CARNEGIE MELLON UNIVERSITY

CARNEGIE INSTITUTE OF TECHNOLOGY
INFORMATION NETWORKING INSTITUTE

Vehicle-to-Vehicle Channel Simulation in a Network Simulator

Gerges Dib

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE

MASTERS OF SCIENCE
IN
INFORMATION NETWORKING

Thesis Advirsor: Prof. Daniel Stancil

Thesis Reader: Prof. Raj Rajkumar

Pittsburgh, PA
April 6, 2009

Abstract

Recent development in vehicle-to-vehicle (V2V) communication has increased the need for simulation to test and analyze protocols for this emerging technology. Packet level network simulators lack a realistic V2V channel module which would allow having accurate simulations. This thesis proposes an efficient method for generating a V2V channel in a packet network simulator. This method, based on V2V channel measurements and models, efficiently generates Nakagami fading whose power spectrum and fading severity change with vehicles velocity and separation. The small-scale fading envelope is used to modulate a large-scale path loss model. The implementation uses a simple pre-computed lookup table to generate the power envelope with minimal calculations in the simulator. This method is then implemented in Network Simulator 2 (NS-2).

Acknowledgements

I would like to thank my thesis advisor Professor Daniel Stancil for his guidance throughout my work for this thesis. This work would not have been possible without his insightful comments and thoughts. His constant encouragement and his strive for excellence make working with him a rewarding experience.

I would like to thank my parents and my sister. Their love and affection was the beacon that lighted my way towards finishing my graduate studies. I am also grateful to all my close friends. I am especially grateful to Samer Sobh who has always been ready to provide his help and support, George Wassouf for making the work during my graduate studies an enjoyable experience, and Konstantina Giannikopoulou for all her tenderness and devotion.

Table of Contents

List of Tables

List of Illustrations

Chapter 1

Introduction

Vehicle-to-Vehicle (V2V) communication will be an essential part of Intelligent Transportation Systems (ITS). It will allow nearby vehicles to communicate without the dependence on any infrastructure or roadside equipment. One of the major applications of V2V is road safety. Such an issue is a major problem in the automotive industry. Many traditional safety technologies such as airbags and seatbelts have been employed to address this issue. Although those technologies could minimize the damage in terms of injuries and fatalities in case of an accident, vehicular safety should go beyond this and provide technologies for preventing such accidents. From this stand, active safety systems have been deployed in the automobile industry, such as Antilock Braking System and Electronic Stability Program. Such systems get to work in critical situations in order to help the driver to keep control over the vehicle. A further step for on-road safety is using V2V to provide drivers with more information about their surroundings and give them early warnings about potential hazards on the road ahead of them. A summary of how V2V can improve on-road safety can be found in [1]. The use of V2V in conjunction with the traditional and active safety technologies would keep accident injuries and fatalities to a minimum.

Beyond road safety, V2V would provide a wide range of real-time services that would improve driving experience. Such services include road traffic information, weather conditions, internet access, automatic highway tolls payment, etc.

Research in V2V has been substantially growing recently after the Federal Communications Commission (FCC) in the United States allocated 75 MHz of spectrum from 5.85 GHz to 5.925 GHz as part of ITS for the use in Dedicated Short Range communications (DSRC) [2]. The IEEE 802.11p standard is currently under development for the use with DSRC technology in this band as well as other international allocations.

As the DSRC standard is being formed, Vehicular Ad Hoc Networks (VANETs) for V2V communication are gaining more importance. A VANET is a decentralized self configuring network providing communication means for an unlimited number of nearby vehicles. Any vehicle, irrespective of its brand, should be able to connect to such a VANET. Current efforts for designing and standardizing protocols for VANET include the California PATH project in the United States [3], the European Fleetnet project [4] and Network-on-Wheels [5]. The main challenge in VANET is developing protocols that would satisfy the performance requirement of high data throughput and low latency essential for safety applications. Studies such as [6], [7] addressed these issues and asses the feasibility and performance of VANET.

1.1 Problem Statement

VANET protocols are currently under development for the use in V2V communication. The process of designing, validating, and evaluating those network protocols require extensive simulations before any large scale deployment or even before conducting any testing using expensive hardware. The simulators should provide results that are as close as possible to the results that would be obtained in a realistic situation. This requires the simulators to provide a realistic V2V communication channel model to be used in the simulation of the designed protocols.

A popular simulator which is greatly used in ad-hoc networks research is the Network Simulator (NS) [8]. NS is an open source discrete event simulator targeted at networking research. It is freely available and highly extensible. It can simulate TCP, routing, multicast and other protocols over wired and wireless media. NS provides functionality to examine network protocols in a controlled environment so that the performance of such protocols could be tested. NS is written in C++ and an object oriented version of the Tool Command Language (Tcl) called OTcl. The objective of this thesis is to extend NS-2 (the second generation of NS) by providing a realistic V2V channel model at the 5.9 GHz band. The current propagation channel models in NS that could be used for wireless communication simulations are the simple two-ray path loss model and a shadowing model. Another

channel model had been added to NS by [9] for generating small scale correlated Ricean fading. Presently, there is no model for a realistic V2V channel.

Several issues should be addressed for building this channel model in NS. A realistic V2V mathematical channel model should be used. This model should be confirmed by measurements to accurately represent the channel. This project is a continuation of the research made by [10], which conducted a thorough study for the V2V channel at 5.9 GHz. It measured, characterized, and proposed a model for the V2V channel. This thesis will implement in NS-2 the model described by [10]. Another issue to be addressed is that the implemented propagation model should be computationally efficient. NS-2 operates on individual packets created in the simulation, and there might be thousands of such packets in one simulation run, which makes it computationally expensive. The addition of a radio frequency propagation channel model should add a minimal amount to the computational complexity in the simulator. Various methods available for simulating wireless channels will be investigated.

1.2 Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 provides some background material about channel modeling and described the V2V channel model that will be used for simulation. Chapter 3 describes a general method that could be used for the efficient generation of the V2V channel model in any network simulator. Chapter 4 first describes the NS-2 architecture and its current wireless model structure. It then shows how the simulation method in Chapter 3 was implemented in NS-2. Chapter 5 analyzes the simulation results from the implemented model and tests the accuracy of the simulation method. Chapter 6 concludes this thesis and discusses some ways to improve and expand this work.

Chapter 2

The Channel Model

The biggest challenge in wireless communications is the performance limitation posed by the wireless channel. A propagating electromagnetic signal is diffracted, reflected, absorbed, and scattered by trees, terrains, buildings, and other objects in the environment. Since such environment is constantly changing over time, the transmitted signal is impaired by the environment in an unpredictable way.

Most texts describing wireless channels [11], [12] divide channel modeling into two parts: *large-scale propagation effects* and *small-scale propagation effects*. The large-scale propagation effects model the gradual, monotonic change of the transmitted signal power with distance. An example of such a model is the free-space path loss model, which describes the $1/r^2$ decrease in signal level caused by the geometrical spreading of the signal power with distance. Another large-scale propagation effect is *shadowing*. It models the variation of the signal power due to presence of large obstacles obstructing the path between the transmitter and receiver.

On the other hand, the small-scale propagation effects model the variation of the received signal due to the destructive and constructive addition of different multipath components of a transmitted signal that is reflected and scattered from objects in the environment. These variations, unlike the ones in the large-scale propagation effects, occur over short distances on the order of the signal wavelength.

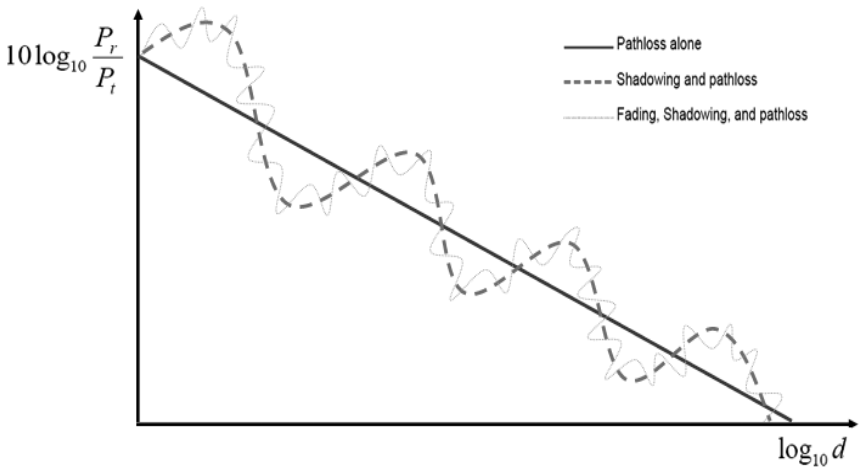

Figure 1 - Pathloss, shadowing, and fading versus distance

The effects of those channel impairments on the power of the transmitted signal are shown in . The rest of this chapter explains the various channel models. Section 2.1 explains in more details the large-scale propagation models, path loss models derived from empirical measurements, and then discusses path loss model specific to the V2V environment. Section 2.2 describes the small-scale propagation models, the statistical characterization of fading, and then it describes the V2V fading model.

2.1 Large-Scale Propagation Effects

2.1.1 Free Space Path loss

Suppose a signal with power P_t is transmitted through free space to a receiver at a distance d from the transmitter. Assuming there is no obstruction between the transmitter and the receiver, the received signal power P_r is shown to be [13]:

$$\boldsymbol{P_r} = \left(\frac{\sqrt{G_l}\lambda}{4\pi d}\right)^2 \boldsymbol{P_t}, \tag{1}$$

where λ is the wavelength of the transmitted signal, and G_l is the product of the transmit and receive antenna gains in the LOS path.

The *free space path loss* is defined as the ratio of the transmitted power to the received power:

$$P_L dB = 10\, log_{10} \frac{P_t}{P_r} = -10\, log_{10} \frac{G_l \lambda^2}{(4\pi d)^2}. \quad (2)$$

This model accurately specifies how the average received power falls with distance in a free space environment. However, such an environment is rarely encountered in practice, and more accurate models are needed for the complex propagation environments encountered by wireless communication systems.

2.1.2 Empirical Path Loss Models

Most of the path loss models for different wireless environments are based on empirical measurements. This is more practical than trying to derive a mathematical model for complex environments where it could become unmanageably cumbersome. The Okumura Model [14] and Hata Model [15] predict the path loss in large urban macrocells in the frequency ranges 150-1500 MHz. The COST 231 model [16] extended the Hata model to frequencies up to 2 GHz.

A common method used in modeling path loss based on empirical measurements is the *dual-slope piecewise linear model*. This model is characterized by a path loss exponent γ_1 above a reference distance d_o and up to a critical distance d_c, and after that the power falls off with path loss exponent γ_2:

$$P(d)\, dB = \begin{cases} P(d_o) - 10\gamma_1 log_{10}\left(\frac{d}{d_o}\right) & if\ d_o \le d \le d_c \\ P(d_o) - 10\gamma_1 log_{10}\left(\frac{d_c}{d_o}\right) - 10\gamma_2 log_{10}\left(\frac{d}{d_c}\right) & if\ d \ge d_c \end{cases}, \quad (3)$$

and $P(d_o)$ is the power at the reference distance d_o. Linear regression is used to find the parameters of this model based on the empirical measurements.

2.1.3 Shadowing

The effect of the random variation of a transmitted signal due to obstruction by objects in the signal path is called shadowing. Such variations are random due to the unknown position and nature of the blocking objects. Statistical models should be used. The most common statistical model used for shadowing is the log-normal distribution, which assumes that the ratio of the transmit to receive power $X = 10\log(P_t/P_r)$ is a random variable given by the distribution:

$$p_X(x) = \frac{1}{\sqrt{2\pi}x\sigma_x} e^{\left[\frac{(10log_{10}x-\mu)^2}{2\sigma_x^2}\right]} \qquad for\ x > 0, \tag{4}$$

where μ is the mean of X and σ_x is the standard deviation of X in dB.

The models for path loss can be superimposed with the shadowing model to show the random variation of the signal as it is attenuated with distance due to the path loss. For example the dual slope piecewise linear model can be superimposed with the log-normal shadowing as follows:

$$P(d) = \begin{cases} P(d_o) - 10\gamma_1 log_{10}\left(\frac{d}{d_o}\right) + X_{\sigma_1} & if\ d_o \le d \le d_c \\ P(d_o) - 10\gamma_1 log_{10}\left(\frac{d_c}{d_o}\right) - 10\gamma_2 log_{10}\left(\frac{d}{d_c}\right) + X_{\sigma_2} & if\ d \ge d_c \end{cases}, \tag{5}$$

where $X_{\sigma 1}$ and $X_{\sigma 2}$ are zero mean normally distributed random variables with standard deviation σ_1 and σ_2 respectively.

2.1.4 The V2V Path Loss Model

An empirical model of the V2V channel path loss given by [17] is of particular interest of this thesis. Thus authors in [17] collected two sets of channel measurements at two different dates in a suburban V2V environment at 5.9 GHz. Those measurements included the received signal strength (RSS) and the respective locations of the receiver and transmitter at the time each RSS measurement was taken. Then a dual slope piecewise linear model, as shown in Eq. 3, was used to approximate the path loss using linear regression on the measured data.

Due to small distance scales and the dominant LOS component in V2V channels, it is unlikely that there would be large obstacles between the transmitter and the receiver and thus there would be very little shadowing. The authors in [17] superimposed random variables $X_{\sigma 1}$ and $X_{\sigma 2}$ to the path loss model as shown in Eq. 5, as an approximate way of modeling the scatter of the measured data and not for modeling the shadowing effect. However, the small-scale effects were not separated in the calculation of this scatter, and thus if the model in Eq. 5 was used, the small-scale effects described in the next section should not be added to this model. Table 1 shows the parameters of the obtained model for the two measurements conducted in [17].

The critical distance d_c is described as the distance from the first Fresnel zone [13] and is calculated as$d_c = \frac{4h_t h_r}{\lambda}$, where h_r and h_t are the antenna heights of the receiver and transmitter respectively. Given from the experimental platform of [17], $h_r = 1.93\ m$ and $h_t = 1.51\ m$, d_c can be calculated to be 225 m. However the fact that a value of $d_c = 100\ m$ had been used (Table 1) is because it makes a better fit for the dual-slope linear model with the measurements. The reason why the value of d_c is less than the theoretical value is due to the presence of other vehicles or pedestrians along the roads, thus creating reflections from points higher than the ground.

Reference [18] extends the same measurements made by [17] to the Highway and Rural propagation environments. The dual-slope piecewise linear model parameters for those environments are summarized in Table 2.

Table 1 – Dual-slope piecewise linear model parameters for suburban environment as calculated from two different measurements

Parameter	Data set 1	Data set 2
γ_1	2.1	2.0
γ_2	3.8	4.0
σ_1 *(dB)*	2.6	5.6
σ_2 *(dB)*	4.4	8.4
d_c *(m)*	100	100

Table 2 - Dual-slope piecewise linear model parameters for highway and rural environments

Parameter	Highway	Rural
γ_1	1.9	2.3
γ_2	4.0	4.0
σ_1 *(dB)*	5.9	5.5
σ_2 *(dB)*	6.6	4.7
d_c *(m)*	220	226

2.2 Small-Scale Propagation Effects

2.2.1 Signal Representation

The small-scale propagation effect which results from the constructive and destructive addition of the multipath components is called *fading*. Since reflecting and scattering objects in the environment are in constant motion and their location and nature is not predictable, fading is characterized by

statistical models. In order to derive the statistics of the received signal, the signal model is described first.

Assume a transmitted signal given by:

$$\boldsymbol{s(t) = Re\{u(t)e^{j2\pi f_c t}\} = Re\{u(t)\}\cos(2\pi f_c t) - Im\{u(t)\}\sin(2\pi f_c t)}, \quad (6)$$

where *u(t)* is the equivalent lowpass signal of *s(t)*, and f_c is the carrier frequency.

