Technical University Berlin

Telecommunication Networks Group

A Gilbert-Elliot Bit Error Model and the Efficient Use in Packet Level Simulation

Jean-Pierre Ebert, Andreas Willig

{ebert,willig}@ft.ee.tu-berlin.de

Berlin, March 1999

TKN Technical Report TKN-99-002

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Abstract

When investigating complex communication protocols and a large number of stations using discrete event simulation, it is of great interest to keep the number of necessary simulation events low, since this number directly translates into simulation runtime. One important target for optimizations is the packet or frame exchange over an underlying transmission medium. The number of simulation events involved in transmitting packets should be as low as possible. On the other hand, it is often of great importance to accurately model the channel characteristics and the exact error process. A prominent example are wireless channels. Unfortunately packet errors are very difficult to model, since they depend on channel and source characteristics (e.g. coding). Often the results of erroneous bits differ with the position of the bit hit by an error. Thus it may be necessary to simulate packet transmission on a bit level to derive the packet error process. Simulation at bit level requires at least one simulation event for every bit instead of a few simulation events per packet. In this paper we show an approach for modeling packet error processes with the accuracy of bit error processes, but having nearly the simulation performance of packet level simulation. In order to show how our approach impacts simulation times of complex protocols, where the packet transmission is only a small part of the model, we simulated the IEEE 802.11 MAC on top of our improved channel model. It can be seen that the reductions in simulation times are impressive.

Contents

Chapter 1

Introduction

Simulation of communication protocols can be a very time consuming task. Therefore, a simulation model should be as abstract and simple as possible while giving sufficiently accurate results. It is desirable to use only a few simulation events per single packet instead of one simulation event per bit transmitted, and thus to use a *packet level channel model* instead of a *bit level channel model*.

For simulation of communication protocols, e.g. Aloha, CSMA/CA, SNOOP-TCP, etc., channel modeling is an important task. In particular RF channels are very error prone (see [2], [10])and the protocol behavior often depends strongly on the channel behavior, especially when not all bits in a packet are equal w.r.t. the resulting consequences when hit by an error. This is often a reason for using bit level channel models.

A second argument for using bit level models stems from the fact that it is often very hard to obtain accurate packet level channel models, since these depend at least on three factors: the bit error behaviour of the channel, coding and packet formats and the source characteristics (e.g. packet length distribution). For this reason packet error processes are often obtained via bitwise simulation of the packets. In figure 1.1 one can see that the packet error characteristics are different for the same bit error pattern, when the packet arrival patterns are different. In particular packet source #1 generates longer packets which are more likely to be hit by a bit error than the shorter packets of source #2. The channel behaviour is in general assumed to be independent from the behaviour of the packet sources.

Using bit level channel models will degrade the performance of the protocol simulation, since for processing of every bit at least one simulation event is necessary. Moreover, if we assume different channels between every pair of mobile stations, the simulation expenses grow with N. In contrast, packet level channel models need only one simulation event for

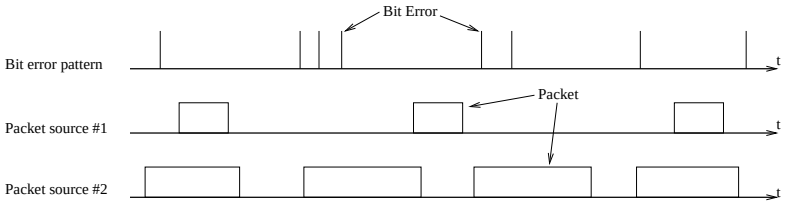


Figure 1.1: Packet error characteristics for the same bit error pattern

processing a packet, where a packet can consist of thousands of bits.

In this paper we demonstrate an approach for modeling packet error processes with the accuracy of bit error processes, but having nearly the simulation performance of packet level simulation. Since we are mainly interested in the behaviour of radio frequency (RF) wireless channels, we focus on a well known class of of bit error models, namely the Gilbert-Elliot RF channel model. Based on the Gilbert/Elliot bit level channel model we derive an equivalent packet level channel model, which yields by construction the same packet error process, as if every packet is simulated bit per bit. We show, that simulation times can be greatly reduced. However, when simulating complex protocols, the transmission channel is only a small part in the simulation. In order to assess the effects of our channel model to the simulation times of complex protocols and scenarios, we have run a simulation of IEEE 802.11 on top of our channel model. Still the savings in simulation times are impressive.

The paper is structured as follows. In the next section 2 we explain the Gilbert-Elliot channel model and how to parameterize it in order to match concrete channels. The Gilbert/Elliot model is first used to derive a bit level channel model. This *straightforward model* is described in section 3. In section 4 we describe two improved models and compare them to the prior model w.r.t. simulation times. The savings for the second model are impressive. In section 5 we describe the effect of the straightforward and the improved bit error models using a complex protocol simulation. For that purpose we have chosen the IEEE 802.11 MAC protocol using the Distributed Coordination Function. In the appendices we give a detailed description and (pseudo-) code of all channel models.

Chapter 2

Gilbert-Elliot RF Channel Model

In this work we consider radio transmission, e.g. using the license free 2.4 GHz ISM band (Industrial, Scientific and Medical band). It is widely accepted that the radio channel is a error prone channel with non-stationary error characteristics. Bit error rates as bad as $\approx 10^{-2,\cdots,-3}$ are reported (for measurements see [2], [4], [3]). The error process in general is constituted by different phenomena:

- Path Loss: the signal power between transmitter and receiver degrades degrades according to $P_r = P_t \cdot \xi \cdot d^{-\alpha}$ where P_r is the signal power at the receiver, P_t is the signal power at the transmitter, d is the distance, ξ is some technology dependent constant and α typically varies between $\alpha = 2$ (best case) and $\alpha = 5$ (bad case). However, the relationship between distance and bit error rate (BER) is typically not linear. Instead, often the mean bit error rate remains almost constant up to a given distance threshold and then degrades rapidly (see e.g. [2]. This is due to receiver behavior, where all signals below a given threshold are discarded.
- Fast Fading due to movement and multi-path propagation.
- Slow Fading due to moving beyond large obstacles.
- Noise and Interference from other networks or devices like microwave ovens.

