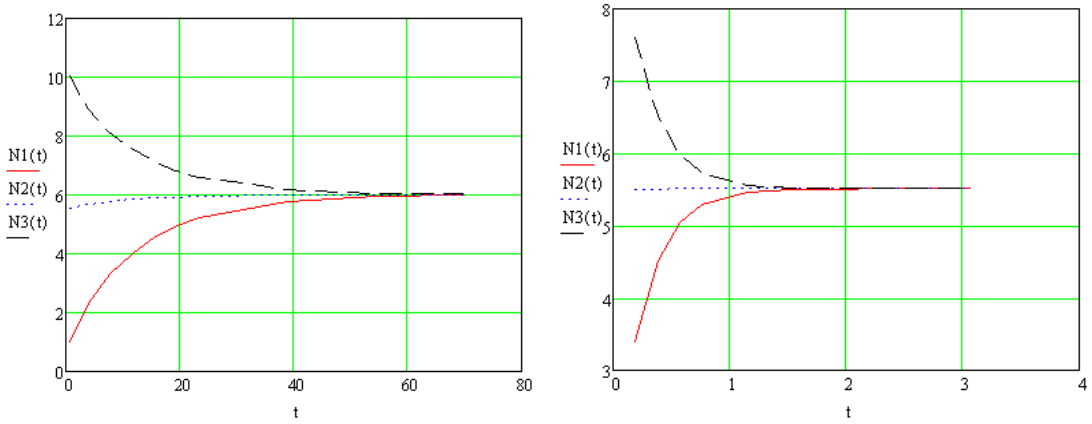
Here we will show only few main figures which will show the tendency of device mean query number. Complete set of figures can be found in Appendix C.

Figure.3.2 and Figure.3.3 show the results of using H.Kobayashi modified model with different initial conditions when initial query number n is zero, 5 and 10, which correspond to $N1(t)$, $N2(t)$ and $N3(t)$ curves respectively.



(a) $C_2 = 1$ (b) $C_2 = 100$

Figure 3.2: Query number in system, $\rho = 0.75$



(a) $C_2 = 1$ (b) $C_2 = 100$

Figure 3.3: Query number in system, $\rho = 0.95$

Tables 3.1, 3.2 and 3.3 show the results of performed approximation. Query number in system n is listed for stationary mode, when it has finished the transition.

C_2	1	5	50	100
n, $\rho = 0.95$	6.013	5.675	5.521	5.511
n, $\rho = 0.75$	7.969	6.533	5.631	5.566

Table 3.1: Query number in system n

C_2	1	5	50	100
P_{rej}, $\rho = 0.95$	0.116	0.099	0.092	0.091
P_{rej}, $\rho = 0.75$	0.258	0.145	0.097	0.094

Table 3.2: denial of service probability P_{rej}

C_2	1	5	50	100
T_{relax}, $\rho = 0.95$	39.0	18.4	2.328	1.152
T_{relax}, $\rho = 0.75$	47.6	18.24	2.464	1.459

Table 3.3: relaxation times T_{relax}

As one can see from the tables and figures adopted Kobayashi model gives very ambiguous results. On one hand we don't know how serving device mean

query number will behave supposing there is self-similar traffic and mean query number n decrease, while variation C_2 increases, should not frighten us. But on the other hand we know that both the mean query number and query loss probability should increase along with utilization factor ρ. Modeling results suppose the opposite. This fact made the author stumble at the considered model application for self-similar traffic modeling too. Also system relaxation time was expected to grow while increasing C_2. As further research will show the doubts were justified and H.Kobayashi model, even modified, cannot be used to model a system with self-similar traffic.

3.3 Simulation

As we've seen in previous section diffusion approximation results are not very convincing. The last resort for transient process study is simulation. In this case simulation involves $M/M/1/K$ and $P/M/1/K$ queuing models implementation, self-similar traffic generation for $P/M/1/K$ case and multiple client access methods simulation.

Current research conclusions are fully backed by the simulation results. Therefore we dedicate a separate chapter to consider this method.

4

Transient process simulation

First we have to simulate $M/M/1/K$ system and compare it with analytical model - they should not differ. If $M/M/1/K$ simulation works flawlessly we can proceed to $P/M/1/K$ simulation, believing it gives trustful results for this kind of queuing system also.

Prior to developing author's own simulation tool few others were considered. Among them was Omnet++, NS3 and other full blown simulation suites. All of them are good and trustworthy tools widely used by universities and other research institutions. But at the same time they require skills, and thus time to acquire them, to be able to control the simulator to get needed results.

However one, rather old but also good tool, provides simple enough scripting language for simulation. That's why it was decided to try it out at the first place.

4.1 GPSS

GPSS (General Purpose Simulation System) was one of the very first simulation systems. It was designed for discrete event modeling in 1960s and was very popular. The popularity of GPSS is due, in part, to its power of expression. A short, easily understood GPSS model would require many lines of coding in other programming language to accomplish a similar goal. GPSS user is

free to concentrate on the important issues in the model being developed since the language itself collects statistics, produces tabulated results and performs a host of mundane tasks one would prefer not to deal with. Even nowadays GPSS is used by college and university students to explore the basics of simulation world. The author took one of the GPSS implementations [8] which allows free usage for students as it's limits are enough to simulate $M/M/1/K$ and $P/M/1/K$ queuing systems.

First of all let's see how GPSS simulation works out and if it's applicable for both $M/M/1$ and $P/M/1$. Figures 4.1 and 4.2 show utilization coefficient changes in time.

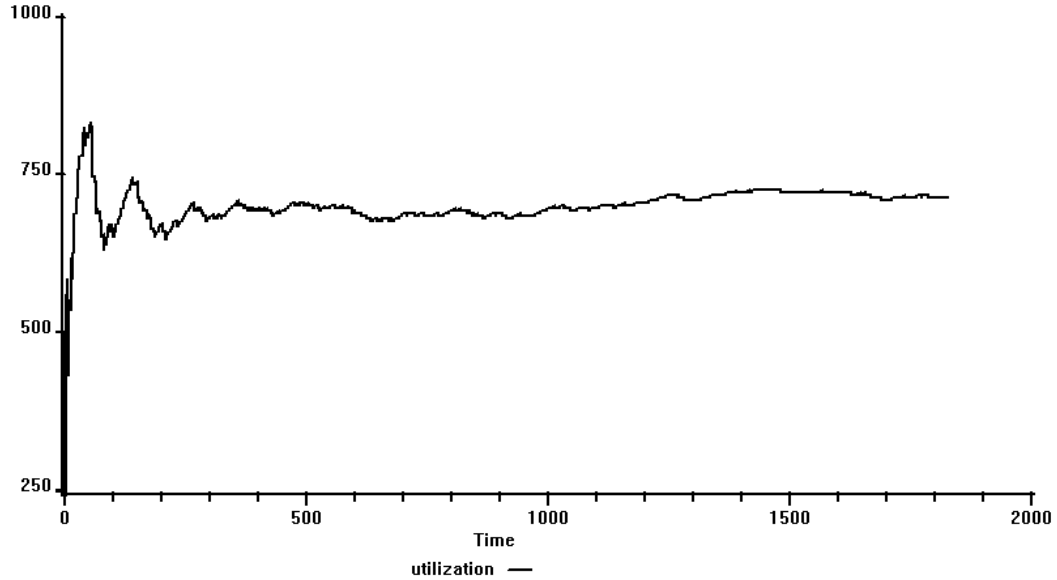


Figure 4.1: $M/M/1$ system utilization, $\lambda = 0.7$, $\mu = 1$

As we can see on Figure 4.1 GPSS gives correct stationary mode utilization result of $\rho \approx 0.7$, which makes us believe that $P/M/1$ simulation can be done using this tool also. We consider more detailed comparison of simulation and analytical expectations in Table 4.1, where $\rho = \lambda/\mu$.

λ	2	1.82	1.58	1.43	1.33	1.17	1.01
ρ, GPSS	0.503	0.554	0.64	0.707	0.757	0.86	0.993
ρ, theory	0.5	0.549	0.633	0.699	0.752	0.855	0.99
error, %	0.6	0.91	1.11	1.14	0.66	0.58	0.30

Table 4.1: GPSS simulation vs. analytical expectations

As one can see from the table above GPSS simulation gives very close results to what one would expect from the theory. But simulation, as seen from the figures already gives one big advantage - it allows to see how system behaves in transitive mode, showing all bursts and anomalies. Now let us see how $P/M/1$ simulation behaves.

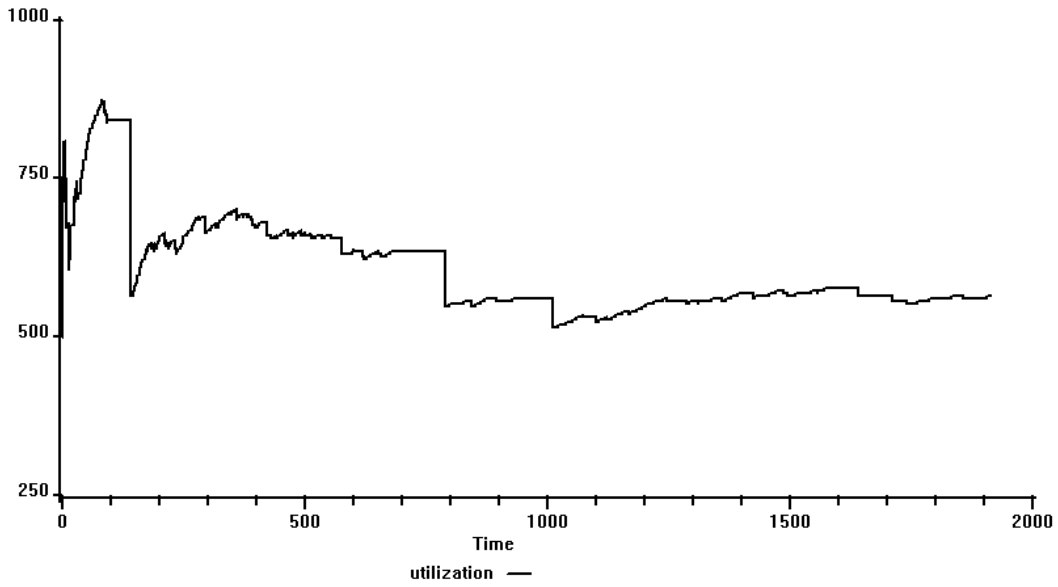


Figure 4.2: $P/M/1$ system utilization, $\lambda = 0.5$, $\mu = 1$

Figure 4.2 shows that even with lower intensity of arrivals $P/M/1$ system utilization behavior is much more complicated, as was expected. This suggests the opposite conclusion from that made by diffusion approximation - relaxation time should grow.

Although GPSS simulation shows impressive results in the sense of very good error rate comparing to the theory and the ability to simulate the system under self-similar traffic load the author stumbled upon the following GPSS tool problems:

- Very poor graphing abilities
- No way to save generated process data for later analysis
- With all respect to developers we don't know how GPSS works internally thus making it impossible to debug the simulation

Despite of the problems described above we have seen that GPSS simulation gives correct results at least for $M/M/1$ system and which completely differ from the diffusion approximation results shown in previous chapter. GPSS

problems and good simulation trial results inspired the author to write it's own simulation program which will be described in the next section.

4.2 Transient process simulation

4.2.1 The model

As already noted we are simulating a pair of open queuing systems with one serving device or server. First one has a query stream with exponentially distributed mean inter-arrival times T_a and exponentially distributed mean service time T_s. In the queuing theory such systems are called $M/M/1$. Second system has self-similar query stream. Self-similarity is achieved with the help of Pareto distribution, i.e. mean query inter-arrival times are Pareto distributed. Mean query service time in the second system is also exponentially distributed and equals to one from the first system - T_s. The author called such a system $P/M/1$.

In order to be able to correctly compare $M/M/1$ and $P/M/1$ simulation results we have to equate mathematical expectations of query inter-arrival times thus obtaining 4.1.

$$T_a = \frac{kx_m}{k-1} \tag{4.1}$$

Where k and x_m are Pareto distribution parameters - shape and location respectively, which are obtained using a given Hurst coefficient H and query arrival intensity $\lambda = 1/T_a$ of $M/M/1$ system. From a well known Hurst parameter formula [100] we can easily obtain k:

$$k = 3 - 2H \tag{4.2}$$

Further, using 4.1 and 4.2 we obtain x_m in 4.3 thus obtaining all the needed parameters to simulate a stream of queries with a given Hurst parameter H.

$$x_m = \frac{kT_a - T_a}{k} = T_a - \frac{T_a}{k} = T_a - \frac{T_a}{3-2H} \tag{4.3}$$

Let us denote the importance of equality of query arrival intensity for simulation results comparison. Also 4.3 can be used only when $k > 1$ and thus $0 \leq H < 1$. However we are interested in H coefficient values at which the incoming traffic will be self-similar, i.e. $0.5 < H < 1$ [100].

Below it will be shown that $M/M/1$ system simulation modeling results and the ones obtained by analytical method are very close, what gives us reason to

rely on $P/M/1$ simulation too, but now let us consider simulation tool more closely.

4.2.2 Simulation tool

During this study a number of programs were written in C, Perl and Bash programming languages. Among them is `mm1_pm1.c` - $M/M/1/K$ and $P/M/1/K$ simulation tool. The basic idea is approximately the same as in GPSS World - the simulation time is advanced by steps, generated arrival and service time lengths T_a and T_s respectively. Depending on current simulation time either the query is added to the queue or is deleted from it. Thus giving us the data about every change in simulated system. See Figure 4.3 for timing example. As we don't have a synchronized simulation clock there is a problem of getting current mean query number which is solved by calculating the area under the queue "curve" and dividing it by elapsed time on every step. Buffer size limits maximum queue size - if the queue size is bigger than the given buffer size then the arrived query is discarded.

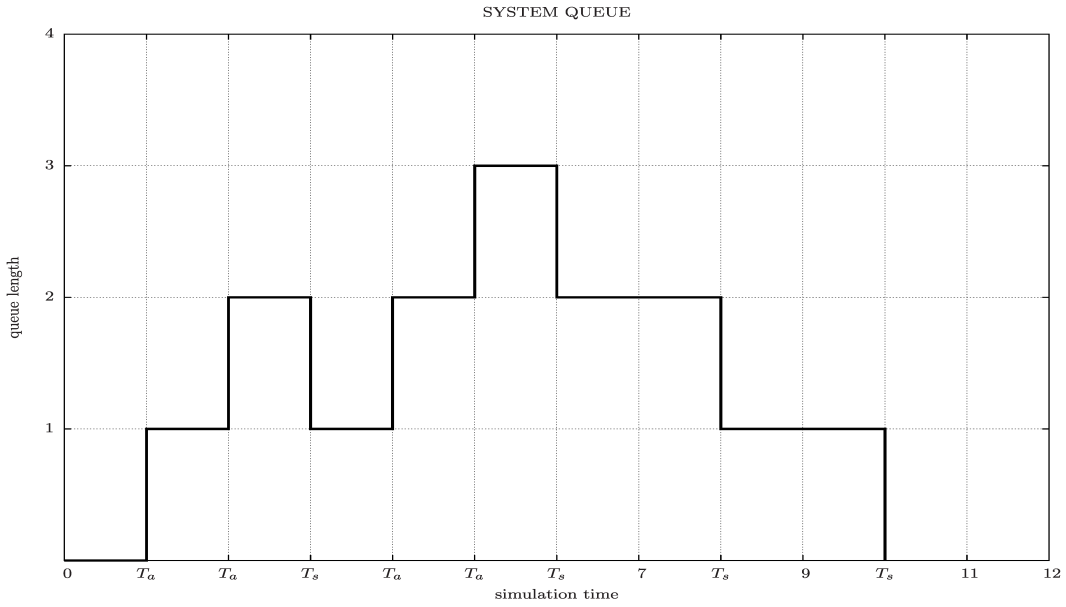


Figure 4.3: Simulated system timing example

The main simulation functionality is represented by the function `int simulate( sim_opt_t *sim, sim_stats_t *stats )` which accepts a pointer to `sim_opt_t` structure (See Listing 4.1) which contains all simulation parameters and pointer to `sim_stats_t` structure (See Listing 4.2) which contains all simulation statistics.

Listing 4.1: Simulation options

```
28 /* Simulation parameters and options */
29 typedef struct sim_opt_t {
30     long unsigned int time;      // simulation time
31     long unsigned int buffer;    // maximum queue size
32     FILE *queue_curr_fp;         // file to save queue data
33     FILE *queue_mean_fp;         // file to save mean queue data
34     FILE *load_fp;               // file to save utilization data
35     double (*arr_cb)( void * ); // callback function for arrivals
36     double (*dep_cb)( void * ); // callback function for departures
37     void *arr_cb_params;         // arrival callback parameters
38     void *dep_cb_params;         // departure callback parameters
39     int clients;                 // number of clients in the system
40 } sim_opt_t;
```

Listing 4.2: Simulation statistics

```
42 /* Simulation statistics */
43 typedef struct sim_stats_t {
44     long unsigned int processed; // served query counter
45     long unsigned int dropped;   // discarded query counter
46     long unsigned int queue;     // queue length counter
47     double queue_mean;           // mean queue length counter
48     double t_busy_tot;           // sustem busy counter
49     double t_sim_tot;            // simulated time counter
50     double t_mean_res;           // mean residence time
51     double load;                 // utilization factor
52     double rate;                 // query service rate
53     double ploss;                // query loss probability
54 } sim_stats_t;
```

Note that for the sake of universality both callback functions accept void pointers thus enabling `simulate()` function user to implement any queuing system with one server but with any arrival and service laws. Please see Listing A.14 lines 452-555 for the `simulate()` function but here let's consider a usage example:

Listing 4.3: simulate() usage example

```
267     /*************************
268      * P/M/1/K simulation part
269      *************************/
270     const char *buffer_pm1_file = "buffer_pm1.out";
271     const char *buffer_mean_pm1_file = "buffer_mean_pm1.out";
272     const char *load_pm1_file = "load_pm1.out";
273
```

```
274     queue_curr_fp = fopen( buffer_pm1_file, ”w” );
275     if ( queue_curr_fp == NULL ) {
276         printf( ”ERROR creating output file (%s)\n”, buffer_pm1_file );
277         exit(EXIT_FAILURE);
278     }
279
280     queue_mean_fp = fopen(buffer_mean_pm1_file, ”w”);
281     if ( queue_mean_fp == NULL ) {
282         printf( ”ERROR creating output file (%s)\n”,
283                         buffer_mean_pm1_file );
284         exit(EXIT_FAILURE);
285     }
286
287     load_fp = fopen( load_pm1_file, ”w” );
288     if ( load_fp == NULL ) {
289         printf( ”ERROR creating output file (%s)\n”, load_pm1_file );
290         exit(EXIT_FAILURE);
291     }
292
293     /* seed to the same random sequence for sane comparision */
294     srand( arguments.seed );
295
296     /*
297      * Calculate shape k and scale(location?) Xm parameters for
298      * Pareto distribution based on mean interarrival time
299      * from M/M/1 simulation and defined Hurst coefficient
300      */
301     double k = 3 - (2 * arguments.hurst);
302     double Xm = (Ta * k - Ta) / k;
303     pareto_params_t pareto_cb_params = { k, Xm };
304
305     sim_stats_t pm1_stats;
306
307     sim_opt_t pm1;
308     pm1.time = arguments.simtime;
309     pm1.buffer = arguments.buffer;
310     pm1.queue_curr_fp = queue_curr_fp;
311     pm1.queue_mean_fp = queue_mean_fp;
312     pm1.load_fp = load_fp;
313     pm1.arr_cb = pareto;
314     pm1.dep_cb = exponential;
315     pm1.arr_cb_params = &pareto_cb_params;
316     pm1.dep_cb_params = &Ts;
317     pm1.clients = arguments.clients;
318
319     simerr = simulate( &pm1, &pm1_stats );
320
```

```
321     fclose(queue_curr_fp);
322     fclose(queue_mean_fp);
323     fclose(load_fp);
324 
325     if (simerr < 0) {
326         fprintf( stdout, "Simulation error\n" );
327         exit(EXIT_FAILURE);
328     }
```

As we see arrivals are configured as Pareto with `pareto()` function (line 313, Listing 4.5) and service law is `exponential()` (line 314, Listing 4.4). Pareto parameters is a structure containing k and x_m parameters considered earlier. Both `pareto()` and `exponential()` use `uniform()` function to pull a random value $0 < u < 1$ (Listing 4.6).

Listing 4.4: exponential()

```
422 double exponential( void *x )
423 {
424     /* Uniform random number from 0 to 1 */
425     double z;
426 
427     uniform( &z );
428 
429     /* log() is natural logarithm */
430     return -(*(double *)x) * log(z);
431 }
```

Listing 4.5: pareto()

```
433 double pareto( void *p )
434 {
435     double z;
436     pareto_params_t *param = (pareto_params_t *)p;
437 
438     uniform( &z );
439 
440     // Generate Pareto rv using the inverse transform sampling
441     // http://en.wikipedia.org/wiki/Pareto_distribution
442     // http://en.wikipedia.org/wiki/Inverse_transform_sampling
443     return param->Xm / pow( z, (1.0 / param->k) );
444 
445     // Generate Pareto rv using Generalized Pareto Law
446     //double mu = Xm;
447     //double sigma = Xm;
448     //double xi = k;
449     //rv = mu + (sigma*(1/pow(z, xi) - 1))/xi;
```

```
450 }
```

Listing 4.6: uniform()

```
405 /* Pull a uniform random value (0 < u < 1) */
406 int uniform( double *u )
407 {
408     *u = 0;
409 
410     // TODO: use something better than just rand()
411     while ((*u == 0) || (*u == 1)) {
412         *u = (double) rand() / RAND_MAX;
413     }
414 
415     return 0;
416 }
```

The `mm1_pm1` program accepts many arguments, allowing to configure the simulation:

```
$ ./mm1_pm1 --help
Usage: mm1_pm1 [OPTION...]

M/M/1/K and P/M/1/K simulation program

-b, --buffer=DOUBLE        Max queue length (default: 1000000)
-c, --clients=INT          Clients in the system (default: 1)
-h, --hurst=FLOAT          Define Hurst coefficient (default: 0.56)
-l, --arrtime=DOUBLE       Define mean interarrival time (default: 10)
-m, --servtime=DOUBLE      Define mean service time (default: 8)
-o, --output=FILE          Output to FILE instead of standard output
-q, --quiet                Be quiet
-s, --seed=INT             Seed value for srand function (default: 8)
-t, --simtime=DOUBLE       Define total simulation time (default: 1500000)
-v, --verbose              Produce verbose output
-?, --help                 Give this help list
    --usage                Give a short usage message
-V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to romans.jerjomins@tet.rtu.lv.
```

If not redefined it outputs all statistics to standard output and in parallel saves all simulation data to defined files:

```
$ ./mm1_pm1
```

```
======================================
   *** M/M/1 simulation results ***
======================================
INPUTS:
  Mean interarrival time = 10.000000 sec
  Mean service time      = 8.000000 sec
======================================
OUTPUTS:
  Total simulated time  = 1500002.9591 sec
  Number of completions = 150937 pkts
  Throughput rate       = 0.100624 pkts/sec
  Server utilization    = 0.807562
  Mean queue length     = 4.222995 pkts
  Mean residence time   = 41.967870 sec
  Buffer misses         = 0 pkts
  Loss probability      = 0.000000
======================================

======================================
   *** P/M/1 simulation results ***
======================================
INPUTS:
  Hurst coefficient      = 0.560000
  Mean service time      = 8.000000 sec
======================================
OUTPUTS:
  k                     = 1.880000
  Xm                    = 4.680851
  Total simulated time  = 1500000.0000 sec
  Number of completions = 149623 pkts
  Throughput rate       = 0.099749 pkts/sec
  Server utilization    = 0.797701
  Mean queue length     = 4.203246 pkts
  Mean residence time   = 42.138369 sec
  Buffer misses         = 0 pkts
  Loss probability      = 0.000000
======================================
```

Note that this and other programs were developed under GNU/Linux operating system and, possibly, will not work as expected under different OS out of the box. The overall simulation program algorithm is reflected on Figure 4.4 below.

Figure 4.4: Simplified simulation algorithm

Full source code for this tool is provided in Listing A.14. Now let us go forward and show the simulation results obtained with described tool developed by the author.

4.3 Simulation modeling results

4.3.1 Simulation vs. analytics

To make sure that our simulation tool works correctly let's first see how the results differ from those obtained by using analytical method and how close mean query number in stationary mode is to the theoretical $\overline{n} = \rho/(1-\rho)$. Figures 4.5 through 4.8 and D.1 through D.3 reflect the results of this test simulation.

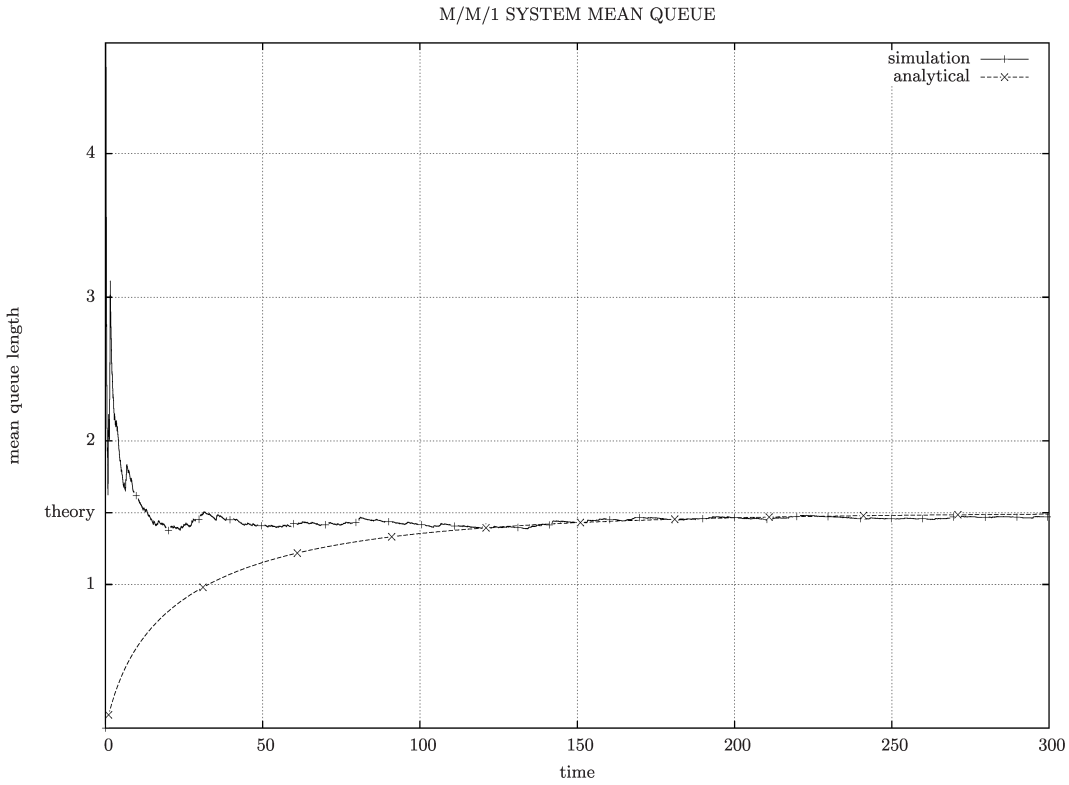


Figure 4.5: $M/M/1$ simulation, $\rho = 0.6$

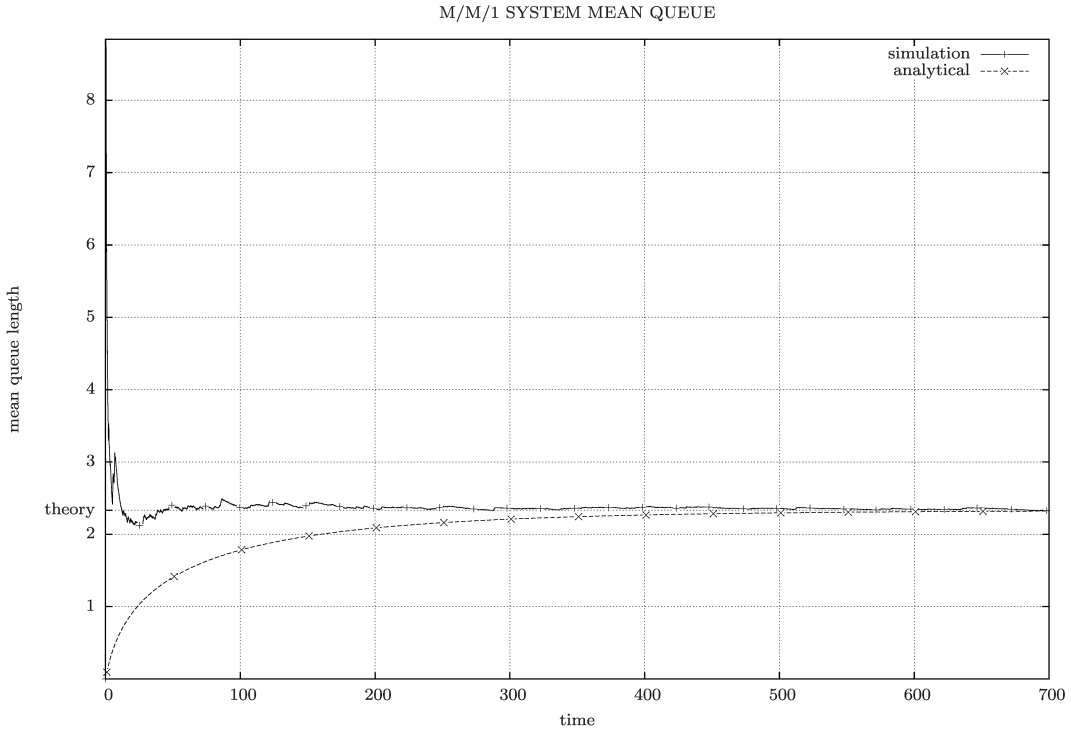


Figure 4.6: $M/M/1$ simulation, $\rho = 0.7$

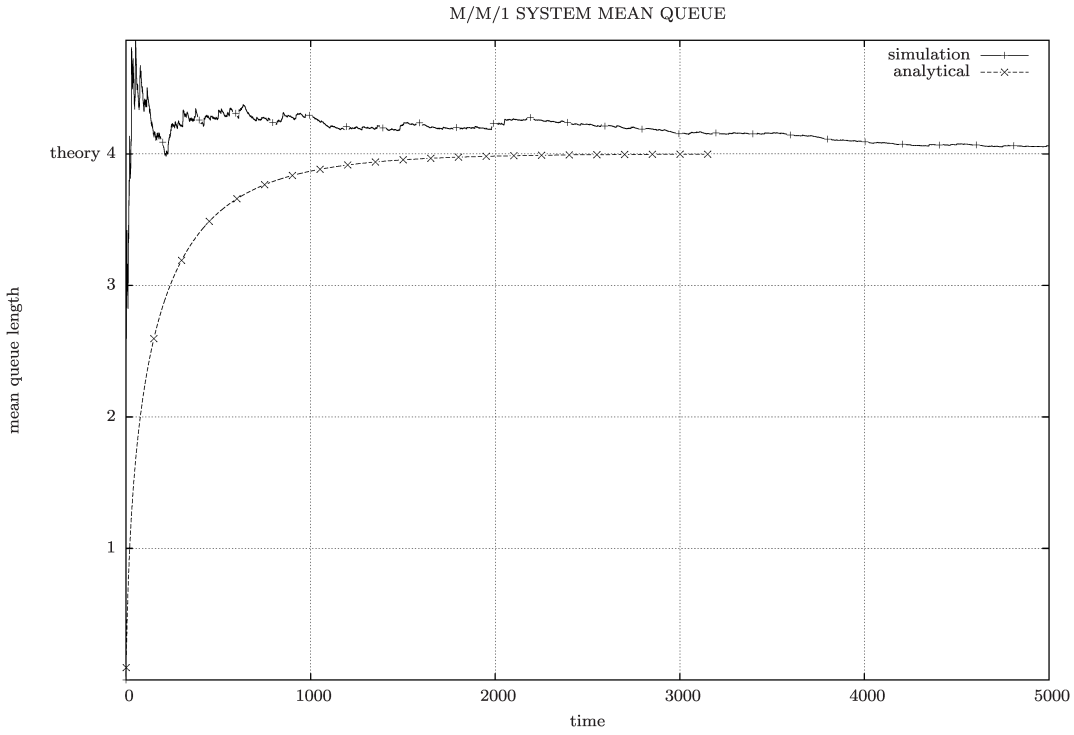


Figure 4.7: $M/M/1$ simulation, $\rho = 0.8$

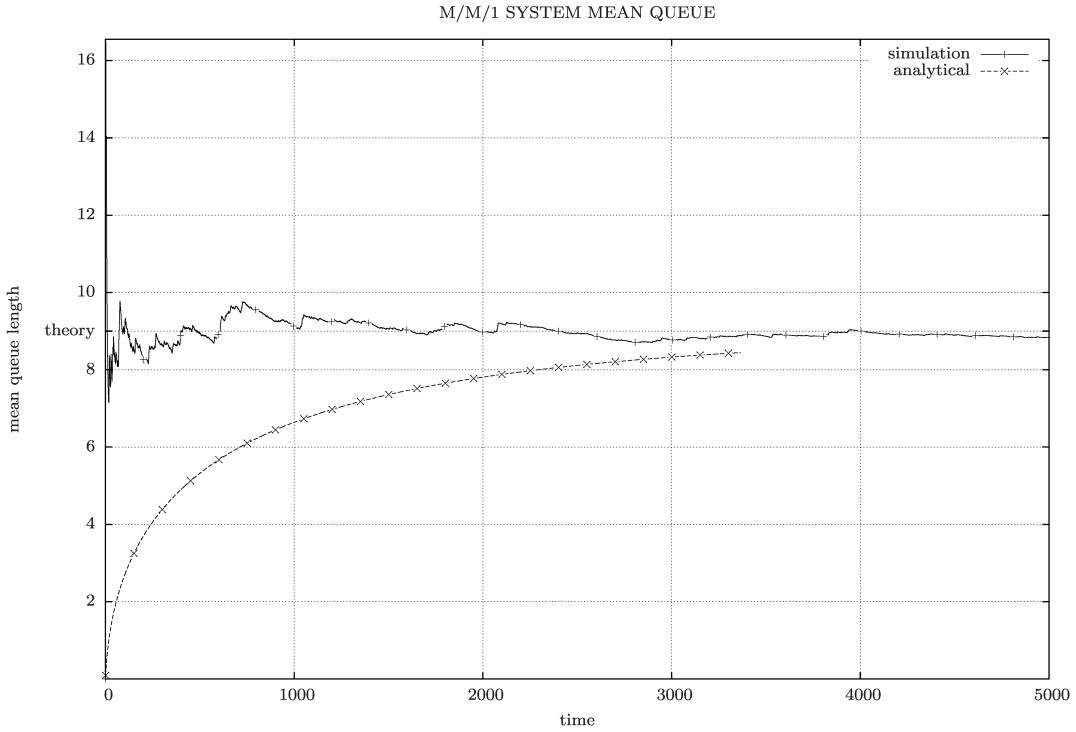


Figure 4.8: $M/M/1$ simulation, $\rho = 0.9$

By eye it seems that simulated resulting mean query number is very close to the theory and relaxation times are close to the analytical method. But let's look at the numbers. Table 4.2 summarizes the results from above figures.

ρ	0.60	0.70	0.75	0.80	0.85	0.90	0.95
$\overline{n}$, theory	1.5	2.333	3	4	5.667	9	19
$\overline{n}$, analytical	1.5	2.333	2.999	3.998	5.623	8.447	13.282
$\overline{n}$, simulation	1.487	2.340	2.962	4.061	5.585	8.898	18.921

Table 4.2: Simulation vs. queuing theory

Although analytical results are exactly the same as expected in theory on lower loads, they quite differ in last two columns. This is due to the calculating errors - Octave functions could not process further and returned error, so the data was cut. On the other hand we see that our simulation tool gives very good output with low error, comparing to the expected theoretical numbers. Sure enough that our simulation tool is working we leave the theoretical expectations and proceed to comparing $M/M/1/K$ and $P/M/1/K$ simulations.