The received signal is the sum of the multipath of *s(t)* introduced by the channel:

$$\boldsymbol{r(t) = Re\left\{\sum_{n=0}^{N(t)} \alpha_n(t)u(t-\tau_n(t))e^{j(2\pi f_c(t-\tau_n(t))+\phi_{D_n})}\right\}}, \quad (7)$$

where *N(t)* is the number of multipath components, $\alpha_n(t)$ is the amplitude of the n^{th} multipath component, $\tau_n(t)$ is the time varying delay of the n^{th} multipath component, and ϕ_{D_n} is the Doppler phase shift. The Doppler phase shift results from the relative velocity between the receiver and the scatterer from where the multipath component came from. The Doppler frequency shift is given by: $f_{D_n}(t) = v\cos\theta_n(t)/\lambda$, where *v* is the velocity of the receiver, and $\theta_n(t)$ is the angle between the direction of motion of the receiver and the signal direction. The corresponding phase shift is $\phi_{D_n} = \int 2\pi f_{D_n}(t)dt$.

Two multipath components are resolvable if the difference of their delay is such that $1/|\tau_1 - \tau_2| \gg B$, where *B* is the bandwidth of the transmitted signal. The *delay spread* of the channel characterizes the delay of the multipath components. One way to define the delay spread is $T_m = \max[\tau_n - \tau_o]$ where τ_o is the delay of the LOS component, and τ_n is the delay of the n^{th} component with a given minimum power of P_{thresh}. The case when $T_m > T_s$ is characterized by wideband fading models. When $T_m \ll T_s$, the channel is characterized by narrowband fading models. The simulation method will only model narrowband fading, where all the multipath components are received within a single symbol time period, and hence no inter symbol interference would occur. Thus the rest of this section describes narrowband fading in more detail.

In narrowband fading, the delay of any multipath component $\tau_n \ll T_s$. Thus any two multipath signals would not be resolvable and it can be assumed that $u(t-\tau_1) \approx u(t-\tau_2)$. In this case the received signal in Eq. 7 can be rewritten as:

$$\boldsymbol{r(t) = Re\left\{u(t)e^{j2\pi f_c t}\sum_{n=0}^{N(t)}\alpha_n(t)e^{-j\left(2\pi f_c \tau_n(t) - \phi_{D_n}(t)\right)}\right\}}. \quad (8)$$

Assuming a single tone wave is being sent (that is *u(t) = 1*), and by separating *r(t)* to in-phase and quadrature components, Eq. 8 becomes:

$$\boldsymbol{r(t) = r_I(t)cos2\pi f_c t - r_Q(t)sin2\pi f_c t}, \quad (9)$$

where

$$\boldsymbol{r_I(t) = \sum_{n=1}^{N(t)}\alpha_n(t)cos\left(2\pi f_c \tau_n(t) - \phi_{D_n}(t)\right)} \quad (10)$$

$$\boldsymbol{r_Q(t) = \sum_{n=1}^{N(t)}\alpha_n(t)sin\left(2\pi f_c \tau_n(t) - \phi_{D_n}(t)\right)}. \quad (11)$$

When the number of multipath *N(t)* is large, $r_I(t)$ and $r_Q(t)$ can be approximated as jointly Gaussian processes using the central limit theorem. This is also true when the phase $\phi_n(t) = 2\pi f_c \tau_n(t) - \phi_{D_n}(t)$ is uniformly distributed.

2.2.2 Autocorrelation function and Power Spectrum

The autocorrelation of each of the in-phase and quadrature components can be obtained as shown in [11]:

$$\boldsymbol{A_{r_Q}(\tau) = A_{r_I}(\tau) = E[r_I(t)r_I(t+\tau)] = \frac{1}{2}\sum_n E[\alpha_n^2]cos\left(2\pi v\tau \cos\frac{\theta_n}{\lambda}\right)}, \quad (12)$$

where *v* is the velocity of the receiver, θ_n is the angel between the n^{th} multipath and the direction of motion of the receiver. Note that $r_Q(\tau)$ and $r_I(\tau)$ are wide-sense stationary random processes since their autocorrelation only depends on the time difference τ.

The autocorrelation of the in-phase and quadrature components could be further simplified by assuming a uniform scattering environment [19]. It is assumed that the multipath components arrive at

the receiver uniformly from all directions with equal power $E[\alpha_n^2] = 2P_r/N$, where P_r is the total received power. Those simplification assumptions result in the autocorrelation function:

$$\boldsymbol{A_{r_Q}(\tau) = A_{r_I}(\tau) = P_r J_o(2\pi f_D \tau)}, \tag{13}$$

where $f_D = v/\lambda$ is the maximum Doppler frequency shift, and J_o is the zero order Bessel function of the first kind. The normalized autocorrelation function is shown in Figure 2. The power spectrum of $r_I(t)$ and $r_Q(t)$ can be calculated by taking the Fourier transform of the autocorrelation function in Eq. 13:

$$\boldsymbol{S_{r_I}(f) = S_{r_Q}(f) = F\left[A_{r_I}(\tau)\right] = \begin{cases} \frac{2P_r}{\pi f_D}\frac{1}{\sqrt{1-(f/f_D)^2}} & |f| \le f_D \\ 0 & else \end{cases}}. \tag{14}$$

The normalized power spectrum is shown in Figure 3.

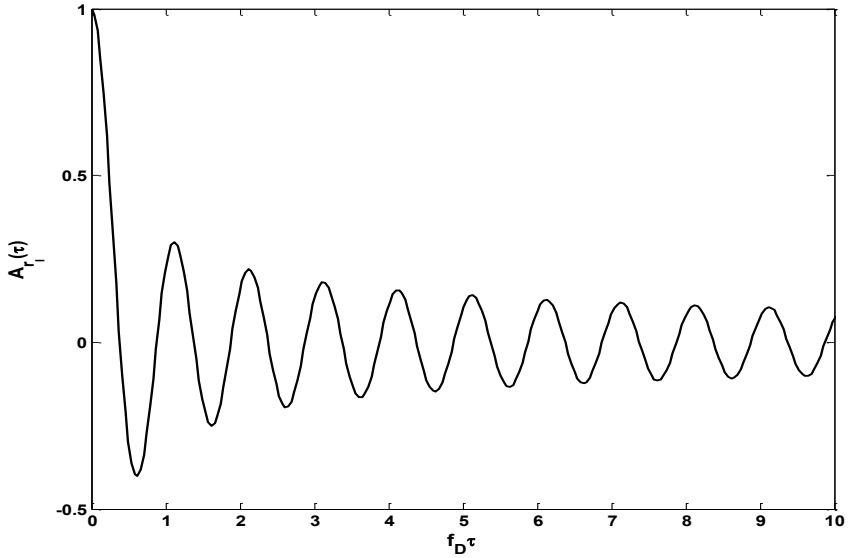

Figure 2 - The autocorrelation function in a uniform scattering environment

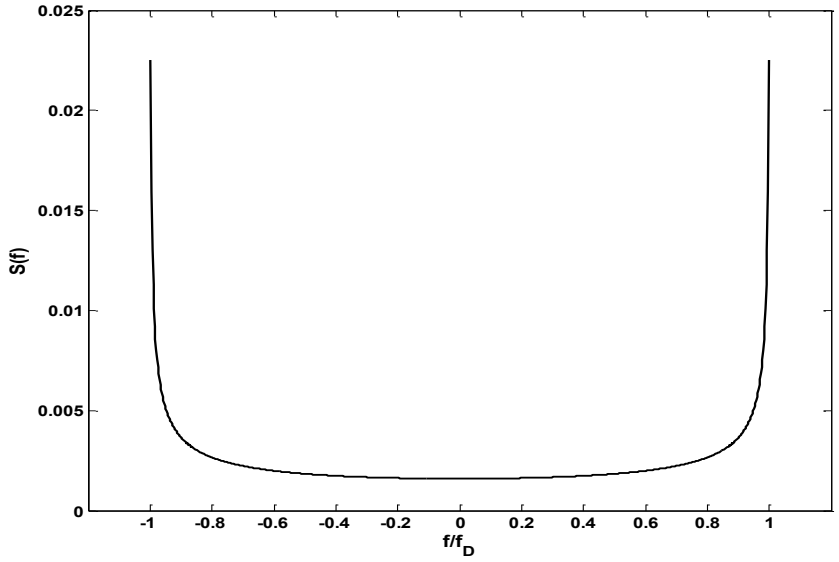

Figure 3 - The Power Spectrum in a uniform scattering environment

2.2.3 Mobile-to-Mobile Autocorrelation and Power Spectrum

The autocorrelation function and the power spectrum in Section 2.2.2 were derived for the case when one of the nodes is stationary. Other efforts to model the power spectrum of the mobile-to-mobile narrow band channel where both the transmitter and receiver are in motion have their origin from the double-ring model introduced in [20]. The double-ring model assumes uniform scattering. The autocorrelation correlation function of the in-phase and quadrature components of the received signal is given by:

$$\boldsymbol{A_{r_Q}(\tau) = A_{r_I}(\tau) = P_r J_o(2\pi f_D \tau) J_o(2\pi a f_D \tau)}, \tag{15}$$

where $a = V_t/V_r$, and the maximum Doppler frequency $f_D = V_t/\lambda$. V_t is the transmitter velocity and V_r is the receiver velocity. The autocorrelation function with different values for a is shown in Figure 4. Note that when $a = 0$, the autocorrelation function reduces to that given by Eq. 13. The power spectrum of the received signal is computed by taking the Fourier transform of the autocorrelation function in Eq. 15:

$$\boldsymbol{S(f) = \frac{\sigma_1^2}{\pi^2 f_D \sqrt{a}} K\left[\frac{(1+a)}{2\sqrt{a}} \sqrt{1 - \left(\frac{f}{(1+a) f_D}\right)^2}\right]}. \tag{16}$$

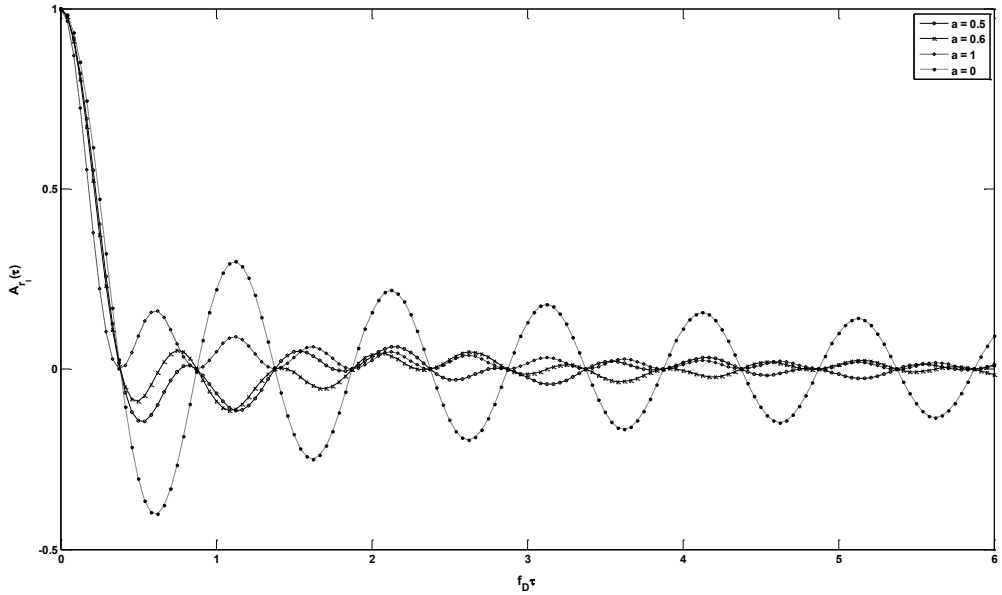

Figure 4 - Autocorrelation function for the double ring model with various values for *a*.

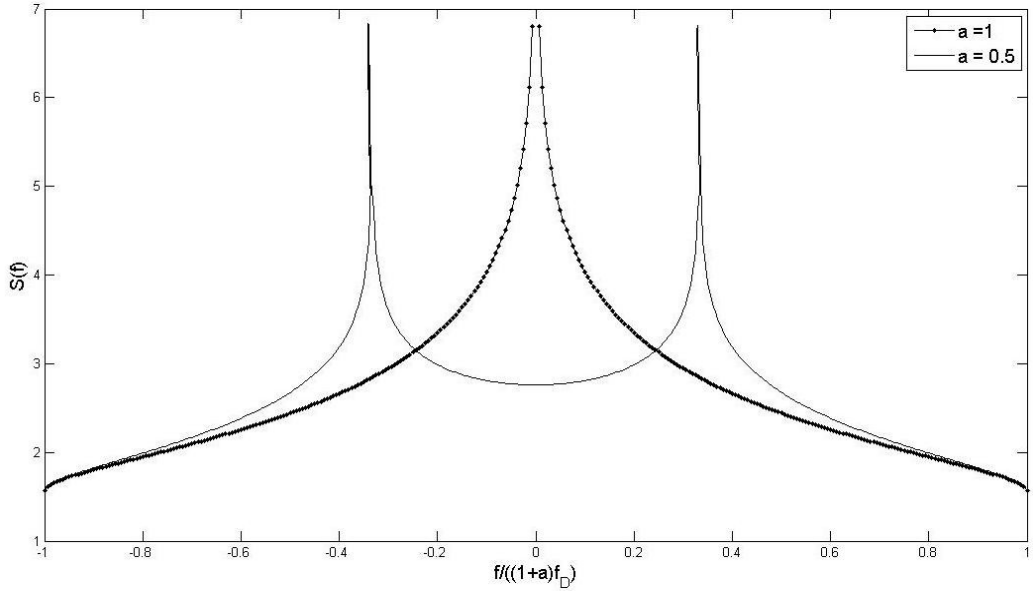

Figure 5 - The power spectrum for the double ring model with various values for *a*.

where K[.] is the complete elliptic integral of the first kind. The power spectrum for difference values of *a* is shown in Figure 5.

2.2.4 V2V Power Spectrum

The double-ring model assumes isotropic scattering, which might not always be the case in on-road environments. Reference [10] models the on-road environment in order to get an accurate power spectrum for unique for the V2V propagation channel. Unlike [20], reference [10] did not assume isotropic scattering, and it took into consideration sources of on road reflections and scattering. The power spectrum is defined as:

$$\left|S\left(f(\theta)\right)\right| = 2AG_r(\theta)G_t(\theta_t)D(d_t)D(d_r)\frac{s\rho\csc^2\theta}{\left|{}^{df}/_{d\theta}\right|} \tag{17}$$

$${}^{df}/_{d\theta} = -\frac{d\theta_t}{d\theta}v_t\sin(\theta_t)/\lambda - v_r sin(\theta)/\lambda \tag{18}$$

$$\frac{d\theta_t}{d\theta} = \frac{s^2\csc^2\theta}{(s\cot\theta - d_{tr})^2 + s^2} \tag{19}$$

and

$$\theta_t = \tan^{-1}\left(\frac{s}{s\cot\theta - d_{tr}}\right), \tag{20}$$

where $A = \frac{P_t\lambda^2\sigma}{(4\pi)^3 {d_o}^4}$, d_o is a reference distance, and σ is the radar cross section of the scattering object. $G_r(\theta)$ and $G_t(\theta_t)$ are the gain of the receiving and transmitting antennas respectively. $D(d)$ is a distance function defined as:

$$D(d) = \begin{cases} \left(\frac{d_o}{d}\right)^2 & if\ d \le d_c \\ \left(\frac{d_o}{d_c}\right)^2\left(\frac{d_c}{d}\right)^4 & if\ d > d_c \end{cases}, \tag{21}$$

s is the distance from the vehicle to the curb, ρ is the density of the scatterers on the roadside, d_{tr} is the distance between the transmitter and receiver, θ is the angle between the scatterer and the receiver, and θ_t is the angle between the scatterer and the transmitter.

2.2.5 The Signal Envelope

After characterizing the power spectrum of the signal, it is of interest to characterize the statistical properties of the received signal envelope. As it was mentioned above, the in-phase and quadrature components of the received signal have a joint Gaussian distribution with zero mean and variance σ^2.