2.1 Modeling Approach

For modeling the error characteristics of a wireless channel between two stations a simple and widely used model is the "Gilbert-Elliot model" [13], [9], [5]: consider a two state Markov chain with the states named *Good* and *Bad* (there is no *Ugly* state), see fig. 2.1. Every state

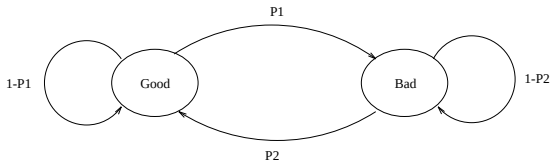


Figure 2.1: A two state Markov channel

is assigned a specific constant bit error rate (BER), e_G in the good state, e_B in the bad state ($e_G \ll e_B$). Within one state bit errors are assumed to occur independently from each other. The bit error rates in general depend on the frequency and coding scheme used and on environmental conditions (e.g. number of paths between source and destination). In the literature often a discrete time Markov chain is used, with state transitions after every channel symbol. Thus the state sojourn time is geometrically distributed. It can be shown, that for a high probability of staying within a state and a large number of symbols the state sojourn time can be approximated by an exponential distribution[1]. For complete specification of the discrete model the two bit error probabilities and the 2×2 stochastic state transition matrix are sufficient. The state transition matrix is completely determined by the values p_{GG} (for the probability that the next state is the good state, given that the current state is also the good state) and p_{BB}. The mean state sojourn time (duration of being in a state) measured in number of steps in this state is given by:

$$T_G = \frac{1}{1 - p_{GG}}$$
$$T_B = \frac{1}{1 - p_{BB}}$$

Some examples from the literature:

- From [6] (a more theoretical study on Polling and ARQ over Wireless channels):
  - $e_G = 10^{-4}$, $e_B = 10^{-3}$, $p_{GG} = 0.995$ and $p_{BB} = 0.96$,
  - state changes occur only at slot boundaries,the slot length is 1 msec,
  - thus the mean duration of good state is 200 msec and the mean duration of bad state is 25 msec
- From Diploma thesis of Jie Volckart (based on [13]):

[1] For this case we assume that the Poisson Distribution is an approximation to the binomial distribution counting the number of state changes within a fixed interval of time (see [7, p.153]). However, for Poisson distributed random variables the inter-arrival times are exponentially distributed.

  - $e_G = 10^{-5}$, $e_B = 10^{-2}$, $p_{GG} = 0.9999918$ and $p_{BB} = 0.999184$
  - The transmission rate is 8 Mbps, the slot size is 8 bytes. It is not known whether state transitions occur at slot boundaries or at symbol (bit) boundaries

- From [1] (a simulation study): mean duration of good period: 1 to 10 sec, mean duration of bad period: 50 to 500 msec (both exponentially distributed), packet loss probability in bad state: 0.8 (with packet size of 512 bytes, link speed = 10 Mbps), packet loss probability in good state = 0.0;

However, the Gilbert/Elliot model is only a special case of the more general model described in the next section, which will allow for derivation of the Gilbert-Elliot channel model parameters based on some fundamental physical properties.

2.2 The N-State Markov Chain Model

This section explains the assumptions and methods necessary for calculation of Markov Chain parameters from physical channel properties. The following is based on [13].

The focus of the model in [13] is on the special case of BPSK coding over a Rayleigh fading channel. The Rayleigh fading process is used for characterization of wireless channels, where

- the receiver and transmitter are not fixed, but move with a moderate speed, with the Doppler frequency being substantially smaller than the symbol rate[2].
- There is a not too small number of signal paths between transmitter and receiver, and the signal strengths are approximately the same[3]. There is no predominant path (e.g. no Line-Of-Sight path).

It seems to be reasonable to assume Rayleigh Fading also in the case where the distance between transmitter and receiver is small (e.g. in Wireless LANs), and there exists multiple other paths in addition to the line-of-sight path, with small absorption coefficients, e.g. due to reflections. This may be the case in office and industrial environments. In contrast to

[2]Between two fading holes the channel is assumed to be in a stationary state. If the time difference between fading holes is large enough, a lot of symbols can be transmitted within the same channel state and the channel obeys bursty characteristics. If the fading frequency is in the range of the symbol frequency, this is not true anymore.

[3]This assumption is necessary to invoke the central limit theorem.

that, if there is one predominant signal path, the channel is said to be a Rice fading channel, which is not covered by the model described here.

The overall assumptions for the N-State Markov Chain model are:

- Rayleigh-Fading, producing a time-varying receiver signal-to-noise ratio (R-SNR)
- BPSK coding
- Time variations of the received signal level are assumed to come from mobility (Doppler effect)

A homogeneous discrete Time Markov Chain is constructed based on these assumptions, where transitions can occur only after every channel symbol. Each state corresponds to a specific channel quality and has its own BER (with independent bit errors). The range of R-SNR is grouped into a finite number (K) of intervals. From R-SNR the bit error probability is directly concluded.

Calculating the Markov Chain Parameters

- Be K the number of states, $\mathcal{S} = \{s_o, .., s_{K-1}\}$ the set of states
- Be $\mathbf{T} := ((t_{i,j}))_{i,j \in \{0,..,K-1\}}$ the time-homogeneous state transition matrix of the underlying Markova chain. $\mathbf{T}$ is a stochastic matrix.
- $\mathbf{p} := (p_0, .., p_{K-1})^t$ is the steady state vector of $\mathbf{T}$
- $\mathbf{e} := (e_0, .., e_{K-1})^t$ is a vector of bit error probabilities e_k for state s_k
- The mean bit error rate e is thus given by $e = \mathbf{p}^t\mathbf{e}$
- For developing a suitable model we need to determine $\mathbf{T}, \mathbf{e}$, and $\mathbf{p}$
- Be A the R-SNR with probability distribution function pdf $p_A(a) = \frac{1}{\rho} \cdot \exp(-\frac{a}{\rho})$ and $0 = A_0 < A_1 < .. < A_K = \infty$ is a partition of the range of R-SNR. ρ is the mean of the R-SNR. The DTMC is defined to be in state s_k at time t iff R-SNR$(t) \in [A_k, A_{k+1})$.
- Be $f_m := \frac{v}{\lambda}$ be the maximum Doppler frequency, where v is the speed of the receiver, λ the wavelength
- Then the number N_a is the expected number of times per second where R-SNR sinks below a given level a:

$$N_a = \sqrt{\frac{2\pi a}{\rho}} \cdot f_m \cdot \exp\left(-\frac{a}{\rho}\right)$$