4.3.2 $M/M/1/K$ vs. $P/M/1/K$

Here we already are much closer to our main interest - transient processes in queuing systems and, especially, in the systems with self-similar incoming query stream (in our specific case it will be $P/M/1/K$). Figures 4.9 through 4.12 and E.1 through E.3 reflect the results of this comparison of $M/M/1/K$ and $P/M/1/K$ simulations.

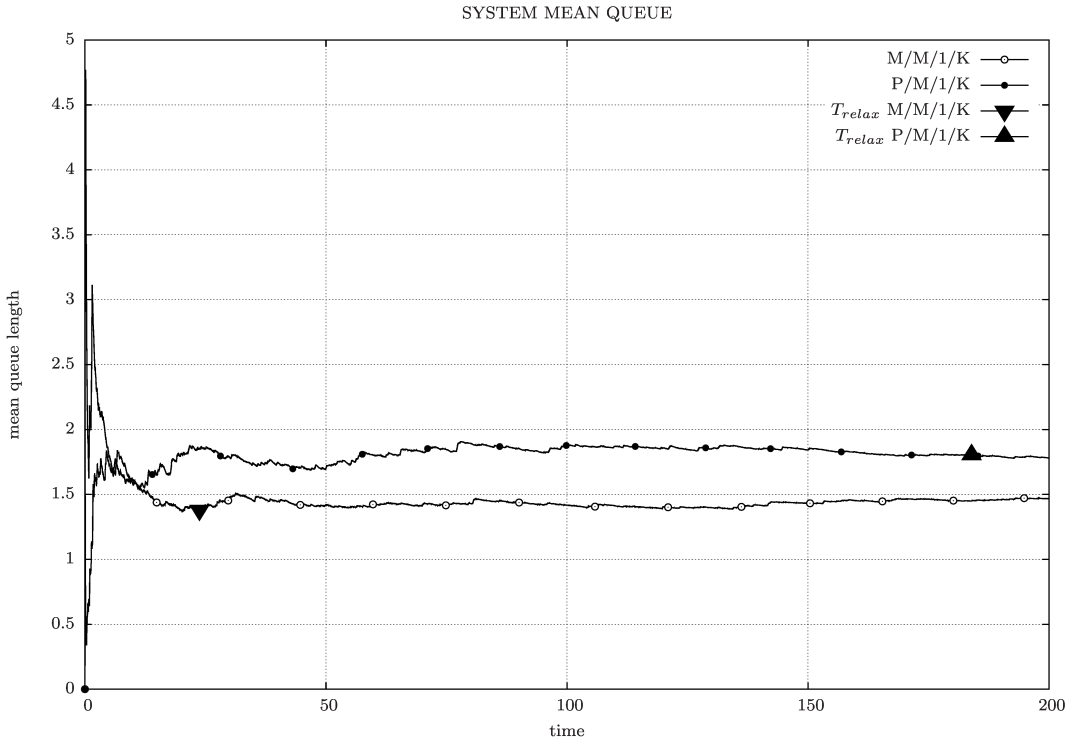


Figure 4.9: $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.6$, $H = 0.7$

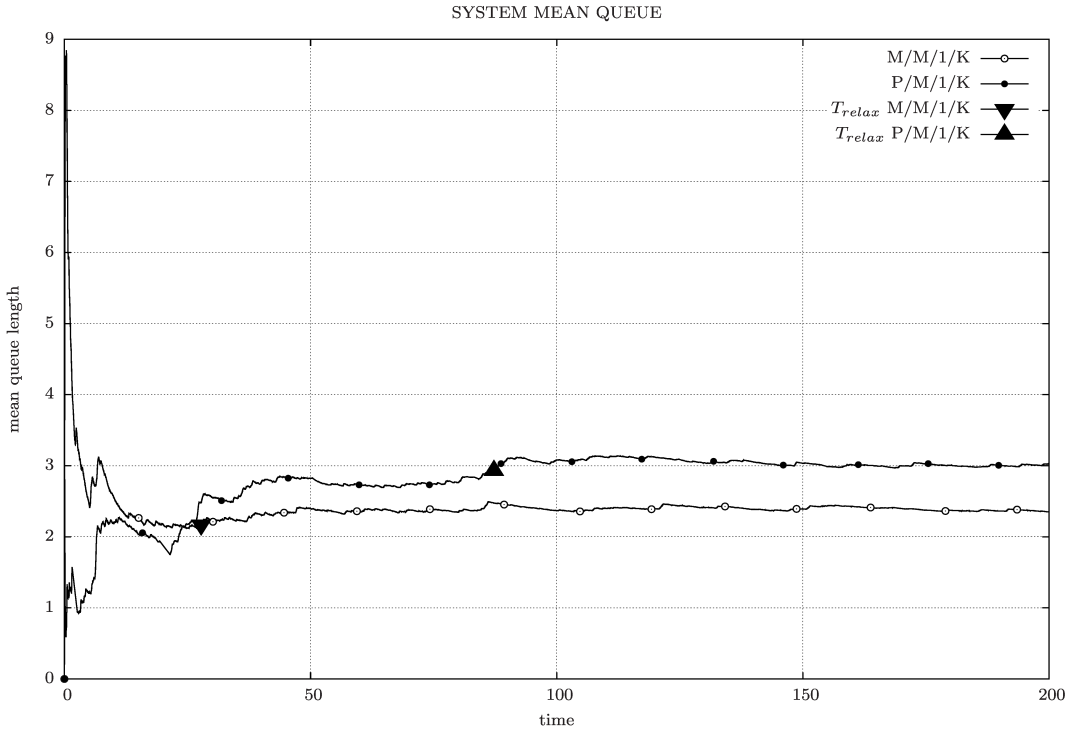


Figure 4.10: $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.7$, $H = 0.7$

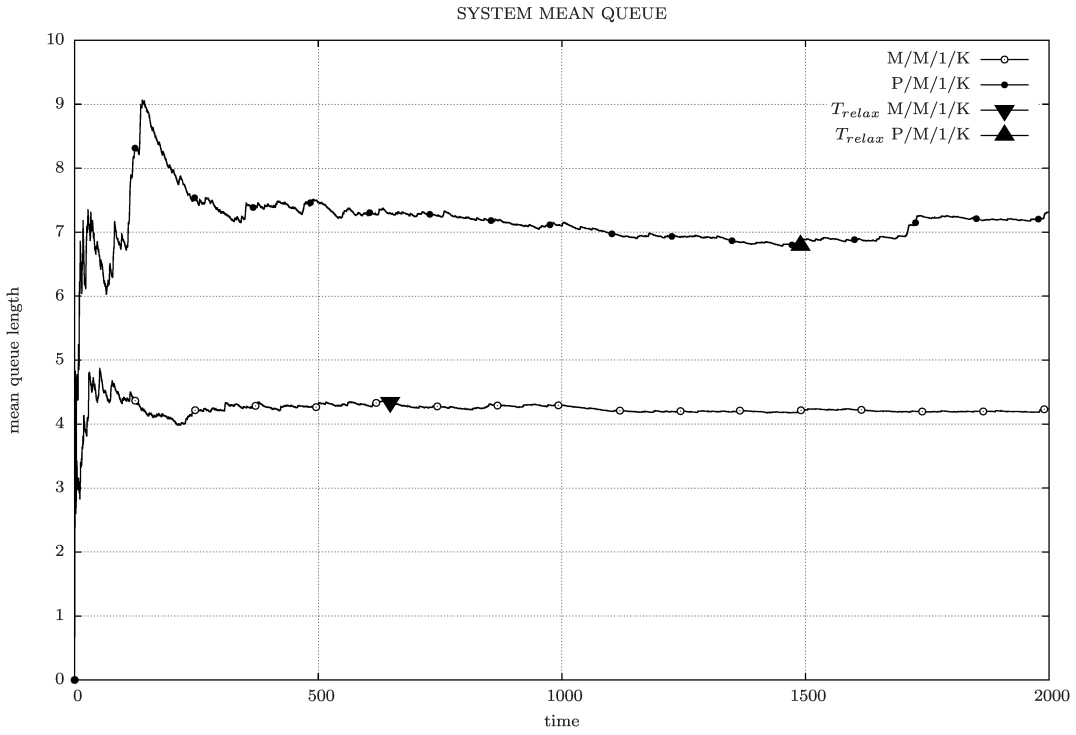


Figure 4.11: $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.8$, $H = 0.7$

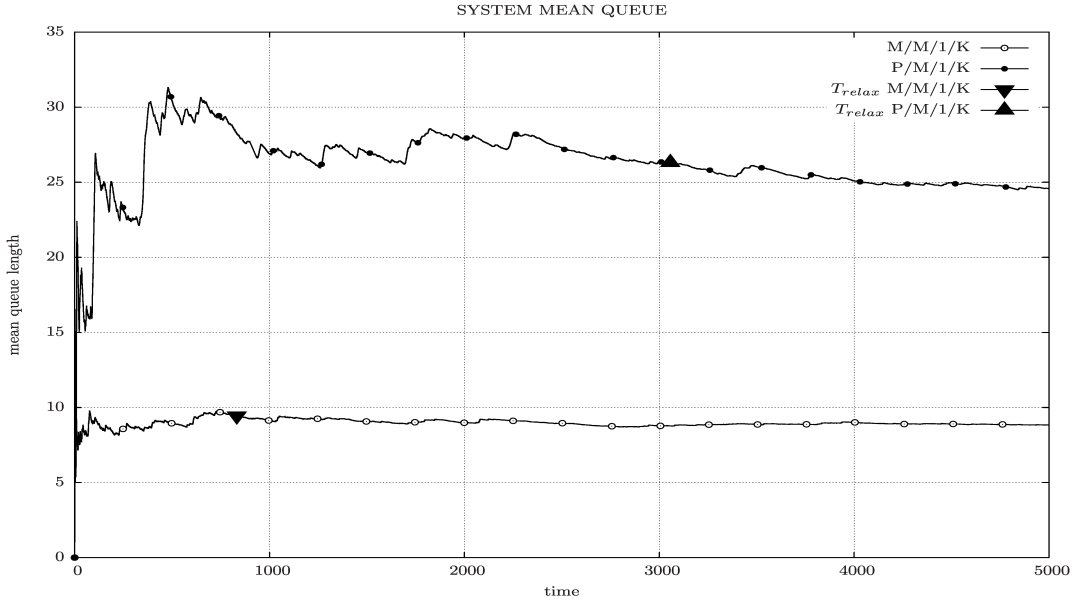


Figure 4.12: $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.9$, $H = 0.7$

Figures above show how $M/M/1/K$ and $P/M/1/K$ systems mean query number behavior differ from each other under various utilization factor ρ. Although we took quite low self-similarity grade $H = 0.7$ the difference is quite noticeable. Tables 4.3 and 4.4 summarize the comparison.

ρ	0.60	0.70	0.75	0.80	0.85	0.90	0.95
$\overline{n}$, $M/M/1/K$	1.487	2.340	2.962	4.061	5.585	8.898	18.921
$\overline{n}$, $P/M/1/K$	1.685	3.151	4.508	7.299	11.672	24.567	77.703

Table 4.3: $M/M/1/K$ vs. $P/M/1/K$ mean query number

ρ	0.60	0.70	0.75	0.80	0.85	0.90	0.95
T_{relax}, $M/M/1/K$	23.75	27.79	134.27	647.64	265.85	832.06	2928.59
T_{relax}, $P/M/1/K$	183.92	87.26	519.55	1489.79	1471.84	3056.63	4286.55

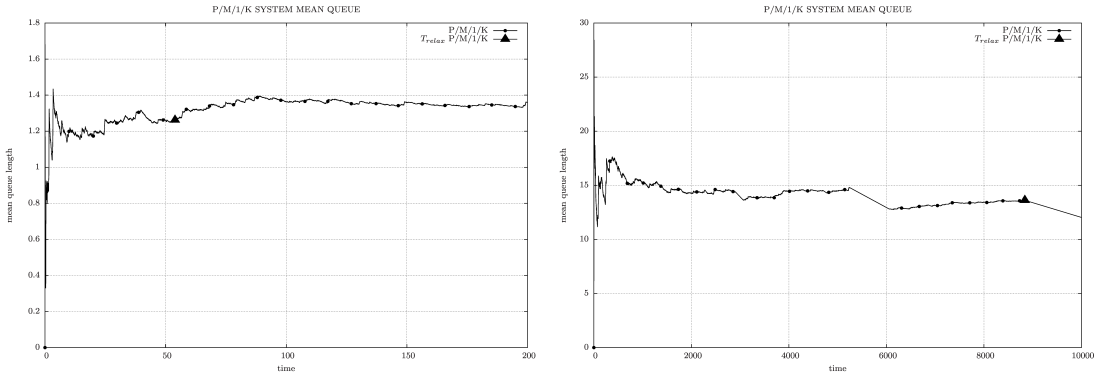
Table 4.4: $M/M/1/K$ vs. $P/M/1/K$ relaxation times

From the tables and figures above we see that both mean query number and relaxation times are much higher for $P/M/1/K$ than for $M/M/1/K$. Here we must note that utilization ρ for Pareto incoming traffic is not quite correct - it differs significantly from $M/M/1/K$ computations, especially when self-similarity parameter H is high. We will consider this fact in the next section.

As it was already known, and we have proved it with current simulation results, $M/M/1/K$ system is legging far behind in the sense of reflecting a system with self-similar incoming traffic, i.e. real wold queuing system. Therefore we switch our attention to simulating $P/M/1/K$ system only.

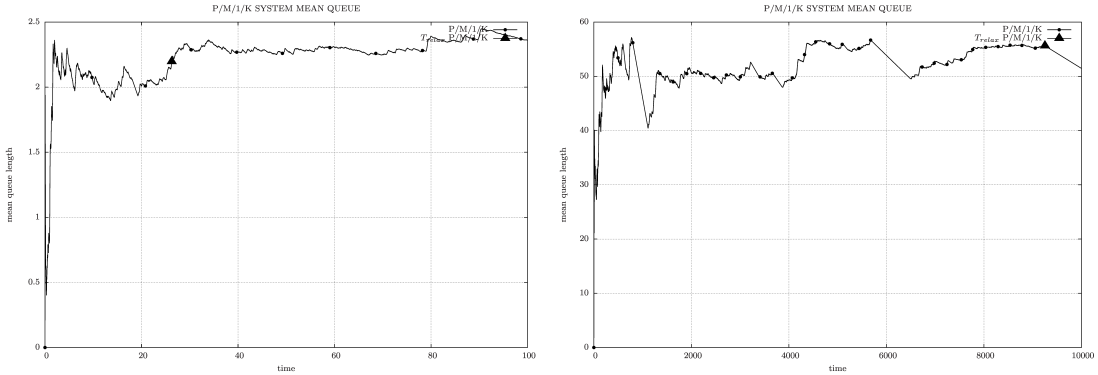
4.3.3 $P/M/1/K$

In this simulation series we change ρ from 0.6 to 0.95 and see what happens if we change self-similarity grade H from 0.6 to 0.9 for each ρ. Here and below we will address to ρ as to a relation of mean service time T_s to mean inter-arrival time T_a, i.e. T_s/T_a or λ/μ, as it would be in a $M/M/1$ system, but the actual utilization factor of $P/M/1$ system, computed during simulation, we will designate as U_p. Figures 4.13 through 4.16 here and F.1 through F.16 in Appendix represent this simulation series results.



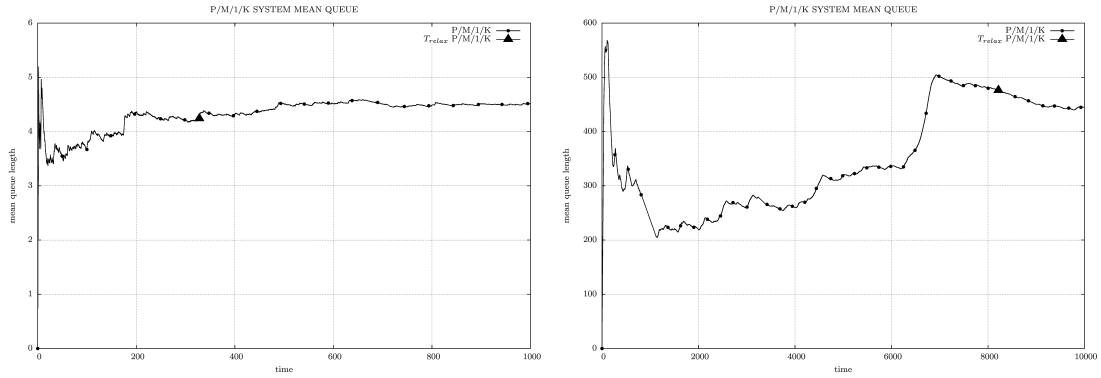
(a) $H = 0.6$ (b) $H = 0.9$

Figure 4.13: $P/M/1/K$ simulation, $\rho = 0.6$



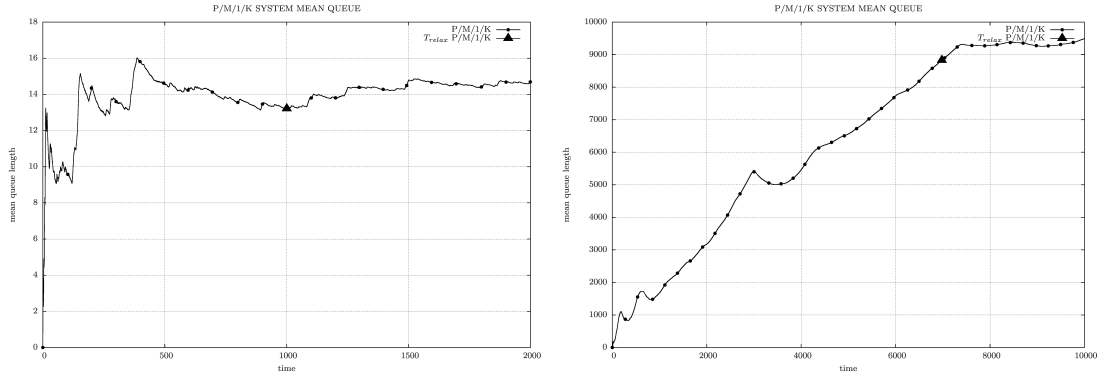
(a) $H = 0.6$ (b) $H = 0.9$

Figure 4.14: $P/M/1/K$ simulation, $\rho = 0.7$



(a) $H = 0.6$ (b) $H = 0.9$

Figure 4.15: $P/M/1/K$ simulation, $\rho = 0.8$



(a) $H = 0.6$ (b) $H = 0.9$

Figure 4.16: $P/M/1/K$ simulation, $\rho = 0.9$

Tables 4.5, 4.6 and 4.7 below summarize the results of these simulation series.

ρ	0.60	0.70	0.75	0.80	0.85	0.90
$H = 0.6$	1.358	2.372	3.194	4.558	7.376	14.203
$H = 0.7$	1.682	3.109	4.526	7.086	11.825	24.598
$H = 0.8$	2.694	5.896	9.493	17.162	33.462	95.155
$H = 0.9$	12.030	51.453	137.776	444.868	1276.517	9489.977

Table 4.5: $P/M/1/K$ mean query number $\overline{n}$ under different parameters

ρ	0.60	0.70	0.75	0.80	0.85	0.90
$H = 0.6$	0.598	0.702	0.750	0.796	0.848	0.901
$H = 0.7$	0.602	0.696	0.754	0.801	0.845	0.885
$H = 0.8$	0.611	0.692	0.742	0.779	0.821	0.868
$H = 0.9$	0.653	0.701	0.694	0.872	0.848	0.992

Table 4.6: $P/M/1/K$ system utilization U_p under different parameters

ρ	0.60	0.70	0.75	0.80	0.85	0.90
$H = 0.6$	53.888	26.272	354.372	327.967	87.822	1000.364
$H = 0.7$	184.353	85.493	436.840	236.371	1480.721	3051.891
$H = 0.8$	113.705	1144.486	1361.760	5586.714	6872.655	8921.949
$H = 0.9$	8838.261	9249.404	8300.975	8215.723	8150.110	6978.514

Table 4.7: $P/M/1/K$ system relaxation time T_{relax} under different parameters

As one would notice there are inadequate numbers in last rows and columns. Also after looking at the corresponding figures it's clear that the system is unstable during all simulated time - it accumulates more and more packets in the buffer and is unable to process them. This is quite typical situation and this is why limited buffers are used in all network equipment. Let's try to introduce a limited buffer to our simulation. As an example we will take Linux kernel [10] buffer sizes. Packet queue buffer size varies from driver to driver but in average the buffers are about 100 packets. We also take only $H = 0.8$ and $H = 0.9$ columns for this simulation as most affected.

The results are combined in Tables 4.8 through 4.11 and figures 4.17 and 4.18 show two most affected systems (for others please see Appendix F).

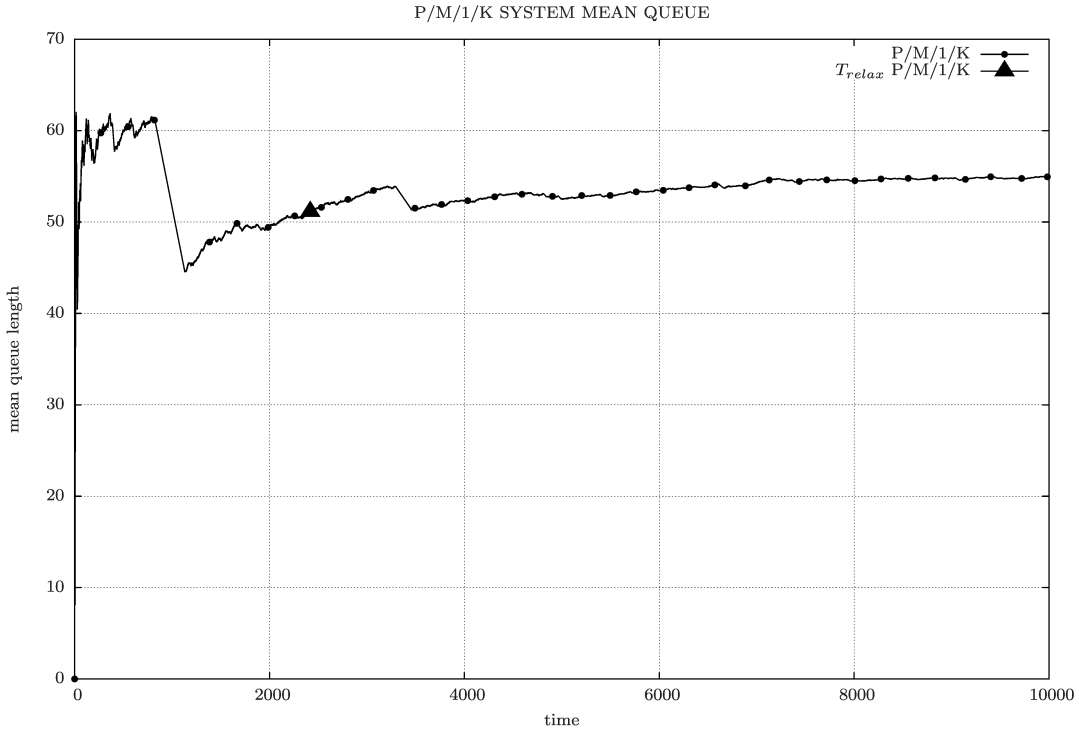


Figure 4.17: $P/M/1/K$ simulation, $\rho = 0.85$, $H = 0.9, K = 100$

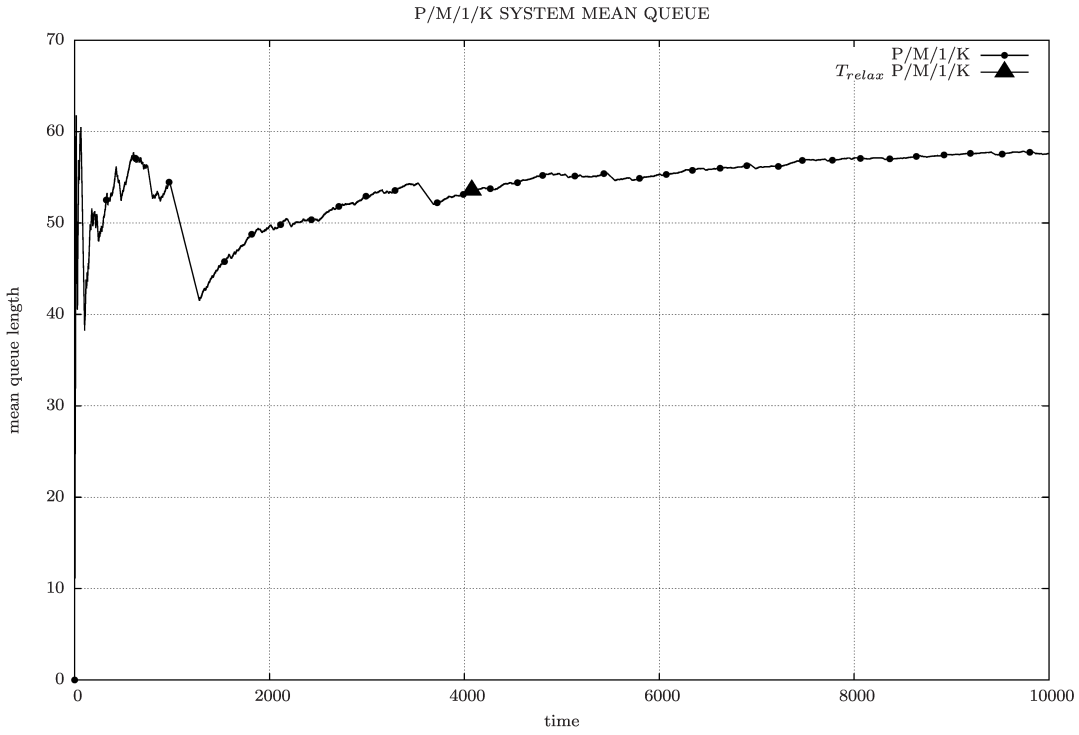


Figure 4.18: $P/M/1/K$ simulation, $\rho = 0.9$, $H = 0.9, K = 100$

ρ	0.60	0.70	0.75	0.80	0.85	0.90
$H = 0.8$	2.694	5.944	9.411	16.375	27.296	42.345
$H = 0.9$	12.573	28.601	36.518	49.335	54.963	57.590

Table 4.8: $P/M/1/K$ mean query number $\overline{n}$, $K = 100$

ρ	0.60	0.70	0.75	0.80	0.85	0.90
$H = 0.8$	0.611	0.692	0.742	0.779	0.823	0.878
$H = 0.9$	0.576	0.624	0.794	0.782	0.806	0.796

Table 4.9: $P/M/1/K$ system utilization U_p, $K = 100$

ρ	0.60	0.70	0.75	0.80	0.85	0.90
$H = 0.8$	113.705	1183.656	1319.189	1949.432	3484.568	1343.704
$H = 0.9$	7920.085	8470.755	8560.601	4176.040	2417.903	4075.193

Table 4.10: $P/M/1/K$ system relaxation time T_{relax}, $K = 100$

ρ	0.60	0.70	0.75	0.80	0.85	0.90
$H = 0.8$	0	0	0	0.001	0.006	0.024
$H = 0.9$	0.002	0.028	0.063	0.098	0.135	0.176

Table 4.11: $P/M/1/K$ packet loss probability P_{loss}, $K = 100$

Above figures and tables show significant improvement in system relaxation times and mean query number (e.g. compare Tables 4.7 and 4.10) but this was a trade-off - up to 17% packet loss appeared in considered systems. It is obvious that increasing packet buffer size will help to lower packet loss probability but it is always a compromise between the memory size and the packet loss probability especially in resource constrained and embedded systems where RAM size is very limited.

Nevertheless increasing the buffer size is not always a good deed. J.Gettys quite recently discovered a problem now called “bufferbloat” [42]. The problem lies in extensive buffering on slow and high latency lines such as DSL. It completely breaks TCP congestion control if a router or switch discards large buffer causing TCP retransmissions and, as a result, much lower overall speed

and higher latency. This is one more problem a solution to which could be a dynamic buffer management using the same techniques as MBAC systems [68]. See also [1] for reference on bufferbloat problem.

4.4 Analysis of obtained results

The previous section shows the results of our simulation but that's not all. The simulation parameters spectrum was much broader and in this section we will try to analyze them.

4.4.1 Relaxation time

As we have stated earlier we are interested in transient process relaxation time T_{relax} which main known parameter is self-similarity degree H. The data we have after conducting about 1200 simulations allows us to construct the graphs of relaxation time T_{relax} dependence on self-similarity parameter H.

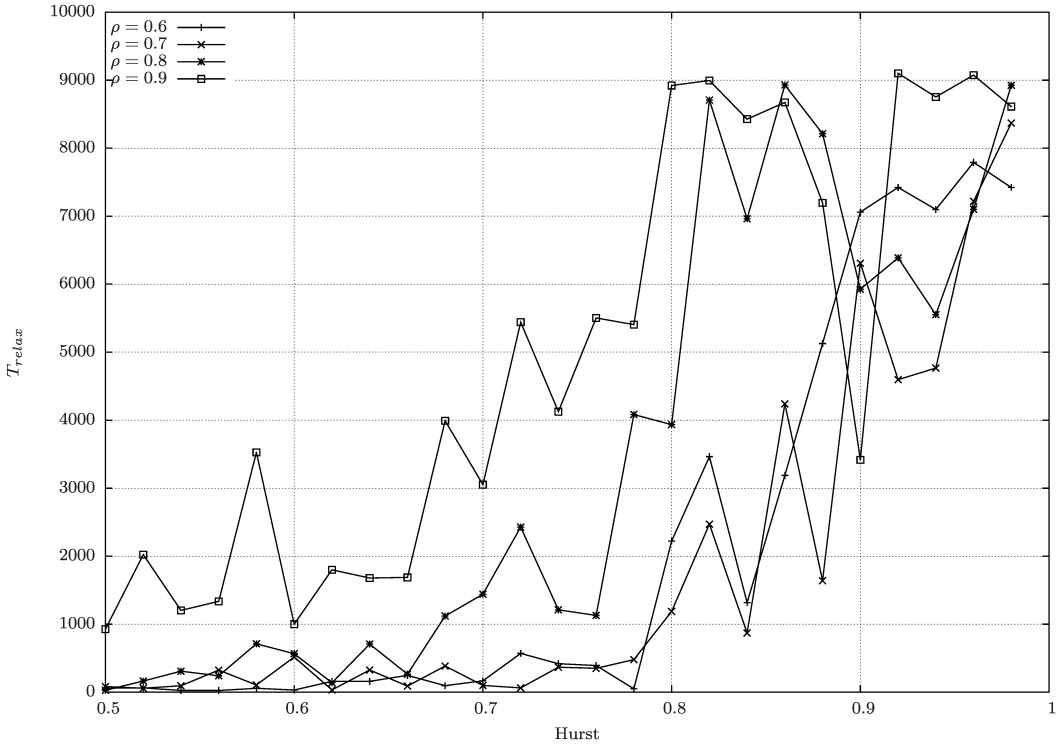


Figure 4.19: T_{relax} dependence on H, different ρ

Figure 4.19 shows how simulated system relaxation time grows along with self-similarity degree. Although having a real world or simulation data is good

there is always a certain interest in approximation this data. Having analytical estimation can ease the further use of these results. By analytical estimation we mean finding an equation which will fit best to our simulation results. Such estimation could be called regression or non linear fit. We performed this analysis using Gnuplot software which gives the ability to define any formula for experimental data fitting. As there are countless number of different formulas to try it was decided to stop on several simple well known formulas which would allow fast and easy computation of relaxation time. The chose fell on linear as the most simple, exponential, logarithmic and Verhulst equations. The later was chosen because it was determined by eye that it could probably fit well our simulation data. Goodness of fit was determined by smallest chi square parameter which basically is squared sum of differences between the formula and the data allowing to see how close the formula approximates the data. Comparative results of chi square values we have put in 4.12.

ρ	0.6	0.7	0.8	0.9
Linear, $f(x) = ax + b$	$1.2e+08$	$8.0e+07$	$2.3e+07$	$5.3e+07$
Exponential, $f(x) = a \cdot e^{-bx} + c$	$3.4e+07$	$2.4e+07$	$2.4e+07$	$6.0e+07$
Logarithmic, $f(x) = a + b \cdot ln(x)$	$2.9e+08$	$3.8e+08$	$2.8e+08$	$1.4e+08$
Verhulst, $f(x) = \frac{ab \cdot e^{cx}}{a+b(\exp(cx)-1)}$	$8.7e+06$	$4.7e+06$	$1.9e+07$	$3.3e+07$

Table 4.12: $T_{relax}(H)$ regression errors

From Table 4.12 we see that Verhulst equation gives the closest approximation to out simulation data. Here a, b and c are equation parameters which were estimated. Verhulst curve can adopt very close to our data in most cases what is seen on Figures 4.20 through 4.23 below.