The envelope of the received signal is given by:

$$|\boldsymbol{r}(\boldsymbol{t})| = \sqrt{\boldsymbol{r}_{\boldsymbol{I}}^{\boldsymbol{2}}(\boldsymbol{t}) + \boldsymbol{r}_{\boldsymbol{Q}}^{\boldsymbol{2}}(\boldsymbol{t})}, \tag{22}$$

which can be shown to have a Rayleigh distribution:

$$\boldsymbol{p}_{\boldsymbol{r}}(\boldsymbol{r}) = \frac{\boldsymbol{r}}{\boldsymbol{\sigma}^{\boldsymbol{2}}} \boldsymbol{e}^{-\frac{\boldsymbol{r}^{\boldsymbol{2}}}{\boldsymbol{2\sigma^2}}} \qquad \boldsymbol{z} \geq \boldsymbol{0}, \tag{23}$$

and the average received power of the signal $P_r = 2\sigma^2$.

If the channel has LOS component, then $r_Q(\tau)$ and $r_I(\tau)$ would not be zero mean, and it is shown that in this case the signal envelope has a Ricean distribution [21]:

$$\boldsymbol{p}_{\boldsymbol{R}}(\boldsymbol{r}) = \frac{\boldsymbol{r}}{\boldsymbol{\sigma}^{\boldsymbol{2}}} \boldsymbol{e}^{-\frac{\left(\boldsymbol{r}^{\boldsymbol{2}}+\boldsymbol{s}^{\boldsymbol{2}}\right)}{\boldsymbol{2\sigma^2}}} \boldsymbol{I}_{\boldsymbol{o}}\left(\frac{\boldsymbol{rs}}{\boldsymbol{\sigma}^{\boldsymbol{2}}}\right) \qquad \boldsymbol{z} \geq \boldsymbol{0}, \tag{24}$$

where I_o is the zero order modified Bessel function of the first kind, and the average received power in this case is: $P_r = s^2 + 2\sigma^2$. Ricean distribution is often characterized by the fading parameter $K = \frac{s^2}{2\sigma^2}$, which is the ratio of the LOS component power to the power of non LOS components.

A general distribution that has been shown to fit well with empirical measurements and that can represent both Rayleigh and Ricean fading by adjusting its parameters is the Nakagami distribution [22]:

$$\boldsymbol{f}_{\boldsymbol{R}}(\boldsymbol{r}) = \frac{\boldsymbol{2m^m r^{2m-1}}}{\boldsymbol{\Gamma(m)\Omega^m}} \boldsymbol{e}^{-(\boldsymbol{m}/\boldsymbol{\Omega})\boldsymbol{r}^{\boldsymbol{2}}}, \tag{25}$$

where $\Omega = E[R^2]$ is the expected power of the signal $r(t)$, and $m = \frac{\Omega^2}{(R^2-\Omega)^2} \geq 0.5$.

In Nakagami fading, m is a parameter that controls the severity of the fading. A value of $m = 1$ results in Rayleigh fading. Values of $0.5 \leq m < 1$ result in fading more severe than Rayleigh, and values of $m > 1$ result in Ricean fading. Figure 6 shows the probability density function (PDF) of the Nakagami distribution for different values of m. Note that as m tends to infinity, the Nakagami-m pdf tends to an impulse.

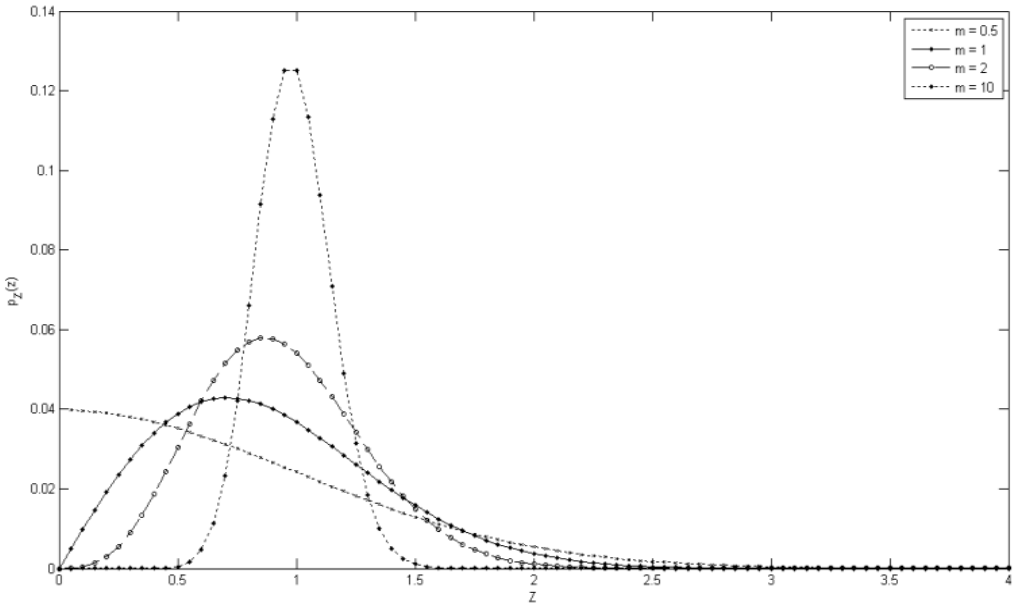

Figure 6 - Nakagami distribution for different values of m

2.2.6 Empirical Channel Characterizations

Empirical studies are important to confirm the fading channel models, especially since scattering and signal obstruction can vary greatly with frequency and with the type of environment for the signal propagation. Various measurements had been made in frequency bands in and outside the 5.9 GHz

band. Empirical studies such as [10], [23], [24], [25], has justified the use of the Nakagami fading model due to its good fit to empirical data. Reference [26] presents measurements taken for a V2V communication link at 2.45 GHz. It specifically shows how the power spectrum changes in measurements obtained in three different environments. References [27], [28], [29] show results of measurements taken for a V2V channel at 5.9 GHz in different environments, and they characterize the statistical properties of the channel and its Doppler spectra. Reference [30] shows how a mathematical channel model could be obtained from empirical data.

Reference [17] uses measurements of the V2V channel at 5.9 GHz to characterize the channel statistics. The authors use the Nakagami distribution to model the received signal envelop due to its generality and its good fit with the acquired data. However, the transmitter and receiver are moving at high speeds in V2V environments. This causes different fading statistics depending on the existence of a line-of-sight (LOS). The m parameter would be changing in inverse proportion with distance since the chance of having LOS decreases as distance increases due to turns or intersections at the road. This results in Rayleigh or Sub-Rayleigh fading. At shorter distances, it is very likely to have a LOS since there would be no obstacles in the road between the vehicles. Reference [17] has shown how m changes with distance by measuring the received signal strength versus distance then dividing the distance into bins, and the amplitude values within each bin are used as the data to fit the Nakagami distribution. Linear regression was then used to get the best fit of the variation of m with distance:

$$m = -1.3log_{10}(d/d_o) + 3.7. \tag{26}$$

2.3 Summary of V2V Channel Model

The above discussion about the characterization of the narrowband wireless channel gives sufficient information to be able to proceed and implement the V2V channel model with accurate statistics and the correct properties. A fading envelope with accurate temporal correlation and correct

fading severity would be used to modulate a large-scale path loss model. The channel model that has been implemented in NS-2 to generate the power envelope is outlined as follows:

- The large-scale path loss is modeled by the dual slope piecewise linear model shown in Eq. 3. The parameters for the equation are given by Tables 1 and 2, depending on the road environment.
- Either superimpose a log-normal model to the path loss model as shown in Eq. 5 with parameters given by Tables 1 and 2, or superimpose a correlated distance varying Nakagami fading as follows:
    - The small-scale propagation effects are modeled using Nakagami fading whose severity changes depending the time varying distance separation as shown in Eq. 26.
    - The temporal correlation for the fading is given by the mobile-to-mobile autocorrelation function given by Eq. 15. The power spectrum is given by equation 16. The power spectrum is time variant depending on a, which changes depending on the velocities of the receiver and transmitter.

Chapter 3

Simulating the V2V Channel Model

This chapter first gives some background on simulating a fading channel. Then it describes a simple and efficient method for simulating a Nakagami fading envelope whose time correlation properties are described by the double-ring model for mobile-to-mobile communication shown in Eq. 15. The time correlation, or the power spectrum, of the Nakagami process varies during the simulation run based on the velocities of the transmitter and the receiver. The *m* parameter of the Nakagami fading process varies according to the separation distance between the transmitter and receiver.

3.1 Background on Simulating a Fading Channel

3.1.1 Simulating a Nakagami Fading Channel

There is no direct way to generate an efficient and high quality Nakagami fading channel with the correct temporal correlation properties. There are two general approaches to generate the Nakagami fading with a known time correlation function. The first method maps a Rayleigh fading channel into the required Nakagami channel using some transfer function. The Rayleigh fading channel could be generated using one of the well established methods mentioned below. However, the time correlation function for the Nakagami process should be also mapped to the one required for the Rayleigh process or for each of the in-phase and quadrature components of the Rayleigh process. This approach is described in [31] and [32].

The second method decomposes the Nakagami process into a sum of Gaussian Processes [33]. The probability density function of a Nakagami process is the amplitude of the sum of squared independent Gaussian processes when *m* is an integer or half integer:

$$\boldsymbol{Z(t)} = \sqrt{\boldsymbol{X_1^2(t)} + \boldsymbol{X_2^2(t)} + \ ... + \boldsymbol{X_n^2(t)}}, \tag{27}$$

where $X_i(t)$, $i = 1, \ldots, n$ are independent and identically distributed (iid) Gaussian processes each with zero mean and variance σ_X^2. The process $Z(t)$ has a Nakagami probability density function with $m = n/2$ and $\Omega = 2m\sigma_X^2$. Then the Nakagami process could be computed in a similar fashion as a Rayleigh process by computing each of the Gaussian components $X_i(t)$ using one of the methods described below. Zhang in [33] extends this method for the case when $2m$ is not an integer by adding a correction term:

$$\boldsymbol{Z(t)} = \sqrt{\boldsymbol{\alpha}\sum_{i=1}^{p} \boldsymbol{X_i^2(t)} + \boldsymbol{\beta X_{p+1}^2(t)}}, \tag{28}$$

where $p = \lfloor 2m \rfloor$ and the coefficient α and β are given by:

$$\boldsymbol{\alpha} = \frac{2pm + \sqrt{2pm(p+1-2m)}}{p(p+1)} \tag{29}$$

$$\boldsymbol{\beta} = \boldsymbol{2m - p\alpha}. \tag{30}$$

The computation of each of the $X_i(t)$ components with the correct time correlation function requires mapping the time correlation function of the Nakagami process into the one required for each of the Gaussian components. The time correlation function of the Nakagami fading $A_z(\tau)$ is mapped to the time correlation function $A_x(\tau)$ of each $X_i(t)$:

$$\boldsymbol{A_z(\tau) = \varphi(m,1)\left\{ \ {}_2F_1\left(-\frac{1}{2}, -\frac{1}{2}; m; A_x^2(\tau) - 1\right)\right\}} \tag{31}$$

where

$$\boldsymbol{\varphi(m,1)} = \frac{\Gamma^2\left(m+\frac{1}{2}\right)}{\Gamma(m)\Gamma(m+1) - \Gamma^2\left(m+\frac{1}{2}\right)} \tag{32}$$

and

$${}_2\boldsymbol{F_1}\left(-\frac{1}{2}, -\frac{1}{2}; \boldsymbol{m}; \sqrt{\boldsymbol{A_x(\tau)}} - \boldsymbol{1}\right) = \sum_{n=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_n \left(-\frac{1}{2}\right)_n}{(m)_n} \frac{\left(A_x^2(\tau)-1\right)^n}{n!} \tag{33}$$

is the hypergeometric function with $(a)_n = a(a+1)\ldots(a+n-1)$. The Newton-Raphson method [34] could be used to solve this equation to obtain $A_x(\tau)$.

3.1.2 Simulating a Rayleigh Fading Channel

There are several approaches to simulate a Rayleigh fading channel. The sum-of-sinusoids method [35] is a direct implementation of Eq. 10 where it adds sinusoids with random phases and amplitudes. The white noise filtering method [36] appropriately filters a white noise Gaussian process to get the correct time correlation. The autoregressive method [37] makes use of the correlation matching properties of auto regression to generate the Rayleigh fading channel. This method is known to have some stability issues since the inverse of a matrix that is near singularity needs to be computed. The Inverse Discrete Fourier Transform (IDFT) method [9], [38], generates complex Gaussian processes in the frequency domain and shapes them using the required power spectrum. The real part of the inverse Fourier transform of the Gaussian processes is then taken and the Rayleigh process is computed using Eq. 22.

3.2 The Simulation Method for the V2V Channel Model

3.2.1 Overview

All the methods described in the previous section involve complex computations that would significantly degrade the performance of the network simulator if they are implemented directly within the simulator. However, the IDFT method for generating the components of a Rayleigh channel is block oriented, where the whole channel is generated at once before performing an inverse Fourier transform on it. This allows the channel to be pre-computed [9] and stored, then the network simulator could reuse it to generate the V2V channel. Since the V2V channel has a Nakagami fading as described in the previous chapter, the Nakagami process could be decomposed as shown in Eq. 27, and its Gaussian components would be obtained from this pre-computed dataset. This would avoid any expensive computations such as the Inverse Fast Fourier Transform. To calculate the fading envelope from the dataset, the simulator would need to be provided some parameters calculated from variables in the simulation scenario. However, the calculation of parameters needs minimal computation. These parameters include the velocity ratio of the receiver and transmitter, maximum

Doppler frequency, *m* parameter of the Nakagami fading, and the large-scale path loss power. The small scale fading is used to modulate the large-scale path loss.

3.2.2 Computing the Nakagami Fading Envelope in the simulator

The Nakagami fading envelope, as mentioned above, can be calculated using the decomposition in Eq. 27 – 30. However, this would require a large number of $X_i(t)$ components to be stored in the dataset, which would make the size of the dataset impractically large. Moreover, it would require a significant amount of computation when *m* becomes large. A solution for this problem is to approximate the Nakagami process as a Ricean process when $m > 1$, by mapping *m* to the Ricean *K* parameter [11]:

$$\boldsymbol{K = m - 1 + \sqrt{m^2 - m}} \tag{34}$$

and then the Nakagami process becomes a Ricean process whose envelope is:

$$\boldsymbol{Z(t) = \sqrt{\left(X_1(t) + \sigma_X\sqrt{2K}\right)^2 + X_2^2(t)}}\,. \tag{35}$$

The corresponding normalized power envelope is then:

$$\boldsymbol{Z^2(t) = \frac{1}{2\sigma_X(K+1)}\left[\left(X_1(t) + \sigma_X\sqrt{2K}\right)^2 + X_2^2(t)\right]}. \tag{36}$$

This allows simulating the Nakagami fading for any real $m > 1$, without the need of storing more than two components in the dataset. Note when *m = 1*, the Nakagami fading becomes Rayleigh fading. When *m* is between $0.5 \leq m \leq 1$, Eqs. 28 – 30 would be used to calculate the fading envelop. Eq. 28 in this case would always have *i = 2*, and it will be reduced to:

$$\boldsymbol{Z(t) = \sqrt{\alpha X_1^2(t) + \beta X_2^2(t)}}. \tag{37}$$

The corresponding normalized power envelope in this case would be:

$$\boldsymbol{Z^2(t) = \frac{1}{\alpha^2+\beta^2}\left[\alpha X_1^2(t) + \beta X_2^2(t)\right]}. \tag{38}$$

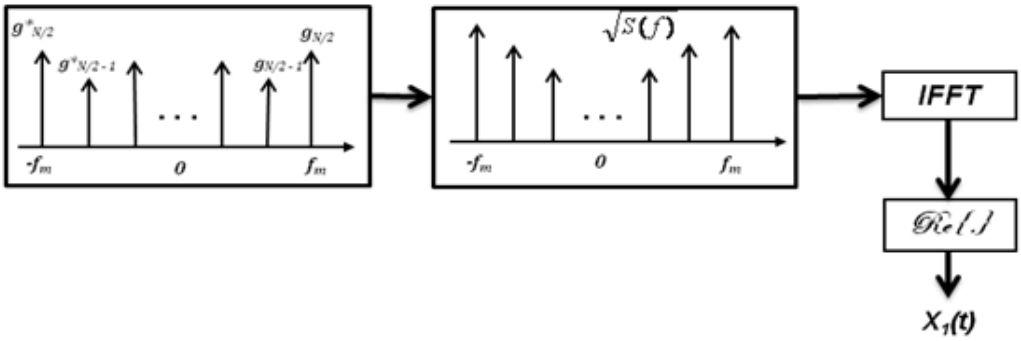

Figure 7 - Generating the a single component for a single value of a

3.2.3 Generating the dataset

As mentioned above, the IDFT method will be used to generate *X(t)*. The power spectrum in Eq. 16 would be used for shaping the generated complex Gaussian random numbers. The need to have a varying spectrum as the velocities of the vehicles change during a simulation run requires storing the same components with power spectra having different values of a. It is chosen to have spectra with a at 0.1 intervals for $0 \leq a \leq 1$. In the case the power envelope in the simulation has a spectrum with a not stored in the dataset, the spectrum with the nearest a is taken. This rounding would not have a significant effect on the correlation properties of the Nakagami fading components. For example, it can be seen from Figure 5 that the autocorrelation functions for $a = 0.5$ and $a = 0.6$ are almost overlapping.