- define

$$F(\alpha) := \int_{-\infty}^{\alpha} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx$$

- The following steps are needed:
  - choose K, λ and v ($v > 0$), and the symbol rate R_t
  - determine the range of R-SNR and define a partition $A_0, .., A_K$ of the range, such that $A_0 = 0$ and $A_K = \infty$
  - choose ρ
  - Then for $k \in \{0, .., K-1\}$

$$e_k = \frac{\int_{A_k}^{A_{k+1}} \frac{1}{\rho} \cdot \exp\left(-\frac{a}{\rho}\right) \cdot (1 - F(\sqrt{2a}))da}{\int_{A_k}^{A_{k+1}} \frac{1}{\rho} \cdot \exp\left(-\frac{a}{\rho}\right) da}$$

  - for $k \in \{0, .., K-1\}$

$$p_k = \exp\left(-\frac{A_k}{\rho}\right) - \exp\left(-\frac{A_{k+1}}{\rho}\right)$$

  - define $R_t^{(k)} := R_t \cdot p_k$

- Proceed with:
  - set

$$N_k := N_a(A_k) = \sqrt{\frac{2\pi A_k}{\rho}} \cdot f_m \cdot \exp\left(-\frac{A_k}{\rho}\right)$$

  - set $t_{i,j} = 0$ for $|i - j| > 1$
  - for $k \in \{0, .., K-2\}$ set

$$t_{k,k+1} = \frac{N_{k+1}}{R_t^{(k)}}$$

  - for $k \in \{1, .., K-1\}$ set

$$t_{k,k-1} = \frac{N_k}{R_t^{(k)}}$$

- And finally:
  - $t_{0,0} = 1 - t_{0,1}$

- $t_{K-1,K-1} = 1 - t_{K-1,K-2}$
- for $k \in \{1, .., K-2\}$:

$$t_{k,k} = 1 - t_{k,k-1} - t_{k,k+1}$$

- For computation of e_k, the following formula behaves better from a numerical point of view:

$$\begin{aligned} e_k &= \frac{\gamma_k - \gamma_{k+1}}{p_k} \\ \gamma_k &= \exp\left(-\frac{A_k}{\rho}\right)(1 - F(\sqrt{2A_k})) \\ &\quad + \sqrt{\frac{\rho}{\rho+1}} F\left(\sqrt{\frac{2A_k(\rho+1)}{\rho}}\right) \end{aligned}$$

In [13], for a given set of parameters, this model is validated against a ray-tracing based simulation.

However, this model has the serious computational problem, that for every channel symbol the current state s_k of the Markov chain and the corresponding BER e_k needs to be determined (the BER can be obtained from a precomputed table), a random experiment has to be performed for checking whether the current bit is in error and a second random experiment is needed for determining the next state. The rest of this report deals with this problem.

The main feature of interest of this simple model is its ability to capture the non-stationary error characteristics of wireless links, where bit errors tend to occur in bursts, i.e. are correlated.

How to choose the number of states?

The above given model description gives no advice on how to choose some of its parameters, namely the number of states K. We have found, that the following heuristic makes sense:

- Consider the case where a MAC protocol tries to ensure, that on a single channel only one station transmits at a time, collisions count as error. Some protocols belonging to this class are TDMA-like protocols, CSMA, CSMA/CA, PRMA, and so forth. If one wants to take only fast fading (rayleigh fading), it seems to be natural to work with only two states (Good and Bad), and to assign the bad state to fading holes, accordingly setting the good state to all other time intervals. This model is appropriate for short distances.

- If the model is built for long distances, one must additionally take slow fading into account, where the signal shows different levels of attenuation, e.g. due to moving beyond a large obstacle. In this case $K = 4$ leads to a model, where the first two states are the good and bad state of the Gilbert/Elliot model in the case where there is a line of sight, the remaining two states are the good and bad state when there is an obstacle between sender and receiver.
- For modeling CDMA systems, where many stations may transmit simultaneously, K should be chosen as the number of stations in the system. Such a model is used e.g. in [8].

Chapter 3

Gilbert-Elliot RF Channel Simulation Model

In this section we describe a bit level channel model based on the Gilbert-Elliot RF channel model described in the preceding section. The input parameters of the simulation model are shown below. We have chosen these parameters according to the IEEE 802.11 DSSS physical layer specification and the assumed indoor application scenario:

In this section we describe a Gilbert-Elliot based model for RF channel simulation, working on a bit-by-bit basis. As underlying simulation engine we use the CSIM (version 18) library [11]. The input parameter of the simulation model are shown below. They are chosen according to the IEEE 802.11 DSSS physical layer specification and the assumed indoor application scenario:

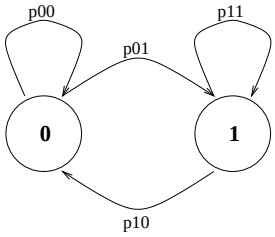
- Vehicle speed: $V = 5$ km/h (1.4 m/s)
- Wave length: $\lambda = 0.125$ m (2.4 GHz)
- Max. Doppler frequency: $f_m = 1.4/0.125 = 11.2$ Hz
- Bit rate: $R_t = 2$ Mbit/s

The target parameters for the simulation model are:

- Number of Markov Chain states: $K = 2$
- Mean of R-SNR: $\rho = 20.5$db
- Threshold SNR (Good $\leftrightarrow$ Bad): $A = 20$db

The values for ρ and A are not qualified through measurements. We have chosen them such that the formulas given above yield a bit error rate of 10^{-5} and relatively long state sojourn times.

¿From the computation method presented in section 2 we will get the parameters for the DTMC, these are shown in figure 3.1. A state of the chain stands for a certain channel state associated with a certain bit error probability.