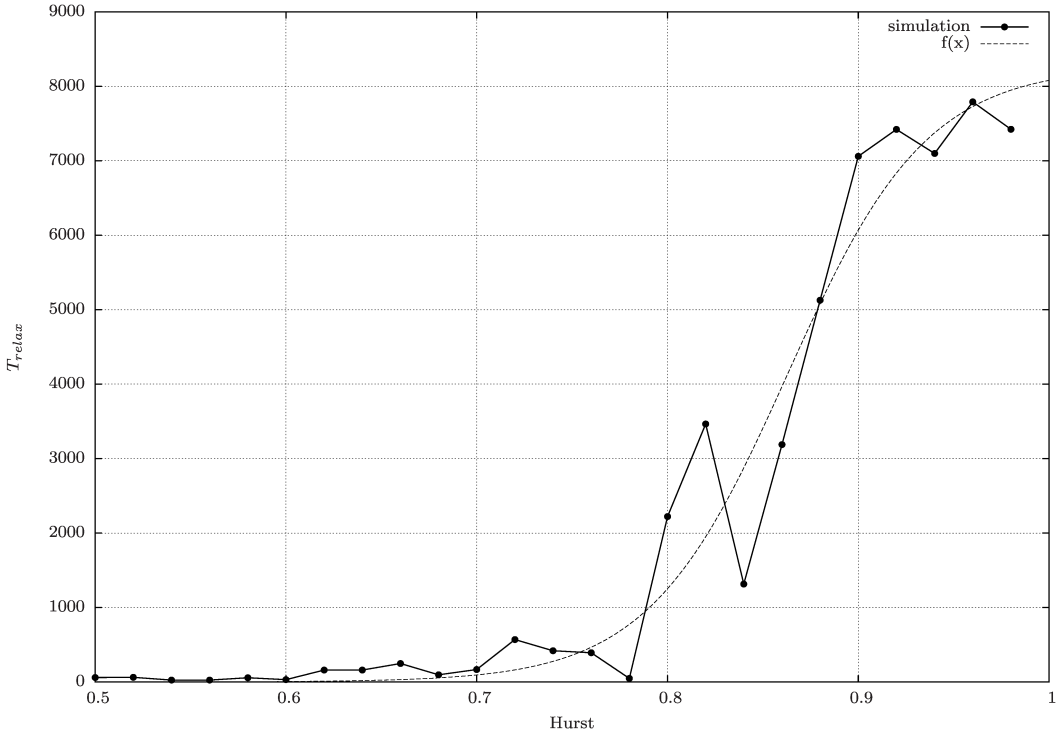


Figure 4.20: $T_{relax}(H)$ Verhulst regression, $\rho = 0.6$

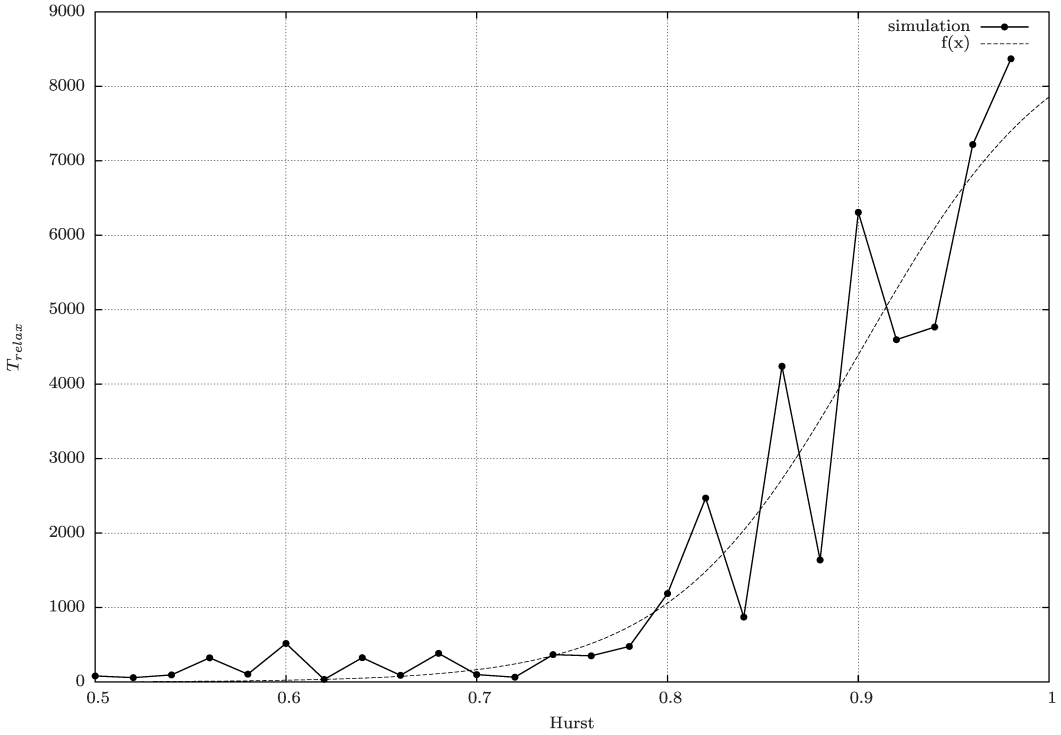


Figure 4.21: $T_{relax}(H)$ Verhulst regression, $\rho = 0.7$

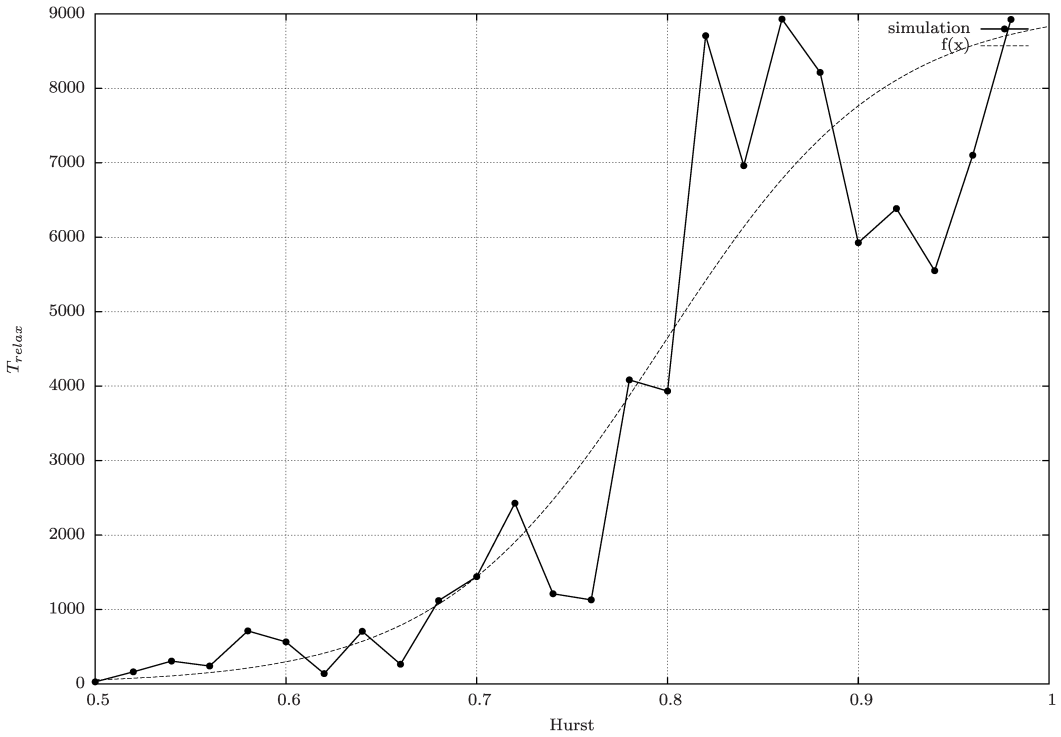


Figure 4.22: $T_{relax}(H)$ Verhulst regression, $\rho = 0.8$

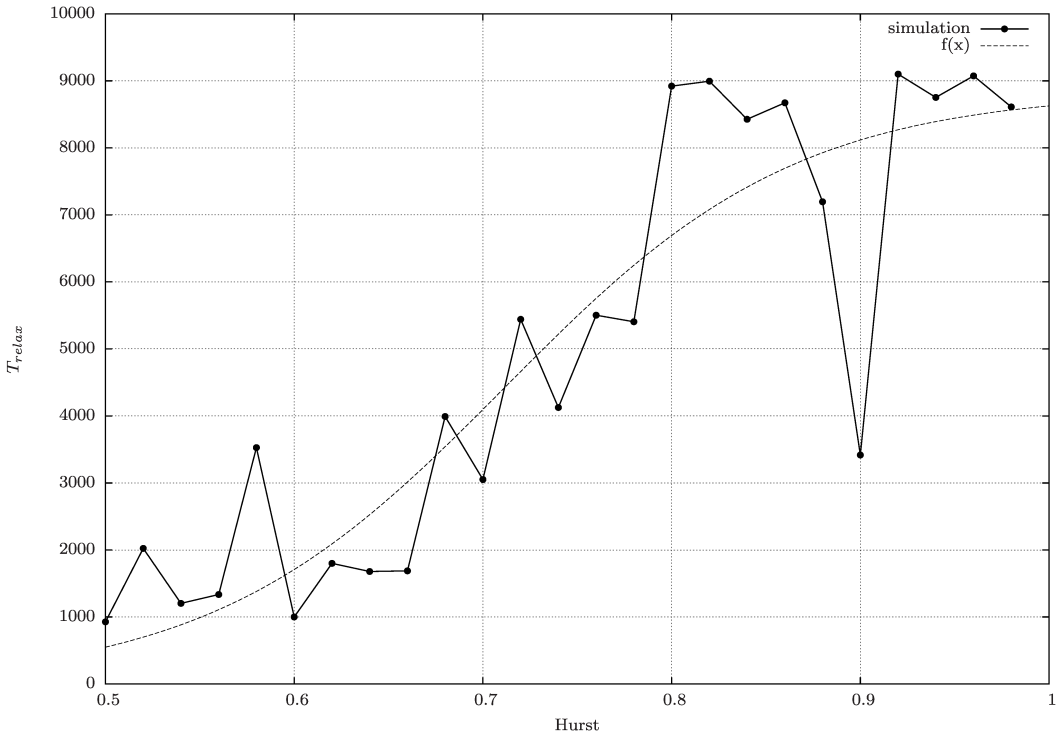


Figure 4.23: $T_{relax}(H)$ Verhulst regression, $\rho = 0.9$

Estimated parameters of Verhulst equation for above figures are given in Table 4.13:

ρ	0.6	0.7	0.8	0.9
a	8270.48 +/- 634.6	9015.42 +/- 2311	9105.39 +/- 1176	8838.93 +/- 998.8
b	4.53e-07 +/- 2.16e-06	0.00018 +/- 0.0009	0.0105 +/- 0.0419	0.96 +/- 2.46
c	27.38 +/- 5.65	19.66 +/- 6.19	17.14 +/- 5.229	12.83 +/- 3.80

Table 4.13: $T_{relax}(H)$ regression estimated parameters

Although Table 4.13 gives us estimated parameters for the fitting formula these are not a common solution to the problem. Finding common solution would involve solving a system of three differential equations for each ρ under the test and then finding relations and correlations between a, b and c of each ρ. Solving this task would probably result in doctoral thesis in mathematics and is actually out of scope of this work. Nevertheless one could use the tables of already calculated parameters for practical purposes.

4.4.2 Self-similar traffic generation

While the above analysis results could be used in an MBAC system the author have found another very interesting property of Verhulst equation which could be used for traffic generation.

Let us go a little back. Verhulst equation 4.6 we used earlier (also called logistic function) is a common solution to differential equation 4.4 used by P-F.Verhulst himself for population growth modeling and, later, by R.May and M.Feigenbaum.

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right) \tag{4.4}$$

The where P is population size, r is growth rate, K is capacity and t is time. Dividing both sides of the equation by K and setting $x = P/K$ gives the differential equation:

$$\frac{dx}{dt} = rx(1 - x) \tag{4.5}$$

Common solution to this equation with P_0 being the initial population is:

$$P(t) = \frac{KP_0e^{rt}}{K + P_0(e^{rt} - 1)} \tag{4.6}$$

There is also a discrete representation of 4.5 called logistic map:

$$x_{n+1} = rx_n(1 - x_n) \tag{4.7}$$

R.May [79] and, later, M.Feigenbaum [35] have studied 4.7 in their work and discovered such interesting property as bifurcation. Plotting many generations of x against r gives us bifurcation diagram. This diagram is also called Feigenbaum fractal (or tree) because each next bifurcation point graphically looks like previous only in smaller scale and plotted resembles a tree.

So, it appears that a system, serving self-similar (or it also could be called fractal) traffic, relaxation time can be well approximated by logistic curve on one hand and the same differential equation produces recurrent relation, which produces one of the most well known fractal, on the other hand. Thus the author expected that some useful results could be obtained using 4.7 for self-similar time sequence generation, which could be used for traffic generation and, further, for wireless network systems relaxation time modeling.

First let's see if fractal orbits can give some self-similarity. By orbit author mean recording at least 100 thousands of x value iterations with fixed r and given x_0 as initial condition. In [35, 79] it was stated that x destabilizes starting from $r = 3.57$. So, we took $r = 3.50$ as first orbit and proceeded till $r = 3.99$ with step 0.01, thus making 50 tests. Unfortunately none of the test gave self-similar sequence regardless of initial value of $0.1 < x_0 < 0.5$. The highest achieved Hurst parameter was around $H = 0.53$ at $r = 3.96$, while for self-similar sequence it should be $0.5 < H < 1$ and as we have already seen in Chapter 2 for real wireless network traffic H is higher than 0.9.

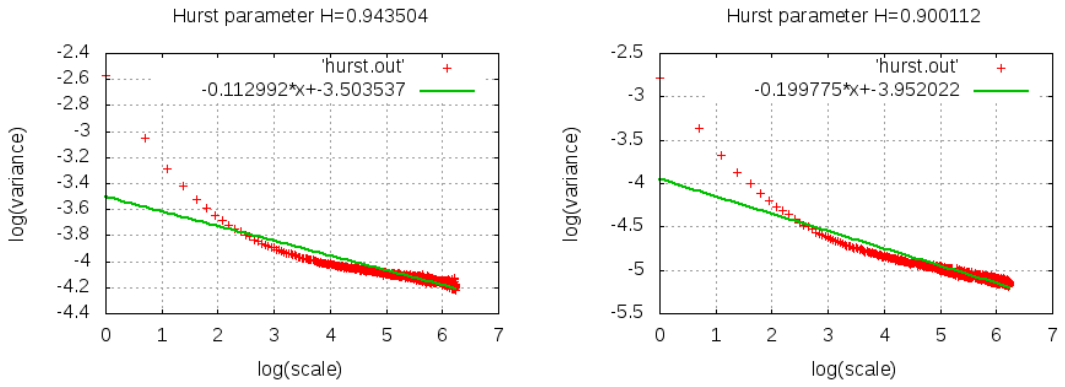
Nevertheless the author tried to use 4.7 in another way - track a given iteration of x while changing r. This method gave very interesting results. The author have performed a set of 16575 tests with dynamic diapason of r starting from $3.50 < r < 3.88$ and ending at $r = 4$ and tracking from 15th to 100th iteration of x, thus conducting 3315 tests for each given $0.1 < x_0 < 0.5$. An overview of results we have put in Table 4.14. which represents a number of obtained self-similar sequences and obtained Hurst parameter values where both variance-time and R/S test methods gave similar results.

Hurst parameter H	Initial x_0				
	0.1	0.2	0.3	0.4	0.5
0.5 - 0.59	0	2	1	1	0
0.6 - 0.69	134	106	179	142	21
0.7 - 0.79	433	374	743	427	316
0.8 - 0.89	1568	1099	1092	1340	586
0.9 - 0.99	904	1319	776	1090	2206

Table 4.14: Generator test overview

From Table 4.14 is clearly seen that more than 70% from overall tests produce highly self-similar sequence and more than 50% from those have Hurst coefficient $H > 0.9$. Almost non of the x iteration tracing method tests gave Hurst parameter $H < 0.6$ what author considers as a very good result.

Figure 4.24 show variance-time plots for estimated Hurst parameter for tracked 90th iteration of x and r starting from 3.6 with initial conditions $x_0 = 0.1$ and $x_0 = 0.5$ correspondingly.



(a) $x_0 = 0.1$, $H = 0.94$ (b) $x_0 = 0.5$, $H = 0.90$

Figure 4.24: Estimated H for 90th iteration of x, $r = 3.6$

It seems natural to the author to generate self-similarity from something that is already self-similar, or fractal, in it's base. Though it's still unclear how to control Hurst parameter to get predicted results directly from using 4.7 but at least already tested parameters can be used to generate self-similar traffic and use it for wireless and other network modeling.

5
Conclusions

5.1 Research summary and contributions

During the work defined objectives were achieved and the following points summarize author's research results:

- Wireless network traffic was collected, analyzed and proved to have self-similar properties with high Hurst coefficient H
- $M/M/1/K$ system relaxation time increases along with system utilization
- H.Kobayashi model, even modified by the author, cannot be used to model a system with self-similar traffic
- $P/M/1/K$ system relaxation time T_{relax} and mean query number $\overline{n}$ is growing along with self-similarity degree H and utilization ρ
- $P/M/1/K$ system relaxation time T_{relax} can be approximated with Verhulst function
- It is possible to generate self-similar time sequence from Verhulst equation

Every point was discussed in international conferences and highlighted in author's nine publications [51, 60, 61, 111–115, 134]

5.2 Possible practical use

Practical use of the current work results could involve:

- System queue size prediction depending on traffic self-similarity degree and system utilization
- System transient mode duration prediction depending on traffic parameters and system utilization
- Networked device development - load planning, device performance parameters planning, memory and queue sizes
- Network planning
- Self-similar network traffic simulation

A
Code listings

Listing A.1: Data capturing

```
#!/usr/bin/perl -W

use Net::SNMP;

my $ip = $ARGV[0];
my $number = $ARGV[1];
if ($ARGV[2]) {
    $community = $ARGV[2];
} else {
    $community = "mrtg"; #default community
}

# Get all the values for current time
($Second, $Minute, $Hour, $Day, $Month, $Year, $WeekDay, $DayOfYear, $IsDST
    )
                              = localtime(time);
$Month = $Month + 1; #this is the true month
if ( $Month < 10 ) {
    $Month = "0".$Month;
}
if ( $Day < 10 ) {
    $Day = "0".$Day;
```

```
22 }
23 if ( $Hour < 10 ) {
24     $Hour = "0".$Hour;
25 }
26 if ( $Minute < 10 ) {
27     $Minute = "0".$Minute;
28 }
29 $d = $Day.$Month."_".$Hour.$Minute;
30
31 my $pout = $ip."_out_".$d.".pkts";
32 my $pin = $ip."_in_".$d.".pkts";
33 open POUT, ">$pout";
34 open PIN, ">$pin";
35
36 ($session,$error) = Net::SNMP->session(Hostname => $ip, Community =>
       $community);
37 die "session error: $error" unless ($session);
38
39 $oid_pkts_out = ".1.3.6.1.2.1.2.2.1.17.";
40 $oid_pkts_in = ".1.3.6.1.2.1.2.2.1.11.";
41
42 $pkts_out1 = get_packets_out();
43 $pkts_in1 = get_packets_in();
44
45 while (1) {
46     sleep 1;
47
48     $pkts_out2 = get_packets_out();
49     $pkts_in2 = get_packets_in();
50
51     if (($pkts_out2 > 0) && ($pkts_out1>0)) {
52         $speed_out = $pkts_out2 - $pkts_out1;
53         $pkts_out1 = $pkts_out2;
54         print POUT $speed_out."\n" ;
55         $pkts_out2 = -1;
56     } else {
57         print POUT "-1\n" ;
58         $pkts_out1 = -1;
59         $pkts_out2 = -1;
60         $pkts_out1 = get_packets_out();
61     }
62
63     if (($pkts_in2 > 0) && ($pkts_in1 > 0)){
64         $speed_in = $pkts_in2 - $pkts_in1;
65         $pkts_in1 = $pkts_in2;
66         print PIN $speed_in."\n" ;
67         $pkts_in2 = -1;
```

```
68      } else {
69          print PIN "-1\n" ;
70          $pkts_in1 = -1;
71          $pkts_in2 = -1;
72          $pkts_in1 = get_packets_in();
73      }
74
75      close POUT;
76      close PIN;
77      open POUT, ">>$pout";
78      open PIN, ">>$pin";
79  }
80  $session->close;
81
82
83  sub get_packets_out {
84      $result = $session->get_request($oid_pkts_out.$number);
85      print POUT "-1\n" unless (defined $result);
86      $packets_out = $result->{$oid_pkts_out.$number};
87      return $packets_out;
88  }
89
90  sub get_packets_in {
91      $result = $session->get_request($oid_pkts_in.$number);
92      print PIN "-1\n" unless (defined $result);
93      $packets_in = $result->{$oid_pkts_in.$number};
94      return $packets_in;
95  }
```

Listing A.2: Data scaling

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main( int argc, char **argv )
5  {
6      if (argc != 3){
7          printf("Bad args!\n");
8          exit(EXIT_FAILURE);
9      }
10
11     FILE *data;
12     data = fopen( argv[1], "r" );
13     if ( !data ) {
14         printf( "Failed to open data file!\n" );
15         exit(EXIT_FAILURE);
16     }
```

```
17
18      int m = atoi(argv[2]);
19      int i, x, x1;
20      float sum, avg;
21
22      while ( fscanf( data, "%i\n", &x ) != EOF ) {
23          sum = 0;
24          avg = 0;
25          for ( i = 1; i < m ; i++ ) {
26              if ( fscanf( data, "%i\n", &x1 ) != EOF ) {
27                  sum += x1;
28              } else {
29                  fclose( data );
30                  exit(EXIT_FAILURE);
31              }
32          }
33
34          sum += x;
35          avg = sum / m;
36          printf( "%f\n", avg );
37
38          x=x1=0;
39      }
40
41      fclose( data );
42      exit(EXIT_SUCCESS);
43  }
```

Listing A.3: Data scaling automation

```
1  #!/bin/bash
2
3  if [ -z "$1" ]; then echo "Give me datadir name!" && exit 1; fi
4  if [ -z "$2" ]; then echo "Give me limit!" && exit 1; fi
5
6  datadir=$1
7  limit=$2
8
9  for (( i = 2; i < $limit; i++ ))
10 do
11     ./avg $datadir/avg1 $i > $datadir/avg$i
12     echo "avg$i done"
13 done
```

Listing A.4: Least Squares Fitting

```
1  #include <stdio.h>
```

```
2  #include <stdlib.h>
3  #include <string.h>
4  #include <getopt.h>
5  #include <gsl/gsl_fit.h>
6
7  #define NAME        "line"
8  #define VERSION     "2.0"
9
10 enum {
11     LINE_MODE_RS,
12     LINE_MODE_VT
13 };
14
15 static const char *opt_string = ":m:i:h";
16 static struct option long_options[] = {
17     { "mode", required_argument, NULL, 'm' },
18     { "input", required_argument, NULL, 'i' },
19     { "help", no_argument, NULL, 'h' },
20     { 0, 0, 0, 0 },
21 };
22
23 void usage ( char *pname ) {
24     char **dp;
25     char *optdoc[] = {
26         "\n",
27         "   Options:\n",
28         "   -m, --mode             mode of operation (rs or vt)\n",
29         "   -i, --input            input data file\n",
30         "   -h, --help             print this help message\n",
31         "\nIf you find bugs, cockroaches or other nasty insects don't\n",
32         "send them to romans.jerjomins@tet.rtu.lv - just kill 'em! ;)\n",
33         0
34     };
35     fprintf( stdout, "%s v%s\n", NAME, VERSION );
36     fprintf( stdout, "Usage: %s [OPTIONS]\n", pname );
37     for ( dp = optdoc; *dp; dp++ ) {
38         fprintf( stdout, "%s", *dp );
39     }
40 }
41
42 int main( int argc, char **argv )
43 {
44     int mode = -1;
45     char infile[ 255 ] = "data.in";
46
47     int opt;
48     int option_index = 0;
```

```
49    while ( (opt = getopt_long( argc, argv,
50                    opt_string, long_options,
51                    &option_index )) != -1 )
52    {
53        switch( opt ) {
54        case 'm':
55            if ( !strncmp( optarg, "rs", 2 ) ) {
56                mode = LINE_MODE_RS;
57            } else if ( !strncmp( optarg, "vt", 2 ) ) {
58                mode = LINE_MODE_VT;
59            }
60            break;
61        case 'i':
62            snprintf( infile, sizeof infile, "%s", optarg );
63            break;
64        case 'h':
65            usage(argv[0]);
66            exit(EXIT_SUCCESS);
67        case '?':
68        default:
69            fprintf( stdout, "%s: option -%c is invalid\n",
70                            argv[0], optopt);
71            exit(EXIT_FAILURE);
72        }
73    }
74    if ( mode < 0 ) {
75        fprintf( stdout, "Invalid operation mode!\n" );
76        usage(argv[0]);
77        exit(EXIT_FAILURE);
78    }
79
80    /* data file as 1st argument */
81    FILE *data;
82    data = fopen( infile, "r" );
83    if ( !data ) {
84        fprintf( stdout, "Failed to open data file!\n" );
85        exit(EXIT_FAILURE);
86    }
87
88    float x, y;
89    int N = 0; //data sequence length
90
91    /* let's find out sequence length */
92    while ( fscanf( data, "%f %f\n", &x, &y ) != EOF ) {
93        N++;
94    }
95    fclose( data );
```

```
96
97      /* initialize data arrays */
98      double X[N], Y[N];
99      memset( X, 0, sizeof X );
100     memset( Y, 0, sizeof Y );
101
102     data = fopen( infile, "r" );
103     if ( !data ) {
104         fprintf( stdout, "Failed to open data file!\n" );
105         exit(EXIT_FAILURE);
106     }
107
108     /* fill arrays with data */
109     int i = 0;
110     while ( fscanf( data, "%f %f\n", &x, &y ) != EOF ) {
111         X[i] = x;
112         Y[i] = y;
113         i++;
114     }
115     fclose( data );
116
117     double c0, c1, cov00, cov01, cov11, chisq;
118     gsl_fit_linear( X, 1, Y, 1, N,
119             &c0, &c1, &cov00, &cov01, &cov11,
120             &chisq );
121
122     switch (mode) {
123         case LINE_MODE_RS:
124             fprintf( stdout, "H=%f\n", c1 );
125             fprintf( stdout, "%f+%f*x\n", c0, c1 );
126             break;
127         case LINE_MODE_VT:
128             fprintf( stdout, "H=%f\n", 1 + c1 / 2 );
129             fprintf( stdout, "%f*x+%f\n", c1, c0 );
130             break;
131     }
132
133     exit(EXIT_SUCCESS);
134 }
```

Listing A.5: Variance-timing

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  int main( int argc, char **argv )
```

```
6  {
7      if ( argc != 3 ) {
8          printf( "Bad args!\n" );
9          exit(EXIT_FAILURE);
10     }
11
12     int rows;
13     FILE *data;
14     float x;
15     float sum = 0;
16     float Mx = 0;
17     double D = 0;
18     double Dx = 0;
19
20     /* data file as 1st argument */
21     data = fopen( argv[1], "r" );
22     if ( !data ) {
23         printf( "Failed to open data file!\n" );
24         exit(EXIT_FAILURE);
25     }
26
27     /* read scale from 2nd argument */
28     int m = atoi(argv[2]);
29
30     rows = 0;
31     while ( fscanf( data, "%f\n", &x ) != EOF ) {
32         sum += x;
33         rows++;
34     }
35     fclose( data );
36
37     /* compute mean */
38     Mx = sum / rows;
39
40     data = fopen( argv[1], "r" );
41     if ( !data ) {
42         printf( "Failed to open data file!\n" );
43         exit(EXIT_FAILURE);
44     }
45
46     while ( fscanf( data, "%f\n", &x ) != EOF ) {
47         D += (x - Mx) * (x - Mx);
48     }
49     fclose( data );
50
51     /* compute variation */
52     Dx = D / rows;
```

```
53
54      /* print logarithm from scale and variation */
55      printf( "%f %f\n", log(m), log(Dx) );
56
57      exit(EXIT_SUCCESS);
58  }
```

Listing A.6: R/S statistics

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  /* Maximum size of time series data array */
7  #define MAX_SIZE 1000000L
8
9  /* Time series read from "in.dat" */
10 double X[MAX_SIZE];
11 /* Number of values in data file */
12 long int N;
13
14 /* Load X array from file */
15 void load_X_array( FILE *file );
16 /* Compute R/S for X of length N */
17 double compute_rs(void);
18
19 int main( int argc, char **argv )
20 {
21     double rs_value;
22     FILE *file = fopen( argv[1], "r" );
23     if ( !file )
24         exit(1);
25
26     load_X_array( file );
27
28     rs_value = compute_rs();
29
30     printf("%f %f\n", log(N), log(rs_value));
31
32     exit(EXIT_SUCCESS);
33 }
34
35 void load_X_array( FILE *file )
36 {
37     char temp_string[1024];
38     N = 0;
```

```
39
40      while (1) {
41          fscanf( file, "%s", temp_string );
42          if ( feof(file) )
43              goto end;
44
45          /* handle comments bounded by "&" symbols */
46          while ( strcmp( temp_string, "&" ) == 0 ) {
47              do {
48                  fscanf( file, "%s", temp_string );
49                  if ( feof(file) )
50                      goto end;
51              } while ( strcmp(temp_string, "&") != 0 );
52
53              fscanf( file, "%s", temp_string);
54              if ( feof(file) )
55                  goto end;
56          }
57
58          /* insert value in array and increment array index */
59          X[N] = atof(temp_string);
60          N++;
61
62          /* check if MAX_SIZE data values exceeded */
63          if ( N >= MAX_SIZE ) {
64              printf( "ERROR: too many data values\n" );
65              exit(1);
66          }
67      }
68
69  end:
70      return;
71  }
72
73  double compute_rs()
74  {
75      double   mom1;     // First moment
76      double   mom2;     // Second moment
77      double   x_bar;    // Mean (X_bar value)
78      double   s;        // Standard deviation (S value)
79      double   w;        // W value
80      double   r;        // R value
81      double   min_w;    // Minimum W value
82      double   max_w;    // Maximum W value
83      double   rs_value; // R/S value to be returned
84      double   sum;      // Temporary sum value
85      long int i, j;     // Loop counters
```

```
86
87      /* compute mean and standard deviation of X */
88      mom1 = mom2 = 0.0;
89      for ( i = 0; i < N; i++) {
90          mom1 += X[i] / N;
91          mom2 += pow( X[i], 2.0 ) / N;
92      }
93      x_bar = mom1;
94      s = sqrt( mom2 - pow(mom1, 2.0) );
95
96      /* find minimum and maximum W values */
97      min_w = max_w = 0.0;
98      sum = 0.0;
99      for ( i = 0; i < N; i++)
100     {
101         sum += X[i];
102
103         w = sum - ((i+1) * x_bar); // fix #1
104         if ( w > max_w )
105             max_w = w;
106         if ( w < min_w )
107             min_w = w;
108     }
109
110     /* Compute R value as maximum W minus minimum W */
111     r = max_w - min_w;
112
113     /* Compute R/S value */
114     rs_value = r / s;
115
116     return rs_value;
117 }
```

Listing A.7: R/S and variance-time output generation

```
1  #!/bin/bash
2
3  if [ -z "$1" ]; then echo "Give me datadir name!" && exit 1; fi
4  if [ -z "$2" ]; then echo "Give me limit!" && exit 1; fi
5  if [ -z "$3" ]; then echo "Give me mode!" && exit 1; fi
6
7  datadir=$1
8  limit=$2
9  mode=$3
10
11 outfile="$mode.out"
12 rm -f $outfile
```

```
13
14 for (( z = 1; z < $limit; z++ ))
15 do
16     ./$mode $datadir/avg$z $z >> $datadir/$outfile
17     echo "$z $mode point done"
18 done
```

Listing A.8: R/S and variance-time plotting

```
1 #!/bin/bash
2
3 if [ -z "$1" ]; then echo "Give me datadir name!" && exit 1; fi
4 if [ -z "$2" ]; then echo "Give me mode!" && exit 1; fi
5 if [ -z "$3" ]; then echo "Give me linemode!" && exit 1; fi
6
7 datadir=$1
8 mode=$2
9 linemode=$3
10
11 file="$datadir/$mode.out"
12 plotfile="$mode.plot"
13 xlabel="log(scale)"
14 ylabel="log($mode)"
15
16 hurst=`./line -i $file -m $linemode | grep H=`
17 line=`./line -i $file -m $linemode | grep "*x"`
18
19 echo "set title 'Hurst parameter $hurst'" > $plotfile
20 echo "set xlabel '$xlabel'" >> $plotfile
21 echo "set ylabel '$ylabel'" >> $plotfile
22 echo "set grid" >> $plotfile
23 echo "set terminal postscript eps monochrome" >> $plotfile
24 echo "set output '$mode.eps'" >> $plotfile
25 echo "plot '$file', $line " >> $plotfile
26 #echo "pause -1" >> $plotfile
27 gnuplot $plotfile
28
29 convert -density 200 $mode.eps $mode.png
30
31 rm -f $plotfile
```

Listing A.9: Run analysis script

```
1 #!/bin/bash
2
3 if [ -z "$1" ]; then echo "Give me file name!" && exit 1; fi
4
```

```
datafile=$1
datadir="data"
limit=513

mkdir -p $datadir
cp $datafile $datadir/avg1

make

./scaling/avg.sh $datadir $limit
./hurst/out.sh $datadir $limit variance
./hurst/out.sh $datadir $limit rs
echo "Plotting variance-time"
./hurst/draw.sh $datadir variance vt
echo "Plotting R/S"
./hurst/draw.sh $datadir rs rs

mkdir -p pic
mv *.eps pic/
mv *.png pic/

make clean
```

Listing A.10: Correlation analysis

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gsl/gsl_statistics.h>

int main( int argc, char *argv[] )
{
    if ( argc != 2 ) {
        printf( "Bad args!\n" );
        exit(EXIT_FAILURE);
    }

    /* data file as 1st argument */
    FILE *data;
    data = fopen( argv[1], "r" );
    if ( !data ) {
        printf("Bad datafile!\n");
        exit(EXIT_FAILURE);
    }

    float X = 0;
    int N = 0;
```

```
23      /* let's count how many values (points) we have */
24      while ( fscanf( data, "%f\n", &X ) != EOF ) {
25          N++;
26      }
27      fclose( data );
28
29
30      int i = 0;
31      /* initialize data array */
32      double data_arr[N];
33      memset( data_arr, 0, sizeof data_arr );
34
35      data = fopen( argv[1], "r" );
36      while ( fscanf( data, "%f\n", &X ) != EOF ) {
37          data_arr[i] = X;
38          i++;
39      }
40      fclose( data );
41
42      double mx = gsl_stats_mean( data_arr, 1, N );
43      double dx = gsl_stats_variance( data_arr, 1, N );
44
45      int k, n;
46      float R, Rk, Xn, Xnk;
47      R = 0;
48      Rk = 0;
49      for ( k = 0; k < N; k++ )
50      {
51          for ( n = 0; n < N; n++ ) {
52              if ( (n + k) < N ) {
53                  Xn = data_arr[n];
54                  Xnk = data_arr[n + k];
55                  R += (Xn - mx) * (Xnk - mx);
56              }
57          }
58
59          Rk = (R / N) / dx;
60          R = 0;
61          printf( "%f\n", Rk );
62      }
63
64      exit(EXIT_SUCCESS);
65  }
```

Listing A.11: Analytical method

```
1  #!/usr/bin/octave -qf
```

```
2
3 arg_list = argv();
4 global Ta = str2double(arg_list{1});
5 global Ts = str2double(arg_list{2});
6 global simtime = str2double(arg_list{3});
7 global l = 1/Ta;
8 global u = 1/Ts;
9 global F = l/u;
10
11
12 function y = f (x)
13     global l; global u; global F;
14     lu = 2 * sqrt(l*u);
15     k = 2:1000;
16     y = e^(-x*(l+u))*( besseli(0,(x*lu)) +
17         1/sqrt(F)*besseli(1,(x*lu)) +
18         (1-F)*sum( 1./F.^(k/2).*besseli(k,(x*lu)) ) );
19 endfunction
20
21 #for t=2900:4000
22 for t = 1:simtime
23     [area] = quad("f", 0, t);
24     n = t * (l-u) + u * area;
25     printf("%i %f\n", t, n);
26 endfor
```

Listing A.12: Approximation method

```
1 #!/usr/bin/octave -qf
2
3 arg_list = argv ();
4 #global N = 10;
5 global N = str2double(arg_list{3});
6 global c1 = 1;
7 global c2 = str2double(arg_list{2});
8 global mu1 = 1;
9 global rho = str2double(arg_list{1});
10 global mu2 = mu1/rho;
11
12 koef = sign((c1+c2*rho)/(1-rho)) * (c1+c2*rho)/(1-rho);
13 #koef = sign((c1+c2*rho)/(rho-1)) * (c1+c2*rho)/(rho-1);
14 #koef = (c1+c2*rho)/(rho-1);
15 global b = (N+1)/koef ;
16
17 #global y0 = [0, b/2, b];
18 global y0=0;
19
```

```
limit = 100;
n=1:limit ;
global lambda = n*pi/b ;

global delta;
if (rho < 1)
    delta = -1;
elseif (rho == 1)
    delta = 0;
elseif (rho > 1)
    delta = 1;
endif

#global Phi = sqrt( (2 .* lambda.^2 )./(b.*(lambda.^2 .+ 1)) ).*( cos(
    lambda.*y0) .+ (delta./lambda).*(sin(lambda.*y0)) ) ;
function PhiY = Phi(y)
    global lambda; global delta; global b;
    PhiY = sqrt( (2 .* lambda.^2 )./(b.*(lambda.^2 .+ 1)) ).*( cos(lambda.*
        y) .+ (delta./lambda).*(sin(lambda.*y)) ) ;
endfunction
#global Phi = PhiY(y0) ;

function PhiY2 = Phi2(y)
    global lambda; global b;
    PhiY2 = sqrt( (2 .* lambda.^2 )./(b.*(lambda.^2 .+ 1)) ).*( cos(lambda
        .*y) .- lambda.*(sin(lambda.*y)) ) ;
endfunction

function P = probability(y)
    global lambda; global delta; global b; global y0; global tau;
    P = y*( ( (2*delta*e^(2*delta*y))/(e^(2*delta*b) - 1) ) + e^(delta*(y-
        y0-(delta*tau/2))) * sum(Phi(y) .* Phi(y0) .* e.^(-(lambda.^2).*tau
        /2)) ) ;
    #P = y.*( ( (2.*delta.*e.^(2.*delta.*y))./(e.^(2.*delta.*b) .- 1) ) .+
        e.^(delta.*(y.-y0.-(delta.*tau./2))) .* sum(Phi(y) .* Phi(y0) .* e
        .^(-(lambda.^2).*tau/2)) ) ;
endfunction

function Py2 = P2(y)
    global lambda; global b; global y0; global tau;
    Py2 = y*( ( (-2*e^(-2*y))/(e^(-2*b) - 1) ) + e^(-y+y0-(tau/2)) * sum(
        Phi2(y) .* Phi2(y0) .* e.^(-(lambda.^2).*tau/2)) ) ;
endfunction

global tau ;
for tau=0:0.01:10
    t=(tau*mu1*(c1+c2*rho))/(1-rho)^2;
```

```
    #t=(tau*mu1*(c1+c2*rho))/(rho-1)^2;
    #t=(tau*(1-rho)^2)/mu1*(c1+c2*rho);
    #t=tau;
    #t = tau * mu1 * (c1+c2*rho) * (1-rho)^2 ;

    [integral]=quad("probability",0,b);
    #[integral]=quad("P2",0,b);
    n01 = integral * koef;
    #n01 = integral;
    #n01 = integral * ( ((mu1*c2 + mu2*c1) / (mu1 - mu2)) + (N+1) );

    printf ("%f %f\n", t, n01);
endfor
```