The steps for calculating a single component in the dataset are shown in Figure 7. The following describes these steps in more details and shows how to iterate through them to generate the other components:

1. Specify the number of samples that will be used to represent the power spectrum, the maximum Doppler frequency f_D, and the frequency spacing (i.e. sampling frequency) of the samples. Note that the power spectrum should be an even function. The inverse of the frequency spacing determines the time duration of the simulation.
2. Set the value of a for the power spectrum in Eq. 16 to zero.

3. Generate a complex Gaussian random sequence for half the samples representing the positive frequency points. Conjugate this sequence to represent the negative frequency points.
4. Evaluate the spectrum from Eq. 16 at the same frequencies as in step 2, and the take its square root.
5. Multiply the generated complex Gaussian sequence with the spectrum samples.
6. Perform an IFFT on the resulting sequence and take the real part to obtain the time domain equivalent.
7. Increase a by 0.1 and repeat steps 4 – 7 until $a = 1$.
8. Repeat steps 2 – 7 to calculate the other time series.

3.2.4 Using different Doppler Frequencies

The dataset contains time series with different spectra for all values of a, however when the transmitter changes velocity, the Dopper spread $f_D = V_t/\lambda$, also changes. This imposes the need to do simulations with f_D different than the value used in constructing the dataset. This can be done by stretching or squeezing the dataset time series [9]. Let f_{D_0} be the maximum Doppler frequency represented in the dataset, f_D' be the desired maximum Doppler frequency for simulation, and Δt be the time spacing in the dataset. The time spacing of the simulation would be:

$$\Delta t' = \frac{f_{D_0}}{f_D'} \Delta t \tag{39}$$

3.2.5 The algorithm

The following summarizes the method for generating the fading envelope for V2V environment:

1. Generate $X_1(t)$ and $X_2(t)$ with different spectra as described above, and store the generated dataset.
2. Get V_r and V_t from the simulation scenario. If $V_t > V_r$, then $a = V_r/V_t$ and $f_D = V_t/\lambda$, otherwise $a = V_t/V_r$ and $f_D = V_r/\lambda$.

3. Expanded or squeeze the dataset time depending on f_D using Eq. 39.
4. Depending on the value of a, select the proper $X_1(t)$ and $X_2(t)$ components from the dataset.
5. In the simulator, calculate m using Eq. 26. The distance is known from the simulation scenario.
6. If $m > 1$, calculate K using Eq. 34. Otherwise, calculate α and β, using Eqs. 29 and 30 respectively, for $p = 1$.
7. Calculate the fading envelope using Eq. 36 if $m > 1$, otherwise use Eq. 38.
8. Add the fading envelope power in dBW to the calculated large-scale path loss power at the corresponding distance using Eq. 3.

Chapter 4

Implementation in the Network Simulator (NS-2)

4.1 Overview

NS-2 is a discrete event simulator. This means that it models the world as a list of events. It takes the next event from the list, runs it until it is done, and fetches the next event in the list. Events occur at an instant of virtual time maintained by a scheduler. However, the actual execution of the event may take arbitrary time.

The architecture of NS-2 is highly modular which makes it easy to extend. It is built using C++ as a compiled language and OTcl as an interpreted language. This dual architecture requires careful attention when adding a new module. The reason it is written in two languages is to separate data and control. The data part, which represents the core for individual packet processing, is written in C++. This makes it fast and gives full control in programming its behavior. OTcl is used for control purposes. It is used for building simulation scenarios and for periodic or triggered actions. This makes it easier to build simulations since OTcl is easy to write, it does not require knowledge of the structure of NS-2. OTcl is a scripting language and thus it does not require NS-2 to be recompiled every time a simulation scenario is changed.

4.2 NS-2 Wireless Model Structure

The major contributions to the wireless simulation features of NS-2 come from Sun Microsystems, the UC Berkeley Daedelus and the CMU Monarch projects. The current wireless simulation architecture is shown in Figure 8. It consists of the `MobileNode` at the core, which is a split object. That means it is written in C++ and OTcl code. The C++ `MobileNode` class inherits form the `class Node`. It adds mobility capabilities so that the node could move in a given topology. It also adds the ability to receive and transmit packets in a wireless channel, thus there is no requirement to

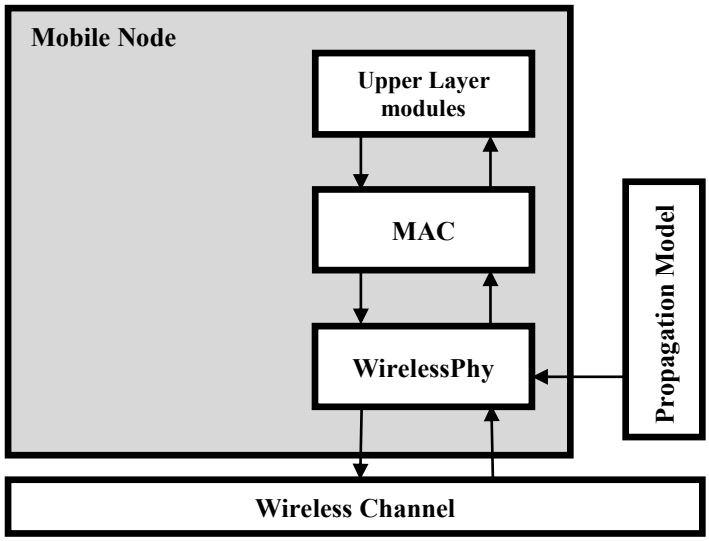

Figure 8 - NS-2 wireless simulation architecture

add links between nodes. A mobile node has a layered interface to send packets up and down from a *wireless channel* that connects it to other mobile nodes. The role of each layer is described below.

Wireless Channel: Interconnects all the mobile nodes in a single simulation scenario. It functions as an interface for the exchange of frames between mobile nodes by creating a copy of a sent frame to each of the intended receiving nodes.

WirelessPhy: When receiving, it fetches the frame from the wireless channel, and calls the *Propagation model* to get the actual reception power P_r of the frame. Then it compares P_r to a given *Carrier Sense Threshold* (CST). If P_r is greater than CST, the frame is passed to the *MAC* layer, otherwise the frame would be marked as not sensed and it is simply discarded. When transmitting, the *WirelessPhy* simply passes a frame from the *MAC* layer to the *wireless channel.*

Propagation Model: Calculates the received power in a transmitted frame and returns it to the *WirelssPhy* layer. Currently the supported propagation models are the Friis free space path loss, two ray ground, two ray ground with shadowing, and Ricean fading. This is the layer that will be extended in this chapter by adding support to the V2V propagation model.

MAC Layer: Responsible for PHY functions such as frame body reception, collision detection, and channel maintenance. There are two MAC protocols implemented in NS-2: 802.11 and TDMA.

Upper Layer Modules: Includes IP routing functionalities, transport layer functionalities and other application level manipulation of the packets.

4.3 The NS-2 Propagation Class

The base class for the propagation model in NS-2 is the `Propagation` Class. It is found in the *~/ns-2.33/mobile* directory of the NS-2 installation. All other propagation models should inherit from this base class. The propagation model for V2V will be implemented in a class named `VanetProp` which inherits from `class Propagation`. `Class Propagation` has three public `virtual` member functions:

```
virtual double Pr(PacketStamp *tx, PacketStamp *rx, WirelessPhy *);
virtual int command(int argc, const char*const* argv);
virtual double getDist(double Pr, double Pt, double Gt, double Gr,
                  double hr, double ht, double L, double lambda).
```

The most important function that does all the actual work in calculating the propagation channel is `Pr`. This function is called by the *WirelessPhy* layer every time a packet is received in order to get the calculated power of received packet which would later be used for deciding if the packet is received correctly or not. Function `Pr` takes three arguments. The first two arguments are pointers to `PacketStamp` objects that represent the transmitting and the receiving mobile nodes. Those two arguments provide information about the communicating mobile nodes. This information is needed in the calculation of the received power, such as the transmitter and receiver positions and the transmit power. The third argument is a pointer to the `WirelessPhy` object that has called this function. This argument provides information specific for the physical link, such as the carrier frequency. Function `Pr` in `Class Propagation` does not perform any calculations and it should be implemented by the classes that inherit from this base class.

The `command` function is called every time a Tcl command for the class `Propagation` is executed. To execute a command from Tcl, a C++ class has to be linked to Tcl. Linking a C++ class to Tcl will be described below. The function `command` allows executing commands from Tcl which allows changing dynamically the behavior of a simulation scenario. The `command` function should be overridden by classes that inherit from the `Propagation` class.

The function `getDist` calculates the distance between the receiver and transmitter given a received and transmitted power. The calculation of the distance depends on the underlying channel model, and thus should be overridden by classes that inherit from `Propagation` class.

Another public member function in the Propagation Class that should not be overridden by upper classes is:

```
double Friis(double Pt, double Gt, double Gr, double lambda, double L, double d).
```

This function returns the received power in the case of free space path loss. It is provided in this base class because it is likely to be used by other propagation models.

4.4 Class VanetProp

The complete code of Class VanetProp included in the files *vanetprop.h* and *vanetprop.cc* is shown in Appendix A. The following subsections explain how the V2V model was implemented in those files.

4.4.1 Linking the C++ code to OTcl

When adding C++ module to NS-2, this module should be linked to Tcl. This enables Tcl to create a shadow object of the corresponding C++ object during simulation. A special static class that inherits from `TclClass` should be added to the C++ module. In `Class VanetProp`, the following code allows such a linkage:

```
static class VanetPropClass: public TclClass {
public:
        VanetPropClass() : TclClass("Propagation/VanetProp") {}
        TclObject* create(int, const char*const*) {
                return (new VanetProp);
        }
} class_VanetProp;
```

This creates a new Tcl object that is a shadow for the current C++ object for the propagation model. The Tcl object, as a convention, is called *Propagation/VanetProp*, which indicates that `VanetProp` inherts from the class `Propagation`.

4.4.2 The Path Loss Model

The path loss model implementation will be described in this subsection, and then the next two subsections will describe how small scale effects are added on top of it. The path loss model uses the dual-slope piecewise linear model shown in Eq. 3. The path loss parameters for V2V suburban environment are shown in Table 1 and that of rural and highway environments are shown in Table 2. The parameters obtained from the two data sets in the suburban environment are averaged. Table 3 shows the parameters for the three environments.

Table 3 - The dual-slope piecewise linear model parameters for the V2V channel in urban, rural and highway environments

Parameter	Urban	Highway	Rural
γ_1	2.05	1.9	2.3
γ_2	3.9	4.0	4.0
d_c *(m)*	100	220	226

Those parameters are entered as default values in NS-2, but they could still be reconfigured in a certain simulation scenario in case other values are needed to be used. The C++ class declaration of `VanetProp` included private variables to store the values of Table 3:

```
double gamma1_suburban, gamma2_suburban, gamma1_highway, gamma2_highway,
gamma1_rural, gamma2_rural;
double dc_suburban, dc_highway, dc_rural;
```

Those C++ variables are bound to Tcl variables in order to make them easy configurable from within a simulation scenario. The binding of the C++ variables to Tcl variables is declared in the constructor of the `VanetProp` class using the `bind()` function:

```
bind("gamma1_suburban_", &gamma1_suburban);
bind("gamma2_suburban_", &gamma2_suburban);
bind("dc_suburban_", &dc_suburban);

bind("gamma1_highway_", &gamma1_highway);
bind("gamma2_highway_", &gamma2_highway);
bind("dc_highway_", &dc_highway);

bind("gamma1_rural_", &gamma1_rural);
bind("gamma2_rural_", &gamma2_rural);
bind("dc_rural_", &dc_rural);
```

The first argument in `bind()` indicates the name of the Tcl variable, and the second argument is the address of the corresponding C++ variable.

Once the C++ variables are bound to the Tcl variables, they could be easily configured within a simulation, without the need to go back into the C++ code to change them. However, it is convenient to store default values for those variables, so that they do not have to be reconfigured in every simulation. The default configuration values for all bound variables in NS-2 are all stored in a file called *ns-defaults.tcl* stored under *~/ns-2.33/tcl/lib* directory. The default values for the variables are set using Table 3:

```
Propagation/VanetProp set gamma1_suburban_ 2.05
Propagation/VanetProp set gamma2_suburban_ 3.90
Propagation/VanetProp set dc_suburban_ 100

Propagation/VanetProp set gamma1_highway_ 1.90
Propagation/VanetProp set gamma2_highway_ 4.00
Propagation/VanetProp set dc_highway_ 220

Propagation/VanetProp set gamma1_rural_ 2.30
Propagation/VanetProp set gamma2_rural_ 4.00
Propagation/VanetProp set dc_rural_ 226
```

Since there are three different environments that could be simulated, the user should be able to specify the type of the propagation environment to be used in the simulation, i.e. suburban, rural or highway. The current propagation environment is stored in a private C++ variable:

```
int environment;        /*      = 0 suburban
                                = 1 highway
                                = 2 rural
                        */
```

The current propagation environment could then be specified during a simulation run using a Tcl `command`. For example, a rural environment could be specified in a simulation scenario:

```
$prop_inst Environment Rural
```

where `$prop_inst` is an instance of the current propagation object in the simulation. When this command is executed, the `command()` function of the `VanetProp` C++ class will be called. The `command()` function should then test if the *Environment* command was executed, and change the value of the variable `environment` accordingly. The following code of the `command()` function tests if `command Environment` is executed and changes the current environment :

```
else if(!strcmp(argv[1], "Environment")){
        if(!strcmp(argv[2], "Suburban")) environment = SUBURBAN;
        else if(!strcmp(argv[2], "Highway")) environment = HIGHWAY;
        else if(!strcmp(argv[2], "Rural")) environment = RURAL;
        else{
                fprintf(stderr, "The specified environment does not exist.
                \nOptions: suburban, highway, or rural.");
                        return TCL_ERROR;
        }
                return TCL_OK;
}
```

The actual implementation of the path loss model is written in the public function

```
double dualSlope(PacketStamp *t, PacketStamp *r, double lambda, double L, double &dist);
```

which is called from function `Pr`. Function `dualSlope` first gets the parameters for the path loss depending on the specified current environment using a `switch` statement:

```
switch(environment){
        case 0: gamma1 = gamma1_suburban;
                gamma2 = gamma2_suburban;
                dc = dc_suburban;
                break;
        case 1: gamma1 = gamma1_highway;
                gamma2 = gamma2_highway;
```

```
                dc = dc_highway;
                break;
        case 2: gamma1 = gamma1_rural;
                gamma2 = gamma2_rural;
                dc = dc_rural;
                break;
        default: gamma1 = 0.0;
                 gamma2 = 0.0;
                 dc = 100.0;
                 break;
}
```