Transition Probabilities:

p00 - 0.999990785635362	p10 - 0.00001325189423
p01 - 0.000009214364637	p11 - .9999867881057613#

Bit Error Probabilities:

State 0 - 0.0001972644427729
State 1 - 0.1^{-35}

Figure 3.1: From Gilbert-Elliot-Model derivated DTMC

A bit error simulation model based on the Gilbert-Elliot model can be quite easily implemented, assuming a certain initial channel state:

1. Perform a random experiment according to the present channel state to determine, whether the bit is erroneous . If the bit is erroneous mark that bit.

2. Perform a random experiment according to the present channel state to determine the next state. In case a state change occur, set the state variable and error probability to the new state.

3. Wait one bit time (perhaps doing some other work) and continue with step 1.

In the following we call a model, which follows this approach, *Straightforward Simulation Model*. Appendix A (**Straightforward Gilbert-Elliot Simulation Model**) contains a CSIM18 presentation of that model. The drawback of this modeling approach is, that at every bit time, two Bernoulli experiments have to be executed: one to determine whether a bit error occurs and the other one to determine a state change. This will slow down packet level communication protocol simulation, since a packet can consist of thousands of bits. A goal for improvement is the reduction of the Bernoulli experiments per packet. Clearly the

minimum number of Bernoulli experiments needed for a single packet is two (no channel state transition during a packet). We want to get as close to this minimum as possible.

Chapter 4

Speeding up the bit error simulation

The first problem, leading to much simulation overhead, is the calculation of the (Markov chain) state change by means of a Bernoulli experiment at every bit time. Applying some statistics leads to an improvement. For instance, one can count the number of independent Bernoulli experiments until an "success" event occurs. This number corresponds to a randov variable X. For a realization of $X = i$ is the probability of i "wrong" experiments is $(1-p)^i$ and for the successful experiment p. The variable X follows a geometric distribution with

- $P\{X = i\} = (1-p)^i p$ for $i = 0, 1, ...,$
- $E[X] = (1-p)/1 - (1-p)$

We can use this knowledge to speed up the simulation. Instead of executing every bit time a Bernoulli experiment to check a state transition in the DTMC, we derive from the geometric distribution the number of (bit) time steps of next (Markov chain) state transition in the DTMC. CSIM18 provides for that purpose a function

- $geometric(p)$ where p is the success probability

which returns an integer number, following the geometrical distribution.

The result of this modification is, that we have to compute just once the next (Markov chain) state every n bits, instead of performing a Bernoulli experiment every bit time. In the following parts of the paper we call this optimization *Improved Simulation Model*. The CSIM18 example, which contains the improvement as proposed above, is presented in Appendix B. This approach still allows to determine the position and the number of bit errors

in a packet. This may be interesting in some communication protocols, where only parts of the packets are error protected, e.g. for the header in ATM packets. In this case one has to know about the position of the bit error, since a packet is only recognized as erroneous, if the header contains an error. Another example is simulation and modeling of error correcting codes, where the transmitted bits differ in their significance. With the help of error correction codes, an erroneous packets can be corrected, if the number of bit errors are below a certain threshold depending on the parameters and power of the code.

Next we attack the calculation of bit errors. Every bit is evaluated with a Bernoulli experiment. A optimization for this case is possible. Consider the case, where one only want to know, whether the packet is correct or wrong, that is, there is respectively no bit error in the packet or at least one. This applies for many communication protocols, which use only a packet error recognition like CRCs (Cyclic Reduncy Check). If one knows the bit error probability and the packet length in bits, it is possible to compute the packet error probability.

Let i be the number of bits and y the bit error probability, then the packet error probability P_{err} is

- $P_{err}(i, y) = 1 - (1 - y)^i$

We can use P_{err} in a single Bernoulli experiment to decide about correctness of the packet. That is, instead of computing the bit error at every bit time, we compute just once for a packet the error. This will reduce complexity and computational expenses of the bit error model leading to an improved simulation performance. However, we have observed that for larger i it is numerically better to use the CSIM binomial random variate generator the following way: one can determine the number of errors in the packet, using a binomial random variable. If the number equals zero, the packet is correct, otherwise it is erroneous.

Of course, the channel state may change during a packet transmission. This can be handled by computing the error probability only of the packet fraction, which belongs to the the current channel state, and for the remaining packet fractions repeat this calculation. The partial results can be combined to determine the packet error probability. For instance, if we have three state transitions of the markov chain during a packet transmission, we compute for the already transmitted part of the packet according to the present bit error probability, whether this part is erroneous. If one or more of the packet parts are erroneous, the whole packet is erroneous. This method can also be applied to determine, whether a certain part of a packet (e.g. header, body) is erroneous.

In the following we call this optimization *Optimized Simulation Model*. A CSIM18 representation of this optimization can be found in Appendix C. The difference between this model and the Improved and the Straightforward Simulation Model is an enormous reduction of necessary Bernoulli experiments. Only a few Bernoulli experiments per packet are performed[1] to determine the correctness of a packet instead of performing a Bernoulli experiment for every bit. Furthermore, the bitwise computation of state change is reduced to one computation every x bits, where x follows a geometric distribution. A reduction of computational expenses and effective simulation time is achieved. Furthermore, from construction it is clear that all models are stochastically equivalent, i.e. they show the same statistics. This is confirmed by the results shown in section 5.

4.1 Model Comparison

In order to compare the presented models and their performance gains with respect to simulation expenses, we measured the CPU time of the simulation programs. The measurement setup was as follows: we have used a 350MHz PentiumII computer running LINUX. There was no user task on the system, only the basic set of LINUX demons was running. We have measured both the "real time" (the time the user sees between starting and stopping the simulation) and the CPU time with the built-in *time* command of the tcsh shell. The runtime of the simulations was controlled by the CSIM run length control mechanism, which runs a simulation until the predefined confidence level of 95 % was reached for a confidence interval width of 10 % of the measured value. This mechanism uses the batch means method. As target for the run length control the channel state probability was chosen. The packet sizes are chosen fixed to be 128 bytes.