Listing A.13: GPSS simulation

```
        INITIAL     X$speed1,0
        INITIAL     X$speed2,0
        INITIAL     X$speed3,0
delay2      VARIABLE    7000
delay3      VARIABLE    14000
lambda1     VARIABLE    0.9
lambda2     VARIABLE    1
lambda3     VARIABLE    0.9
mikro       VARIABLE    1
        ; Terminal 1
        GENERATE    (Pareto(1,V$lambda1,1.4))
        ASSIGN      source,1
        TRANSFER    ,pro
        ; Terminal 2
        GENERATE    (Pareto(1,V$lambda2,1.4)),,V$delay2
        ASSIGN      source,3
        TRANSFER    ,pro
        ; Terminal 3
        GENERATE    (Pareto(1,V$lambda3,1.4)),,V$delay3
        ASSIGN      source,5

        ; Server
pro     QUEUE       wait
        SEIZE       proc
        DEPART      wait
        ADVANCE     (Exponential(1,0,V$mikro))
        TRANSFER    P,source,13
        QUEUE       first
        TRANSFER    ,out
        QUEUE       second
        TRANSFER    ,out
```

```
32          QUEUE       third
33 out      RELEASE     proc
34          TRANSFER    P,source,20
35          DEPART      first
36          TRANSFER    ,term
37          DEPART      second
38          TRANSFER    ,term
39          DEPART      third
40 term         SAVEVALUE   speed1,(QC$first/AC1)
41          SAVEVALUE   speed2,(QC$second/(AC1-V$delay2))
42          SAVEVALUE   speed3,(QC$third/(AC1-V$delay3))
43          SAVEVALUE   speed,((X$speed1+X$speed2+X$speed3))
44          TERMINATE
45          ; Timer
46          GENERATE    20000,,,1
47          TERMINATE   1
```

Listing A.14: Simulation program

```
1  /*
2   * M/M/1/K vs. P/M/1/K queue simulation
3   *
4   * This program is adapted from Figure 1.6 in Simulating Computer Systems,
5   * Techniqyes and Tools by M. H. MacDougall (1987).
6   */
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10 #include <string.h>
11 #include <time.h>
12 #include <argp.h>
13 #include <float.h>
14
15 #define VERSION         ”0.3”
16
17 /* Default values */
18 #define DEFAULT_VERBOSE     0
19 #define DEFAULT_QUIET       0
20 #define DEFAULT_CLIENTS     1
21 #define DEFAULT_SIMTIME     1500000
22 #define DEFAULT_ARRTIME     10.00
23 #define DEFAULT_SERVTIME    8.00
24 #define DEFAULT_BUFFER      1000000
25 #define DEFAULT_SEED        8
26 #define DEFAULT_HURST       0.56
27
28 /* Simulation parameters and options */
```

```
29 typedef struct sim_opt_t {
30     long unsigned int time;      // simulation time
31     long unsigned int buffer;    // maximum queue size
32     FILE *queue_curr_fp;         // file to save queue data
33     FILE *queue_mean_fp;         // file to save mean queue data
34     FILE *load_fp;               // file to save utilization data
35     double (*arr_cb)( void * ); // callback function for arrivals
36     double (*dep_cb)( void * ); // callback function for departures
37     void *arr_cb_params;         // arrival callback parameters
38     void *dep_cb_params;         // departure callback parameters
39     int clients;                 // number of clients in the system
40 } sim_opt_t;
41
42 /* Simulation statistics */
43 typedef struct sim_stats_t {
44     long unsigned int processed; // served query counter
45     long unsigned int dropped;   // discarded query counter
46     long unsigned int queue;     // queue length counter
47     double queue_mean;           // mean queue length counter
48     double t_busy_tot;           // sustem busy counter
49     double t_sim_tot;            // simulated time counter
50     double t_mean_res;           // mean residence time
51     double load;                 // utilization factor
52     double rate;                 // query service rate
53     double ploss;                // query loss probability
54 } sim_stats_t;
55
56 /* Pareto distribution parameters */
57 typedef struct pareto_params_t {
58     double k;
59     double Xm;
60 } pareto_params_t;
61
62 /* Function prototypes */
63 int uniform( double *u );
64 double exponential( void *x );
65 double pareto( void *p );
66 int simulate( sim_opt_t *sim, sim_stats_t *stats );
67
68
69 /* Argument parsing */
70 const char *argp_program_version = VERSION;
71 const char *argp_program_bug_address = "romans.jerjomins@tet.rtu.lv";
72
73 /* Program documentation. */
74 static char doc[] = "\nM/M/1/K and P/M/1/K simulation program";
75
```

```
76 /* A description of the arguments we accept. */
77 static char args_doc[] = "";
78
79 /* The options we understand. */
80 static struct argp_option options[] = {
81     {"quiet",    'q', 0,        0, "Be quiet" },
82     {"verbose",  'v', 0,        0, "Produce verbose output" },
83     {"clients",  'c', "INT",    0, "Clients in the system (default: 1)" },
84     {"simtime",  't', "DOUBLE", 0,
85         "Define total simulation time (default: 1500000)" },
86     {"arrtime",  'l', "DOUBLE", 0,
87         "Define mean interarrival time (default: 10)" },
88     {"servtime", 'm', "DOUBLE", 0,
89         "Define mean service time (default: 8)" },
90     {"buffer",   'b', "DOUBLE", 0,
91         "Max queue length (default: 1000000)" },
92     {"seed",     's', "INT",    0,
93         "Seed value for srand function (default: 8)" },
94     {"hurst",    'h', "FLOAT",  0,
95         "Define Hurst coefficient (default: 0.56)" },
96     {"output",   'o', "FILE",   0,
97         "Output to FILE instead of standard output" },
98     { 0, 0, 0, 0, 0 }
99 };
100
101 /* Used by main to communicate with parse_opt. */
102 struct arguments
103 {
104     int quiet, verbose, seed;
105     char *output_file;
106     float hurst;
107     double simtime, arrtime, servtime, buffer;
108     int clients;
109 };
110
111 /* Parse a single option. */
112 static error_t parse_opt ( int key, char *arg, struct argp_state *state )
113 {
114     struct arguments *arguments = state->input;
115     switch (key) {
116         case 'q':
117             arguments->quiet = 1;
118             break;
119         case 'v':
120             arguments->verbose = 1;
121             break;
122         case 'c':
```

```
123             arguments->clients = atoi(arg);
124             break;
125         case 't':
126             arguments->simtime = atof(arg);
127             break;
128         case 'l':
129             arguments->arrtime = atof(arg);
130             break;
131         case 'm':
132             arguments->servtime = atof(arg);
133             break;
134         case 'b':
135             arguments->buffer = atof(arg);
136             break;
137         case 's':
138             arguments->seed = atoi(arg);
139             break;
140         case 'h':
141             arguments->hurst = atof(arg);
142             break;
143         case 'o':
144             arguments->output_file = arg;
145             break;
146         default:
147             return ARGP_ERR_UNKNOWN;
148     }
149
150     return 0;
151 }
152
153 /* Our argp parser. */
154 static struct argp argp = { options, parse_opt, args_doc, doc };
155
156
157 int main( int argc, char **argv ) {
158
159     struct arguments arguments;
160     /* Default values. */
161     arguments.quiet = DEFAULT_QUIET;
162     arguments.verbose = DEFAULT_VERBOSE;
163     arguments.clients = DEFAULT_CLIENTS;
164     arguments.simtime = DEFAULT_SIMTIME;
165     arguments.arrtime = DEFAULT_ARRTIME;
166     arguments.servtime = DEFAULT_SERVTIME;
167     arguments.buffer = DEFAULT_BUFFER;
168     arguments.seed = DEFAULT_SEED;
169     arguments.hurst = DEFAULT_HURST;
```

```
170     arguments.output_file = NULL;
171     argp_parse (&argp, argc, argv, 0, 0, &arguments);
172
173     /* Common variables */
174     int simerr;
175     FILE *queue_curr_fp;
176     FILE *queue_mean_fp;
177     FILE *load_fp;
178
179     double Ta = arguments.arrtime;    // Mean time between arrivals
180     double Ts = arguments.servtime;   // Mean service time
181
182
183     /*************************
184      * M/M/1/K simulation part
185      *************************/
186     const char *buffer_mm1_file = "buffer_mm1.out";
187     const char *buffer_mean_mm1_file = "buffer_mean_mm1.out";
188     const char *load_mm1_file = "load_mm1.out";
189
190     queue_curr_fp = fopen( buffer_mm1_file, "w" );
191     if ( queue_curr_fp == NULL ) {
192         printf( "ERROR creating output file (%s)\n", buffer_mm1_file );
193         exit(EXIT_FAILURE);
194     }
195
196     queue_mean_fp = fopen( buffer_mean_mm1_file, "w" );
197     if ( queue_mean_fp == NULL ) {
198         printf( "ERROR creating output file (%s)\n",
199                         buffer_mean_mm1_file );
200         exit(EXIT_FAILURE);
201     }
202
203     load_fp = fopen( load_mm1_file, "w" );
204     if ( load_fp == NULL ) {
205         printf( "ERROR creating output file (%s)\n", load_mm1_file );
206         exit(EXIT_FAILURE);
207     }
208
209     /* randomize rand() with known seed */
210     srand( arguments.seed );
211
212     sim_stats_t mm1_stats;
213
214     sim_opt_t mm1;
215     mm1.time = arguments.simtime;
216     mm1.buffer = arguments.buffer;
```

```
217     mm1.queue_curr_fp = queue_curr_fp;
218     mm1.queue_mean_fp = queue_mean_fp;
219     mm1.load_fp = load_fp;
220     mm1.arr_cb = exponential;
221     mm1.dep_cb = exponential;
222     mm1.arr_cb_params = &Ta;
223     mm1.dep_cb_params = &Ts;
224     mm1.clients = arguments.clients;
225
226     simerr = simulate( &mm1, &mm1_stats );
227
228     fclose(queue_curr_fp);
229     fclose(queue_mean_fp);
230     fclose(load_fp);
231
232     if (simerr < 0) {
233         fprintf( stdout, "Simulation error\n" );
234         exit(EXIT_FAILURE);
235     }
236
237     if ( !arguments.quiet ) {
238         printf("======================================\n");
239         printf("   *** M/M/1 simulation results ***   \n");
240         printf("======================================\n");
241         printf("INPUTS:                               \n");
242         printf("  Mean interarrival time = %f sec     \n",Ta);
243         printf("  Mean service time      = %f sec     \n",Ts);
244         printf("======================================\n");
245         printf("OUTPUTS:                              \n");
246         printf("  Total simulated time  = %3.4f sec   \n",
247                             mm1_stats.t_sim_tot);
248         printf("  Number of completions = %lu pkts    \n",
249                             mm1_stats.processed);
250         printf("  Throughput rate       = %f pkts/sec \n",
251                             mm1_stats.rate);
252         printf("  Server utilization    = %f          \n",
253                             mm1_stats.load);
254         printf("  Mean queue length     = %f pkts     \n",
255                             mm1_stats.queue_mean);
256         printf("  Mean residence time   = %f sec      \n",
257                             mm1_stats.t_mean_res);
258         printf("  Buffer misses         = %lu pkts    \n",
259                             mm1_stats.dropped);
260         printf("  Loss probability      = %f          \n",
261                             mm1_stats.ploss );
262         printf("======================================\n");
263         printf("\n\n");
```

```
264     }
265
266
267     /*************************
268      * P/M/1/K simulation part
269      *************************/
270     const char *buffer_pm1_file = "buffer_pm1.out";
271     const char *buffer_mean_pm1_file = "buffer_mean_pm1.out";
272     const char *load_pm1_file = "load_pm1.out";
273
274     queue_curr_fp = fopen( buffer_pm1_file, "w" );
275     if ( queue_curr_fp == NULL ) {
276         printf( "ERROR creating output file (%s)\n", buffer_pm1_file );
277         exit(EXIT_FAILURE);
278     }
279
280     queue_mean_fp = fopen(buffer_mean_pm1_file, "w");
281     if ( queue_mean_fp == NULL ) {
282         printf( "ERROR creating output file (%s)\n",
283                         buffer_mean_pm1_file );
284         exit(EXIT_FAILURE);
285     }
286
287     load_fp = fopen( load_pm1_file, "w" );
288     if ( load_fp == NULL ) {
289         printf( "ERROR creating output file (%s)\n", load_pm1_file );
290         exit(EXIT_FAILURE);
291     }
292
293     /* seed to the same random sequence for sane comparision */
294     srand( arguments.seed );
295
296     /*
297      * Calculate shape k and scale(location?) Xm parameters for
298      * Pareto distribution based on mean interarrival time
299      * from M/M/1 simulation and defined Hurst coefficient
300      */
301     double k = 3 - (2 * arguments.hurst);
302     double Xm = (Ta * k - Ta) / k;
303     pareto_params_t pareto_cb_params = { k, Xm };
304
305     sim_stats_t pm1_stats;
306
307     sim_opt_t pm1;
308     pm1.time = arguments.simtime;
309     pm1.buffer = arguments.buffer;
310     pm1.queue_curr_fp = queue_curr_fp;
```

```
311     pm1.queue_mean_fp = queue_mean_fp;
312     pm1.load_fp = load_fp;
313     pm1.arr_cb = pareto;
314     pm1.dep_cb = exponential;
315     pm1.arr_cb_params = &pareto_cb_params;
316     pm1.dep_cb_params = &Ts;
317     pm1.clients = arguments.clients;
318
319     simerr = simulate( &pm1, &pm1_stats );
320
321     fclose(queue_curr_fp);
322     fclose(queue_mean_fp);
323     fclose(load_fp);
324
325     if (simerr < 0) {
326         fprintf( stdout, "Simulation error\n" );
327         exit(EXIT_FAILURE);
328     }
329
330     if (!arguments.quiet){
331         printf("======================================\n");
332         printf("   *** P/M/1 simulation results ***   \n");
333         printf("======================================\n");
334         printf("INPUTS:                               \n");
335         printf("  Hurst coefficient      = %f         \n",
336                             arguments.hurst );
337         printf("  Mean service time      = %f sec     \n",Ts);
338         printf("======================================\n");
339         printf("OUTPUTS:                              \n");
340         printf("  k                      = %f         \n",k);
341         printf("  Xm                     = %f         \n",Xm);
342         printf("  Total simulated time   = %3.4f sec  \n",
343                             pm1_stats.t_sim_tot );
344         printf("  Number of completions = %ld pkts    \n",
345                             pm1_stats.processed );
346         printf("  Throughput rate        = %f pkts/sec \n",
347                             pm1_stats.rate );
348         printf("  Server utilization     = %f         \n",
349                             pm1_stats.load );
350         printf("  Mean queue length      = %f pkts    \n",
351                             pm1_stats.queue_mean );
352         printf("  Mean residence time    = %f sec     \n",
353                             pm1_stats.t_mean_res );
354         printf("  Buffer misses          = %lu pkts   \n",
355                             pm1_stats.dropped );
356         printf("  Loss probability       = %f         \n",
357                             pm1_stats.ploss );
```

```
358         printf("=====================================\n");
359         printf("\n\n");
360     }
361
362
363     /* Dump interesting values to output stream */
364     FILE *output_file;
365     if (arguments.output_file)
366         output_file = fopen( arguments.output_file, "w" );
367     else
368         output_file = stdout;
369
370     if ( output_file == NULL) {
371         fprintf( stdout, "Error opening output file\n" );
372         exit(EXIT_FAILURE);
373     }
374
375     fprintf
376     (
377         output_file,
378             "SimT=%.0f\n"
379             "Ta=%.3f\n"
380             "Ts=%.3f\n"
381             "H=%.2f\n"
382             "Um=%.3f\n"
383             "Up=%.3f\n"
384             "Mm=%.3f\n"
385             "Mp=%.3f\n"
386             "Plm=%.3f\n"
387             "Plp=%.3f\n"
388             "Rm=%.3f\n"
389             "Rp=%.3f\n",
390                 arguments.simtime,
391                 Ta, Ts,
392                 arguments.hurst,
393                 mm1_stats.load, pm1_stats.load,
394                 mm1_stats.queue_mean, pm1_stats.queue_mean,
395                 mm1_stats.ploss, pm1_stats.ploss,
396                 mm1_stats.rate, pm1_stats.rate
397     );
398
399     if (arguments.output_file)
400         fclose(output_file);
401
402     exit(EXIT_SUCCESS);
403 }
404
```

```
405 /* Pull a uniform random value (0 < u < 1) */
406 int uniform( double *u )
407 {
408     *u = 0;
409 
410     // TODO: use something better than just rand()
411     while ((*u == 0) || (*u == 1)) {
412         *u = (double) rand() / RAND_MAX;
413     }
414 
415     return 0;
416 }
417 
418 /*
419  * generate exponentially distributed random values using inverse method
420  * x - mean value of distribution
421  */
422 double exponential( void *x )
423 {
424     /* Uniform random number from 0 to 1 */
425     double z;
426 
427     uniform( &z );
428 
429     /* log() is natural logarithm */
430     return -(*(double *)x) * log(z);
431 }
432 
433 double pareto( void *p )
434 {
435     double z;
436     pareto_params_t *param = (pareto_params_t *)p;
437 
438     uniform( &z );
439 
440     // Generate Pareto rv using the inverse transform sampling
441     // http://en.wikipedia.org/wiki/Pareto_distribution
442     // http://en.wikipedia.org/wiki/Inverse_transform_sampling
443     return param->Xm / pow( z, (1.0 / param->k) );
444 
445     // Generate Pareto rv using Generalized Pareto Law
446     //double mu = Xm;
447     //double sigma = Xm;
448     //double xi = k;
449     //rv = mu + (sigma*(1/pow(z, xi) - 1))/xi;
450 }
451 
```

```
452 int simulate( sim_opt_t *sim, sim_stats_t *stats )
453 {
454     if ( !sim || !stats)
455         return -1;
456
457     if ( sim->time < 0 || sim->buffer < 0 ||
458                 !sim->queue_curr_fp || !sim->queue_mean_fp)
459         return -1;
460
461     if ( !sim->queue_curr_fp || !sim->queue_mean_fp || !sim->load_fp )
462         return -1;
463
464     int i;
465     double rv = DBL_MAX;
466     double rv_tmp = 0.0;
467     double t_curr = 0.0;              // Simulation timeline
468     double t_arr = 0.0;               // Time for next event #1 (arrival)
469     double t_dep = (double)sim->time; // Time for next event #2 (departure)
470     double t_last = 0.0;              // last event time
471     double queue_area = 0.0;          // Area of number of packets
472
473     stats->processed = 0;             // Number of packets served
474     stats->dropped = 0;               // Number of packets dropped
475     stats->queue = 0;                 // Queue length
476     stats->queue_mean = 0.0;          // Mean queue length
477     stats->t_busy_tot = 0.0;          // Total busy time
478     stats->t_sim_tot = 0.0;           // Simulation total time
479     stats->t_mean_res = 0.0;          // Mean residence time
480     stats->load = 0.0;
481     stats->rate = 0.0;
482     stats->ploss = 0.0;
483
484     while ( t_curr < sim->time ) {
485         fprintf( sim->queue_curr_fp, "%f %lu\n", t_curr, stats->queue );
486
487         if ( t_arr < t_dep ) { // Event #1 (arrival)
488             t_curr = t_arr;
489             /* Update area under the curve */
490             queue_area += stats->queue * (t_curr - t_last);
491
492             if ( stats->queue > 0 )
493                 stats->t_busy_tot += t_curr - t_last;
494
495             if ( stats->queue < sim->buffer )
496                 stats->queue++;
497             else
498                 stats->dropped++;
```

```
499
500             for ( i = 0; i < sim->clients; i++ ) {
501                 rv_tmp = sim->arr_cb( sim->arr_cb_params );
502                 if ( rv_tmp < rv )
503                     rv = rv_tmp;
504             }
505
506             t_arr = t_curr + rv;
507             rv = DBL_MAX;
508
509             if ( stats->queue == 1 )
510                 t_dep = t_curr +
511                     sim->dep_cb( sim->dep_cb_params );
512
513         } else { // Event #2 (departure)
514             t_curr = t_dep;
515             queue_area += stats->queue * (t_curr - t_last);
516             stats->queue--;
517             stats->processed++;
518
519             if ( stats->queue > 0 )
520                 t_dep = t_curr +
521                     sim->dep_cb( sim->dep_cb_params );
522             else
523                 t_dep = sim->time;
524
525             stats->t_busy_tot += t_curr - t_last;
526         }
527         t_last = t_curr; // t_last = "last event time" for next event
528
529         if ( t_curr > 0 )
530             stats->queue_mean = queue_area / t_curr;
531         else
532             stats->queue_mean = 0;
533
534         stats->load = stats->t_busy_tot / t_curr;
535
536         fprintf( sim->queue_mean_fp, "%f %f\n",
537                     t_curr, stats->queue_mean );
538         fprintf( sim->load_fp, "%f %f\n", t_curr, stats->load );
539     }
540
541     stats->t_sim_tot = t_curr;
542
543     if ( stats->processed > 0 )
544         stats->rate = stats->processed / stats->t_sim_tot;
545
```

```
546     if ( stats->rate > 0 )
547         stats->t_mean_res = stats->queue_mean / stats->rate;
548 
549     if ( stats->processed > 0 || stats->dropped > 0 )
550         stats->ploss = ( 1.0 / (stats->processed + stats->dropped) )
551                             * stats->dropped;
552 
553 
554     return 0;
555 }
```

Listing A.15: Relaxation time calculation program

```
1  #!/usr/bin/perl -w
2  
3  my $sim_data="sim.dat";
4  my $mm1_data="buffer_mean_mm1.out";
5  my $pm1_data="buffer_mean_pm1.out";
6  my $window_size=20;
7  my $relax_threshold=0.07;
8  my $mm1_mean=0;
9  my $pm1_mean=0;
10 
11 my $Mm=`grep Mm= $sim_data`;
12 my $Mp=`grep Mp= $sim_data`;
13 
14 my $mm1_data_arr_size=`cat $mm1_data | wc -l`;
15 my $pm1_data_arr_size=`cat $pm1_data | wc -l`;
16 
17 if ( $Mm =~ m/^Mm=(.*)/ ) {
18     $mm1_mean = $1;
19 }
20 if ( $Mp =~ m/^Mp=(.*)/ ) {
21     $pm1_mean = $1;
22 }
23 
24 print $mm1_mean."\n".$pm1_mean."\n";
25 
26 # Initialize arrays?
27 for ( $i = 0; $i < $window_size; $i++ ) {
28     $window[$i] = 0;
29 }
30 for ( $i = 0; $i < $mm1_data_arr_size; $i++ ) {
31     $mm1_data_arr[$i] = 0;
32 }
33 for ( $i = 0; $i < $mm1_data_arr_size; $i++ ) {
34     $mm1_time_arr[$i] = 0;
```

```
35 }
36 for ( $i = 0; $i < $pm1_data_arr_size; $i++ ) {
37     $pm1_data_arr[$i] = 0;
38 }
39 for ( $i = 0; $i < $pm1_data_arr_size; $i++ ) {
40     $pm1_time_arr[$i] = 0;
41 }
42
43 my $Trelax = 0;
44 if ( open( MM1DATA, $mm1_data ) ) {
45     while (<MM1DATA>) {
46         chomp;
47         my ($time, $value) = split();
48         pop( @mm1_data_arr );
49         unshift( @mm1_data_arr, $value );
50         pop( @mm1_time_arr );
51         unshift( @mm1_time_arr, $time );
52     }
53     close(MM1DATA);
54 }
55
56 my $window_sum = 0;
57 for ( $z = 0; $z < $mm1_data_arr_size; $z++ ) {
58     shift( @window );
59     push( @window, $mm1_data_arr[$z] );
60     if ( $z >= $window_size ) {
61         $window_sum = 0;
62         for ( $i = 0; $i < $window_size; $i++ ) {
63             $window_sum = $window_sum + $window[$i];
64         }
65         if( abs( $window_sum / $window_size - $mm1_mean ) > $mm1_mean *
               $relax_threshold ) {
66             $Trelax = $mm1_time_arr[$z];
67         }
68     }
69     if( $Trelax > 0 ) {
70         last;
71     }
72 }
73
74 my $Trm = $Trelax;
75 print "Trm=".$Trm."\n";
76
77 if ( open( PM1DATA, $pm1_data ) ) {
78     while (<PM1DATA>) {
79         chomp;
80         my ($time, $value) = split();
```

```
81          pop( @pm1_data_arr );
82          unshift( @pm1_data_arr, $value );
83          pop( @pm1_time_arr );
84          unshift( @pm1_time_arr, $time );
85      }
86      close(PM1DATA);
87  }
88
89  $Trelax = 0;
90  $window_sum = 0;
91  for ( $z = 0; $z < $pm1_data_arr_size; $z++ ) {
92      shift( @window );
93      push( @window, $pm1_data_arr[$z] );
94      if ( $z >= $window_size ) {
95          $window_sum = 0;
96          for ( $i = 0; $i < $window_size; $i++ ) {
97              $window_sum = $window_sum + $window[$i];
98          }
99          if( abs( $window_sum / $window_size - $pm1_mean ) > $pm1_mean *
                $relax_threshold ) {
100             $Trelax = $pm1_time_arr[$z];
101         }
102     }
103     if ( $Trelax > 0 ) {
104         last;
105     }
106 }
107
108 my $Trp = $Trelax;
109 print "Trp=".$Trp."\n";
110
111 system('echo Trm='.$Trm.' > tr.dat');
112 system('echo Trp='.$Trp.' >> tr.dat');
```

B
Wireless traffic

Outgoing traffic for router 1 (the same as analysed in Section.2).

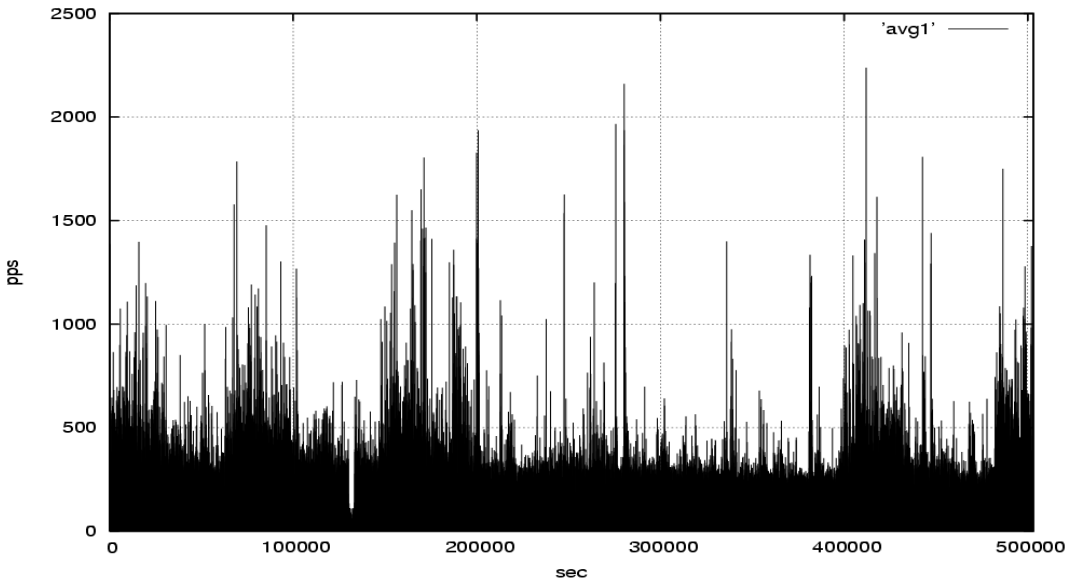


Figure B.1: Wireless network traffic, outgoing, six days period (packets per second)

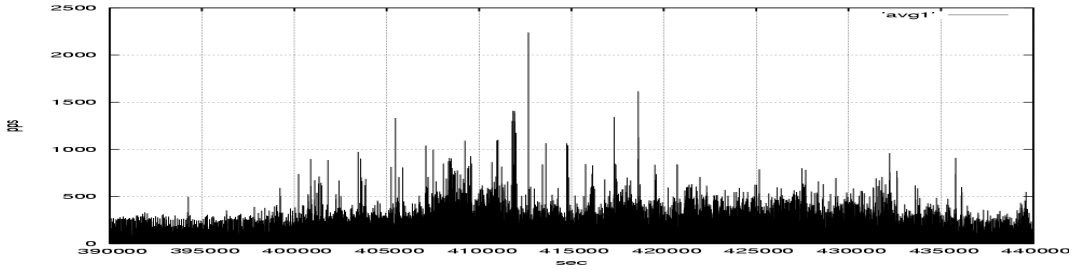


Figure B.2: Wireless network traffic, outgoing, one day period (packets per second)

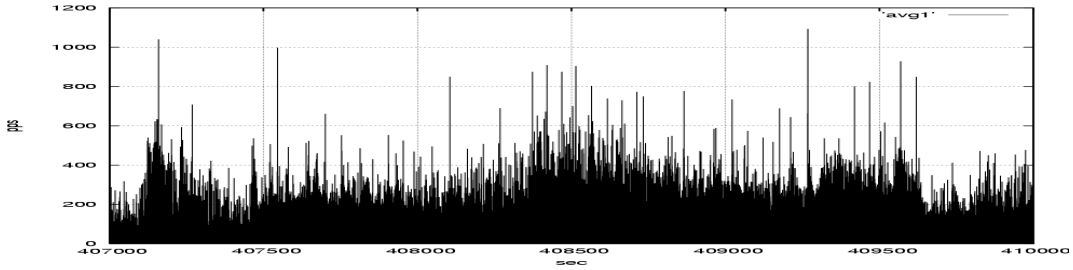


Figure B.3: Wireless network traffic, outgoing, one hour period (packets per second)

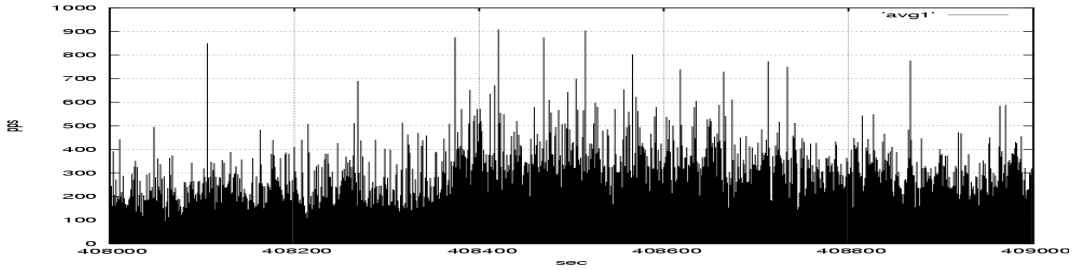


Figure B.4: Wireless network traffic, outgoing, 20 minutes period (packets per second)

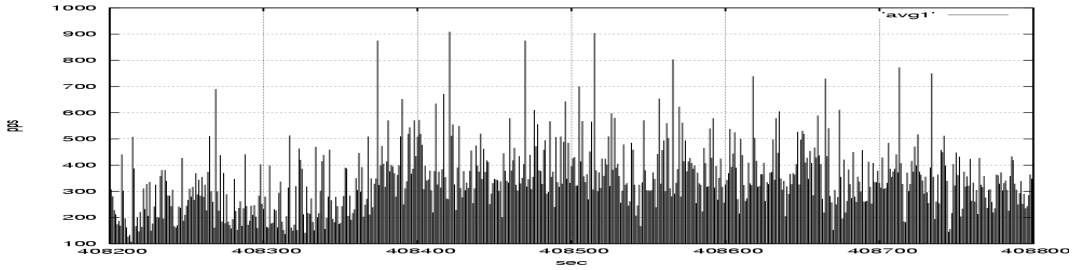


Figure B.5: Wireless network traffic, outgoing, 10 minutes period (packets per second)

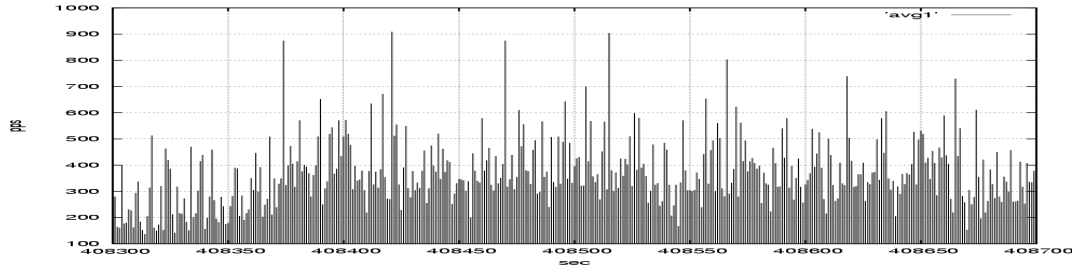


Figure B.6: Wireless network traffic, outgoing, 7 minutes period (packets per second)

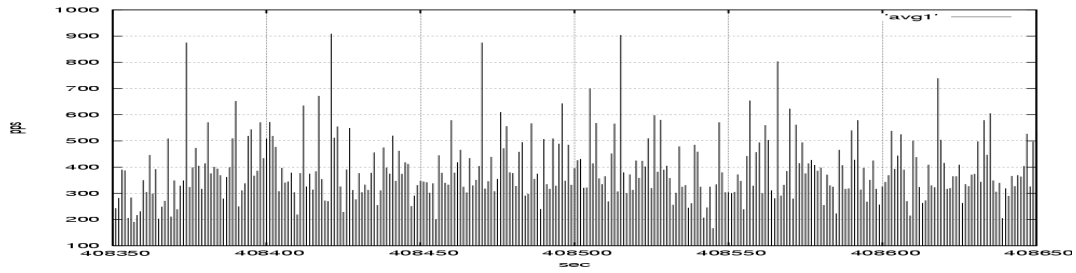


Figure B.7: Wireless network traffic, outgoing, 5 minutes period (packets per second)

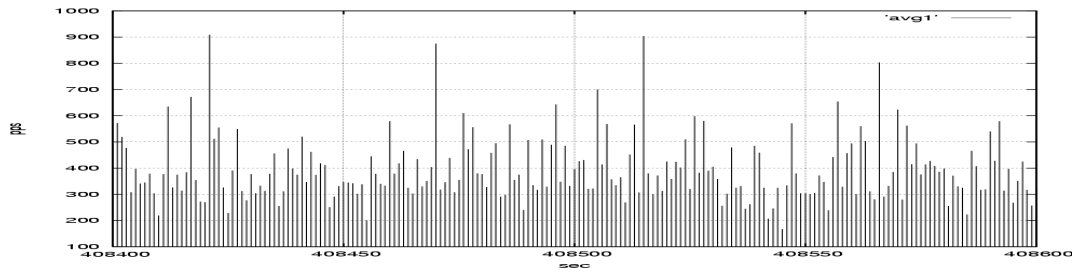


Figure B.8: Wireless network traffic, outgoing, 3 minutes period (packets per second)

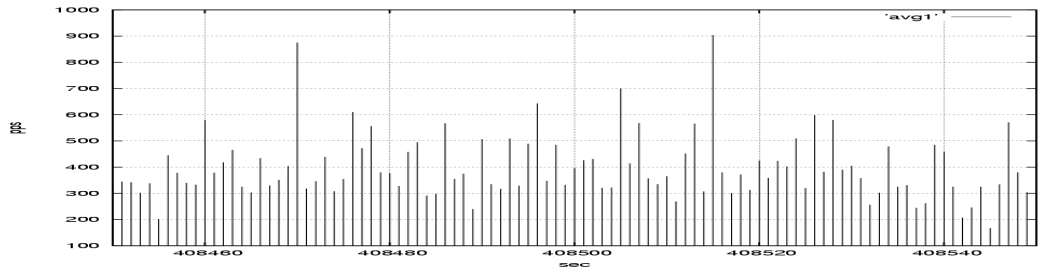


Figure B.9: Wireless network traffic, outgoing, 2 minutes period (packets per second)

Hurst parameter estimation for router 1.