Second, it calculates the distance between the receiver and transmitter and stores it in `dist`:

```
t->getNode()->getLoc(&tx, &ty, &tz);
r->getNode()->getLoc(&rx, &ry, &rz);

//add the relative distance of antenna to the node
rx += r->getAntenna()->getX();
ry += r->getAntenna()->getY();
rz += r->getAntenna()->getZ();
tx += t->getAntenna()->getX();
ty += t->getAntenna()->getY();
tz += t->getAntenna()->getZ();

double dX = rx - tx;
double dY = ry - ty;
double dZ = rz - tz;
dist = sqrt(dX * dX + dY * dY + dZ * dZ);
```

Then it gets the gains of the receiver and transmitter antennas:

```
double Gt = t->getAntenna()->getTxGain(dX, dY, dZ, lambda);
double Gr = r->getAntenna()->getRxGain(dX, dY, dZ, lambda);
```

and then the received power is calculated using the dual-slope piecewise linear model:

```
if(dist <= d0){
       pl = 10*log10(Friis(t->getTxPr(), Gt, Gr, lambda, L, dist));
}
else if(dist <= dc){
       pl = 10*log10(P0) - 10*gamma1*log10(dist/d0);
}
else {
       pl = 10*log10(P0) - 10*gamma1*log10(dc/d0) -
           10*gamma2*log10(dist/dc);
}

return pl;
```

Note that when the distance is less than the reference distance, the free space path loss model is used, and `P0` is the power calculated at the reference distance:

```
double P0 = Friis(t->getTxPr(), Gt, Gr, lambda, L, d0);
```

4.4.3 The Fading Model

4.4.3.1 Loading the dataset

The general method used to generate V2V fading process was outlined in Section 3.2.5. First, the dataset was generated using MatLab. The code for generating this dataset is shown in Appendix B. The name of the data file should be given as an input using a Tcl command from the simulation scenario:

```
$prop_inst LoadPropFile env.txt
```

where `$prop_inst` is an instance of the current propagation environment which is a `VanetProp` object, and then the arguments `LoadPropFile` and `env.txt` are sent to the `command` function of class `VanetProp`. The part of `command` function that is executed when this command is called:

```
else if(!strcmp(argv[1], "LoadPropFile")){
       if((file_loadin = loadFile(argv[2])) == TCL_OK)
              initialized = 1;
       return file_loadin;
}
```

This checks if the `LoadPropFile` is provided as the first argument, and then it calls the function `loadFile` giving it the second argument of the command, which should be the data file name. It also sets the value of the variable `initialized` to 1, which enables the addition of the fading effects. The values of the parameters used to build the dataset are stored in the data file. Those parameters include the Doppler spread f_D, the sampling frequency f_s, the total number of samples for each component N, and the number of components *a_col*. Function `loadFile` first checks the file for those parameters and stores them in local private variables of the class `VanetProp`. The actual data components from the file are stored in a two dimensional array *envelop_data*. Each column in the two dimensional array stores a single component. The array is initialized at startup, before the simulator virtual time begins.

4.4.3.2 Supporting multiple nodes

Function `Pr` implements the fading model and returns the correct power value of the envelope depending on the current simulator virtual time. One issue that should be taken into consideration is that multiple nodes cannot have the same channel, since this is not physically correct. To avoid this, the current time index for each channel between two nodes is stored in a two dimensional array `node_time_index`. If it is the first time communication occurs on this the channel (i.e. between a specific transmitter and receiver), a unique starting point is used to index the dataset. This starting point is selected depending on the node IDs of the receiver and transmitter. Another similar array `node_sched_time` keeps track of what was the scheduler virtual time the last time the channel was used. The following code shows how to uniquely set the starting index from the components in the dataset:

```
if(node_time_index[n1id][n2id] == -1.0){
       if (max_node_id) {
              node_time_index[n1id][n2id] = int(floor(double((n1id-
                1)*N)/double(max_node_id) +
                 double((n2id)*N)/double(max_node_id*max_node_id)));
              node_sched_time[n1id][n2id] = 0;
       }
       .
       .
       .
}
```

Variables `n1id` and `n2id` are the IDs of the receiver and transmitter. Variable `n1id` should be greater than `n2id`.

4.4.3.3 Getting the time index

To get the proper fading power time index according to the simulator virtual time, the current simulator time is squeezed or stretched depending on the current value of the Doppler spread f_D, and then added to the last accessed time index of the current communication channel. First, the velocities of the receiver and transmitter nodes are fetched:

```
v1 = t->getNode()->speed();
v2 = r->getNode()->speed();
```

and then the Doppler spread is calculated:

```
fm = (v2 > v1) ? (v2/lambda) : (v1/lambda);
```

and the simulator time is then scaled according to the calculated Doppler spread using Eq. 39. This is then added to the previous value in the array `node_time_index` for the current channel. Array `node_sched_time` is also updated with the current scheduler time:

```
node_time_index[n1id][n2id] += (Scheduler::instance().clock() -
                            node_sched_time[n1id][n2id])*fs*fm/fm0;
node_time_index[n1id][n2id] -= double(N) *
          floor(node_time_index[n1id][n2id]/double(N));
node_sched_time[n1id][n2id] = Scheduler::instance().clock();
```

4.4.3.4 Calculating the fading envelope power

To be able to calculate the Nakagami fading envelop, *m* should be calculated first. This is done using Eq. 26:

```
m = -1.3*log10(dist/d0) + 3.7;
```

Then the value of *a* is calculated:

```
if( v1 == 0.0 && v2 == 0.0) a = 0;
else a = (v2 > v1) ? (v1/v2) : (v2/v1);
```

and then transformed into an index to be able to select the proper power spectrum from the array `envelop_data`:

```
int a_ind = a/0.10 + 0.5;
```

the components of the Nakagami power envelope are then fetched from the dataset:

```
x = Interpolate(n1id, n2id, a_ind);
y = Interpolate(n1id, n2id, 11+a_ind);
x = x ? x : 0.000001;
y = y ? y : 0.000001;
```

Note that the components `x` and `y` might happen to be zero, and this might cause errors in the simulator if a zero value for the power is returned. In this case the components are given a small value 0.000001 to avoid such errors. The time sample that is calculated as shown in section 4.4.3.3 is used

to index the rows of array `envelop_data` and the value of `a_ind` is used to index the columns of the array. However, the selected time sample that is stored in `node_time_index[n1id][n2id]` could have a fractional value, and thus the value to be fetched is between two consecutive time samples in the datafile. Interpolation is used in case this happens. Function `Interpolate` returns the interpolated comoponent value. This function uses Lagendre's polynomials to interpolate between the three samples that are in the vicinity of the index stored in `node_time_index[n1id][n2id]`.

The power sample is then calculated using Eqs. 34, 36, if $m > 1$:

```
if(m > 1){
       K = m - 1.0 + sqrt(m*m - m);
       x += sqrt(2*K);
       fad_pwr = (x*x + y*y)/(2*(K+1));
}
```

and using Eqs. 29, 30, 38, if $0.5 \leq m \leq 1$:

```
else{
       if(m < 0.5) m = 0.5;
       alpha = m + sqrt(m*(1-m));
       beta = 2*m - alpha;
       fad_pwr = (alpha*x*x + beta*y*y)/(alpha*alpha+beta*beta);
}
```

4.4.4 The Log-Normal Model

The log-normal model is used when the fading model described in Section 4.4.3 is not enabled. This happens when the data file is not loaded, i.e. the command:

```
$prop_inst LoadPropFile env.txt
```

is not declared in the Tcl simulation file. In this case, the integer variable `initialized` would have a value of zero. This will keep the fading effects disabled and enable the addition of the Log-Normal Model. The log-normal model is added on top of the path loss as shown in Eq. 5. The code for calculating the dual slope piece wise linear model shown in section 4.4.2 is slightly modified to add the log-normal model in case no data file is specified:

```
scatter = 0;

if(dist <= d0){
       pl = 10*log10(Friis(t->getTxPr(), Gt, Gr, lambda, L, dist));
       if(!initialized) scatter = ranVar->normal(0.0, std_db1);
}
else if(dist <= dc){
       pl = 10*log10(P0) - 10*gamma1*log10(dist/d0);
       if(!initialized) scatter = ranVar->normal(0.0, std_db1);
}
else {
       pl = 10*log10(P0) - 10*gamma1*log10(dc/d0) -
             10*gamma2*log10(dist/dc);
       if(!initialized) scatter = ranVar->normal(0.0, std_db2);

}
return (pl+scatter);
```

`Class ranVar` is used for random number generation in NS-2. It is used to obtain a number drawn from a normal distribution with a given mean and standard deviation for representing the scatter of the path loss data. The values of the standard deviations `std_db1` and `std_db2` are obtained according to Table 4.

Table 4 - The stand deviation of the log-normal model for different environments

Parameter	Urban	Highway	Rural
σ_1 (dB)	4.1	5.9	5.5
σ_2 (dB)	6.4	6.6	4.7

4.4.5 The Returned Power Value

The calculated fading power value `fad_pwr` as shown in section 4.4.3.4 is used to modulate the calculated path loss power value `pl` as shown in section 4.4.2. The power returned back to the *WirelessPhy* layer is the products of those two values:

```
return pl*fad_pwr;
```

Chapter 5

Simulation Results

The simulation results using the new V2V channel model has been tested to confirm their accuracy. This was done by comparing the simulation statistics to the theoretical statistics. Those statistics include the probability density function and the autocorrelation function of the power envelope. However, the fact that the statistics of the fading envelope are constantly changing during the simulation makes it difficult to verify the simulation results when the transmitter and receiver are moving and varying their velocities. A test simulation scenario in NS-2 is shown in Appendix C. In this scenario, the following parameters were specified:

- Rural environment
- Transmitter velocity: 0 m/s
- Receiver velocity: 20 m/s
- Omnidirectional antennas
- Antenna height: 1.51 m (for both receiver and transmitter)
- Transmit power: 0.2818 W
- Frequency: 5.9 GHz

A tracing function was used to separately record the path loss power, scatter power, and fading power of the received power envelope. The distance, time, and sample indices were also recorded. This recorded data was then analyzed using MatLab to study the performance of the model.

5.1 The Signal Power Envelope

In the conducted simulation scenario, the receiver is constantly moving away from the transmitter. Thus plotting the power envelope versus distance is equivalent of plotting it versus the time. Each component of the power envelope is observed separately to confirm that each part of the model is

correct. Figure 9 shows the path loss power as distance increases. It shows clearly how the path loss exponent changes at the critical distance at 226 m (rural environment). The power at distance 1 m (reference distance) is calculated using the free space path loss model. From the graph it is shown to be around -53 dB.

Figure 10 shows the scatter superimposed on the path loss when fading is disabled. This scatter is modeled by the log normal distribution. In the rural environment the variance of the scatter after the critical distance is less than before the critical distance. It can be seen from Figure 10 that the scatter narrows down after the critical distance.

Figure 11 shows the fading envelop when Nakagami fading is enabled and superimposed on the path loss model versus the *log* of distance. Notice that the fading severity increases as the distance increases, as expected according to Eq. 26. Figure 12 shows the fading power envelope resulting from the simulation. This is the same as Figure 11, but with power is plotted versus linear distance.