The results are shown in table 4.1 (CPU time) and table 4.2 (real time). We can clearly see the improvement in simulation time. Comparing the Straightforward and the Optimized Model in table 4.1 the difference is in the order of 9. The improvement is a result of the reduction of necessary Bernoulli experiments. The drastic improvement between the Improved and Optimized model results from the fact, that the computation for determining the bit errors is replaced be a very simple compution for determining the packet error. In the optimization for determining the state change the relative complex bitwise Bernoulli experiment is replaced by a more complex but less frequently used computation of the time of switching to the next state, which is based on a geometric distribution.

[1] If no state change occur during a packet transmission, only one Bernoulli experiment is necessary to determine a packet error

To check the validity of the models, we have compared the mean bit error probabilities delivered by the straightforward model and the improved model with the analytical solution as described in section 2. The results are shown in table 4.3. The results equals very good each other. We further checked the simulation results for error state probability and sojourn times as compared to the analytical solution. In table 4.4 it is shown that the results differ only marginally.

CPU Time in Seconds		
Straightf. Model	**Improved Model**	**Optimized Model**
2454.960	286.010	0.130

Table 4.1: CPU time of simulation models

Real Time in Minutes		
Straightf. Model	**Improved Model**	**Optimized Model**
46:01.02	5:14.14	0:00.41

Table 4.2: Real time of simulation models

Mean Bit Error Probability		
Analytical Solution	**Straightf. Model**	**Improved Model**
0.0000116	0.0000117	0.0000117

Table 4.3: Mean bit error probability

	Channel State Probability		Sojourn Time in ms	
Model	**State0**	**State1**	**State0**	**State1**
Analytics	0.589857	0.410142	0.054262	0.037729
Straightf.	0.59702	0.402978	0.05468	0.036913
Improved	0.592986	0.407014	0.054988	0.037743
Optimized	0.590147	0.409853	0.054318	0.037724

Table 4.4: Error state probability and sojourn time

Chapter 5

Simulating IEEE 802.11 DCF

We have investigated the influence of the error models used to the runtimes needed for simulation of higher layer protocols. For that purpose we took a complex media access protocol: IEEE 802.11 using DCF (Distributed Coordination Function). We do not describe the IEEE 802.11 model in detail, since we just want to know how the optimization of the bit error model presented before scale with a complex packet protocol model. For information and details of IEEE 802.11 the reader is referred to [12].

5.1 IEEE 802.11 DCF Model and Parameter

IEEE 802.11 is a standard for wireless in-house communication. We only want to present the simulation setup and parameters in this section.

For obtaining results on consumed CPU time and real time we integrated the proposed bit error models in the model of IEEE 802.11 DCF. We simulated two mobiles sending packets to each other. The run length of the simulation again was controlled by the CSIM run length control mechanism with the same confidence level and accuracy values for the inter-arrival times of erroneous packets. Both mobiles were fed by source processes, where the packet inter-arrival time is a Poisson process with the mean of 1500 μsec and the packet size is also determined by a Poisson process with mean of 128 bytes. The parameters of the MAC protocol were set according to the 2Mbit/s DSSS-PHY (Direct Sequence Spread Spectrum) of the IEEE 802.11 standard. The simulations were performed only with the Straightforward and the Optimized Model.

The following figure 5.1 shows the simulation setup.

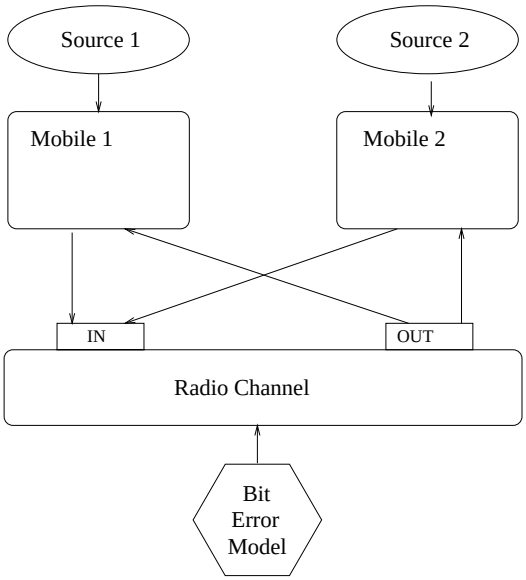


Figure 5.1: Simulation Model

5.2 Comparison

First we compared several simulation results of the two models as listed in table 5.1.

Various simulation results of IEEE 802.11 DCF using		
Results	**Straightf. Model**	**Optimized Model**
# of generated packets	369590	368980
# of erroneous packets	5000	5000
# of successful trans. packets	246208	245896
# of collisions	8106	7997
Normalized system load	0.67	0.67
Normalized system throughput	0.45	0.45
Channel access delay in ms	1.419	1.418

Table 5.1: Various simulation results

The results are very close to each other. So we assume, that the models deliver stochastically equal simulation results. Furthermore we compared the distribution of the inter-passage time between to consecutive erroneous packets. The observed mean values are $55.373ms$ and $55.280ms$ for IEEE 802.11 DCF simulation with the Straightforward and the Optimized

Model, respectively. Furthermore the distribution curves of inter-packet times as shown below are very close to each other, as shown below:

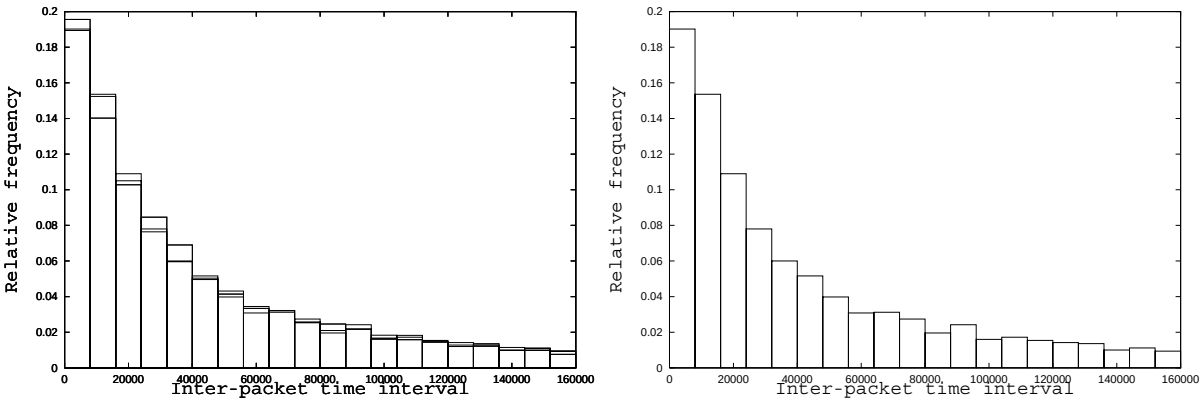


Figure 5.2: Inter packet time distribution of erroneous packets: Simulation with Straightforward (left) and Optimized Model (right)