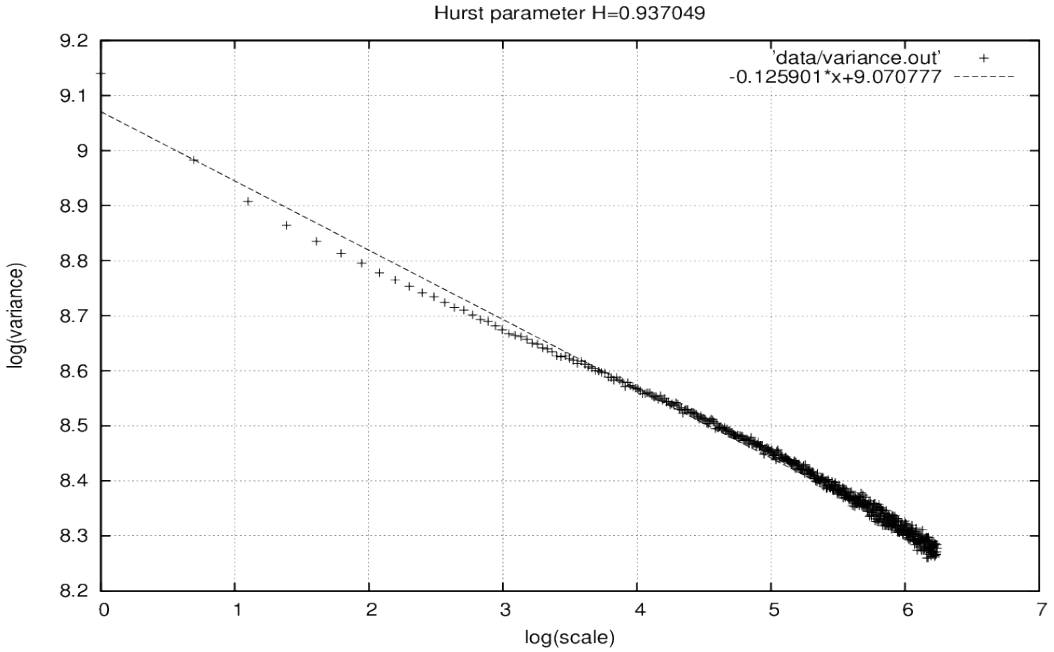


Figure B.10: variance-time plot of wireless network traffic

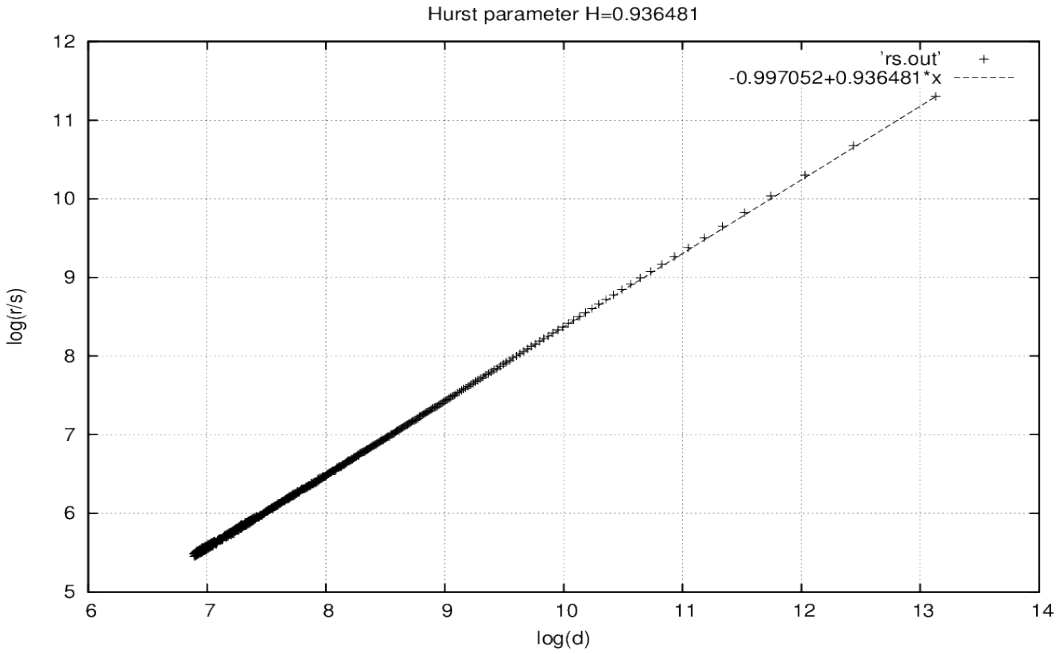


Figure B.11: R/S statistics plot of wireless network traffic

Autocorrelation of traffic from second router.

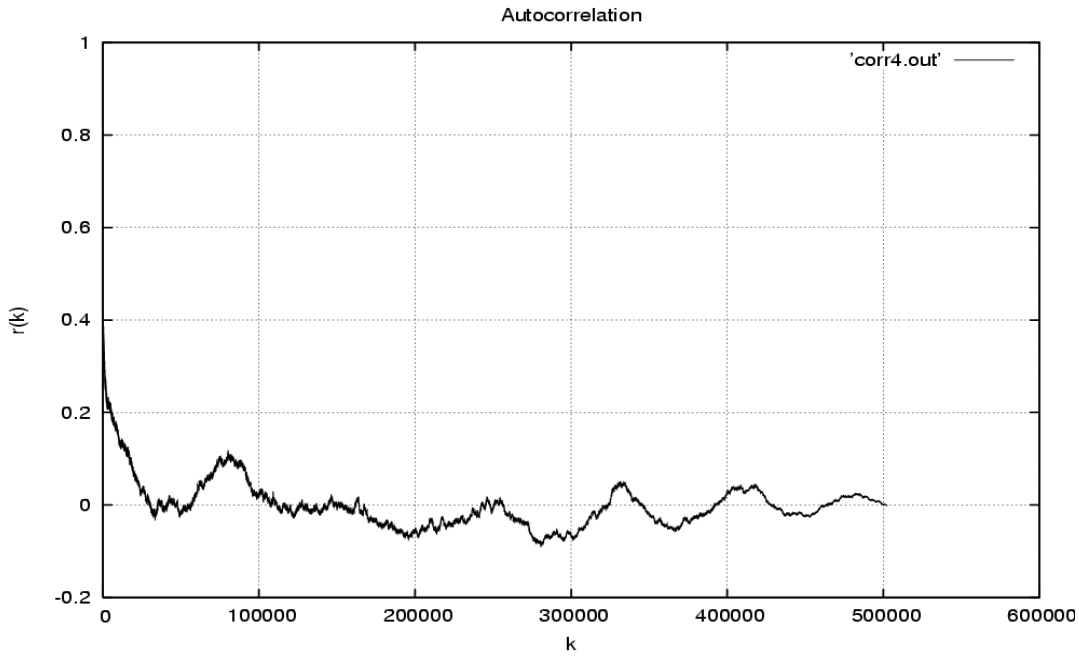


Figure B.12: Autocorrelation of wireless network traffic, full range

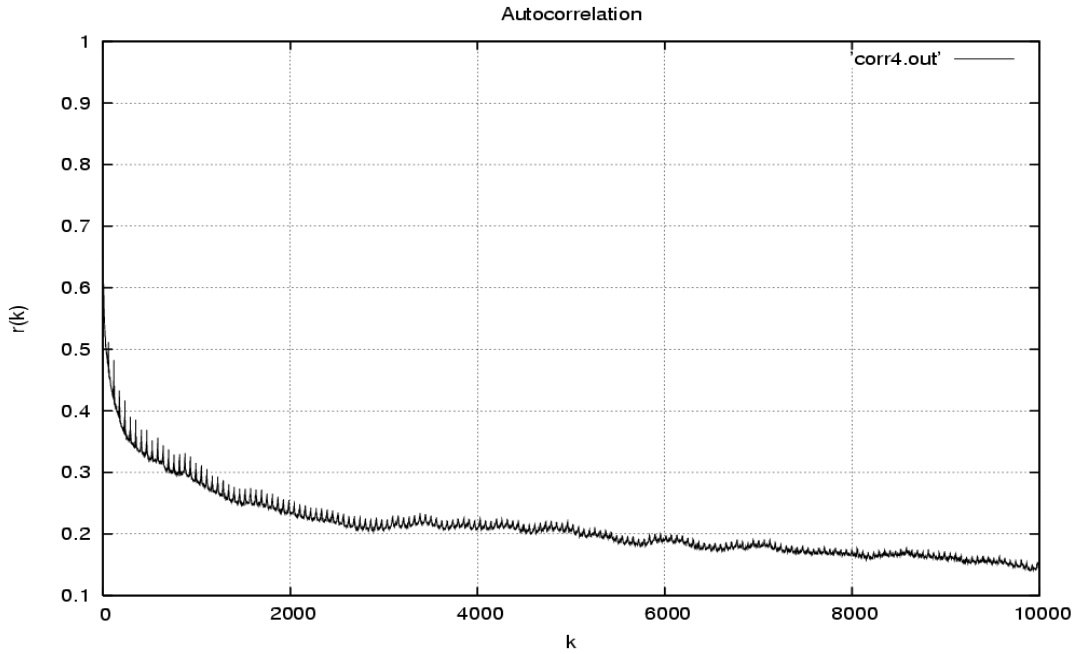
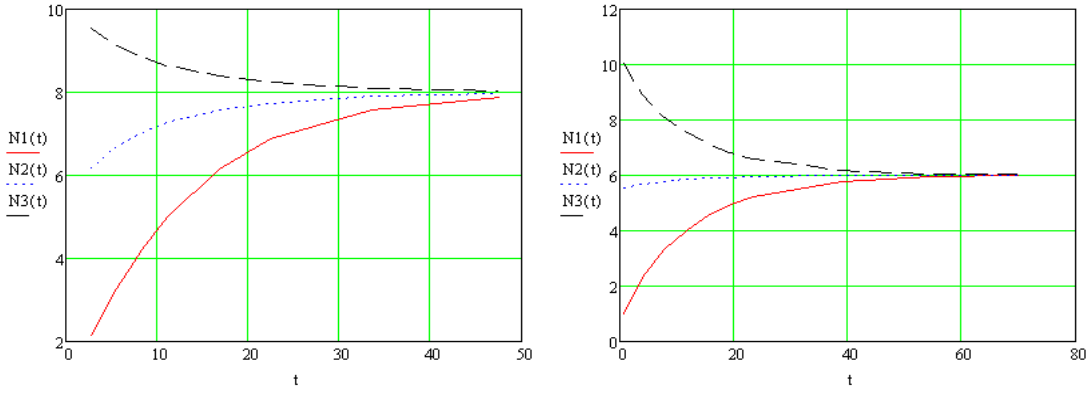