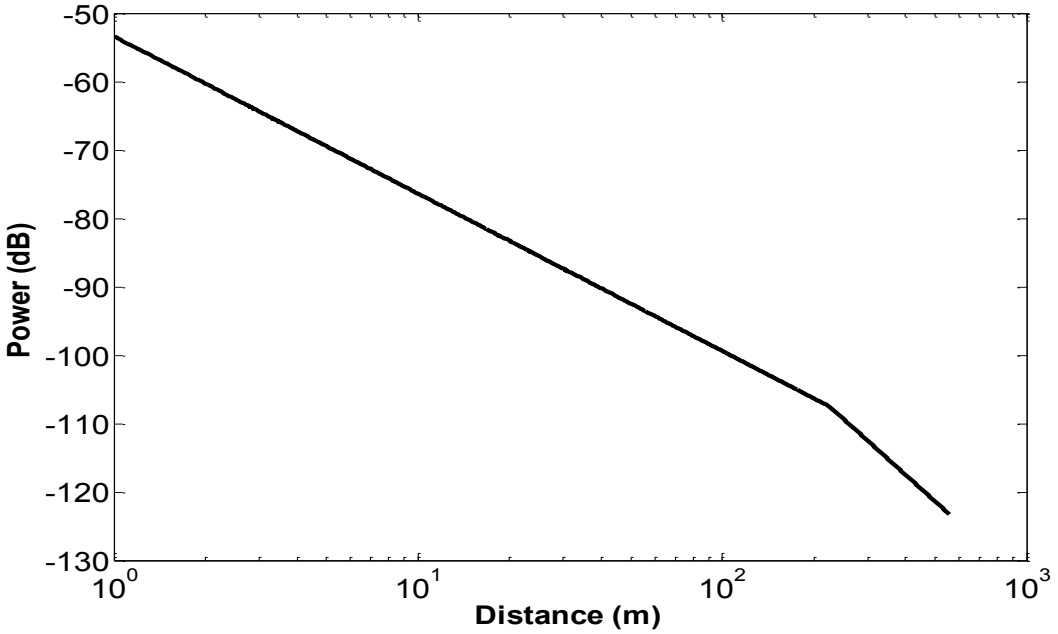

Figure 9 - Path loss power versus the log of distance

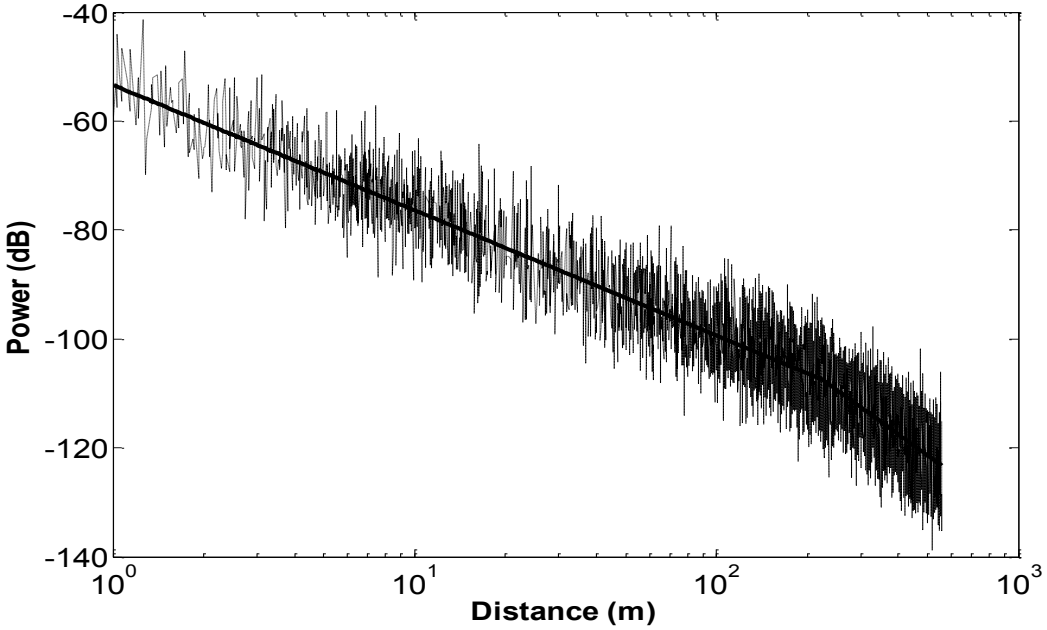

Figure 10 - Log-normal scattering superimposed on the path loss

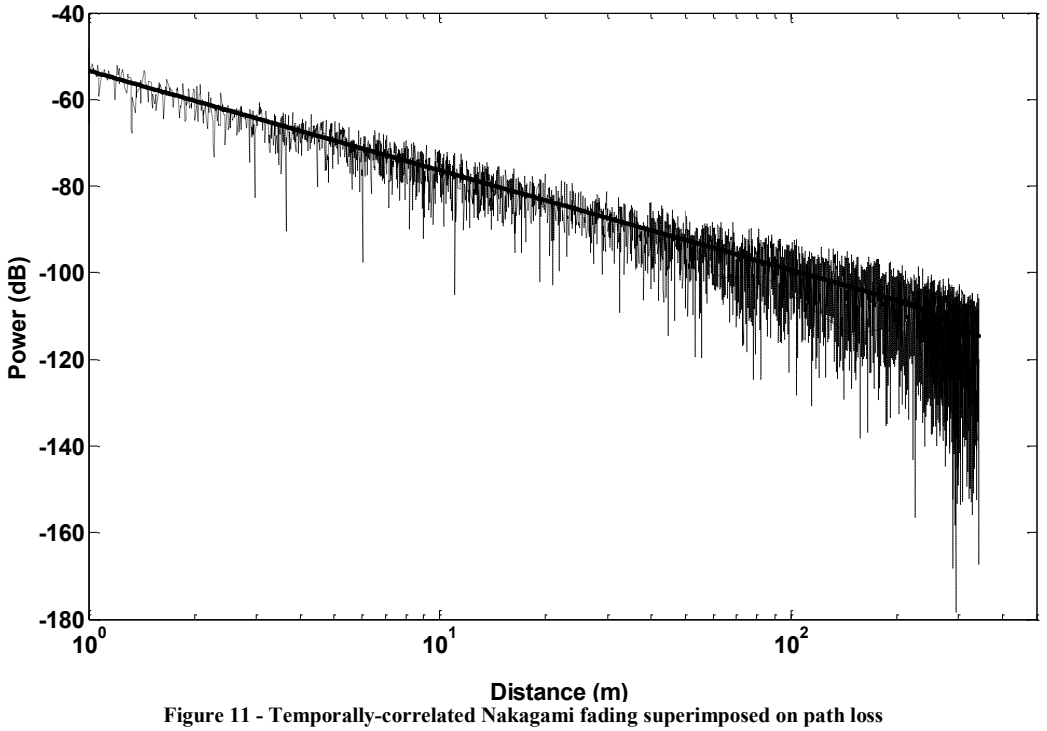

Figure 11 - Temporally-correlated Nakagami fading superimposed on path loss

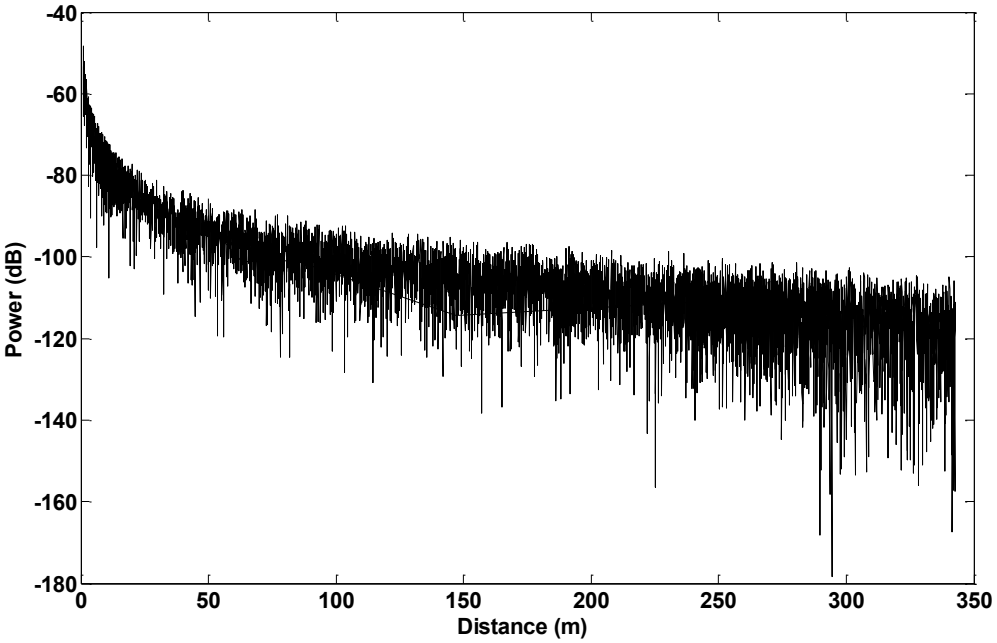

Figure 12 - The fading power envelop versus linear distance

5.2 Fading Probability Density Function

The probability density function (PDF) given in Eq. 25 is for the fading signal envelope *Z(t)*. However, in NS-2 the power envelope is returned, which is the square of the signal amplitude $Z^2(t)$. The Nakagami PDF of $Z^2(t)$ is defined as:

$$p_{Z^2}(r) = \left(\frac{m}{\Omega}\right)^m \frac{r^{m-1}}{\Gamma(m)} e^{\frac{-mr}{\Omega}} \tag{40}$$

A simulation scenario was conducted similar to the one described above. However, the transmitter is moving at a constant velocity of 20 m/s in the same direction as the receiver, thus keeping the distance between them constant. This will make the fading parameter *m* constant. This allows plotting the PDF of $Z^2(t)$ for a given *m*. The distance was kept constant at 50 m which would give $m = 1.4913$ according to Eq. 26. shows that the distribution of the simulated power envelop closely resembles the theoretical distribution given in Eq. 40 for $m = 1.4913$. Note $\Omega = 1$ since the large scale effects are not taken into consideration. The PDF of the simulated fading envelop was plotted by counting all the

power values that lie within small equidistant intervals and then dividing this value by the total number of samples in the simulation.

The results from the simulation scenario described at the start of this chapter were used to characterize the PDF of the power envelope in order to be able to know how the distribution would look when the distance is varying (which would cause *m* to vary). However, to be able to know if the resulting distribution is accurate, the corresponding theoretical distribution is needed to be known. In Eq. 40 the random variable is Z^2, but each value of Z^2 is obtained for a different value of *m* in the simulation. Figure 14 shows how *m* changes during the simulation as the distance increases between the receiver and transmitter. The mean of *m* during the simulation run is calculated to be 0.97, and the variance is 0.52. Figure 15 shows the PDF of the simulated power envelop and the Nakagami PDF for $m = 0.97$. It can be seen that the simulated envelope resembles the Nakagami PDF with the mean value of *m*. The simulated envelop PDF also lies between the Nakagami PDFs for $m = 0.97 \pm 0.52$ (not shown in Figure 15).

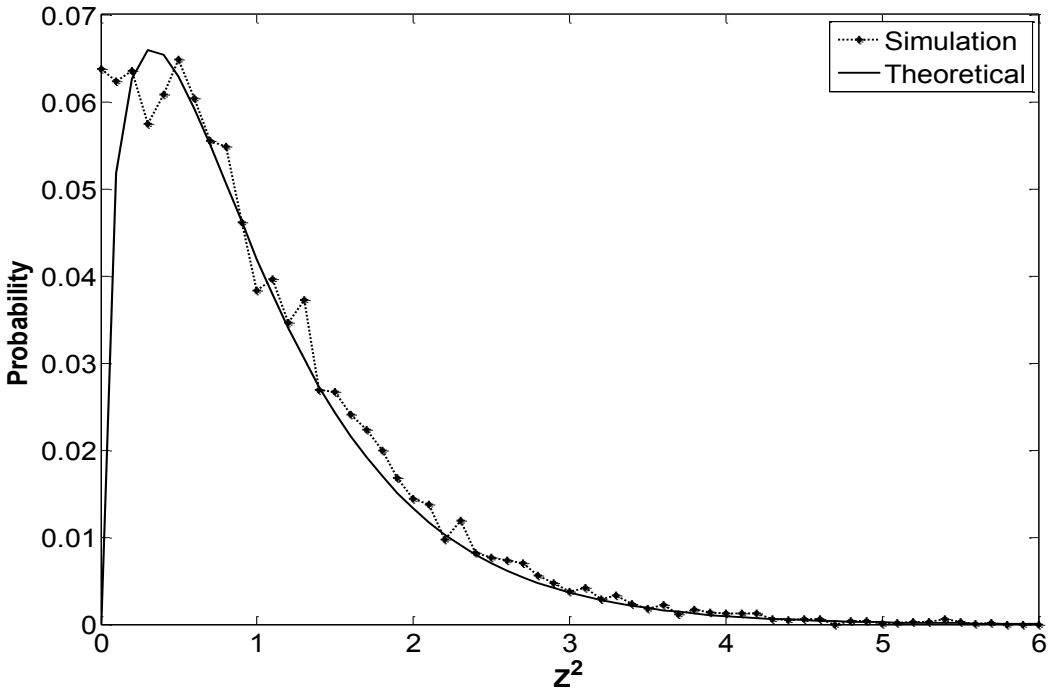

Figure 13 - Comparison of simulated and theoretical Nakagami probability density function when m = 1.49

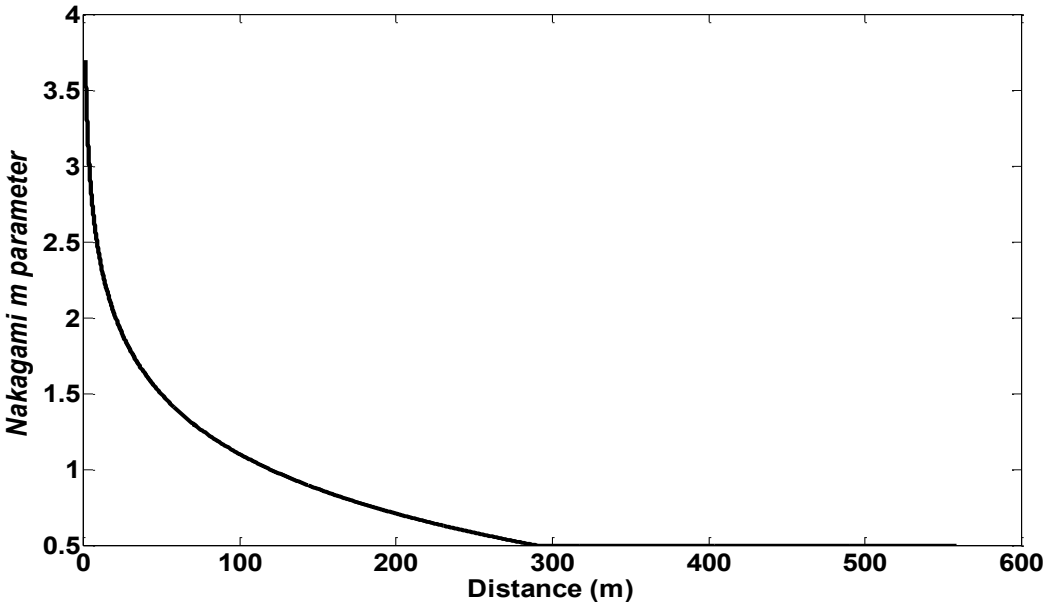

Figure 14 - The m parameter as a function of distance as resulting from the simulation

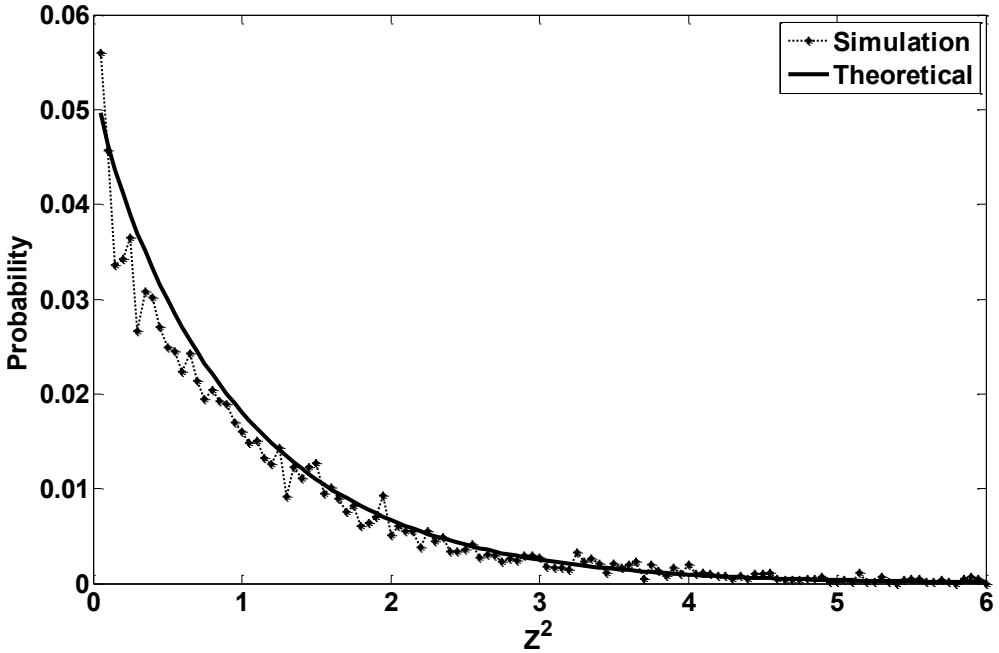

Figure 15 - Comparison of simulated and theoretical Nakagami probability density function when *m* is varying. The theoretical curve is for *m = 0.97*.

5.4 The autocorrelation function

The autocorrelation function for the power envelope components ($X_1(t)$ and $X_2(t)$) is given by Eq. 15. However, the autocorrelation function of the power envelope is required to be derived. The Nakagami fading power envelope autocorrelation function could be concluded using Eq. 31. This function can be approximated as $A_Z(\tau) = A_x^2(\tau)$. The normalized autocorrelation function of the power envelope would then be:

$$A_Z(\tau) = [J_0(2\pi f_D \tau) J_0(2\pi a f_D \tau)]^2 \quad (41)$$

When the transmitter is stationary and the receiver is moving at a constant velocity, the autocorrelation function does not change during the simulation run. The parameters for the autocorrelation function for the values mentioned at the start of this chapter are: a = 0, and $f_D = V_R/\lambda$ = 20/0.0508 = 393.333 Hz. Figure 16 shows the autocorrelation of the simulation results, along with the corresponding theoretical autocorrelation.

Another scenario is when the transmitter and receiver are both moving at the same velocity of 20 m/s. This yields an autocorrelation function with a = 1, and f_D = 393.3 Hz. The simulated and theoretical autocorrelation functions are shown in Figure 17. It can be seen that the simulation results are similar to the expected results.