Now we compare the the CPU and real time used for both simulation models (table 5.2 and 5.3). These times are measured with the same setup as described above (*time* command of tcsh), with a fixed packet length of 128 bytes. The results show that the simulation can be speeded up significantly with an appropriate modeling approach. The gains in simulation speed will likely be higher, if the packet size distribution has higher mean values, thus a greater fraction of large packets. In table 5.3 we show the CPU times needed for both models. We can conclude that the choice of the channel model can have an overwhelming impact.

CPU Time in Seconds	
Straightf. Model	**Optimized Model**
2044.050	23.550

Table 5.2: CPU time of simulation models

Real Time in h:min.sec	
Straightf. Model	**Optimized Model**
37:41.45	0:26.06

Table 5.3: Real time of simulation models

Chapter 6

Conclusion

In this paper we have developed an approach for speeding up simulations of wireless networks. The nice thing of our approach is that the simulation can use a relatively coarse time resolution (on the order of packets) while achieving the error modeling accuracy of bit-by-bit simulations. Furthermore, the modeling of errors is based on physical parameters of the underlying network. We have explained our approach for the special and common case of a two state markov chain, however, it is straightforward to apply the methodology to the case of N states.

We have evaluated our approach and it shows impressive reductions in simulation times, while being (by construction) stochastically equivalent to the straightforward bit-by-bit channel model. In order to show that these gains are also visible in more complex network simulations, where besides the channel there are also computationally expensive protocol processing tasks, we have incorporated our channel models into a simulator for the IEEE 802.11 DCF protocol. Also in this case the reductions in simulation time are significant.

Bibliography

[1] Pravin Bhagwat, Partha Bhattacharya, Arvind Krishna, and Satish K. Tripathi. Using channel state dependent packet scheduling to improve TCP throughput over wireless LANs. *Wireless Networks*, 3(1):91–102, March 1997.

[2] Kenneth L. Blackard, Theodore S. Rappaport, and Charles W. Bostian. Measurements and models of radio frequency impulsive noise for indoor wireless communications. *IEEE Journal on Selected Areas in Communications*, 11(7):991–1001, September 1993.

[3] D. Duchamp and N.F.Reynolds. Measured performance of wireless lan. In *Proc. of 17th Conf. on Local Computer Networks, Minneapolis*, 1992.

[4] David Eckhard and Peter Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. In *Proc. of ACM SIGCOMM'96 Conference,*, pages 243–254, Stanford University, California, August 1996.

[5] E. O. Elliot. Estimates of error rates for codes on burst-noise channels. *Bell Syst. Tech. J.*, 42:1977–1997, September 1963.

[6] Romano Fantacci and Massimo Scardi. Performance evaluation of preemptive polling schemes and arq techniques for indoor wireless networks. *IEEE Transactions on Vehicular Technology*, 45(2):248–257, May 1996.

[7] William Feller. *An Introduction to Probability Theory and Its Applications - Volume I.* John Wiley, New York, third edition, 1968.

[8] Frank Fitzek, Berthold Rathke, Morten Schläger, and Adam Wolizs. Quality of service support for real-time multimedia applications over wireless links using the simultaneous mac-packet transmission (smpt) in a cdma environment. In *Proc. Fifth International Workshop on Mobile Multimedia Communication (MoMuC'98)*, Berlin, Germany, October 1998.

[9] E. N. Gilbert. Capacity of a burst-noise channel. *Bell Syst. Tech. J.*, 39:1253–1265, September 1960.

[10] Hang Liu, Hairuo Ma, Magda El Zarki, and Sanjay Gupta. Error control schemes for networks: An overview. *MONET – Mobile Networks and Applications*, 2(2):167–182, 1997.

[11] Mesquite Software, Inc., T. Braker Lane, Austin, Texas. *CSIM18 Simulation Engine – Users Guide*, 1997.

[12] The Editors of IEEE 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Draft Standard.* IEEE, 1997.

[13] H.S. Wang and N. Moayeri. Finite state markov channel - a useful model for radio communication channels. *IEEE Transactions on Vehicular Technology*, 44(1):163–171, February 1995.

Appendix A

A Straightforward Simulation Model

For modeling purposes we have used the simulation tool CSIM18. It provides a slim library with modeling routines and the event handling system. CSIM18 incorporates no graphical user interface. Two very important characteristics of CSIM18 are very good simulation performance and stability. The modeling approach of CSIM18 is process oriented. For more information see *http://www.mesquite.com*.

```
...
long nr_of_bits=0;
long nr_of_err_bits;
STREAM error;
...

void channel_error()
{
  short int err_state=0;
  double ran1, ran2;

  create("error");

  while (1)
    {
      /* compute bit error and error state */
      ran1=stream_uniform(error, 0.0, 1.0); // error indication
      ran2=stream_uniform(error, 0.0, 1.0); // state change
      switch (err_state)
        {

        case 0:
          if (ran1 > 0.00001972644427) err_ind=NO_ERROR; else err_ind=ERROR;
          if (ran2 <= 0.00000921436463) err_state=1;
          break;

        case 1:
          err_ind=NO_ERROR;
          if (ran2 <= 0.0000132518942386) err_state=0;
          break;
```

```
      };

     /* wait a bit duration */
     hold(1/C_SPEED);
     nr_of_bits++;
     if (err_ind==ERROR) nr_of_err_bits++;
   };
}
```

The function `channel_error` represents a CSIM18 process, which is instantiated with the `create` function. After instantiation two Bernoulli experiments are executed. The first one

```
ran1=stream_uniform(error, 0.0, 1.0)
```

computes for the present time slot, which corresponds to a single bit, whether the bit is erroneous. The second one

```
ran2=stream_uniform(error, 0.0, 1.0)
```

determines the following state of the DTMC. Both experiments are done with respect to the current channel state. After one bit time `hold(1/C_SPEED)` the experiment is repeated.