Figure B.13: Autocorrelation of wireless network traffic, first 10000 values

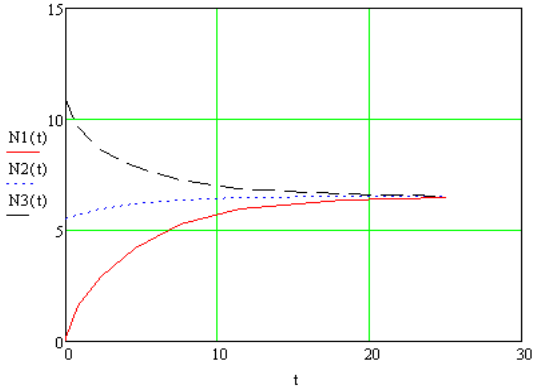
C

Diffusion approximation results

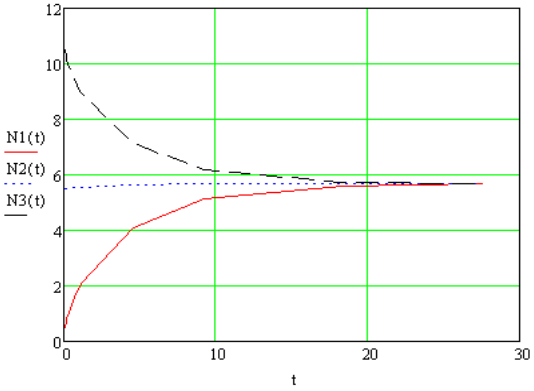


(a) $\rho = 0.75$ (b) $\rho = 0.95$

Figure C.1: Query number in system, $C_2 = 1$

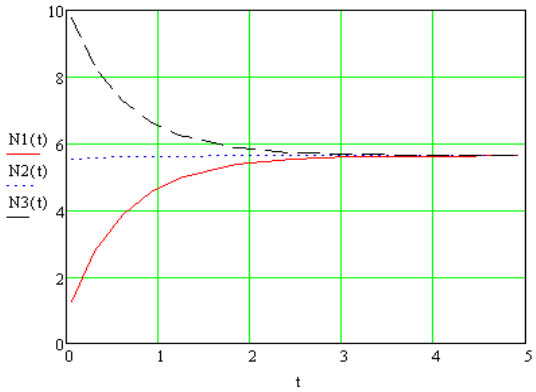


(a) $\rho = 0.75$

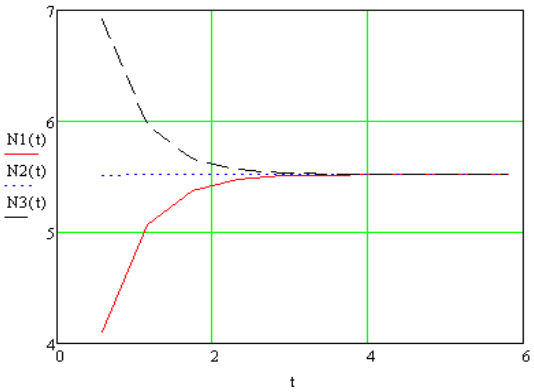


(b) $\rho = 0.95$

Figure C.2: Query number in system, $C_2 = 5$

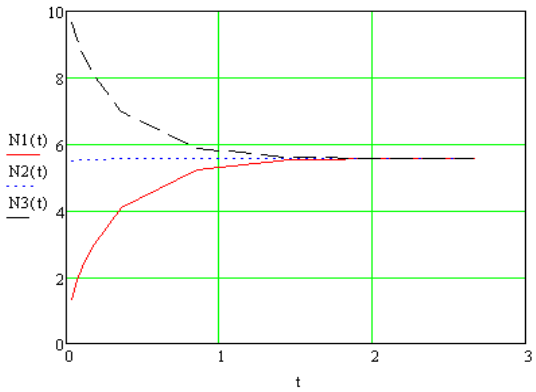


(a) $\rho = 0.75$

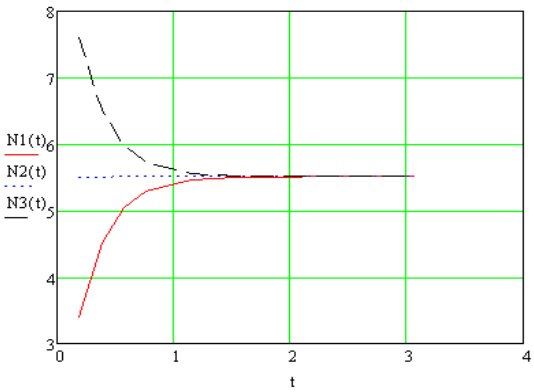


(b) $\rho = 0.95$

Figure C.3: Query number in system, $C_2 = 50$

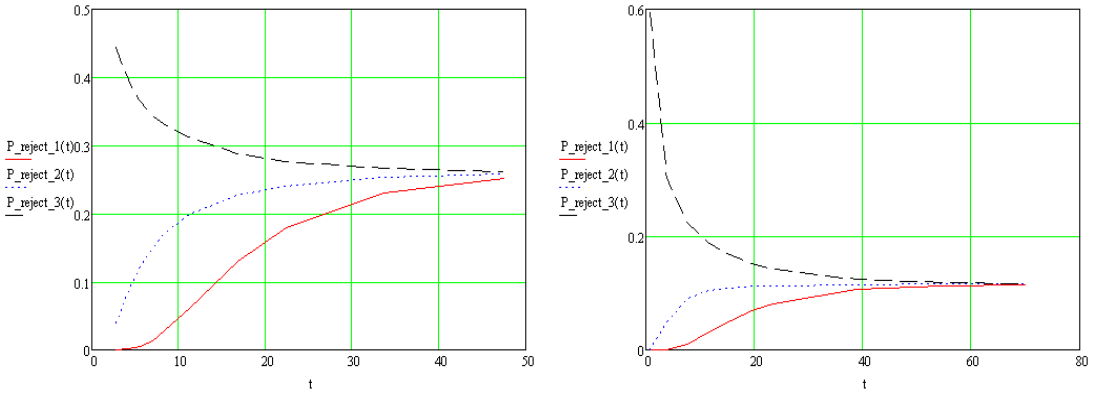


(a) $\rho = 0.75$



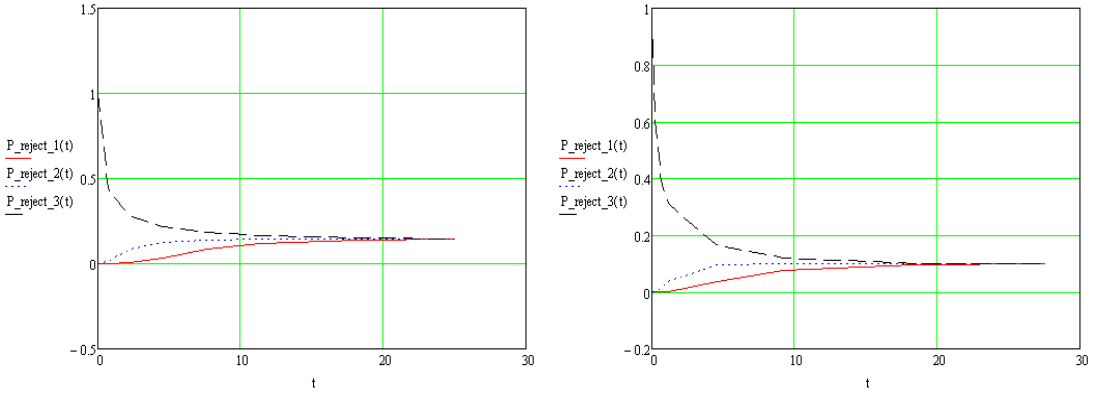
(b) $\rho = 0.95$

Figure C.4: Query number in system, $C_2 = 100$



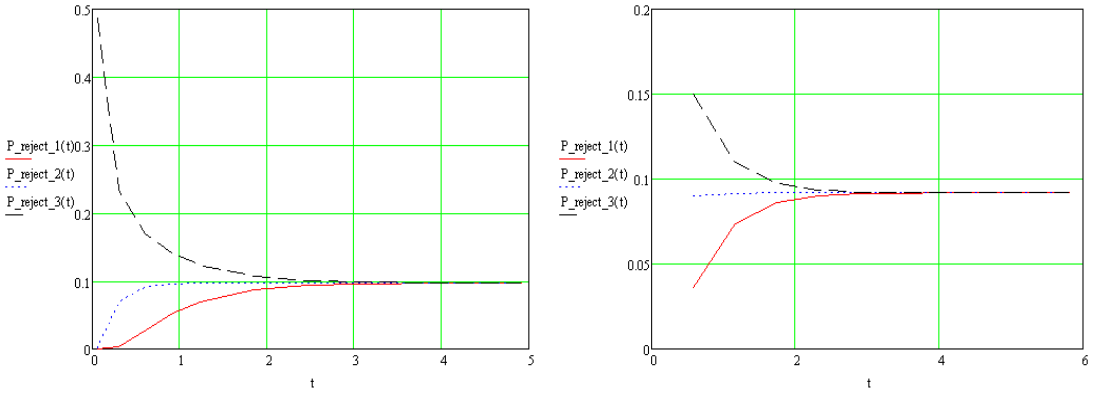
(a) $\rho = 0.75$ (b) $\rho = 0.95$

Figure C.5: Query loss probability, $C_2 = 1$



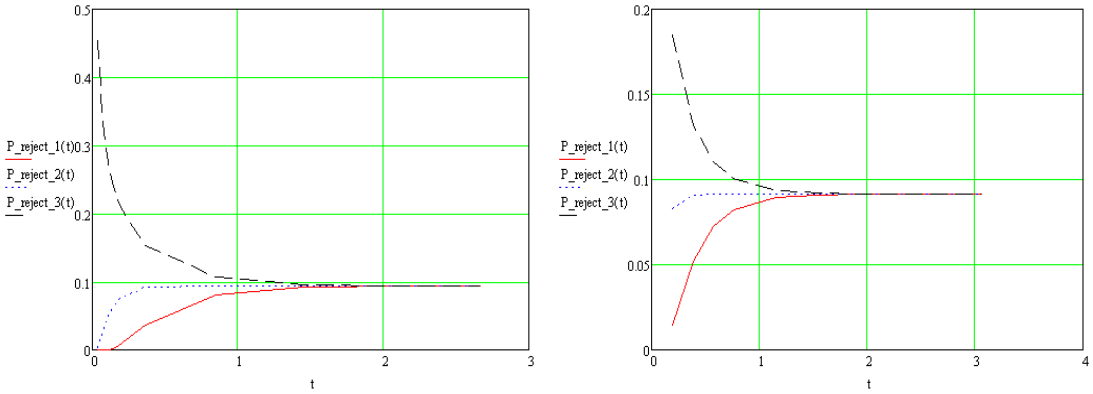
(a) $\rho = 0.75$ (b) $\rho = 0.95$

Figure C.6: Query loss probability, $C_2 = 5$



(a) $\rho = 0.75$ (b) $\rho = 0.95$

Figure C.7: Query loss probability, $C_2 = 50$



(a) $\rho = 0.75$ (b) $\rho = 0.95$

Figure C.8: Query loss probability, $C_2 = 100$

D

Simulation vs. analytics modeling

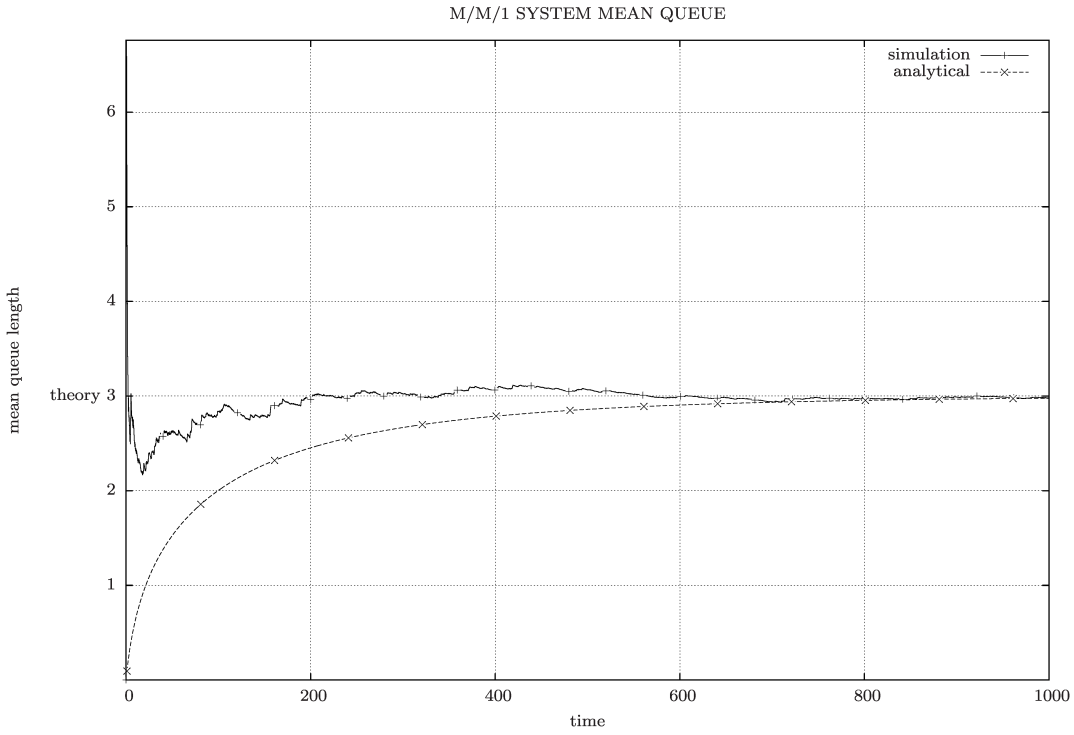


Figure D.1: $M/M/1$ simulation, $\rho = 0.75$

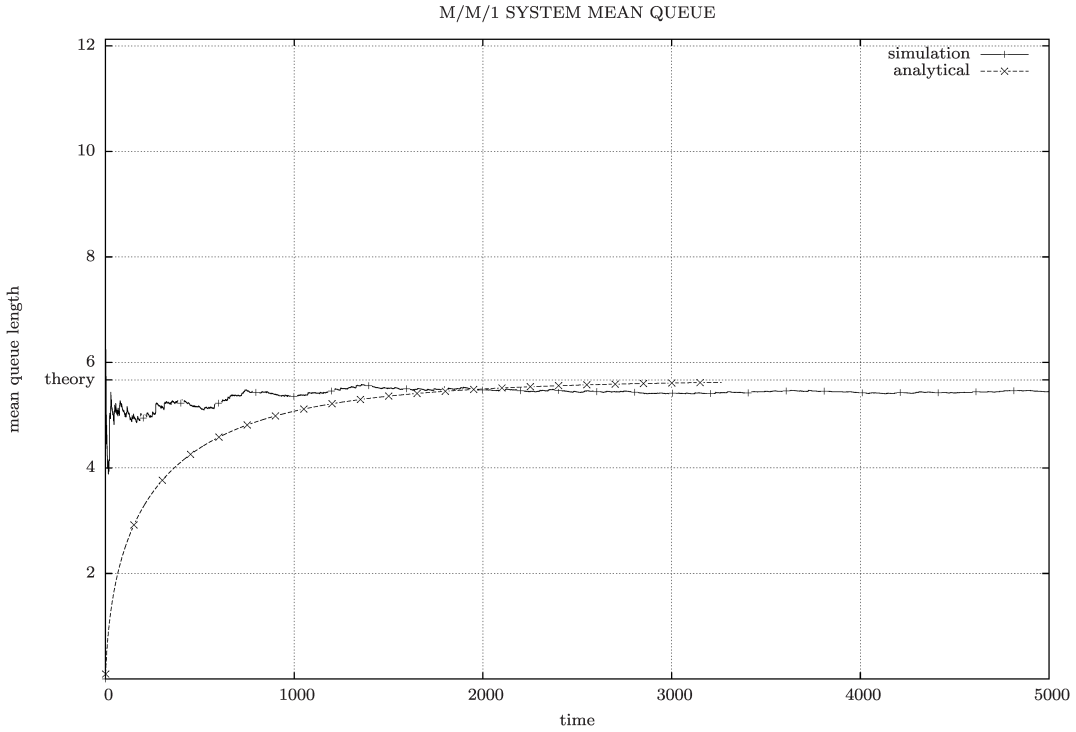


Figure D.2: $M/M/1$ simulation, $\rho = 0.85$

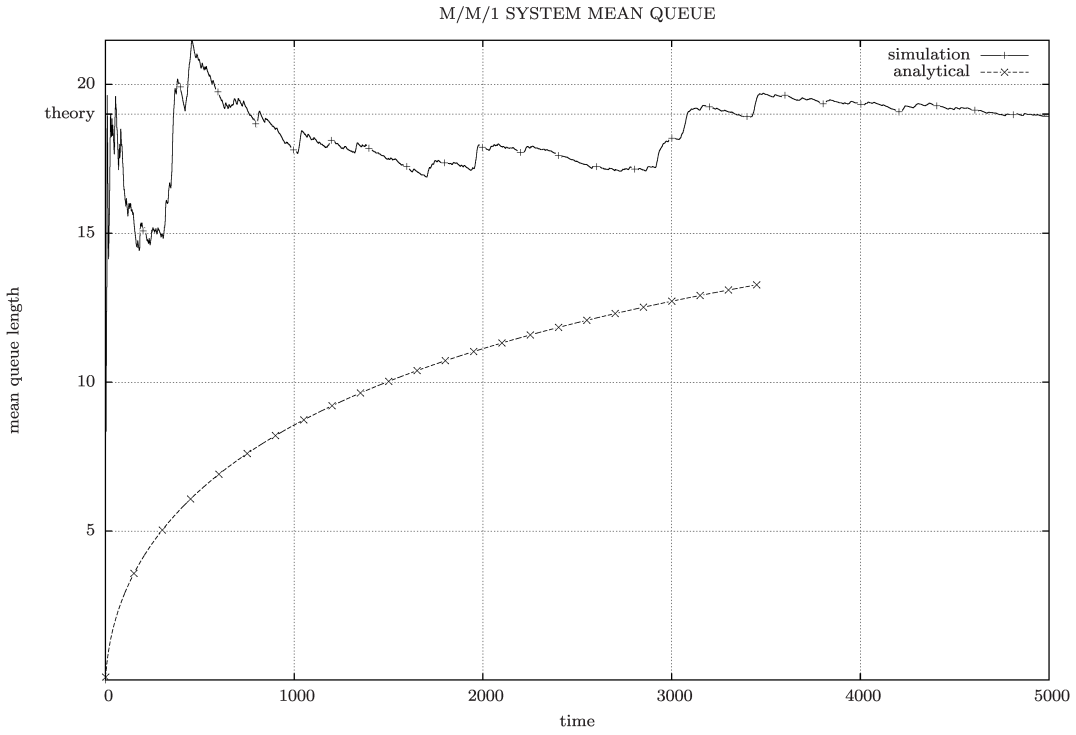


Figure D.3: $M/M/1$ simulation, $\rho = 0.95$

E

$M/M/1/K$ vs. $P/M/1/K$ modeling

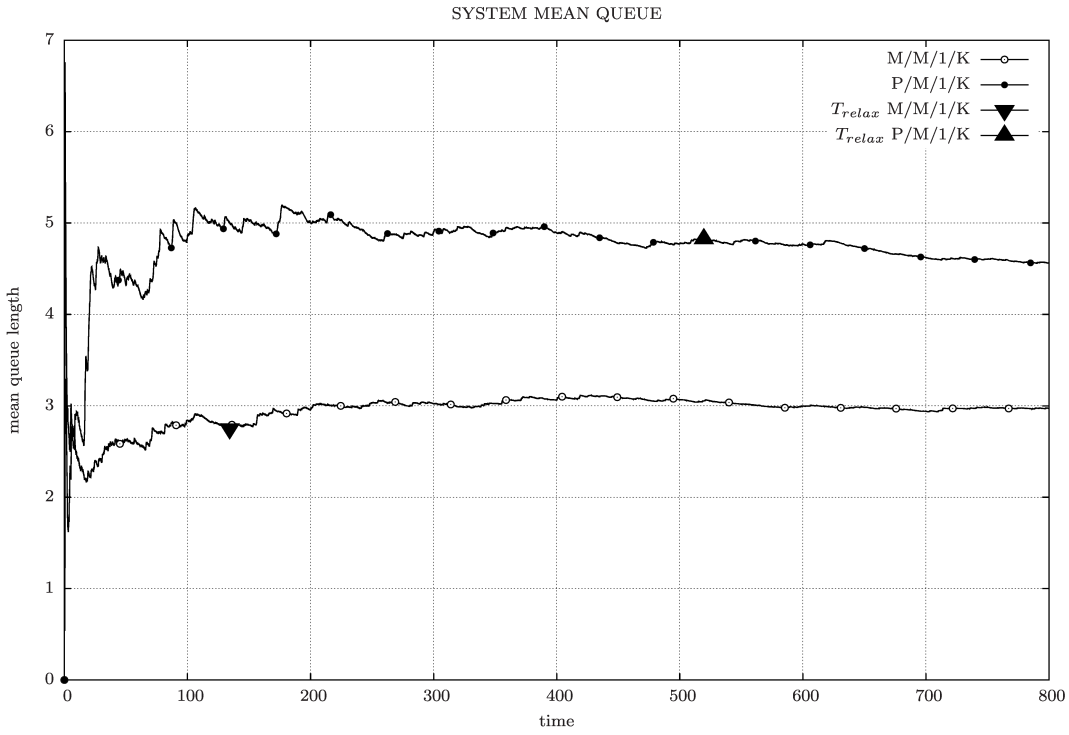


Figure E.1: $M/M/1$ vs. $P/M/1/K$, $\rho = 0.75$, $H = 0.7$

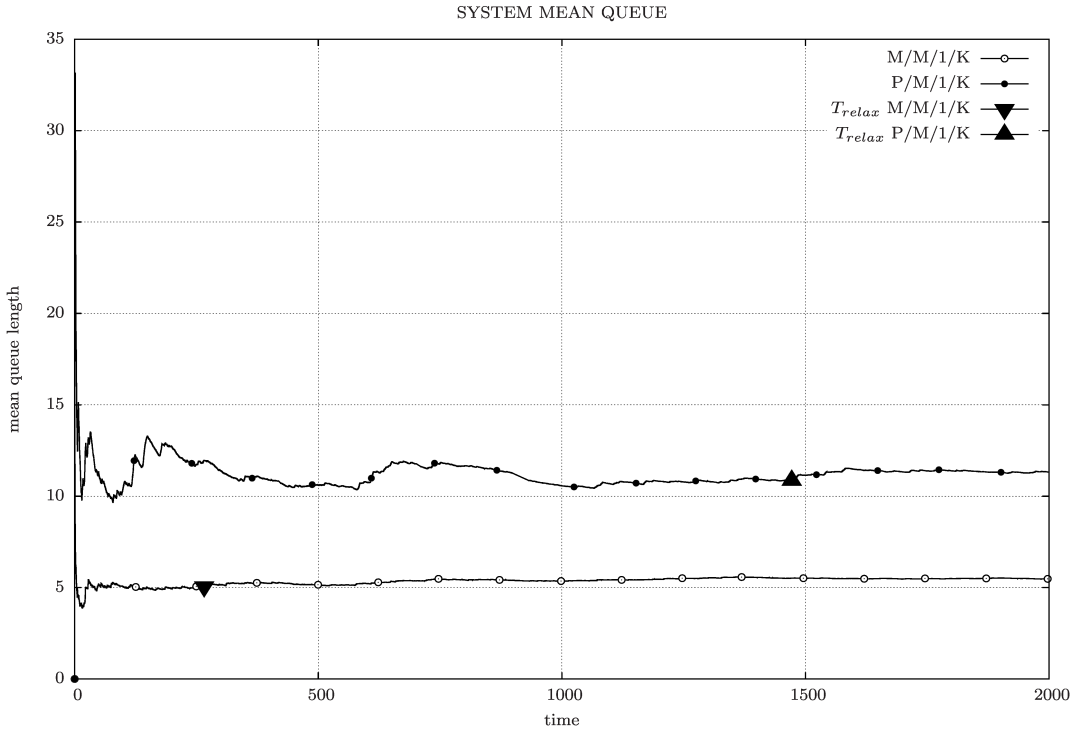


Figure E.2: $M/M/1$ vs. $P/M/1/K$, $\rho = 0.85$, $H = 0.7$

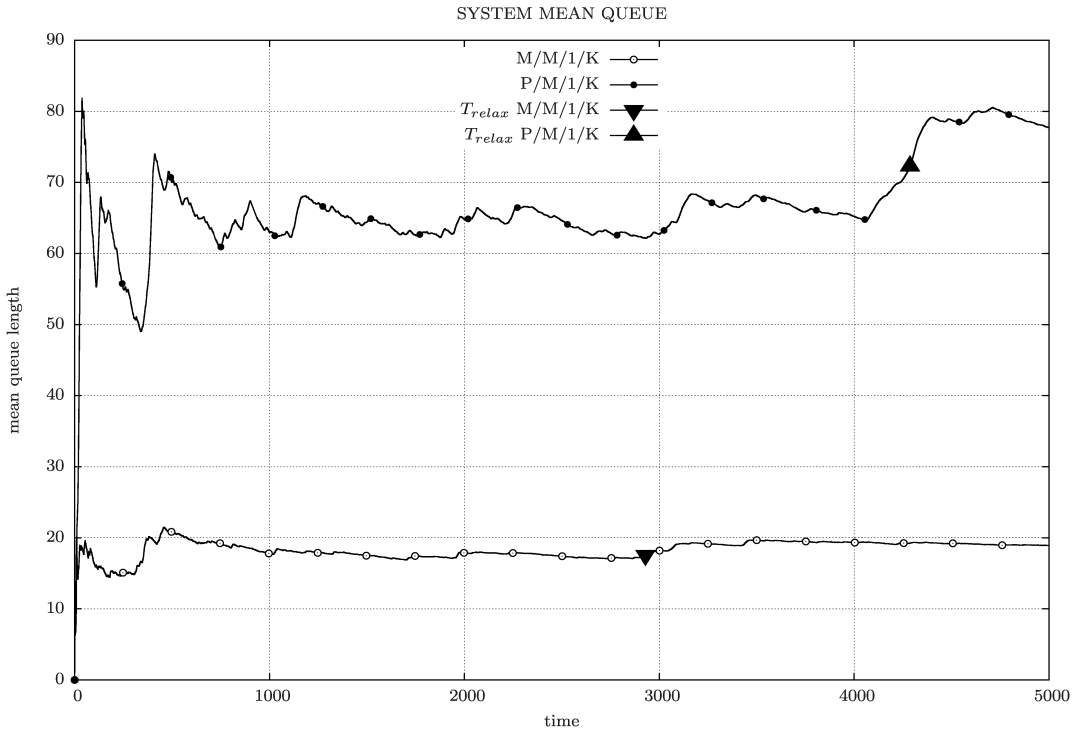


Figure E.3: $M/M/1$ vs. $P/M/1/K$, $\rho = 0.95$, $H = 0.7$

F

$P/M/1/K$ modeling results

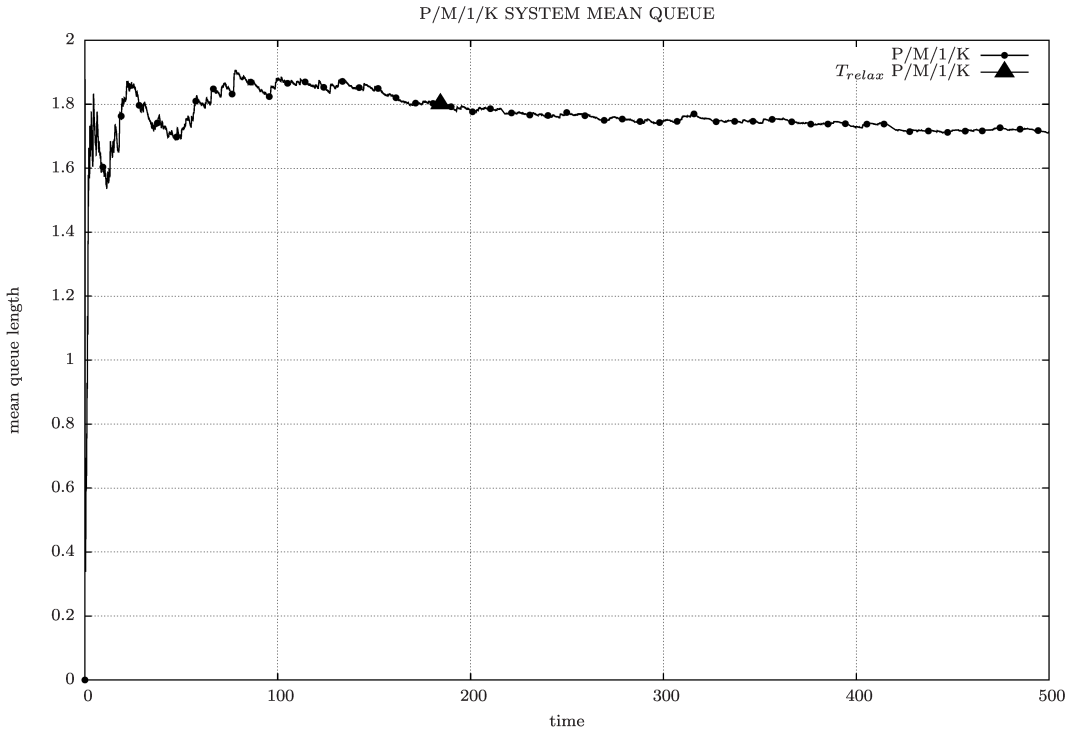


Figure F.1: $P/M/1/K$, $\rho = 0.6$, $H = 0.7$

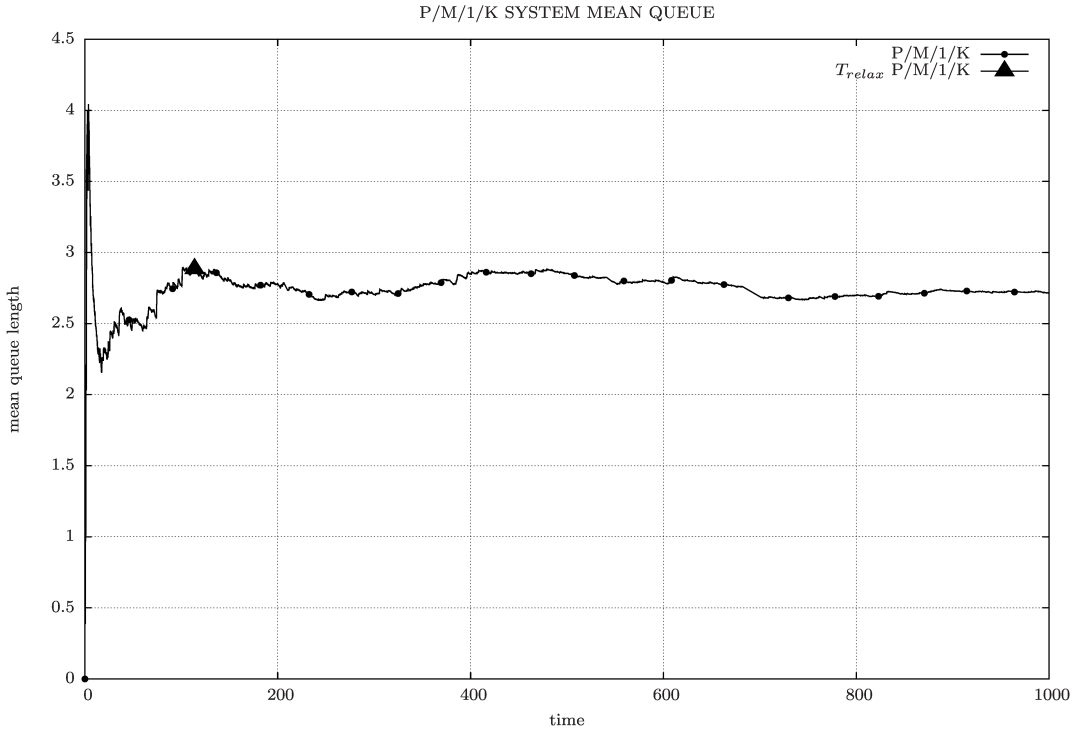


Figure F.2: $P/M/1/K$, $\rho = 0.6$, $H = 0.8$

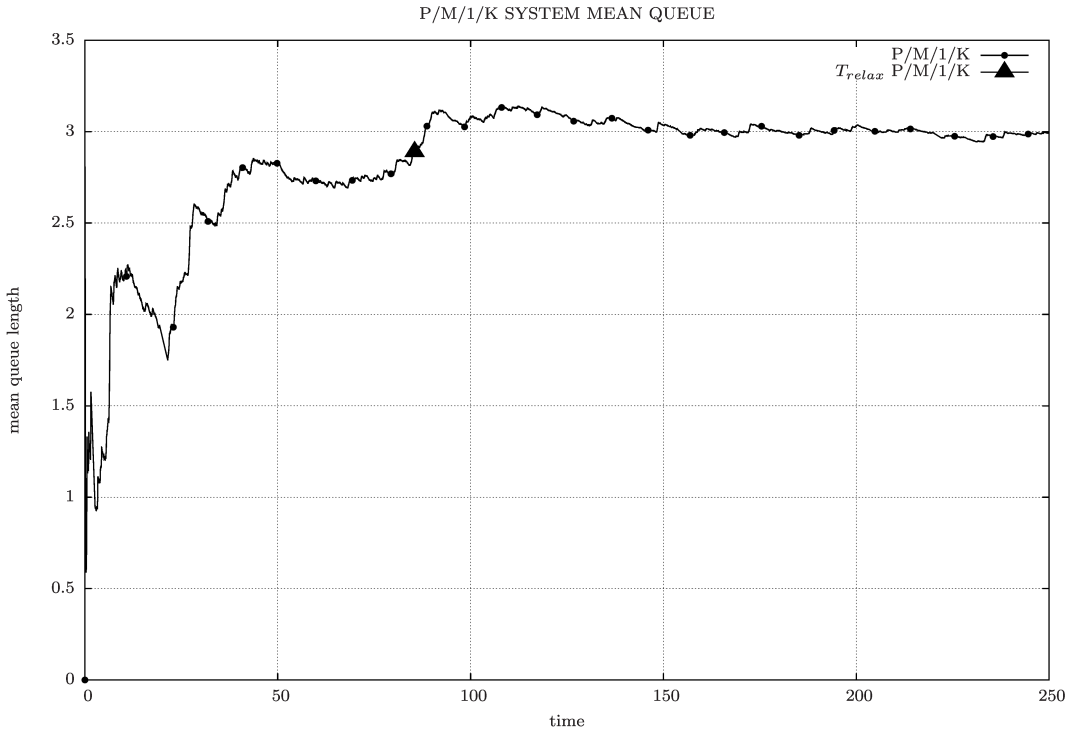


Figure F.3: $P/M/1/K$, $\rho = 0.7$, $H = 0.7$

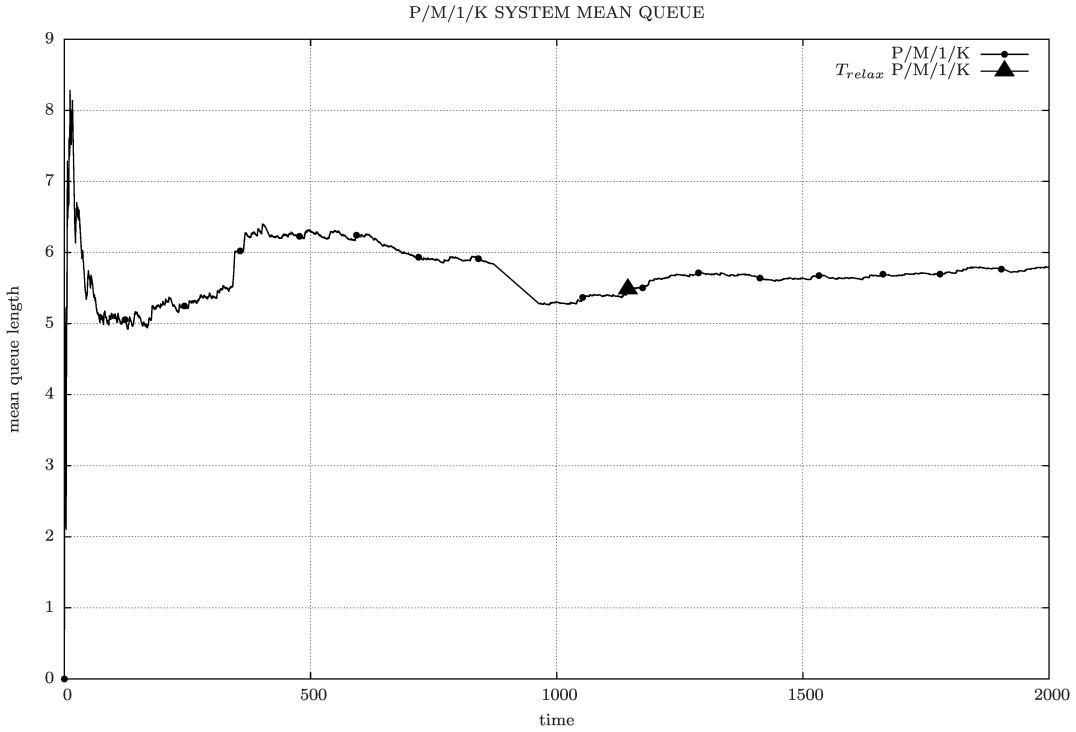


Figure F.4: $P/M/1/K$, $\rho = 0.7$, $H = 0.8$

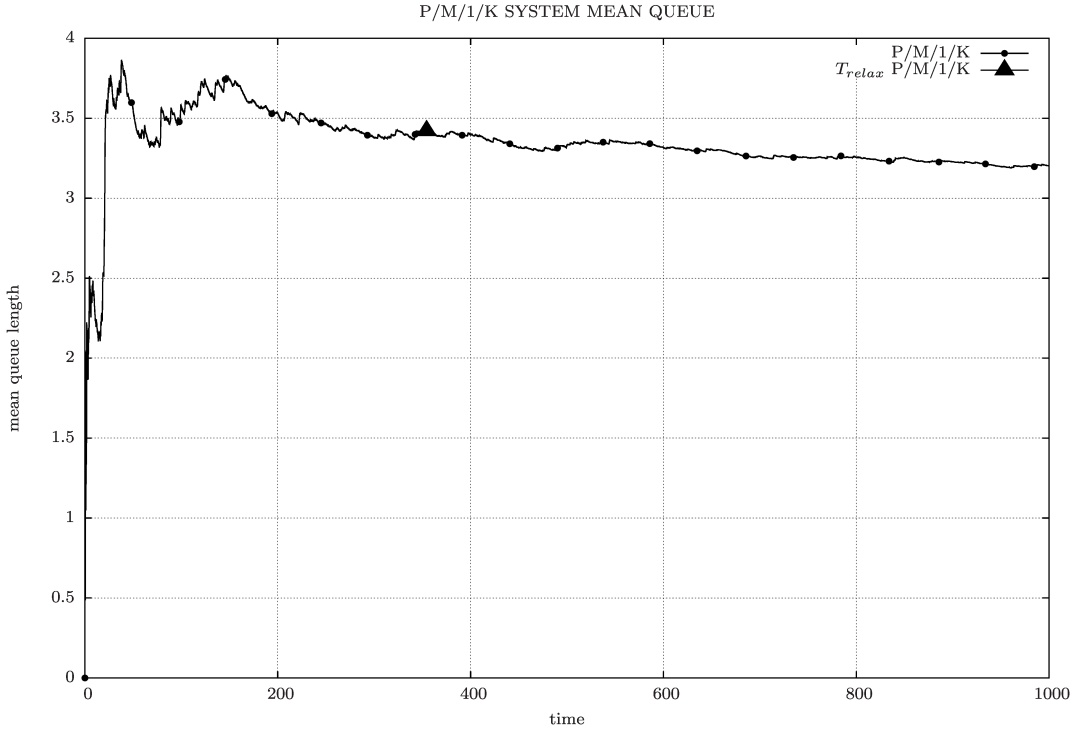


Figure F.5: $P/M/1/K$, $\rho = 0.75$, $H = 0.6$

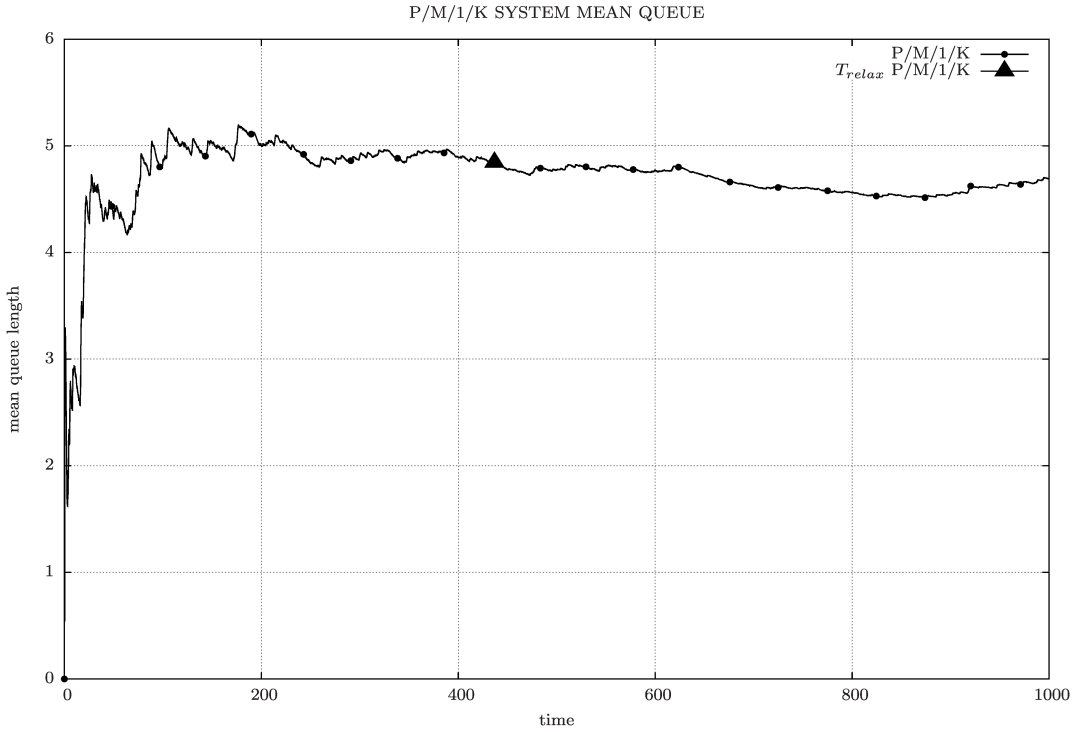


Figure F.6: $P/M/1/K$, $\rho = 0.75$, $H = 0.7$

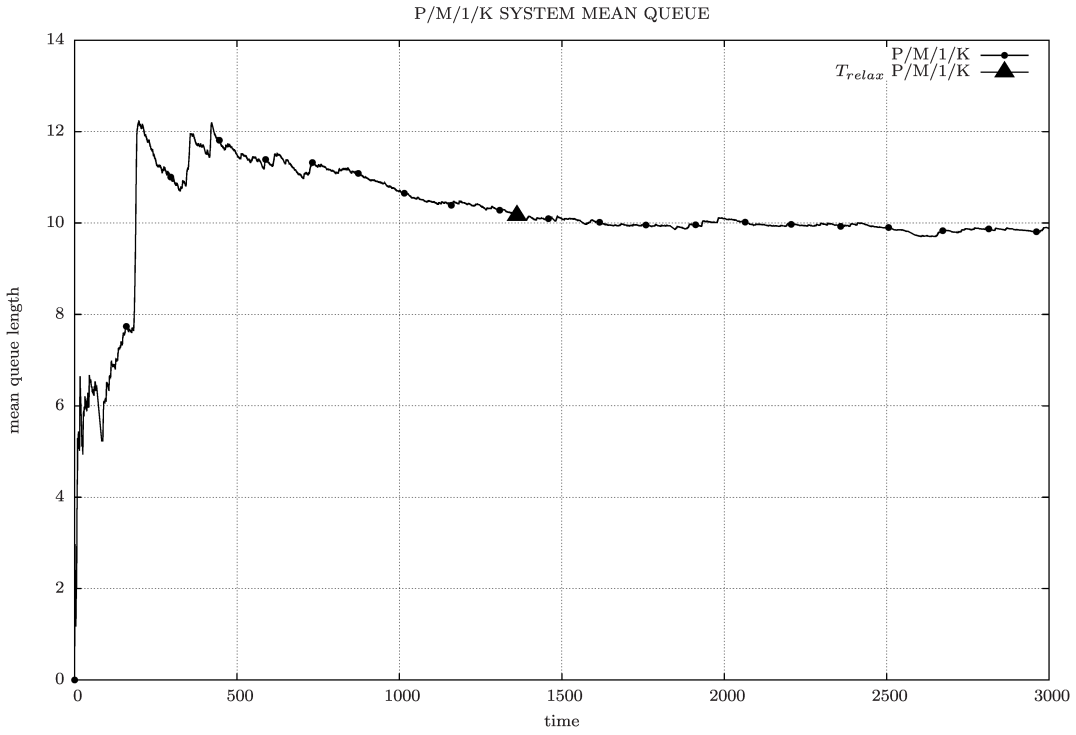


Figure F.7: $P/M/1/K$, $\rho = 0.75$, $H = 0.8$

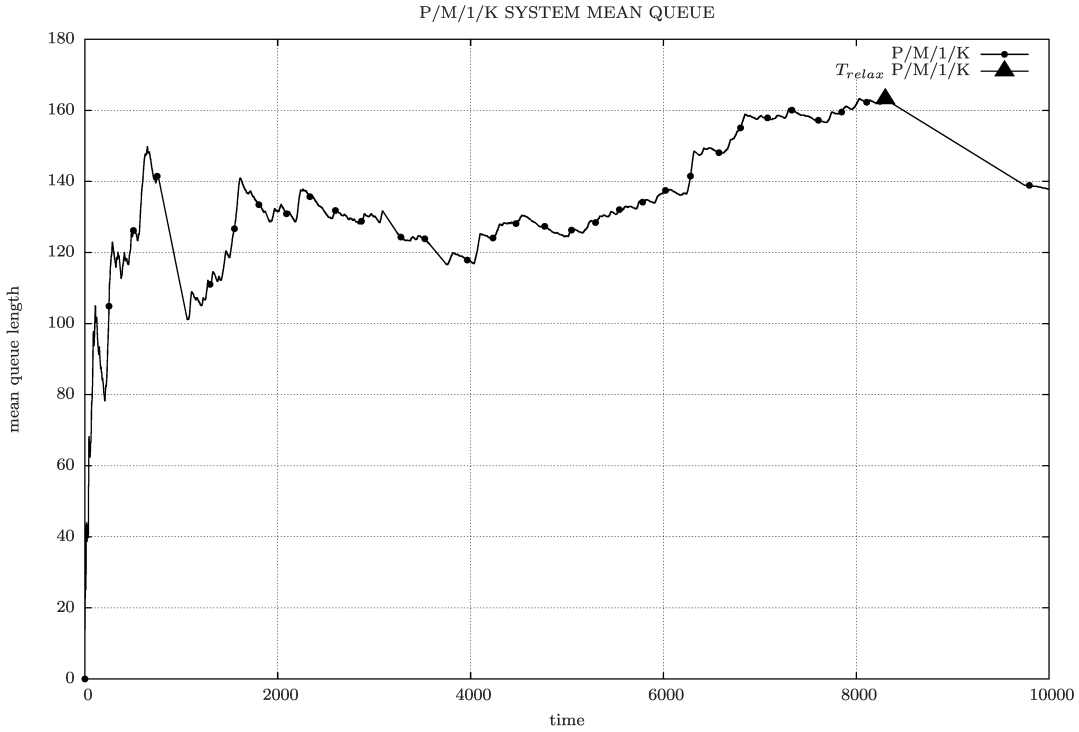


Figure F.8: $P/M/1/K$, $\rho = 0.75$, $H = 0.9$

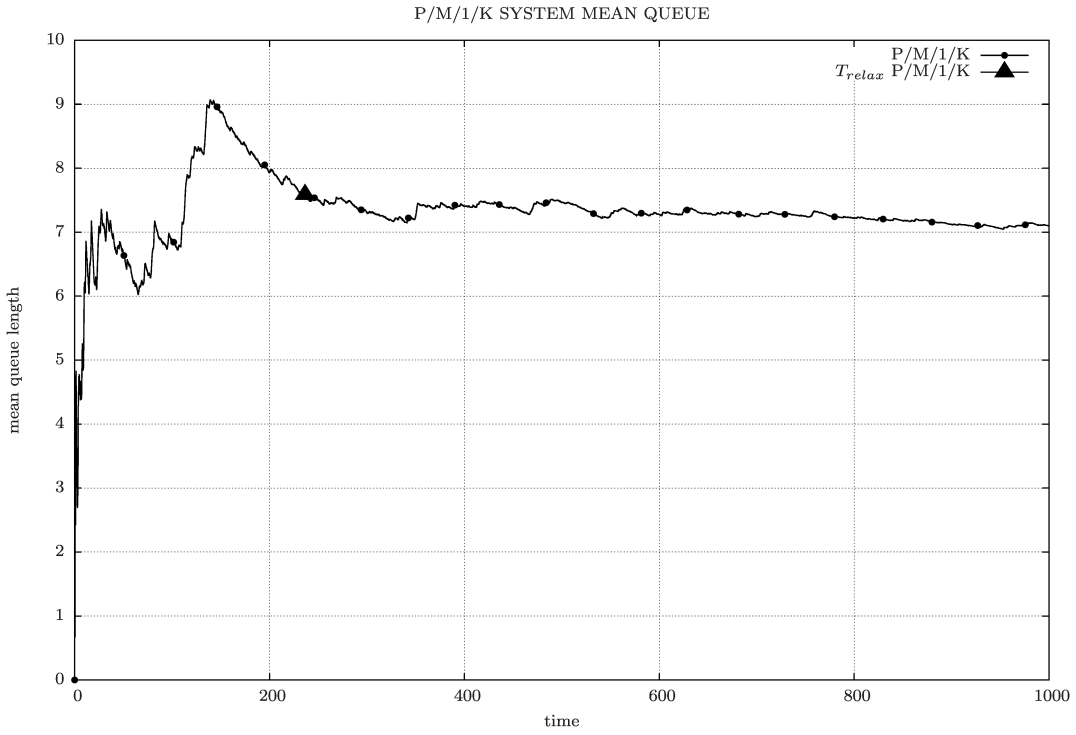


Figure F.9: $P/M/1/K$, $\rho = 0.8$, $H = 0.7$

Figure F.10: $P/M/1/K$, $\rho = 0.8$, $H = 0.8$

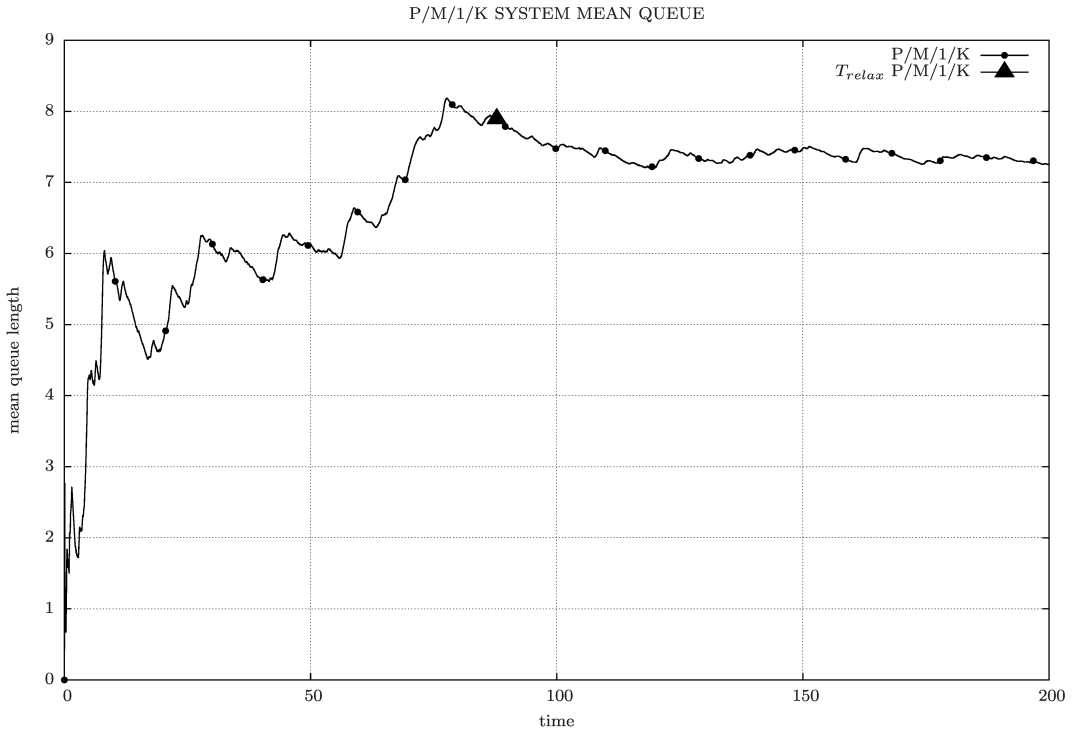


Figure F.11: $P/M/1/K$, $\rho = 0.85$, $H = 0.6$

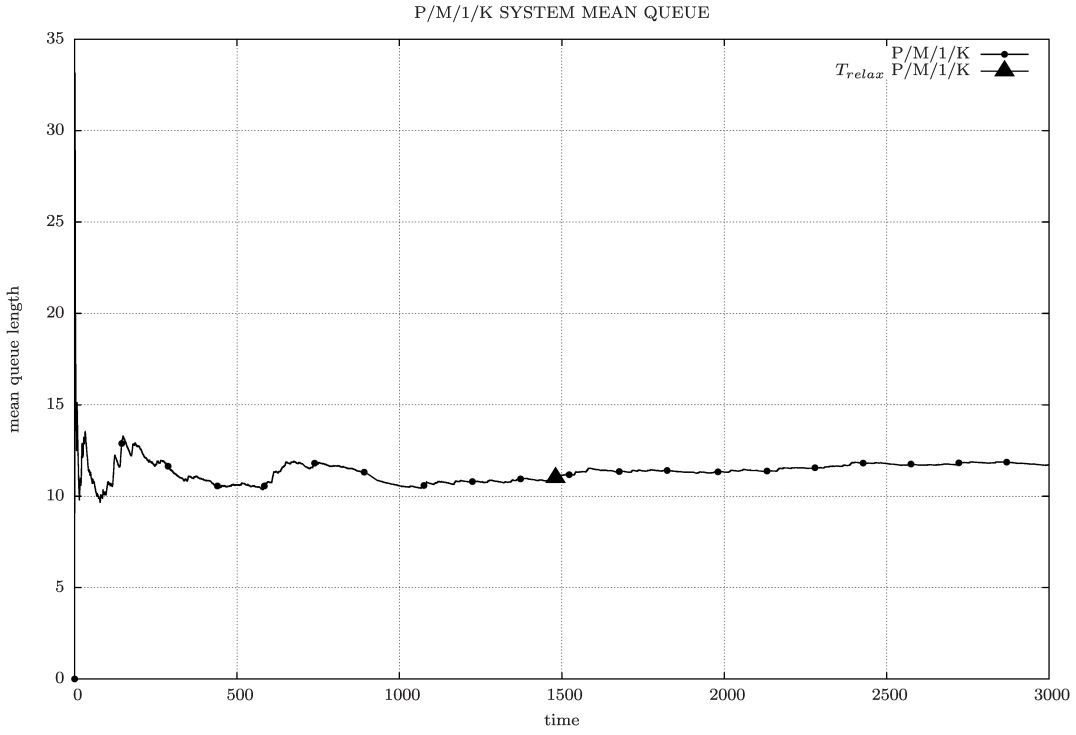


Figure F.12: $P/M/1/K$, $\rho = 0.85$, $H = 0.7$

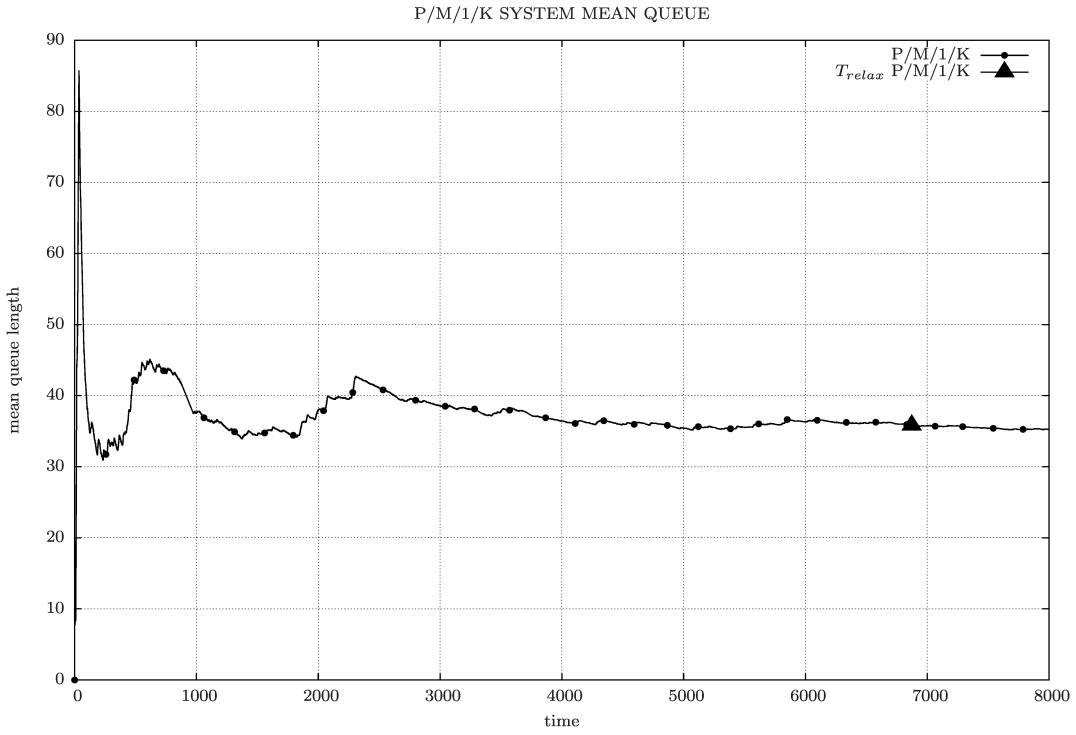