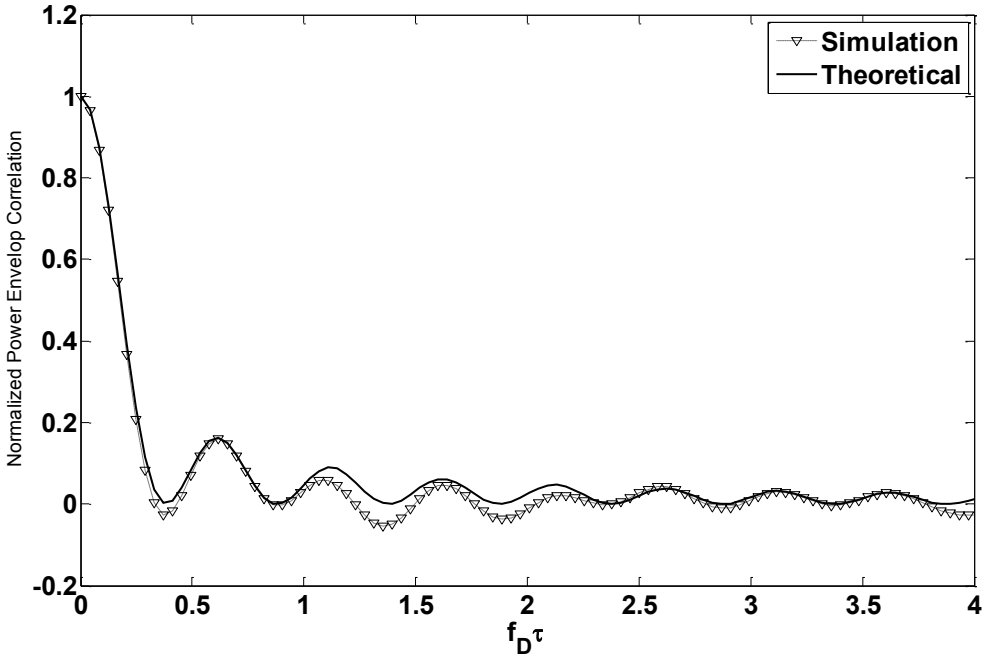

Figure 16 - Theoretical and simulated autocorrelation funtion for the fading power envelop

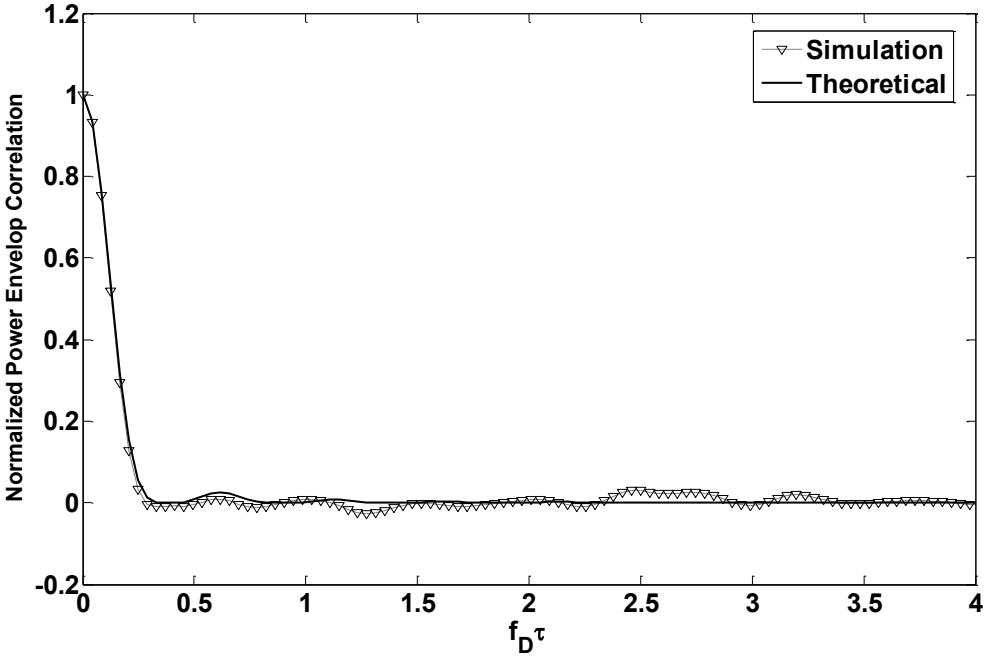

Figure 17 - Theoretical and simulated autocorrelation function for the fading power envelop for *a = 1*.

Chapter 6

Conclusion and Future Work

As V2V technology is emerging and becoming an integral part of ITS, new network protocols need to be tested and evaluated for such technology. Network simulation is an important testing methodology which provides flexible and inexpensive means for evaluating new network protocols. However, network simulators need to provide a realistic propagation model to have accurate results that would allow correct analysis of new network protocol performance. The V2V channel differs from other wireless channels since the transmitter and receiver are both moving in a confined space, and their velocities are constantly changing.

This thesis documented the V2V large-scale and small-scale propagation channel effects based on previous measurements and studies. It then proposes an efficient method for generating small-scale fading for the V2V channel in a network simulator. A Nakagami process was decomposed into two Gaussian components. Each component was replicated several times and then each replica was shaped by a different spectrum to obtain the time correlation for various values of the vehicles velocities. Those values were then stored to be used during simulation. The dataset can be reused in long simulation runs. A key contribution of this work is the realization of a computationally efficient fading model that can be varied during the simulation run to describe the effects of changing distance between vehicles in a link, and changing vehicle speeds. The method was implemented in NS-2, and the fading envelope resulting from the method is superimposed on a large-scale path loss model for V2V. The simulation results accurately modeled the statistics of the V2V channel model.

6.1 Future Work

Implementation in NS-3. This thesis implements the V2V Channel Model in NS-2.33. Although the second distribution of NS is currently the most popular, NS-3 has already been released. NS-3 is a new simulator with a totally different architecture than NS-2, and there is not compatibility between

the two simulators. NS-3 is written entirely in C++ which removes some complexities of NS-2 due to its dual architecture. Simulation scripts in NS-3 could be written in C++ or Python. The different architecture of NS-3 would require porting the current NS-2 V2V channel model to make it work in NS-3.

Using a more accurate power spectrum. The V2V channel measurements made in [10] has shown that although the double ring power spectrum model is a good fit with the measured data, a better spectrum was derived specifically for V2V environment. This power Spectrum is given in Eqs. 17 – 20. This spectrum depends on V_t, V_r, and on the distance between the receiver and transmitter d_{tr}. This would require a modification of the simulation method described in this thesis to be able to make this power spectrum vary with time as the wireless nodes move and vary their velocities in the simulation scenario.

Appendix A

Class VanetProp

Class Declaration: File *vanetprop.h*

```
#ifndef V2VDSRC_H
#define V2VDSRC_H

#include <wireless-phy.h>
#include <packet-stamp.h>
#include <propagation.h>
#include <rng.h>

#define BUFF_SIZE 4096
#define SUBURBAN 0
#define HIGHWAY 1
#define RURAL 2

class VanetProp : public Propagation {
       public:
              VanetProp();
              virtual double Pr(PacketStamp *t, PacketStamp *r, WirelessPhy *ifp);
              double dualSlope(PacketStamp *t, PacketStamp *r, double lambda, double
L, double &dist, double &scatter);
              virtual int command(int argc, const char*const* argv);
              //get interference distance
              virtual double getDist(double Pr, double Pt, double Gt, double Gr,
                       double hr, double ht, double L, double lambda);
              ~VanetProp();

       private:
              RNG *ranVar;
              int seed;
              int initialized;

              //pathloss exponents for the large-scale model
              double gamma1_suburban, gamma2_suburban, gamma1_highway,
gamma2_highway, gamma1_rural, gamma2_rural;
              //the standard deviation for the log normal shadowing
              double std_db1_suburban, std_db2_suburban, std_db1_highway,
std_db2_highway, std_db1_rural, std_db2_rural;
              //the critical distance for the dual slope model
              double dc_suburban, dc_highway, dc_rural;

              double **envelop_data;        //this array is used to load data from
the
              int a_col;                    // file number of spectrums
              double **node_time_index;     //those 2 variables are used to store
the
              double **node_sched_time;     // previous time for nodes.
              double fs;                    //sampling rate of data in the file
              int N;                        //number of samples in file
              double fm0;                   //maximum doppler frequency in file
              double fm;                    //maximum doppler frequency in scenario
              int max_node_id;
              int environment;              /*      = 0 suburban
                                                    = 1 highway
                                                    = 2 rural
                                            */
```

```
                int d0;                         // the reference distance
                BaseTrace *pwr_tracer;
                void trace(double index, double dist, double pl, double fad_prw,
double scatter);
                int loadFile(const char *file_name);
                double Interpolate(int, int, int a_ind);
};

#endif
```

Class Implementation: File *vanetprop.cc*

```
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>

#include <antenna.h>
#include <vanetprop.h>
#include <ctype.h>

static class VanetPropClass: public TclClass {
public:
        VanetPropClass() : TclClass("Propagation/VanetProp") {}
        TclObject* create(int, const char*const*) {
                return (new VanetProp);
        }
} class_VanetProp;


VanetProp::VanetProp() : Propagation()
{
        pwr_tracer = 0;
        initialized = 0;
        N = fm = fs = 0;
        a_col = 0;
        node_time_index = new double*[1];
        node_time_index[0] = new double[1];
        node_sched_time = new double*[1];
        node_sched_time[0] = new double[1];

        /******set some values by default********/
        d0 = 1.0;                       //set reference distance d0 = 1 m;
        environment = SUBURBAN;              //set default environment to suburban
        /*************************************/

        bind("gamma1_suburban_", &gamma1_suburban);
        bind("gamma2_suburban_", &gamma2_suburban);
        bind("std_db1_suburban_", &std_db1_suburban);
        bind("std_db2_suburban_", &std_db2_suburban);
        bind("dc_suburban_", &dc_suburban);

        bind("gamma1_highway_", &gamma1_highway);
        bind("gamma2_highway_", &gamma2_highway);
        bind("std_db1_highway_", &std_db1_highway);
        bind("std_db2_highway_", &std_db2_highway);
        bind("dc_highway_", &dc_highway);

        bind("gamma1_rural_", &gamma1_rural);
        bind("gamma2_rural_", &gamma2_rural);
        bind("std_db1_rural_", &std_db1_rural);
        bind("std_db2_rural_", &std_db2_rural);
```

```
        bind("dc_rural_", &dc_rural);

        bind("seed_", &seed);

        ranVar = new RNG;
        ranVar->set_seed(RNG::PREDEF_SEED_SOURCE, seed);
}

VanetProp::~VanetProp()
{
        int i;
        for(i = 0; i < N; ++i){
                delete [] envelop_data[i];
                envelop_data[i] = 0;
        }
        delete [] envelop_data;
        envelop_data = 0;

        for(i = 0; i < max_node_id; ++i){
                delete [] node_time_index[i];
                delete [] node_sched_time[i];
                node_time_index[i] = 0;
                node_sched_time[i] = 0;
        }
        delete [] node_time_index;
        delete [] node_sched_time;
        node_time_index = 0;
        node_sched_time = 0;
}

int VanetProp::command(int argc, const char* const *argv)
{
        TclObject *obj;
        int file_loadin;
        int i, j;

        if(argc == 3){
                if(!strcmp(argv[1], "Tracer")){
                        if( (obj = TclObject::lookup(argv[2])) == 0) {
                                fprintf(stderr, "Propagation: %s lookup of %s
failed\n", argv[1], argv[2]);
                                return TCL_ERROR;
                        }
                        pwr_tracer = (BaseTrace *) obj;
                        assert(pwr_tracer);
                        return TCL_OK;
                }
                else if(!strcmp(argv[1], "LoadPropFile")){
                        if((file_loadin = loadFile(argv[2])) == TCL_OK)
                                initialized = 1;
                        return file_loadin;
                }
                else if(!strcmp(argv[1], "Environment")){
                        if(!strcmp(argv[2], "Suburban")) environment = SUBURBAN;
                        else if(!strcmp(argv[2], "Highway")) environment = HIGHWAY;
                        else if(!strcmp(argv[2], "Rural")) environment = RURAL;
                        else{
                                fprintf(stderr, "The specified environment does not
exist. \nOptions: suburban, highway, or rural.");
                                return TCL_ERROR;
                        }
                        return TCL_OK;
                }
                else if(!strcmp(argv[1], "MaxNodeID")){
                        max_node_id = atoi(argv[2]);
                        delete node_time_index;
                        node_time_index = 0;
```

```
                        node_time_index = new double*[max_node_id+1];
                        node_sched_time = new double*[max_node_id+1];

                        for(i = 0; i < max_node_id; ++i){
                                node_time_index[i] = new double[max_node_id+1];
                                node_sched_time[i] = new double[max_node_id+1];
                                for(j = 0; j < max_node_id; ++j){
                                        node_time_index[i][j] = -1;
                                        node_sched_time[i][j] = -1;
                                }
                        }
                        return TCL_OK;
                }
        }
        return Propagation::command(argc, argv);
}

int VanetProp::loadFile(const char *file_name)
{
        string buff;
        char *tkns;
        char cbuff[BUFF_SIZE];
        char arg1[BUFF_SIZE];
        char arg2[BUFF_SIZE];
        int found_data = 0;
        int i = 0, j = 0;

        //open the specified file
        ifstream fin(file_name);

        if(!fin.is_open()){
                printf("%s: File cannot be opened.", file_name);
                return TCL_ERROR;
        }

        while(!fin.eof())
        {
                getline(fin, buff);

                if( (buff[0] != '#') && !buff.empty())
                {
                        strcpy(cbuff, buff.c_str());
                        if(!found_data) {
                                sscanf(cbuff, "%s %s", arg1, arg2);

                                if(!strcmp(arg1,"fm")){
                                        fm0 = atof(arg2);
                                }
                                else if(!strcmp(arg1, "a_col")){
                                        a_col = atoi(arg2);
                                }
                                else if(!strcmp(arg1,"N")){
                                        N = atoi(arg2);
                                        envelop_data = new double*[N];
                                }
                                else if(!strcmp(arg1,"fs")){
                                        fs = atof(arg2);
                                }

                                else if(!strcmp(arg1, "DATA")){
                                        found_data = 1;
                                        printf("Data found.\n");
                                }
                                else{
                                        printf("Error in input file.\n");
                                        return TCL_ERROR;
                                }
                        }
```

```
                        else{
                                envelop_data[i] = new double[a_col];
                                tkns = strtok(cbuff, " \t\n\r");
                                j = 0;
                                while(tkns != NULL) {
                                        envelop_data[i][j++] = atof(tkns);
                                        tkns = strtok(NULL, " \t\n\r");

                                }
                                ++i;
                        }
                }
        }
        fin.close();
        return TCL_OK;
}

double VanetProp::dualSlope(PacketStamp *t, PacketStamp *r, double lambda, double L,
double &dist, double &scatter)
{
        double tx, ty, tz;     //transmitter coordinates
        double  rx, ry, rz;    //receiver coordinates
        double gamma1, gamma2, std_db1, std_db2, dc;

        switch(environment){
                case 0: gamma1 = gamma1_suburban;
                        gamma2 = gamma2_suburban;
                        std_db1 = std_db1_suburban;
                        std_db2 = std_db2_suburban;
                        dc = dc_suburban;
                        break;
                case 1: gamma1 = gamma1_highway;
                        gamma2 = gamma2_highway;
                        std_db1 = std_db1_highway;
                        std_db2 = std_db2_highway;
                        dc = dc_highway;
                        break;
                case 2: gamma1 = gamma1_rural;
                        gamma2 = gamma2_rural;
                        std_db1 = std_db1_rural;
                        std_db2 = std_db2_rural;
                        dc = dc_rural;
                        break;
                default: gamma1 = 0.0;
                         gamma2 = 0.0;
                         std_db1 = 0.0;
                         std_db2 = 0.0;
                         dc = 100.0;
                         break;
        }
        t->getNode()->getLoc(&tx, &ty, &tz);
        r->getNode()->getLoc(&rx, &ry, &rz);

        //add the relative distance of antenna to the node
        rx += r->getAntenna()->getX();
        ry += r->getAntenna()->getY();
        rz += r->getAntenna()->getZ();
        tx += t->getAntenna()->getX();
        ty += t->getAntenna()->getY();
        tz += t->getAntenna()->getZ();

        double dX = rx - tx;
        double dY = ry - ty;
        double dZ = rz - tz;
        dist = sqrt(dX * dX + dY * dY + dZ * dZ);    //distance between receiver and
transmitter

        // get antenna gain
```

```
        double Gt = t->getAntenna()->getTxGain(dX, dY, dZ, lambda);
        double Gr = r->getAntenna()->getRxGain(dX, dY, dZ, lambda);

        double P0 = Friis(t->getTxPr(), Gt, Gr, lambda, L, d0);     //use the
freepath equation to get power (in dB) at distance do;

        //calculate the critical distance
        //double dc = (4 * tz * rz)/lambda;
        double pl;
        scatter = 0;
        //get the longterm average power
        if(dist <= d0){
                pl = 10*log10(Friis(t->getTxPr(), Gt, Gr, lambda, L, dist));
                if(!initialized) scatter = ranVar->normal(0.0, std_db1);
        }
        else if(dist <= dc){
                pl = 10*log10(P0) - 10*gamma1*log10(dist/d0);
                if(!initialized) scatter = ranVar->normal(0.0, std_db1);
        }
        else {
                pl = 10*log10(P0) - 10*gamma1*log10(dc/d0) - 10*gamma2*log10(dist/dc);
                if(!initialized) scatter = ranVar->normal(0.0, std_db2);
        }
        return (pl+scatter);
}

double VanetProp::Pr(PacketStamp *t, PacketStamp *r, WirelessPhy *ifp)
{
        double L = ifp->getL();                         //system loss
        double lambda = ifp->getLambda();     //wavelength
        double dist = 0;
        double K;
        double m;
        double x, y;
        double alpha, beta;
        double v1, v2, a;
        double scatter = 0;
        double pl = pow(10,dualSlope(t, r, lambda, L, dist, scatter)/10);
        double fad_pwr = 1;
        int n1id = 0, n2id = 0, tmp;

        //find the small scale fading

        if(initialized)
        {
                n1id = t->getNode()->nodeid();
                n2id = r->getNode()->nodeid();

                //Find the maximum of the two and make n1 always higher
                if (n1id < n2id) {
                        tmp = n1id;
                        n1id = n2id;
                        n2id = tmp;
                }
                //set the starting point of the current channel
                if(node_time_index[n1id][n2id] == -1.0){
                        if (max_node_id) {

                                node_time_index[n1id][n2id] = int(floor(double((n1id-
1)*N)/double(max_node_id) +

        double((n2id)*N)/double(max_node_id*max_node_id)));
                                node_sched_time[n1id][n2id] = 0;
                        }
                        else{
                                node_time_index[n1id][n2id] = 0;
                                node_sched_time[n1id][n2id] = 0;
                        }
```

```
                }

                v1 = t->getNode()->speed();
                v2 = r->getNode()->speed();

                if( v1 == 0.0 && v2 == 0.0) a = 0;
                else a = (v2 > v1) ? (v1/v2) : (v2/v1);

                fm = (v2 > v1) ? (v2/lambda) : (v1/lambda);  //get the doppler spread

                node_time_index[n1id][n2id] += (Scheduler::instance().clock() -
                                        node_sched_time[n1id][n2id])*fs*fm/fm0;
                node_time_index[n1id][n2id] -=
double(N)*floor(node_time_index[n1id][n2id]/double(N));
                node_sched_time[n1id][n2id] = Scheduler::instance().clock();

                //the parameters to get the nakagami envelop
                m = -1.3*log10(dist/d0) + 3.7;

                int a_ind = a/0.10 + 0.5;

                //get quadrature and in phase components
                x = Interpolate(n1id, n2id, a_ind);
                y = Interpolate(n1id, n2id, 11+a_ind);
                x = x ? x : 0.000001;
                y = y ? y : 0.000001;

                //calculate the power envelop
                if(m > 1){
                        K = m - 1.0 + sqrt(m*m - m);
                        x += sqrt(2*K);
                        fad_pwr = (x*x + y*y)/(2*(K+1));
                }
                else{
                        if(m < 0.5) m = 0.5;
                        alpha = m + sqrt(m*(1-m));
                        beta = 2*m - alpha;
                        fad_pwr = (alpha*x*x + beta*y*y)/(alpha*alpha+beta*beta);
                }
        }

        trace(node_time_index[n1id][n2id], dist, pl, fad_pwr, scatter);

        return pl*fad_pwr;
}


double VanetProp::Interpolate(int n1id, int n2id, int a_ind)
{
        double interp;
        double X0, X1, X2, X3;
        int ind0, ind1, ind2, ind3;

        ind1 = int(floor(node_time_index[n1id][n2id]));
        ind0 = (ind1-1+N) % N;
        ind2 = (ind1+1) % N;
        ind3 = (ind1+2) % N;

        X1 = node_time_index[n1id][n2id] - ind1;
        X0 = X1+1.0;
        X2 = X1-1.0;
        X3 = X1-2.0;

        interp = envelop_data[ind0][a_ind]*X1*X2*X3/(-6.0) +
                envelop_data[ind1][a_ind]*X0*X2*X3*(0.5) +
                envelop_data[ind2][a_ind]*X0*X1*X3*(-0.5) +
                envelop_data[ind3][a_ind]*X0*X1*X2/6.0;
        return interp;
```

```
}

void VanetProp::trace(double index, double dist, double pl, double fad_pwr, double
scatter)
{
        if(pwr_tracer) {
                sprintf(pwr_tracer->buffer(), "%f \t %f \t %f \t  %f \t %f \t %f",
index, Scheduler::instance().clock(), dist, 10*log10(pl), 10*log10(fad_pwr),
scatter);
                pwr_tracer->dump();
        }
}

double VanetProp::getDist(double Pr, double Pt, double Gt, double Gr, double hr,
double ht, double L, double lambda)
{
        return sqrt(sqrt(Pt * Gt * Gr * (hr * hr * ht * ht) / Pr));
}
```