Appendix B

An Improved Simulation Model

An optimized version of the Gilbert-Elliot bit error model is shown below:

```
...
long nr_of_bits=0;
long nr_of_err_bits=0;
STREAM error;
...

void channel_error()
{
  short int err_state=0;
  double ran1;
  long i,x;

  create("error");

  while (1)
    {

      /* compute bit error and error state */
      switch (err_state)
        {

        case 0:
          /* step to state 1 after x time */
          x=stream_geometric(error,0.00000921436463);
          hold(x/C_SPEED);
          err_state=1;
          /* This is for statistic: Count Bit errors and bits */
          for (i=1; i<=x; i++)
            {
              ran1=stream_uniform(error, 0.0, 1.0);
              if (ran1 < 0.00001972644427) nr_of_err_bits++;
            };
          nr_of_bits +=x;
          break;

        case 1:
          /* step to state 0 after x time */
          x=stream_geometric(error, 0.0000132518942386);
          hold(x/C_SPEED);
          err_state=0;
          /* This is for statistic: Count the bits;
          no errors in this state*/
          nr_of_bits +=x;
```

```
        break;

      };
   };
}
```

Considering a certain state (case 0 or 1) of the DTMC we compute the sojourn time in that state with `x=stream_geometric(error,0.000010120867254)`, measuring in numbers of bits. After the waiting time (`hold(x/C_SPEED)`), where `C_SPEED` is the transmission rate, the state is changed. In the case that a state has both a predecessor and a successor state (e.g. a DTMC with eight states[1]), an operation would be necessary to decide in which state to go next. This is easily accomplished by the normalized transition probabilities to the predecessor and successor state. For the computed `x` bits, the number of bit errors are evaluated (in the `for` statement).

[1] As mentioned in section 2, a Gilbert-Elliot channel model can have an arbitrary number of states. We have chosen a model with only two states, since it turned out to be sufficient for our parameters.

Appendix C

An Optimized Simulation Model

The optimized CSIM18 version of the Gilbert-Elliot Channel Model is shown below:

```
...
long nr_of_bits=0;
long nr_of_err_bits=0;
STREAM error;
double bit_err_prob; // contains current bit error probability
EVENT State_Change;  // this event indicate a state change
...

void channel_error()
{
  short int err_state=0;
  double ran1;
  long i,x,y;

  create("error");
  bit_err_prob = 0.00001972644427;

  while (1)
    {
      /* compute bit error and error state */

      switch (err_state)
        {

        case 0:
          /* step to state 1 after x time */
          x=stream_geometric(error,0.00000921436463);
          hold(x/C_SPEED); err_state=1;
          /* Set bit error prob of state 1 and indicate state change */
           bit_err_prob = 0.0; set(State_Change);
           nr_of_bits +=x;
          break;

        case 1:
          /* step to state 0 after x time */
          x=stream_geometric(error, 0.0000132518942386);
          hold(x/C_SPEED); err_state=0;
          /* Set bit error prob of state 0 and indicate state change */
          bit_err_prob = 0.00001972644427; set(State_Change);
          nr_of_bits +=x;
          break;
        };
    };
```

```
}
```

In this routine the `for`-statement of the Improved Simulation Model (Appendix B) is replaced by an indication of the current bit error probability (`bit_err_prob = 0.xxxxx`) and state change

```
set(State_Change)
```

This can be used by a packet model to determine the current bit error probability and a state change during the transmission of packets.

Appendix D

An Error-Position aware Model

Introduction

A more advanced algorithm for simulation addresses two specific features:

- it provides information about the position of bit errors and upper layer software routines can be called in order to evaluate the specific effects of bit errors on that particular position. An example using this feature is transmission with start- and stop bits, where an error in a start bit is recognized immediately, long time before any checksum is evaluated.
- it can properly handle multiple changes of the wireless channel state during a single packet.

The algorithm is given in this appendix, using C++ syntax and the CSIM library. Event processing occurs at the packet boundaries, at every time the medium switches its state within the Gilbert/Elliot model, and at every time, a bit error occurs.

The algorithm works the following way: An independent process `biterrorrate_modulator` performs switching of the channel state. It always stores the time, when the next switching will happen, in the variable `tSwitchTime`. Furthermore, the current bit error rate is stored in a global variable `dCurrentBER`. Now, when a packet or frame needs to be sent, the process `frame_sender` determines the portion of the frame which will be transmitted until the next medium switch will happen. For this portion of the frame the occurrence of errors is simulated (procedure `simulate_k_bits`). When this is done, `frame_sender` waits for the switching event and after that repeats this algorithm for the remaining portion of the frame.

The input for the procedure `simulate_k_bits` is given by the current bit error rate (BER) and by the index of the starting and stopping bit of the portion currently to transmit (w.r.t. to the whole frame), giving k bits to transmit. By construction, no channel state switch will occur during this portion, thus bit errors occur independently of each other. Since we want not to simulate bit-by-bit, we decided to use the following approach:

- At first a binomial random experiment is used to determine the number n of bit errors during the current portion.
- If this number is 0, we let pass simulation time according to the duration of the portion.
- If this number is greater than 0, we need to determine n pairwise different error positions within the portion to transmit. These error positions should be "equally distributed".
- After determining the error positions (and thus the time of their occurrence) each error is simulated, by letting the simulator for each error pass the time between two error events (or between start of the portion and the first error event) and then to notify the upper layers on the error.