Figure F.13: $P/M/1/K$, $\rho = 0.85$, $H = 0.8$

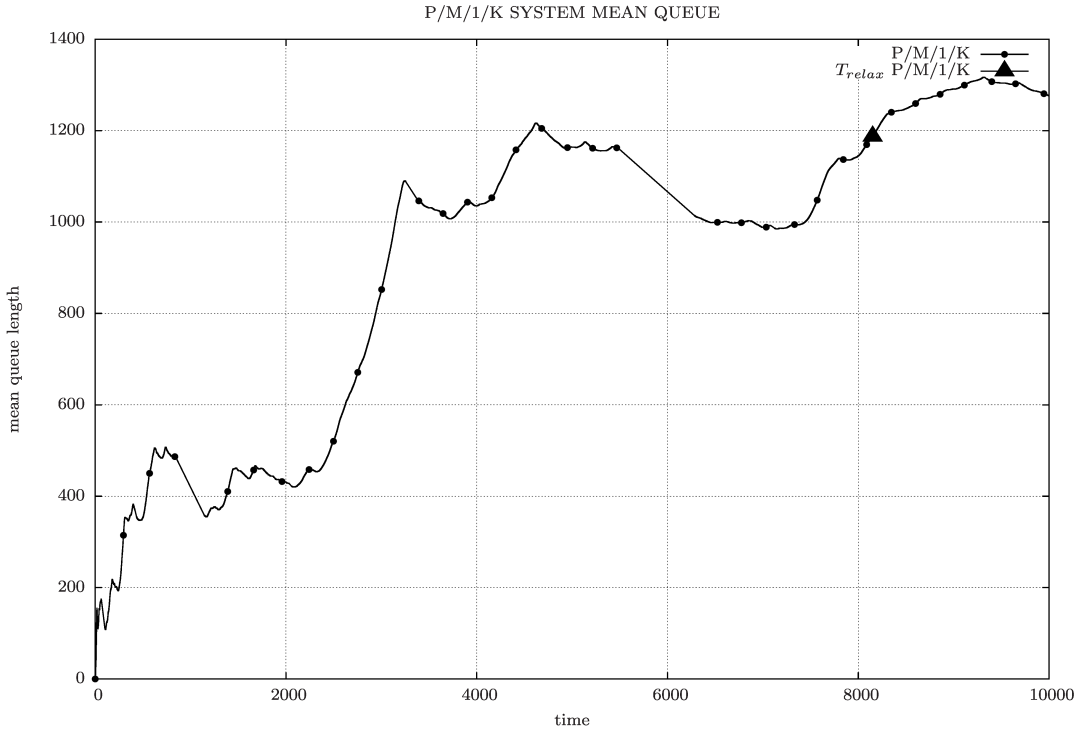


Figure F.14: $P/M/1/K$, $\rho = 0.85$, $H = 0.9$

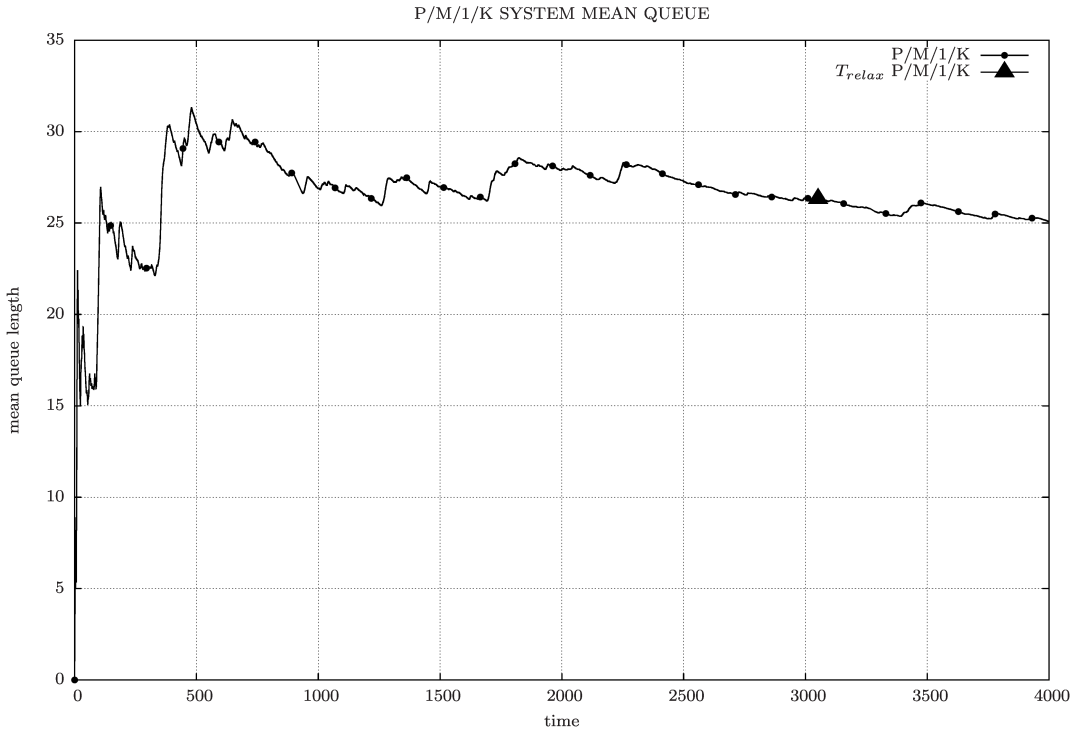


Figure F.15: $P/M/1/K$, $\rho = 0.9$, $H = 0.7$

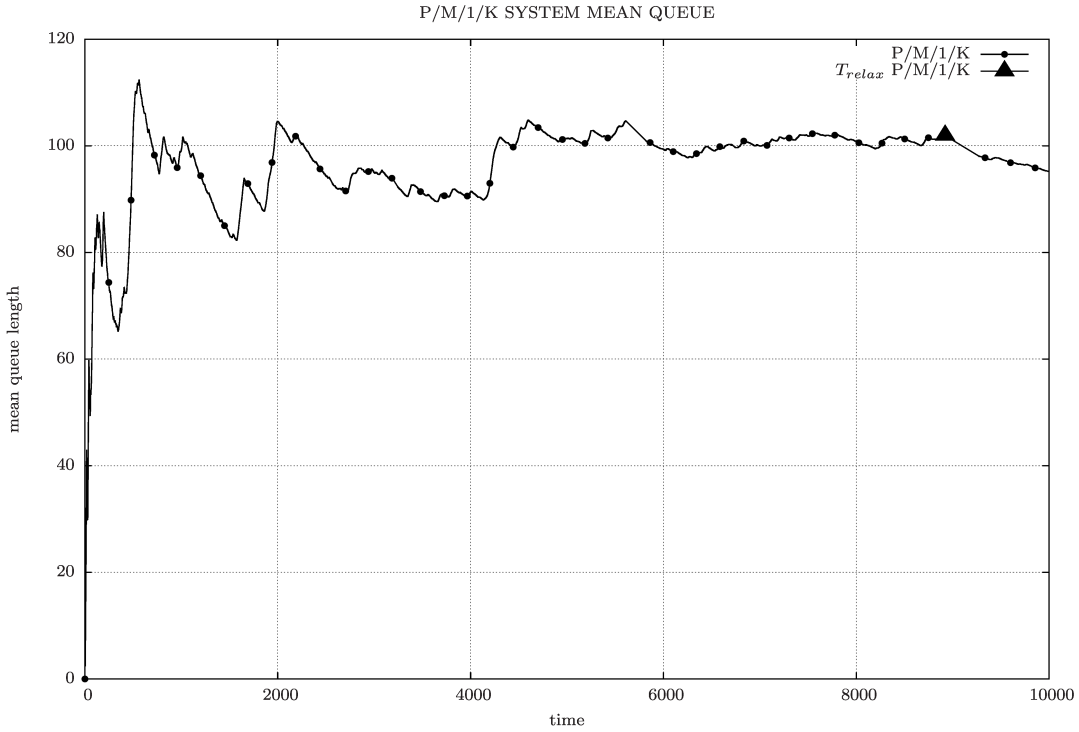


Figure F.16: $P/M/1/K$, $\rho = 0.9$, $H = 0.8$

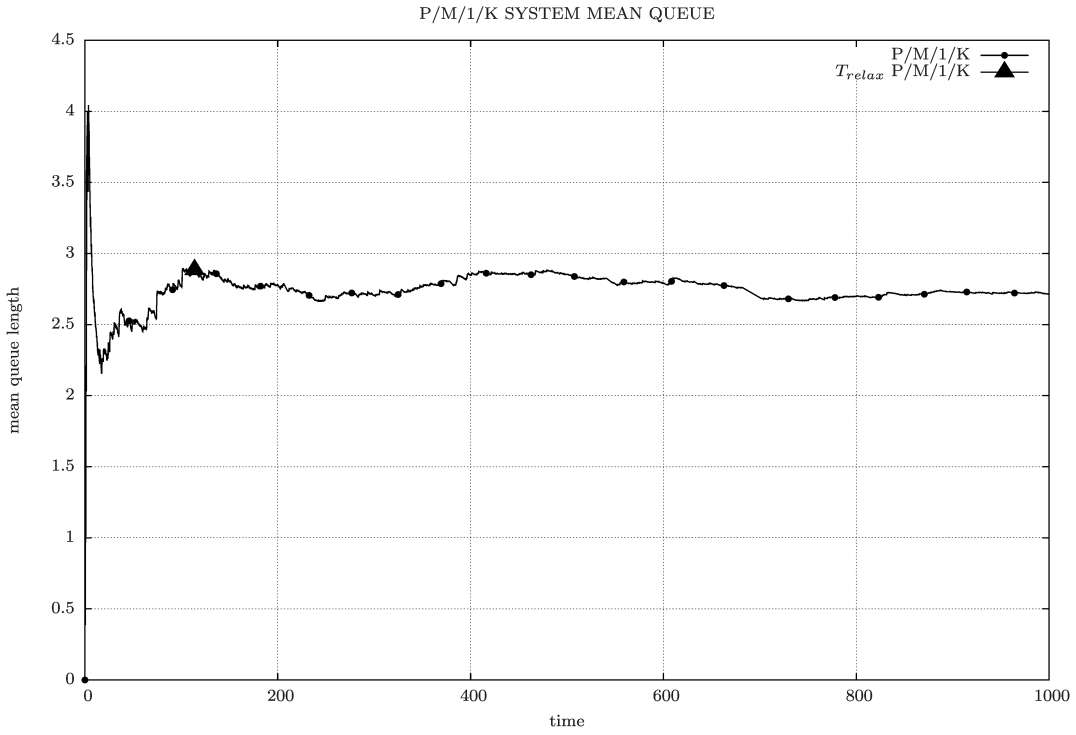


Figure F.17: $P/M/1/K$ simulation, $\rho = 0.6$, $H = 0.8, K = 100$

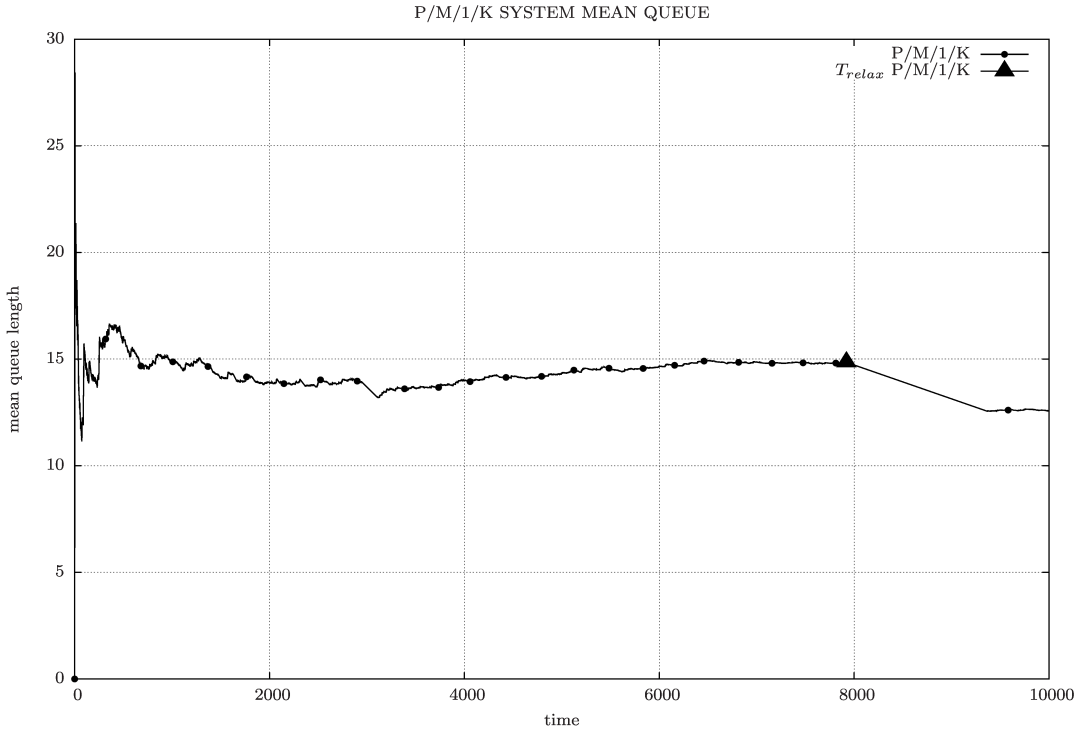


Figure F.18: $P/M/1/K$ simulation, $\rho = 0.6$, $H = 0.9, K = 100$

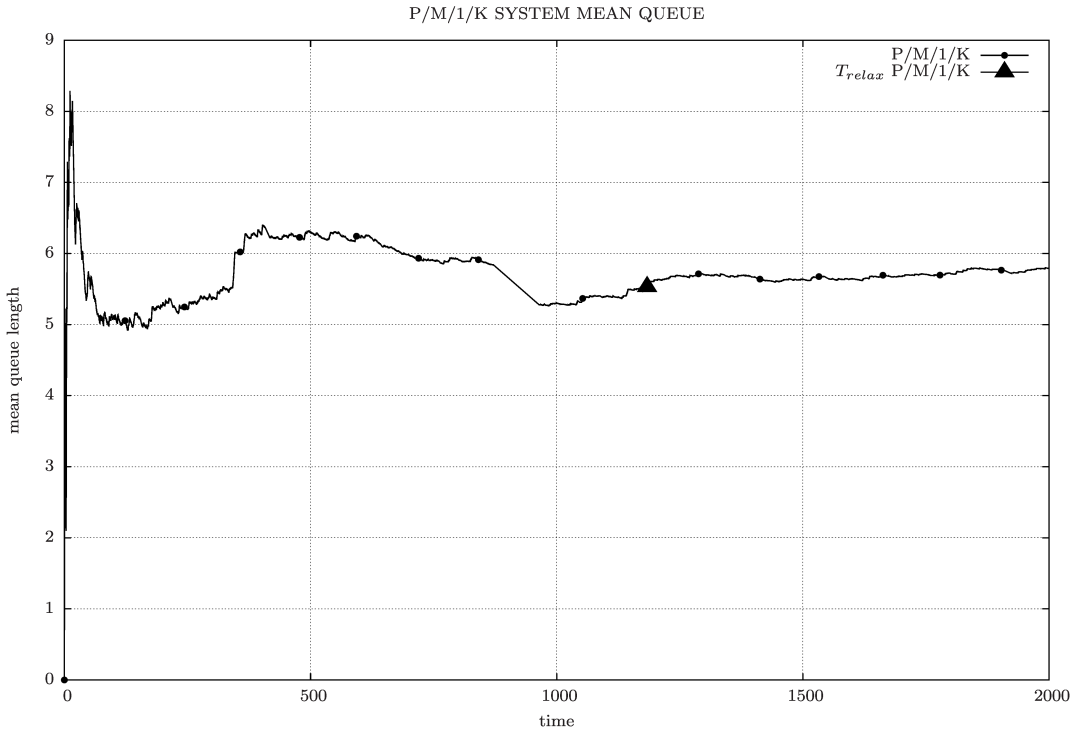


Figure F.19: $P/M/1/K$ simulation, $\rho = 0.7$, $H = 0.8, K = 100$

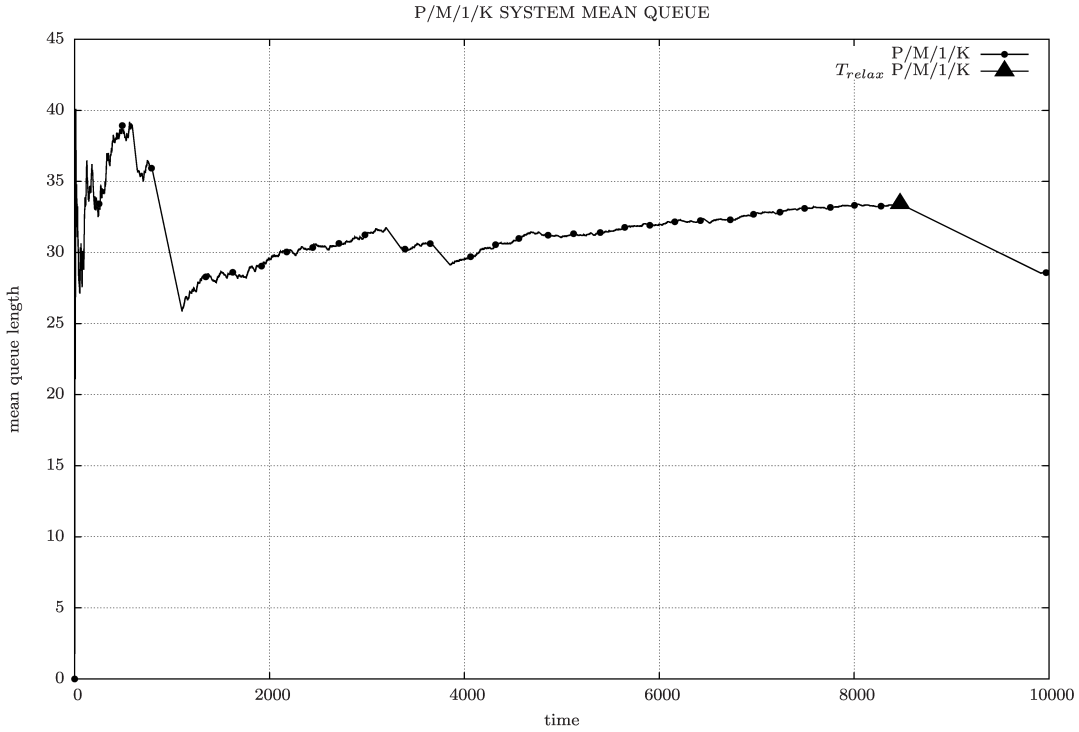


Figure F.20: $P/M/1/K$ simulation, $\rho = 0.7$, $H = 0.9, K = 100$

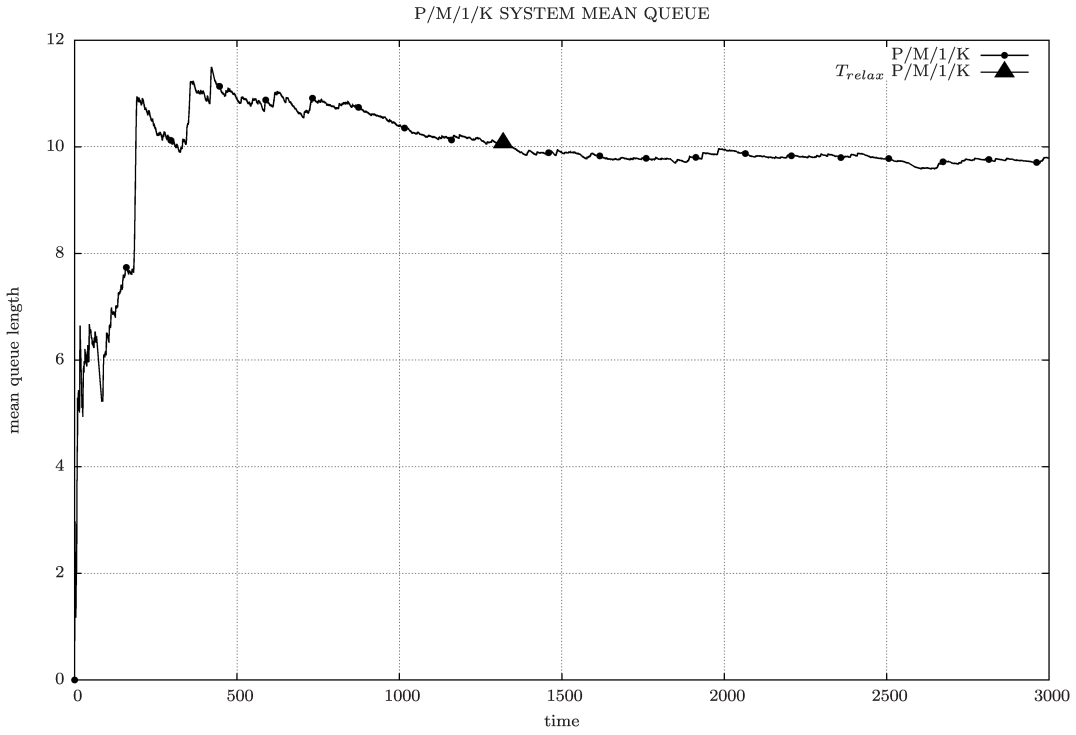


Figure F.21: $P/M/1/K$ simulation, $\rho = 0.75$, $H = 0.8, K = 100$

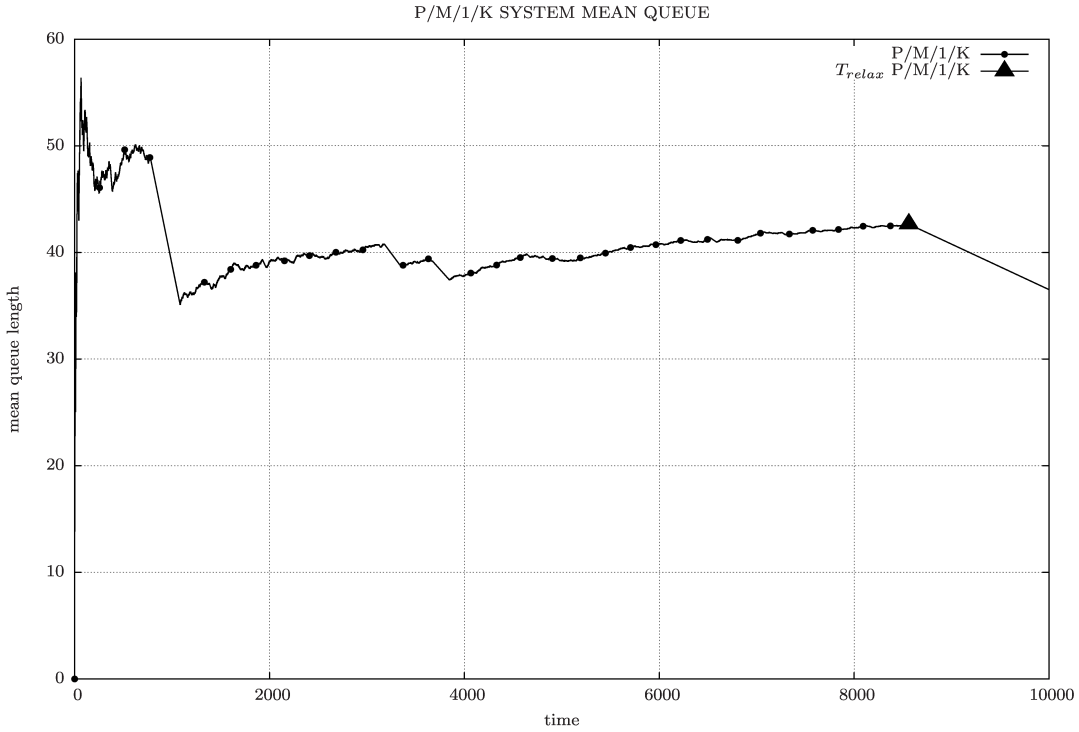


Figure F.22: $P/M/1/K$ simulation, $\rho = 0.75$, $H = 0.9, K = 100$

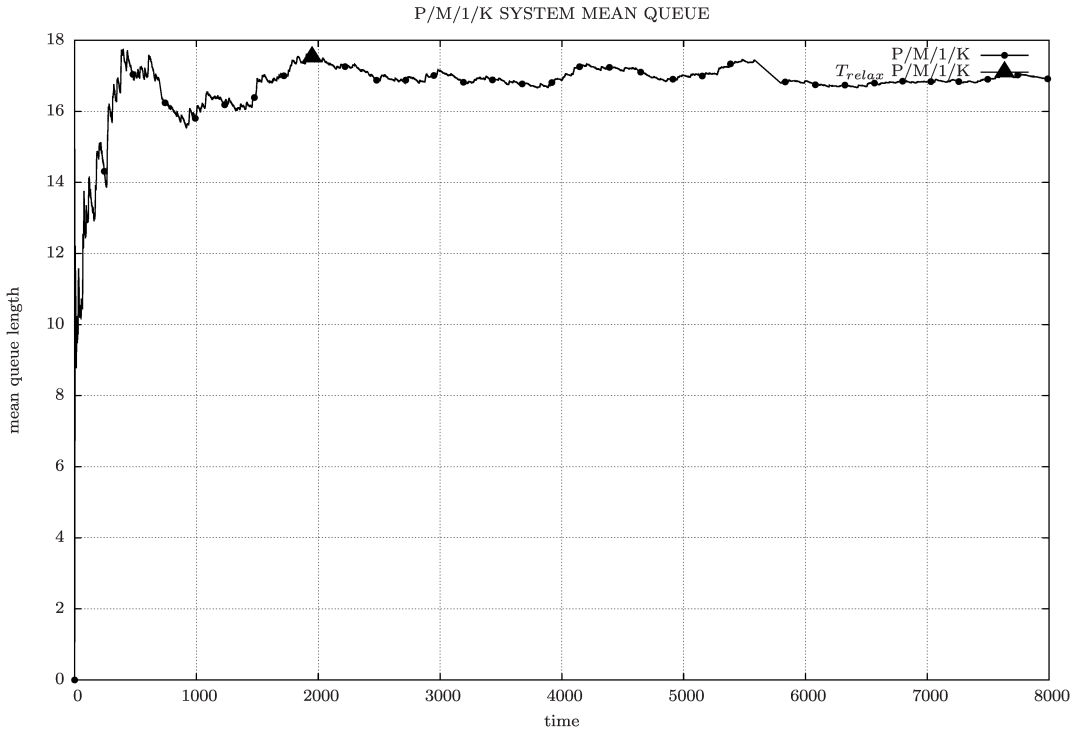


Figure F.23: $P/M/1/K$ simulation, $\rho = 0.8$, $H = 0.8, K = 100$

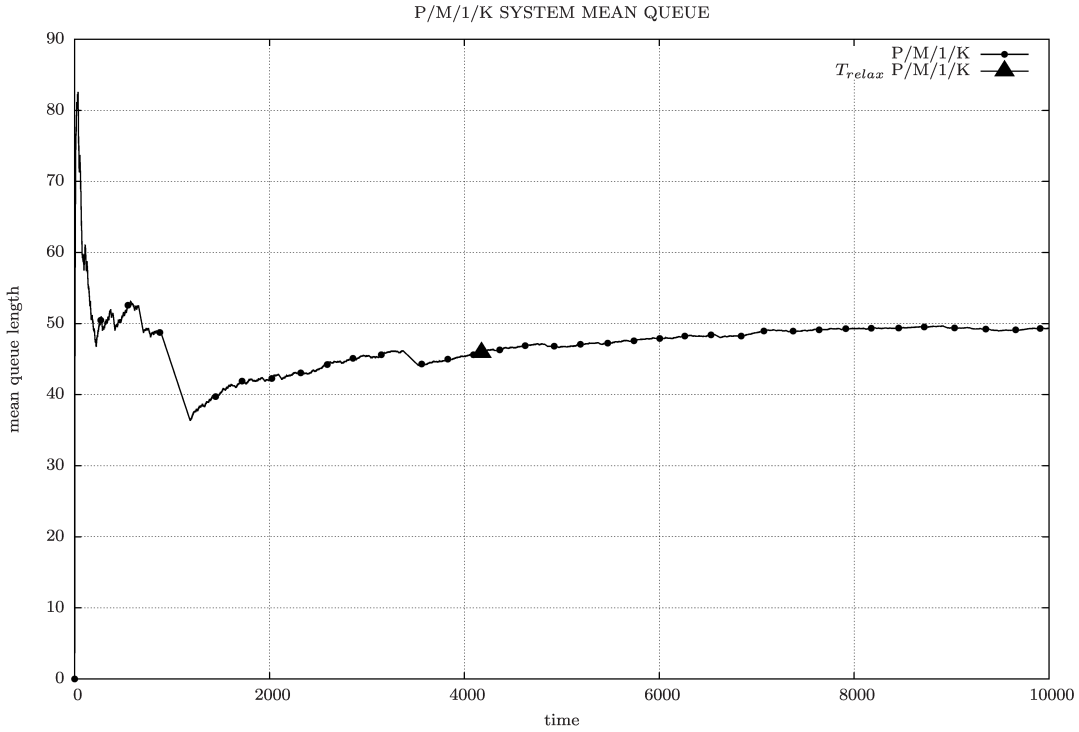


Figure F.24: $P/M/1/K$ simulation, $\rho = 0.8$, $H = 0.9, K = 100$

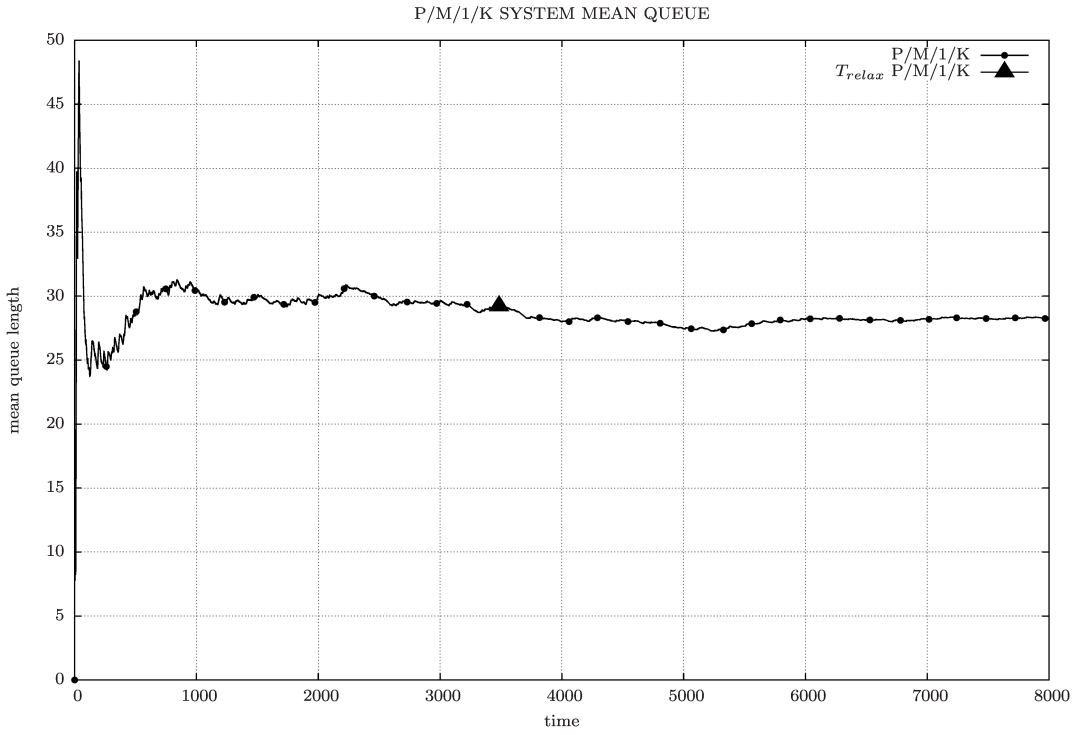


Figure F.25: $P/M/1/K$ simulation, $\rho = 0.85$, $H = 0.8, K = 100$

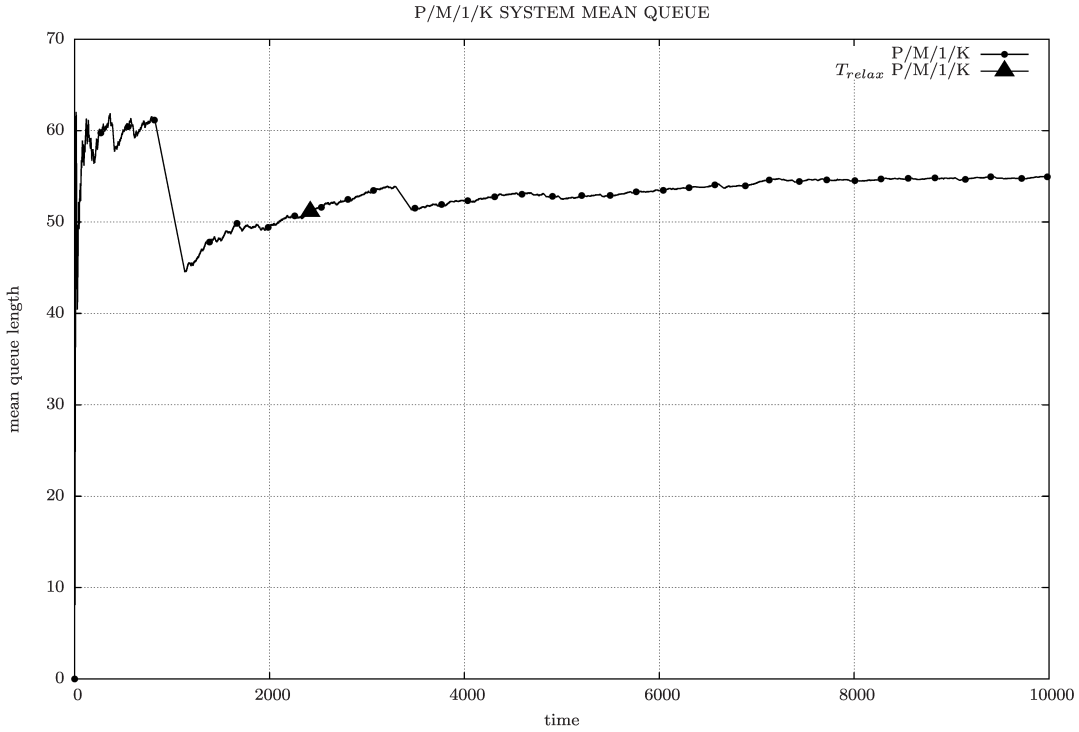


Figure F.26: $P/M/1/K$ simulation, $\rho = 0.85$, $H = 0.9, K = 100$

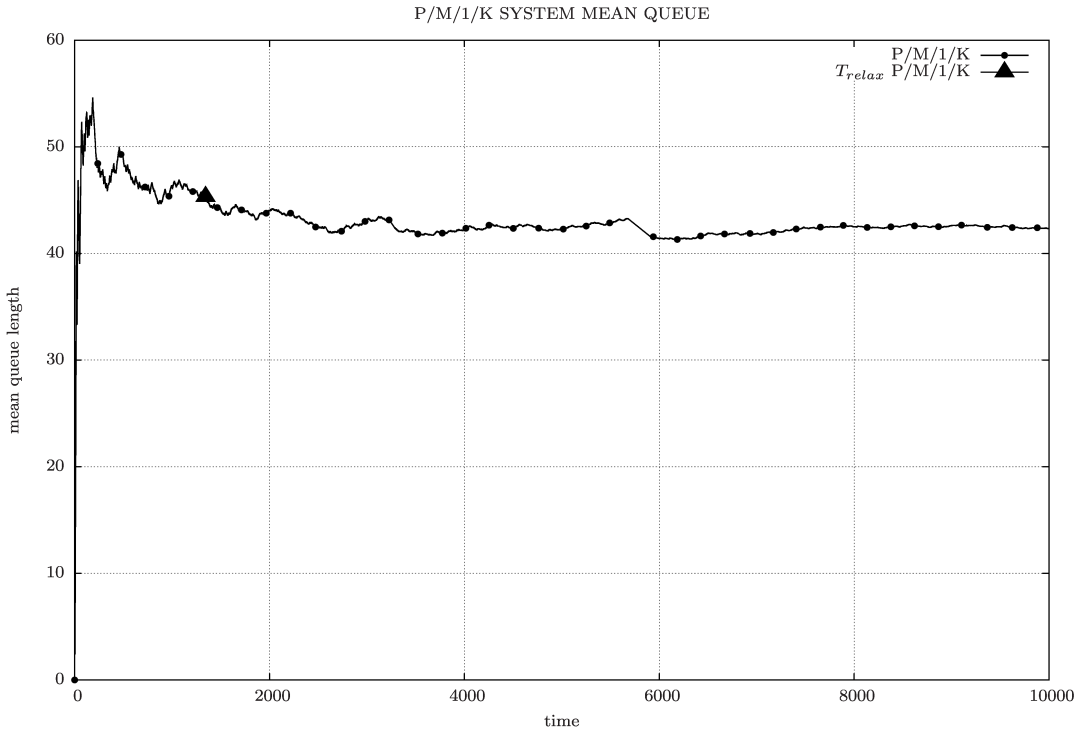


Figure F.27: $P/M/1/K$ simulation, $\rho = 0.9$, $H = 0.8, K = 100$

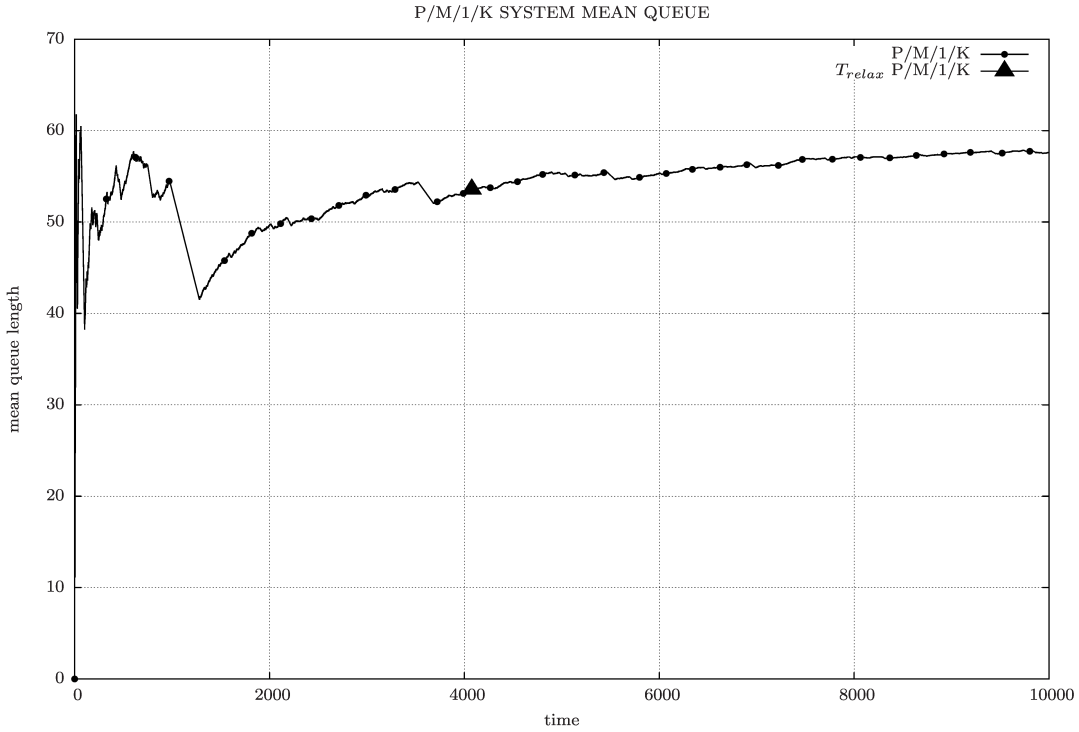


Figure F.28: $P/M/1/K$ simulation, $\rho = 0.9$, $H = 0.9, K = 100$

List of Figures

1.1 Transient processes in wireless networks 3
1.2 Self-similar WEB traffic 9
1.3 Parameters for Hurst analysis 10

2.1 Wireless network traffic, six days period 19
2.2 Wireless network traffic, one day period 20
2.3 Wireless network traffic, one hour period 20
2.4 Wireless network traffic, 20 minutes period 20
2.5 Wireless network traffic, 10 minutes period 20
2.6 Wireless network traffic, 7 minutes period 21
2.7 Wireless network traffic, 5 minutes period 21
2.8 Wireless network traffic, 3 minutes period 21
2.9 Wireless network traffic, 2 minutes period 21
2.10 Example variance-time plot 23
2.11 Example R/S statistics plot 24
2.12 Wireless network traffic, aggregation ratio 2 27
2.13 Wireless network traffic, aggregation ratio 4 27
2.14 Wireless network traffic, aggregation ratio 8 27
2.15 Wireless network traffic, aggregation ratio 16 28
2.16 Wireless network traffic, aggregation ratio 32 28
2.17 Wireless network traffic, aggregation ratio 64 28
2.18 Wireless network traffic, aggregation ratio 128 28
2.19 Wireless network traffic, aggregation ratio 256 29
2.20 Wireless network traffic, aggregation ratio 512 29
2.21 variance-time plot 29
2.22 R/S statistics plot 30
2.23 Autocorrelation of wireless network traffic, full range 30
2.24 Autocorrelation of wireless network traffic, first 10000 values . . 31

3.1 $M/M/1$ system mean query number behavior under different loads 33

3.2 Query number in system, $\rho = 0.75$ . . . 37
3.3 Query number in system, $\rho = 0.95$ . . . 38

4.1 $M/M/1$ system utilization, $\lambda = 0.7$, $\mu = 1$ . . . 42
4.2 $P/M/1$ system utilization, $\lambda = 0.5$, $\mu = 1$ . . . 43
4.3 Simulated system timing example . . . 45
4.4 Simplified simulation algorithm . . . 51
4.5 $M/M/1$ simulation, $\rho = 0.6$ . . . 52
4.6 $M/M/1$ simulation, $\rho = 0.7$ . . . 53
4.7 $M/M/1$ simulation, $\rho = 0.8$ . . . 53
4.8 $M/M/1$ simulation, $\rho = 0.9$ . . . 54
4.9 $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.6$, $H = 0.7$ . . . 55
4.10 $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.7$, $H = 0.7$ . . . 56
4.11 $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.8$, $H = 0.7$ . . . 56
4.12 $M/M/1/K$ vs. $P/M/1/K$, $\rho = 0.9$, $H = 0.7$ . . . 57
4.13 $P/M/1/K$ simulation, $\rho = 0.6$ . . . 58
4.14 $P/M/1/K$ simulation, $\rho = 0.7$ . . . 58
4.15 $P/M/1/K$ simulation, $\rho = 0.8$ . . . 59
4.16 $P/M/1/K$ simulation, $\rho = 0.9$ . . . 59
4.17 $P/M/1/K$ simulation, $\rho = 0.85$, $H = 0.9, K = 100$ . . . 61
4.18 $P/M/1/K$ simulation, $\rho = 0.9$, $H = 0.9, K = 100$ . . . 61
4.19 T_{relax} dependence on H, different ρ . . . 63
4.20 $T_{relax}(H)$ Verhulst regression, $\rho = 0.6$ . . . 65
4.21 $T_{relax}(H)$ Verhulst regression, $\rho = 0.7$ . . . 65
4.22 $T_{relax}(H)$ Verhulst regression, $\rho = 0.8$ . . . 66
4.23 $T_{relax}(H)$ Verhulst regression, $\rho = 0.9$ . . . 66
4.24 Estimated H for 90th iteration of x, $r = 3.6$ . . . 69

B.1 Wireless network traffic, outgoing, six days period . . . 104
B.2 Wireless network traffic, outgoing, one day period . . . 105
B.3 Wireless network traffic, outgoing, one hour period . . . 105
B.4 Wireless network traffic, outgoing, 20 minutes period . . . 105
B.5 Wireless network traffic, outgoing, 10 minutes period . . . 105
B.6 Wireless network traffic, outgoing, 7 minutes period . . . 106
B.7 Wireless network traffic, outgoing, 5 minutes period . . . 106
B.8 Wireless network traffic, outgoing, 3 minutes period . . . 106
B.9 Wireless network traffic, outgoing, 2 minutes period . . . 106
B.10 variance-time plot . . . 107
B.11 R/S statistics plot . . . 107

B.12 Autocorrelation of wireless network traffic, full range 108
B.13 Autocorrelation of wireless network traffic, first 10000 values . . 108

C.1 Query number in system, $C_2 = 1$ 109
C.2 Query number in system, $C_2 = 5$ 110
C.3 Query number in system, $C_2 = 50$ 110
C.4 Query number in system, $C_2 = 100$ 110
C.5 Query loss probability, $C_2 = 1$ 111
C.6 Query loss probability, $C_2 = 5$ 111
C.7 Query loss probability, $C_2 = 50$ 111
C.8 Query loss probability, $C_2 = 100$ 112

D.1 $M/M/1$ simulation, $\rho = 0.75$ 113
D.2 $M/M/1$ simulation, $\rho = 0.85$ 114
D.3 $M/M/1$ simulation, $\rho = 0.95$ 114

E.1 $M/M/1$ vs. $P/M/1/K$, $\rho = 0.75$, $H = 0.7$ 115
E.2 $M/M/1$ vs. $P/M/1/K$, $\rho = 0.85$, $H = 0.7$ 116
E.3 $M/M/1$ vs. $P/M/1/K$, $\rho = 0.95$, $H = 0.7$ 116

F.1 $P/M/1/K$, $\rho = 0.6$, $H = 0.7$ 117
F.2 $P/M/1/K$, $\rho = 0.6$, $H = 0.8$ 118
F.3 $P/M/1/K$, $\rho = 0.7$, $H = 0.7$ 118
F.4 $P/M/1/K$, $\rho = 0.7$, $H = 0.8$ 119
F.5 $P/M/1/K$, $\rho = 0.75$, $H = 0.6$ 119
F.6 $P/M/1/K$, $\rho = 0.75$, $H = 0.7$ 120
F.7 $P/M/1/K$, $\rho = 0.75$, $H = 0.8$ 120
F.8 $P/M/1/K$, $\rho = 0.75$, $H = 0.9$ 121
F.9 $P/M/1/K$, $\rho = 0.8$, $H = 0.7$ 121
F.10 $P/M/1/K$, $\rho = 0.8$, $H = 0.8$ 122
F.11 $P/M/1/K$, $\rho = 0.85$, $H = 0.6$ 122
F.12 $P/M/1/K$, $\rho = 0.85$, $H = 0.7$ 123
F.13 $P/M/1/K$, $\rho = 0.85$, $H = 0.8$ 123
F.14 $P/M/1/K$, $\rho = 0.85$, $H = 0.9$ 124
F.15 $P/M/1/K$, $\rho = 0.9$, $H = 0.7$ 124
F.16 $P/M/1/K$, $\rho = 0.9$, $H = 0.8$ 125
F.17 $P/M/1/K$ simulation, $\rho = 0.6$, $H = 0.8, K = 100$ 125
F.18 $P/M/1/K$ simulation, $\rho = 0.6$, $H = 0.9, K = 100$ 126
F.19 $P/M/1/K$ simulation, $\rho = 0.7$, $H = 0.8, K = 100$ 126
F.20 $P/M/1/K$ simulation, $\rho = 0.7$, $H = 0.9, K = 100$ 127

F.21 $P/M/1/K$ simulation, $\rho = 0.75$, $H = 0.8, K = 100$ 127
F.22 $P/M/1/K$ simulation, $\rho = 0.75$, $H = 0.9, K = 100$ 128
F.23 $P/M/1/K$ simulation, $\rho = 0.8$, $H = 0.8, K = 100$ 128
F.24 $P/M/1/K$ simulation, $\rho = 0.8$, $H = 0.9, K = 100$ 129
F.25 $P/M/1/K$ simulation, $\rho = 0.85$, $H = 0.8, K = 100$ 129
F.26 $P/M/1/K$ simulation, $\rho = 0.85$, $H = 0.9, K = 100$ 130
F.27 $P/M/1/K$ simulation, $\rho = 0.9$, $H = 0.8, K = 100$ 130
F.28 $P/M/1/K$ simulation, $\rho = 0.9$, $H = 0.9, K = 100$ 131

List of Tables

3.1 Query number in system n . . . 38
3.2 denial of service probability P_{rej} . . . 38
3.3 relaxation times T_{relax} . . . 38

4.1 GPSS simulation vs. analytical expectations . . . 42
4.2 Simulation vs. queuing theory . . . 54
4.3 $M/M/1/K$ vs. $P/M/1/K$ mean query number . . . 57
4.4 $M/M/1/K$ vs. $P/M/1/K$ relaxation times . . . 57
4.5 $P/M/1/K$ mean query number $\overline{n}$ under different parameters . . . 59
4.6 $P/M/1/K$ system utilization U_p under different parameters . . . 60
4.7 $P/M/1/K$ system relaxation time T_{relax} under different parameters 60
4.8 $P/M/1/K$ mean query number $\overline{n}$, $K = 100$ . . . 62
4.9 $P/M/1/K$ system utilization U_p, $K = 100$ . . . 62
4.10 $P/M/1/K$ system relaxation time T_{relax}, $K = 100$ . . . 62
4.11 $P/M/1/K$ packet loss probability P_{loss}, $K = 100$ . . . 62
4.12 $T_{relax}(H)$ regression errors . . . 64
4.13 $T_{relax}(H)$ regression estimated parameters . . . 67
4.14 Generator test overview . . . 69

Acronyms

3G 3rd Generation (mobile network).

4G 4th Generation (mobile network).

CPU Central Processing Unit.

DSL Digital Subscriber Line.

GPRS General Packet Radio Service.

GPSS General Purpose Simulation System.

GSL GNU Scientific Library.

GSM Global System for Mobile Communications.

IEEE Institute of Electrical and Electronics Engineers.

IP Internet Protocol.

MBAC Measurement Based Admission Control.

Mbps Megabits per second.

MIB Management Information Base.

OS Operating System.

PC Personal Computer.

QoS Quality of Service.

RAM Random Access Memory.

SNMP Simple Network Management Protocol.

TCP Transport Control Protocol.

UDP User Datagram Protocol.

Wi-Fi Wireless Fidelity.

WWW World Wide Web.

List of Symbols

C variation coefficient.

H Hurst parameter.

I_x first kind, order x modified Bessel function.

K queue capacity, buffer size.

k shape parameter of Pareto distribution.

λ packet inter-arrival rate.

μ packet service rate.

n query number in the system.

$\overline{n}(t)$ mean query number in the system.

P_{loss} packet loss probability.

P_{rej} denial of service probability.

Ψ utilization coefficient.

ρ utilization coefficient.

$R(k)$ autocorrelation function.

$r(k)$ autocorrelation coefficient.

$r^{(m)}(k)$ autocorrelation function of a process $X_{(m)}$.

T_a packet interarrival time.

T_{relax} system relaxation time.

$var(X^m)$ variance of a process $X_{(m)}$.

$X_{(m)}$ a process X aggregated by m values.

x_m scale parameter of Pareto distribution.

References

[1] The bufferbloat projects. http://www.bufferbloat.net/.

[2] E-book readers comparison. http://en.wikipedia.org/wiki/Comparison_of_e-book_readers.

[3] Gnu bash. http://www.gnu.org/software/bash/.

[4] Gnu compiler collection. http://www.gnu.org/software/gcc/.

[5] Gnu octave. http://www.gnu.org/software/octave/.

[6] Gnu scientific library. http://www.gnu.org/software/gsl/.

[7] Gnuplot. http://www.gnuplot.info/.

[8] Gpss world. http://www.minutemansoftware.com/simulation.htm.

[9] Ieee 802.11u. http://standards.ieee.org/findstds/standard/802.11u-2011.html.

[10] Linux kernel archives. http://kernel.org/.

[11] Mrtg. http://oss.oetiker.ch/mrtg/.

[12] Ptc mathcad. http://www.ptc.com/product/mathcad/.

[13] Routerboard. http://routerboard.com/.

[14] Snmp. http://docwiki.cisco.com/wiki/Simple_Network_Management_Protocol.

[15] Robert J. Adler, R.E. Feldman, and M.S. Taqqu. *A practical guide to heavy tails: statistical techniques and applications*. Birkhäuser, 1998.

[16] E.E. Afify. Estimation of parameters for pareto distribution. *Far East Journal of Theoretical Statistics*, 9(2):157–162, 2003.

[17] I. Akyildiz, Y. Altunbasak, F. Fekri, and R. Sivakumar. AdaptNet: an adaptive protocol suite for the next-generation wireless Internet. *Communications Magazine, IEEE*, 42(3):128–136, 2004.

[18] O. Andrisano, R. Verdone, and M. Nakagawa. Intelligent transportation systems: the role of third generation mobile radio networks. *Communications Magazine, IEEE*, 38(9):144 –151, sep 2000.

[19] D. Arnold. Fitting a logistic curve to data. *College of the Redwoods, working material*, 2002.

[20] A. Asars, M. Kulikovs, and E. Petersons. Buffer size and output bandwidth optimization in a MBAC system. *Automatic Control and Computer Sciences*, 43(5):241–246, 2009.

[21] Thomas Bohnert and Edmundo Monteiro. A comment on simulating lrd traffic with pareto on/off sources. In *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, CoNEXT '05, pages 228–229, New York, NY, USA, 2005. ACM.

[22] G. Bolch. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience Publication. Wiley, 1998.

[23] H.J. Chao and X. Guo. *Quality of service control in high-speed networks*. Wiley, 2002.

[24] J. Chen, F. Paganini, R. Wang, MY Sanadidi, and M. Gerla. Fluid-flow analysis of tcp westwood with red. In *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, volume 7, pages 4064–4068. IEEE, 2003.

[25] M.E. Crovella and A. Bestavros. Explaining world wide web traffic self-similarity. Technical report, Boston University Computer Science Department, 1995.

[26] Tadeusz Czachorski and Ferhan Pekergin. Diffusion approximation as a modelling tool. In Demetres Kouvatsos, editor, *Network Performance Engineering*, volume 5233 of *Lecture Notes in Computer Science*, page 447–476. Springer Berlin / Heidelberg, 2011.

[27] A. Das and R. Srikant. Diffusion approximations for a single node accessed by congestion-controlled sources. *Automatic Control, IEEE Transactions on*, 45(10):1783–1799, oct. 2000.

[28] P. De, A. Raniwala, S. Sharma, and T.C. Chiueh. Design considerations for a multihop wireless network testbed. *Communications Magazine, IEEE*, 43(10):102 – 109, oct. 2005.

[29] L. Deri et al. Improving passive packet capture: beyond device polling. In *Proceedings of SANE*, volume 2004, page 85, 2004.

[30] P. Droz and J.Y. Le Boudec. A high-speed self-similar atm vbr traffic generator. In *Global Telecommunications Conference, 1996. GLOBECOM'96.'Communications: The Key to Global Prosperity*, volume 1, pages 586–590. IEEE, 1996.

[31] N.A. El-Fishawy. Quality of service investigation for multimedia transmission over wireless local area networks. In *Radio Science Conference, 2004. NRSC 2004. Proceedings of the Twenty-First National*, pages C3 – 1–8, march 2004.

[32] P. Embrechts and M. Maejima. *Selfsimilar processes*. Princeton series in applied mathematics. Princeton University Press, 2002.

[33] M. Emmelmann, B. Bochow, and C. Kellum. *Vehicular Networking: Automotive Applications and Beyond*, volume 2 of *Intelligent Transport Systems*. Wiley, 2010.

[34] S. Evans. *Telecommunications network modelling, planning and design*. BT communications technology series. Institution of Electrical Engineers, 2003.

[35] M.J. Feigenbaum. Quantitative universality for a class of nonlinear transformations. *Journal of statistical physics*, 19(1):25–52, 1978.

[36] A. Feldmann. Characteristics of tcp connection arrivals. *Self-Similar Network Traffic and Performance Evaluation*, pages 367–399, 2000.

[37] M.J. Fischer, D.M.B. Masi, D. Gross, and J.F. Shortle. One-parameter pareto, two-parameter pareto, three-parameter pareto: is there a modeling difference? *The Telecommunications Review*, pages 79–92, 2005.

[38] M.W. Garrett and W. Willinger. Analysis, modeling and generation of self-similar vbr video traffic. In *ACM SIGCOMM Computer Communication Review*, volume 24, pages 269–280. ACM, 1994.

[39] RG Garroppo, S. Giordano, S. Miduri, M. Pagano, and F. Russo. Statistical multiplexing of self-similar vbr videoconferencing traffic. In *Global*

Telecommunications Conference, 1997. GLOBECOM'97., IEEE, volume 3, pages 1756–1760. IEEE, 1997.

[40] R. Gass, J. Scott, and C. Diot. Measurements of in-motion 802.11 networking. In *Mobile Computing Systems and Applications, 2006. WMCSA'06. Proceedings. 7th IEEE Workshop on*, pages 69–74. IEEE, 2005.

[41] M. Gerla, M.Y. Sanadidi, Ren Wang, A. Zanella, C. Casetti, and S. Mascolo. Tcp westwood: congestion window control using bandwidth estimation. In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, volume 3, pages 1698 –1702 vol.3, 2001.

[42] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15:96, 95, 2011.

[43] M. Greiner, M. Jobmann, and L. Lipsky. The importance of power-tail distributions for modeling queueing systems. *Operations Research*, pages 313–326, 1999.

[44] S. Ha, I. Rhee, and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[45] H. Haken. *Synergetics: an introduction : nonequilibrium phase transitions and self-organization in physics, chemistry, and biology*. Number v. 2 in Springer series in synergetics. Springer-Verlag, 1978.

[46] W.C. Hardy. *QoS: measurement and evaluation of telecommunications quality of service*. QSDG magazine. Wiley, 2001.

[47] H. Hartenstein and K. Laberteaux. *VANET: vehicular applications and inter-networking technologies*. Intelligent Transport Systems. Wiley, 2010.

[48] J. Hespanha, S. Bohacek, K. Obraczka, and J. Lee. Hybrid modeling of tcp congestion control. *Hybrid Systems: Computation and Control*, pages 291–304, 2001.

[49] J. Hillebrand, C. Prehofer, R. Bless, and M. Zitterbart. Quality-of-service signaling for next-generation ip-based mobile networks. *Communications Magazine, IEEE*, 42(6):72 – 79, june 2004.

[50] C.M. Huang and Y.S. Chen. *Telematics communication technologies and vehicular networks: wireless architectures and applications*. Premier Reference Source. Information Science Reference, 2009.

[51] S. Ilnickis, E. Petersons, and R. Jerjomins. Server non-stationary behaviour research at near to self-similar query stream influence. *Electronics And Electrical Engineering*, 59(3):46–51, 2005.

[52] A. Ipatovs. *Experimental and Analytical Goodput Evaluation of Drive-thru Internet Systems*. PhD thesis, Riga Technical University, 2012.

[53] A. Ipatovs and E. Petersons. Performance Evaluation of WLAN depending on Number of Workstations and Protocols. *Electronics And Electrical Engineering*, 76(8):72–76, 2006.

[54] A. Ipatovs and E. Petersons. Real speed of data transfer and the signal-to-noise ratio in a wireless net for connection with mobile objects. *Automatic Control and Computer Sciences*, 43(1):40–46, 2009.

[55] A. Ipatovs, E. Petersons, and J. Jansons. Model for Wireless Base Station Goodput Evaluation in Vehicular Communication Systems. *Electronics And Electrical Engineering*, 111(5):19–22, 2011.

[56] M. Izzetoglu, B. Yazici, B. Onaral, and N. Bilgutay. Kalman filtering for self-similar processes. In *Statistical Signal Processing, 2001. Proceedings of the 11th IEEE Signal Processing Workshop on*, pages 82–85. IEEE, 2001.

[57] J. Jansons, A. Ipatovs, and E. Petersons. Estimation of Doppler Shift for IEEE 802.11 g Standard. In *Baltic Conference Advanced Topics in Telecommunication, University of Rostock*, page 73–82, 2009.

[58] HD Jeong, D. McNickle, and K. Pawlikowski. A generator of pseudo-random self-similar sequences based on sra. 1998.

[59] H.D.J. Jeong, J.S.R. Lee, D. McNickle, and K. Pawlikowski. Suggestions of efficient self-similar generators. *Simulation Modelling Practice and Theory*, 15(3):328–353, 2007.

[60] R. Jerjomins and E. Petersons. Client-server model non-stationary behaviour research at near self-similar query stream influence under the condition of overloaded terminal system. *Electronics And Electrical Engineering*, 71(7):35–38, 2006.

[61] R. Jerjomins and E. Petersons. Server non-stationary behaviour research in a wireless network under the condition of self-similar traffic. In *Riga Technical University 48th International Scientific Conference, Electronics and Telecommunications*, page 9–15. Riga Technical University, 2007.

[62] M. Jiang, M. Nikolic, S. Hardy, and L. Trajkovic. Impact of self-similarity on wireless data network performance. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 2, pages 477–481. IEEE, 2001.

[63] L. Kleinrock. *Queueing systems: Computer applications*. Number vol. 2 in Wiley-Interscience Publication. Wiley, 1976.

[64] L. Kleinrock. *Queueing Systems: Theory*. Number vol. 1 in A Wiley-Interscience publication. Wiley, 1976.

[65] Hisashi Kobayashi. Application of the diffusion approximation to queueing networks i: Equilibrium queue distributions. *J. ACM*, 21:316–328, April 1974.

[66] Hisashi Kobayashi. Application of the diffusion approximation to queueing networks ii: Nonequilibrium distributions and applications to computer modeling. *J. ACM*, 21:459–469, July 1974.

[67] M. Kulikovs. Statistical parameters estimation of the self-similar input traffic for the Measurement-based Admission Control. *TELECOMMUNICATIONS AND ELECTRONICS*, page 37, 2008.

[68] M. Kulikovs. *Research and development of effective managing algorithms in admission control systems for telecommunication networks*. PhD thesis, Riga Technical University, 2010.

[69] M. Kulikovs and E. Petersons. Real-Time Traffic Analyzer for Measurement-Based Admission Control. In *Telecommunications, 2009. AICT'09. Fifth Advanced International Conference on*, page 72–75. IEEE, 2009.

[70] M. Kulikovs and E. Petersons. Optimal dispatching of the flows falling in the same priority class. *Automatic Control and Computer Sciences*, 44(1):42–46, 2010.

[71] M. Kulikovs, E. Petersons, J. Roth, and J. Gutierrez. Remarks on Packet Loss Probability for the Network Traffic with Self-Similar Behaviour. *Proc. of Wireless Applications and Computing and Telecommunications, Networks and Systems*, page 85–90, 2008.

[72] M. Kulikovs, E. Petersons, and S. Sharkovsky. Integral measurement process of incoming traffic for Measurement-Based Admission Control.

In *Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8 International Conference on*, page 183–186. IEEE, 2010.

[73] M. Kulikovs, S.P.E. Sharkovsky, and E. Petersons. Comparative Studies of Methods for Accurate Hurst Parameter Estimation. *Electronics and Electrical Engineering*, (7):103, 2010.

[74] Rachid Laalaoua, Tülin Atmaca, Stanisław Jędrús, and Tadeusz Czachórski. Diffusion model of red control mechanism. In Pascal Lorenz, editor, *Networking — ICN 2001*, volume 2093 of *Lecture Notes in Computer Science*, page 107—116. Springer Berlin / Heidelberg, 2001.

[75] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of ethernet traffic. *ACM SIGCOMM Computer Communication Review*, 23(4):183–193, 1993.

[76] Petri Mähönen, Janne Riihijärvi, Marina Petrova, and Zach Shelby. Hop-by-hop toward future mobile broadband IP. *IEEE Communications Magazine*, 42(3):138–146, 2004.

[77] S.I. Maniatis, E.G. Nikolouzou, and I.S. Venieris. End-to-end qos specification issues in the converged all-ip wired and wireless environment. *Communications Magazine, IEEE*, 42(6):80 – 86, june 2004.

[78] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 287–297, New York, NY, USA, 2001. ACM.

[79] R.M. May et al. Simple mathematical models with very complicated dynamics. *Nature*, 261(5560):459–467, 1976.

[80] SW Ng and E. Chan. Equation-based tcp-friendly congestion control under lossy environment. *Journal of Systems Architecture*, 51(9):542–569, 2005.

[81] Qiang Ni, A. Vinel, Yang Xiao, A. Turlikov, and Tao Jiang. Wireless broadband access: Wimax and beyond - investigation of bandwidth request mechanisms under point-to-multipoint mode of wimax networks. *Communications Magazine, IEEE*, 45(5):132 –138, may 2007.

[82] I. Nikolaidis, C.A. Cooper, K.S. Perumalla, and R.M. Fujimoto. Time-parallel generation of self-similar atm traffic. In *Proceedings of the 29th conference on Winter simulation*, pages 1071–1078. IEEE Computer Society, 1997.

[83] I. Norros. On the use of fractional brownian motion in the theory of connectionless networks. *Selected Areas in Communications, IEEE Journal on*, 13(6):953–962, 1995.

[84] J. Ott and D. Kutscher. Drive-thru internet: Ieee 802.11 b for automobile users. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1. IEEE, 2004.

[85] J. Ott and D. Kutscher. A modular access gateway for managing intermittent connectivity in vehicular communications. *European transactions on telecommunications*, 17(2):159–174, 2006.

[86] K. Park and W. Willinger. *Self-similar network traffic and performance evaluation*. Wiley-Interscience publication. Wiley, 2000.

[87] H.D.J.J.K. Pawlikowski and D.C. McNickle. Generation of self-similar time series for simulation studies of telecommunication networks.

[88] E. Perahia. Ieee 802.11n development: History, process, and technology. *Communications Magazine, IEEE*, 46(7):48 –55, july 2008.

[89] V. Petroff. Self-similar network traffic: From chaos and fractals to forecasting and qos. *NEW2AN. œ St. Petersburg*, pages 110–118, 2004.

[90] S.W. Philbrick. A practical guide to the single parameter pareto distribution. *PCAS LXXII*, 44:85, 1985.

[91] R. Popescu-Zeletin, I. Radusch, and M.A. Rigani. *Vehicular-2-X Communication: State-of-the-Art and Research in Mobile Vehicular Ad hoc Networks*. Springer, 2010.

[92] J. PROCHASKA and R. VARGIC. Using digital filtration for hurst parameter estimation. *Radioengineering*, 18(2):238–241, 2009.

[93] A. Raniwala and Tzi cker Chiueh. Architecture and algorithms for an ieee 802.11-based multi-channel wireless mesh network. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2223 – 2234 vol. 3, march 2005.

[94] W.J. Reed. The pareto, zipf and other power laws. *Economics Letters*, 74(1):15–19, 2001.

[95] T.G. Robertazzi. *Computer networks and systems: queueing theory and performance evaluation*. Telecommunication networks and computer systems. Springer, 2000.

[96] O. Rose. Estimation of the hurst parameter of long-range dependent time series. *University of Wurzburg, Institute of Computer Science Research Report Series.—February*, 1996.

[97] O.I. Sheluhin, S.M. Smolskiy, and A.V. Osin. *Self-similar processes in telecommunications*. Wiley, 2007.

[98] Y. Shu, F. Xue, Z. Jin, and O. Yang. The impact of self-similar traffic on network delay. *Journal of Computer Science and Technology*, 14(6):585–589, 1999.

[99] B. Sklar. *Digital Communications. 2001*. Prentice Hall.

[100] W. Stallings. *High-speed networks and internets: performance and quality of service*. William Stallings books on computer and data communications technology. Prentice Hall, 2002.

[101] W. Stallings. *Wireless communications and networking*. William Stallings books on computer and data communications technology. Prentice Hall, 2002.

[102] J.W. Stoutenborough and P. Johnson. Pareto distribution, 2006.

[103] A. Sykes. *An introduction to regression analysis*. Law School, University of Chicago, 1993.

[104] M. Tekala. Tcp westwood with limited congestion window. In *Advanced Computer Control, 2009. ICACC'09. International Conference on*, pages 687–692. IEEE, 2009.

[105] S. Trivedi, S. Jaiswal, R. Kumar, and S. Rao. Comparative performance evaluation of tcp hybla and tcp cubic for satellite communication under low error conditions. In *Internet Multimedia Services Architecture and Application (IMSAA), 2010 IEEE 4th International Conference on*, pages 1–5. IEEE, 2010.

[106] P. Ulanovs and E. Petersons. Modeling methods of self-similar traffic for network performance evaluation. In *Scientific Proceedings of RTU. Series*, volume 7, pages 43–49, 2002.

[107] H. Velayos and G. Karlsson. Statistical analysis of the ieee 802.11 mac service time. In *Proceedings of the 19th International Teletraffic Congress*, 2005.

[108] L. Villaseñor-González, C. Portillo-Jiménez, and J. Sánchez-García. A performance study of the ieee 802.11 g phy and mac layers over heterogeneous and homogeneous wlans. *Ingeniería, investigación y tecnología*, 8(1):45–57, 2007.

[109] J. Wang and S. Keshav. Efficient and accurate ethernet simulation. In *Local Computer Networks, 1999. LCN'99. Conference on*, pages 182–191. IEEE, 1999.

[110] W.H. Xi, T. Whitley, A. Munro, and M. Barton. Modeling and simulation of mac for qos in ieee 802.11 e using opnet modeler. *Networks &J Protocols Group, CCR, Department of Electrical & Electronic Engineering, University of Bristol*, 2006.

[111] R. Yeryomin and E. Petersons. Self-similar traffic in wireless networks. In *Proc. of International Conference Mathematical Methods of Optimization of Telecommunication Networks.–Minsk: Belarusian State University Press*, volume 18, page 49–55, 2005.

[112] R. Yeryomin and E. Petersons. Server relaxation time at near to self-similar query stream influence. In *Proc. of International Conference Mathematical Methods of Optimization of Telecommunication Networks.–Minsk: Belarusian State University Press*, volume 19, page 87–91, 2007.

[113] R. Yeryomin and E. Petersons. Transient process relaxation time research under the condition of self-similar traffic input in wireless networks. *Automatic Control and Computer Sciences*, 43(3):138–147, 2009.

[114] R. Yeryomin and E. Petersons. Analytical estimation of self-similar wireless traffic relaxation time and hurst coefficient dependence. *Electronics And Electrical Engineering*, 108(2):31–34, 2011.

[115] R. Yeryomin and E. Petersons. Generating self-similar traffic for wireless network simulation. In *Internet Communications (BCFIC Riga), 2011 Baltic Congress on Future*, page 218–220. IEEE, 2011.

[116] J. Yu and AR Petropulu. Is high-speed wireless network traffic self-similar? In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 2, pages ii–425. IEEE, 2004.

[117] A. Zanella, G. Procissi, M. Gerla, and MY Sanadidi. Tcp westwood: Analytic model and performance evaluation. In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 3, pages 1703–1707. IEEE, 2001.

[118] Jing Zhu and S. Roy. Mac for dedicated short range communications in intelligent transport system. *Communications Magazine, IEEE*, 41(12):60 – 67, dec. 2003.

[119] В.Д. Боев. *Моделирование систем. Инструментальные средства GPSS World*. БХВ-Петербург, 2004.

[120] А.Д. Вентцель. *Курс теории случайных процессов*. Наука, 1975.

[121] Е.С. Вентцель. *Введение в исследование операций*. Советское радио, 1964.

[122] Е.С. Вентцель. *Теория вероятностей*. Наука, 1969.

[123] В.М. Вишневский. *Теоретические основы проектирования компьютерных сетей*. Техносфера, 2003.

[124] В.М. Вишневский. *Широкополосные беспроводные сети передачи информации*. Техносфера, 2005.

[125] А.Я. Городецкий and В.С. Заборовский. *Фрактальные процессы в компьютерных сетях*. Санкт-Петербургский государственный политехнический университет, 2000.

[126] Р.Ж. Ессен. *Методы статистических обследований*. Финансы и статистика, 1985.

[127] Н.Ш. Кремер, Н.Д. Эриашвили, А.Н. Романов, and Министерство образования РФ. *Теория вероятностей и математическая статистика:*. Юнити, 2000.

[128] А. Кофман и Р. Крюон. *Массовое обслуживание. Теория и приложения*. Мир, 1965.

[129] Е.А. Кучерявый. *Управление трафиком и качество обслуживания в сети интернет*. Наука и техника, 2004.

[130] Б.Р. Левин. *Теоретические основы статистической радиотехники*. Number v. 1. Советское Радио, 1974.

[131] А.В. ОСИН. *Влияние самоподобности речевого трафика на качество обслуживания в телекоммуникационных сетях*. PhD thesis, 2005.

[132] Е.С. Вентцель и Л.А. Овчаров. *Задачи и упражнения по теории вероятностей*. Академия, 2003.

[133] В. Быстров и Э. Петерсон. Аналитическая оценка вероятности потери пакетов в коммуникационных системах с самоподобным входным потоком. *Автоматика и вычислительная техника*, 38(4):46–53, 2008.

[134] Р. Еремин и Э. Петерсонс. Исследование времени релаксации переходного процесса при обслуживании самоподобного входного трафика в беспроводных сетях. *Автоматика и вычислительная техника*, 43(3):36–46, 2009.

[135] В.В. Петров. То, что вы хотели знать о самоподобном телетрафике, но стеснялись спросить. *М.: МЭИ, ИРЭ*, 2003.

[136] В.В. Петров. *Структура телетрафика и алгоритм обеспечения качества обслуживания при влиянии эффекта самоподобия*. PhD thesis, [Моск. энергет. ин-т (Техн. ун-т)] Москва, 2004.

[137] В.В. Петров and Е.А. Богатырев. Статистический анализ сетевого трафика. In *Радиоэлектроника, электротехника и энергетика: Тез. докл. Десятой Междунар. научно–техн. конференции студентов и аспирантов*, volume 1, 2003.

[138] П.П. Бочаров и А.В. Печинкин. *Теория массового обслуживания*. РУДН, 1995.

[139] Ю.А. Розанов. *Случайные процессы: Краткий курс. [Учеб. пособие для физ.-мат. и физ.-тех. специальностей вузов]*. Наука, 1971.

[140] Э. Таненбаум. *Компьютерные сети:*. Классика Computer Science. Питер, 2005.

[141] И. Шахнович. *Современные технологии беспроводной связи*. Техносфера, 2006.