So the remaining question is how we can determine exactly n distinct positions within k bits such that these are equally distributed. The most obvious approach is to create uniform random numbers for the range between minimum and maximum bit index in a loop, until exactly n distinct positions are found. To this behalf, the result of every experiment must be stored and it must be checked, whether the random number is already contained in the storage. However, this is not a good choice, since it is hard to deterministically bound the number of loop iterations needed, especially if n is not small as compared to k. So we used the following approximation: the k bits are subdivided into n intervals of equal size, the remaining bits are called *slack* and are not considered anymore. In every interval we generate a uniform random number within that interval and store the value. It can be argued, that the resulting bit errors are not uniformly distributed, especially if n is large (leading to a large slack). However, we believe that this approximation suffices, since in our experiments, for frames of 10000 bit size the difference between the distributions of inter-error-times for the approximation and a brute-force "bit-by-bit" simulation were marginal, even for relatively high bit error rates of 10^{-3}. Furthermore, especially in a wireless scenario larger frame lengths are rarely used.

Source Code

```
void biterrorrate_modulator (void)
{
  create ("biterrorrate_modulator");

  double         dTmp;

  while (TRUE)
    {
      dCurrentBER     = simPara.dGoodBER;
      dTmp            = expntl(simPara.dDurationGood);
      tSwitchTime     = clock + dTmp;
      eSwitch.set();
      hold(dTmp);
```

```
      dCurrentBER     = simPara.dBadBER;
      dTmp            = expntl(simPara.dDurationBad);
      tSwitchTime     = clock + dTmp;
      eSwitch.set();
      hold(dTmp);
    }
}


// ----------------------------------------

BOOL afCurrentlyTransmitting  [NUMBER_OF_MAC_ADDRESSES];

// ------------------------------

void signal_receive_error (...)
{
  // signals an error to upper layers at the time it occurs
  .....
}

// ------------------------------

void signal_error_event (...)
{
  // signals an error to upper layers after the end of packet
  // transmission (medium goes idle)
  .....
}

// ------------------------------

void deliver_frame (....)
{
  // make some final tests (e.g. checksum) and deliver the
  // frame
  .....
}

// ------------------------------

BOOL simulate_k_bits (double                  dBER,
                      USHORT                  usBitIdxLow,
                      USHORT                  usBitIdxHigh)
  // simulates k Bits
  // the return values indicates, whether an error was generated
{
  double  dStopTime;
  long    lLastPos, lNumBits, lNumErrs, lIntLen, lSlack;
```

```
long    lMin, lMax;
int     i;
long   *alErrPos;
BOOL    fErrorSignalled;

fErrorSignalled = FALSE;

lNumBits = usBitIdxHigh - usBitIdxLow + 1;
dStopTime = lNumBits * simPara.dBitTime + clock;

lNumErrs = binomial (dBER, lNumBits);

if (lNumErrs == 0)
  {
    // no errors ... let time pass
    hold (lNumBits * simPara.dBitTime);
    return FALSE;
  }

alErrPos = new long [lNumErrs];

// now determine the positions of the bit errors
// the interval of BitIdxLow to BitIdxHigh is divided into lNumErrs
// Subintervals of equal size. In each subinterval one bit error
// position will be determined with uniform distribution.
// if the slack is zero then the whole distribution of the
// bit errors is uniform

lIntLen = lNumBits / lNumErrs;
lSlack  = lNumBits % lNumErrs;

for (i = 1; i <= lNumErrs; i++)
  {
    lMin = usBitIdxLow + (i-1)*lIntLen;
    lMax = usBitIdxLow + i*lIntLen - 1;

    alErrPos[i-1] = uniform_int (lMin, lMax);
  }

// now we simulate the bit errors
lLastPos = usBitIdxLow;
i = 0;
while (i < lNumErrs)
  {
     // let pass time until we reach the bit error
     // position
     hold ( ((alErrPos[i] - lLastPos) + 1) * simPara.dBitTime);
     lLastPos = alErrPos[i] + 1;
```

```
      // and signal error
      signal_receive_error (...);
      fErrorSignalled = TRUE;
      i++;
    }

  delete [] alErrPos;

  // let the remaining time pass....
  hold (dStopTime - clock);

  return fErrorSignalled;
}

// -----------------------------

void frame_sender (USHORT idx)
{
  create ("frame_sender");

  while (TRUE)
    {
      // getting frame from upper layers, pointed to
      // by pFrame

      afCurrentlyTransmitting[idx] = TRUE;

      {
        double dDuration;
        USHORT usConsumedBits;
        USHORT usNumBits;
        USHORT usBitsToSimulate;
        BOOL   fErrorSignalled;
        BOOL   fCollFound;
        BOOL   fTmp;

        // Collision Detection
        fCollFound      = FALSE;
        fErrorSignalled = FALSE;
        for (i=0; i<simPara.usNumberOfStations; i++)
          {
            if ((i != idx) && (afCurrentlyTransmitting[i]))
              {
                fCollFound = TRUE;
              }
          }
        if (fCollFound)
          {
            signal_receive_error (....);
```

```
            fErrorSignalled = TRUE;
          }

        // first reset the eSwitch event
        eSwitch.clear();

        usNumBits       = (pFrame->usLen) * 8;
        dDuration       = usNumBits * simPara.dBitTime;
        usConsumedBits  = 0;

        while ((dDuration > 0.0) && (usConsumedBits < usNumBits))
          {
            if (dDuration <= (tSwitchTime - clock))
              {
                // no medium switch will happen in the remaining
                // portion of the frame

                dDuration = 0.0;
                fTmp = simulate_k_bits(dCurrentBER, usConsumedBits, usNumBits-1);
                fErrorSignalled = fErrorSignalled || fTmp;
              }
            else
              {
                // at least one medium switch will happen during
                // the current frame.

                dDuration = dDuration - (tSwitchTime - clock);
                usBitsToSimulate = (USHORT) ceil(((tSwitchTime - clock) * simPara.dBitrate));

                fTmp = simulate_k_bits(dCurrentBER,
                                       usConsumedBits,
                                       usConsumedBits + usBitsToSimulate - 1);

                fErrorSignalled = fErrorSignalled || fTmp;
                usConsumedBits = usConsumedBits + usBitsToSimulate;

                // wait for the switch event
                eSwitch.wait();
              }
          }

        // make some final tests (e.g. checksum) and deliver the
        // frame
        deliver_frame (...);
      }

      .....
    }
}
```