Appendix B

Generating the Dataset

The MatLab function `NakDataset` generates the dataset to be used in the Network Simulator.

```
function NakDataset(fd, fs, N, filename)
   t = (0:(N-1))./fs;
   f = 0:fs/(N-1):fs;
   a = 0:0.1:1;

   fid = fopen(filename, 'wt');
   fprintf(fid, 'fm \t %.8f \nN \t %i \nfs \t %.8f \na_col \t %i \n', fd, N, fs,
   length(a));
   fprintf(fid, 'DATA\n');

   g = (1/sqrt(2))*randn(2,N/2-1) + sqrt(-1)*(1/sqrt(2))*randn(2,N/2-1);
   gs = conj(g);
   G = [zeros(2,1),fliplr(gs),zeros(2,1),g];

   x = zeros(N, length(a));
   y = zeros(N, length(a));
   str = '';

   for i=1:length(a)
       %calculate the autocorrelation function for current a
       c = besselj(0, 2*pi*fd*t).*besselj(0, 2*pi*fd*a(i)*t);
       %get the power spectrum
       S = real(fft(c));
       S( (f > (1+a(i))*fd) & (f < (f(end) - (1+a(i))*fd))) = 0;
       Sroot = sqrt(S);
       I = G(1,:).*Sroot;
       Q = G(2,:).*Sroot;
       %get the compoenents in the time domain
       x(:,i) = real(ifft(I'));
       y(:,i) = real(ifft(Q'));
       x(:,i) = x(:,i)./sqrt(var(x(:,i)));
       y(:,i) = y(:,i)./sqrt(var(y(:,i)));
       str = strcat(str, '%f\t%f\t');
   end

   str = strcat(str, '\n');
   fprintf(fid, str, [x y]);
   fclose(fid);
```

Appendix C

An NS-2 Simulation Scenario

```
#set the required parameters for the VANET wireless                          #
###############################################################################
Antenna/OmniAntenna set X_ 0  ;                                              #
Antenna/OmniAntenna set Y_ 0  ;                                              #
Antenna/OmniAntenna set Z_ 1.51       ;                                      #
Antenna/OmniAntenna set Gt_ 1.0       ;                                      #
Antenna/OmniAntenna set Gr_ 1.0       ;                                      #
Phy/WirelessPhy set CPThresh_ 10.0    ;                                      #
Phy/WirelessPhy set CSThresh_ 1.559e-11      ;                               #
Phy/WirelessPhy set RXThresh_ 3.61705e-09    ;                               #
Phy/WirelessPhy set bandwidth_ 2e6           ;                               #
Phy/WirelessPhy set Pt_ 0.2818                       ;                       #
Phy/WirelessPhy set freq_ 5.90e+9            ;                               #
Phy/WirelessPhy set L_ 1.0                   ;                               #
                                                                             #
                                                                             #
###############################################################################
#wireless system of two moving nodes
#using the VANET propagation model

#set array val() with wireless network parameters
set val(netif)         Phy/WirelessPhy               ;
set val(mac)           Mac/802_11            ;
set val(ifq)           Queue/DropTail/PriQueue       ;
set val(ll)            LL                    ;
set val(ant)           Antenna/OmniAntenna   ;
set val(ifqlen)               50                    ;
set val(chan)          Channel/WirelessChannel       ;
set val(prop)          Propagation/VanetProp  ;
set val(rp)            DSDV                  ;
set val(nn)            2                     ;
set val(max_x)         1100                  ;
set val(max_y)         1100                  ;
set val(MaxNodeID)     [expr {$val(nn)-1}]   ;
###############################################################################

#initialize global variables
set ns_ [new Simulator]
#set the tracing object
set tracefd [open simple.tr w]
$ns_ trace-all $tracefd
#set the topology
set val(topo) [new Topography]
$val(topo) load_flatgrid $val(max_x) $val(max_y)
#create GOD
create-god     $val(nn)

#configure the wireless nodes
$ns_ node-config       -adhocRouting  $val(rp)       \
                       -llType        $val(ll)       \
                       -macType       $val(mac)      \
                       -ifqType       $val(ifq)      \
                       -ifqLen        $val(ifqlen)   \
                       -antType       $val(ant)      \
                       -propType      $val(prop)     \
                       -phyType       $val(netif)    \
                       -channelType   $val(chan)     \
                       -topoInstance  $val(topo)     \
                       -agentTrace    ON             \
                       -routerTrace   OFF            \
                       -macTrace      OFF            \
```

```
                    -movementTrace oFF

#create the actual nodes
for {set i 0} {$i < $val(nn)} {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0
}

#provide initial coordinates of the nodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 6.0
$node_(1) set Y_ 2.0
$node_(1) set Z_ 0.0

#initialize the signal power tracing
set prop_inst [$ns_ set propInstance_]
set pwrfd [open vanetpwr.tr w]
set pwrTrace [new BaseTrace]
$pwrTrace attach $pwrfd
$pwrTrace set src_ 0
$prop_inst Tracer $pwrTrace

#configure the VANET channel propagation module
$prop_inst LoadPropFile  env.txt;
$prop_inst Environment Rural
$prop_inst MaxNodeID $val(MaxNodeID)

#produce simple node movement: node_(1) moves away from node_(0), with speed changing
$ns_ at 10.0 "$node_(1) setdest 1000.0 2.0 20"


##################################
#setup traffic flow between nodes#
##################################

#TCP connection between node_(0) and node_(1)
set udp [new Agent/UDP]
$udp set class_ 2
set sink [new Agent/Null]
$ns_ attach-agent $node_(0) $udp
$ns_ attach-agent $node_(1) $sink
$ns_ connect $udp $sink
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp


$ns_ at 10.0 "$cbr0 start"

#tell nodes when simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
        $ns_ at 150.0 "$node_($i) reset";
}

$ns_ at 150.0 "stop"
$ns_ at 150.01 "puts \"NS EXITING...\"; $ns_ halt"

proc stop {} {
        global ns_ tracefd pwrfd
        $ns_ flush-trace
        close $tracefd
        close $pwrfd
}


#start simulation
$ns_ run
```

References

[1] W. Chen and S. Cai, "Ad hoc peer-to-peer network architecture for vehicle safety communications," IEEE Communication Magazine, pp. 100-107, April 2005.

[2] "Standard specification for telecommunications and information exchange between roadside and vehicle systems - 5GHz band dedicated short range communications (DSRC) medium access control (MAC) and physical layer (PHY) specifications, ASTM E2213-03," Sep. 2003.

[3] "The California PATH project," in *http://www.path.berkeley.edu*.

[4] "The Fleetnet project," in *http://www.et2.tu-harburg.de/fleetnet/*.

[5] "Network on Wheels project," *http://www.network-on-wheels.de*.

[6] Jeremy J. Blum, Azim Eskandarian, and Lance J. Hoffman, **"**Challenges of intervehicle ad hoc networks," *IEEE Trans. on Intelligent Transportation Systems*, vol. 5, pp. 347-351, 2004.

[7] Yin Jijun et al., "Performance evaluation of safety applications over DSRC vehicular ad hoc networks," *VANET'04: Proceedings of the 1st ACM international workshop on vehicular ad hoc networks*, pp. 1-9, 2004.

[8] "The Network Simulator," in *http://nsnam.isi.edu*.

[9] Ratish J. Punnoose, Pavel V. Nikitin, and Daniel D. Stancil, **"**Efficient simulation of ricean fading within a packet simulator," *IEEE Vehicular Technology Conference*, vol. 2, 2000.

[10] Lin Cheng, PHYSICAL LAYER MODELING AND ANALYSIS FOR VEHICLE-TO-VEHICLE NETWORKS, Ph.D. thesis, Carnegie Mellon University, 2007.

[11] Andrea Goldsmith**,** *Wireless Communications.* New York : Cambridge University Press, 2005.

[12] T. S. Rappaport**,** *Wireless communications: principles and practice.* Upper Sadle River : Prentice Hall, 2001.

[13] J. D. Parsons, *The Mobile Radio Propagation Channel.* New York : Wiley, 1994.

[14] T. Okumura, E. Ohmori, and K. Fukuda, **"**Field strength and its variability in VHF and UHF land mobile service," *Rev. Elec. Commun. Lab.*, 1968.

[15] M. Hata, "Epirical formula for propagation loss in land mobile radio services," *IEEE Trans. Veh. Tech.*, pp. 317-25, August 1980.

[16] European Cooperative in the Field of Science and Technical Research EURO-COST 231, "Urban Transmission loss models for mobile radio in the 900 and 1800 MHz bands," The Hague, 1991.

[17] L. Cheng, B. Henty, D. Stancil, F. Bai, and P. Mudalige, **"**Mobile vehicle-to-vehicle narrow-band channel measurement and characterization of the 5.9 GHz dedicated short range communcation (DSRC) frequency band," *IEEE Journal on Selected Areas in Communications* , vol. 25, pp. 1501-1516, Oct. 2007.

[18] Lin Cheng, et al., "Highway and rural propagation channel modeling for vehicle-to-vehicle communications at 5.9 GHz," *Antennas and Propagation Society International Symposium*, 2008.

[19] M. J. Gans, "A power-spectral theory of propagation in the mobile radio environment," *IEEE Trans. Veh. Technol.,* vol. VT-21, pp. 27-38, Feb. 1972.

[20] A. S. Akki and F. Haber, "A statistical model of the mobile-to-mobile land communication channel," *IEEE Trans. on Vehicle Technology*, 1986.

[21] S. O. Rice, "Mathematical analysis of random noise," *Bell System Tech. J.*, pp. 46-156, Jan. 1945.

[22] M. Nakagami, **"**The m-distribution, a general formula of intensity distribution of rapid fading," in *Statistical methods in radio wave propagation*, W. G. Hoffman, Ed, Oxford, UK: Pergamon, 1960.

[23] T. Aulin, "Characteristics of a digital mobile radio channel," *IEEE Trans. Veh. Technol*, vol. VT-30, pp. 45-53, May 1981.

[24] U. Charash, "Reception through Nakagami fading multipath channels with random delays," *IEEE Trans. Commun*, vol. 27, April 1979.

[25] H. Suzuki, "A statistical model for urban multipath," *IEEE Trans. Commun.,* vol. 25, pp. 673-680, July 1977.

[26] Guillermo Acosta, Kathleen Tokuda, and Mary Ann Ingram, "Measured joint doppler-delay power profiles for vehicle-to-vehicle communications at 2.4 GHz," *GLOBECOM '04. IEEE*, vol. 6, pp. 3813 – 3817, 2004.

[27] Guillermo Acosta and Mary Ann Ingram, "Doubly Selective Vehicle-to-Vehicle Channel Measurements and Modeling at 5.9 GHz," *Wireless Personal Multimedia Communications Conference*, Sept. 2006.

[28] D. W. Matolak, I. Sen, and W. Xiong, "Channel Modeling for V2V Communications," *Third Annual Conf. on Mobile and Ubiquitous Systems*, July 2006.

[29] I. Tan, Wanbin Tang, K. Laberteaux, A. Bahai, "Communications, Measurement and Analysis of Wireless Channel Impairments in DSRC Vehicular," *IEEE International Conf. on Communications*, May 2008.

[30] W. Mohr, "Modeling of Wideband Mobile Radio Channels Based on Propagation Measurements," *PIMRC '95, IEEE*, vol. 2, pp. 397 – 401, Sept. 1995.

[31] C. Cheng and Norman C. Beaulieu, "Efficient nakagami-m fading channel simulation," *IEEE Trans. on. Veh. Tech.*, vol. 54, pp. 413 – 424, March 2005.

[32] Scot D. Gordon and James A. Ritcey, "Generating correlated nakagami fading channels," *IEEE Conf. Record of the Thirtieth Asilomar Conf. on Signals, Systems and Computers,* vol. 1, pp. 684 – 688, Nov. 1996.

[33] Q. T. Zhang, "A decomposition technique for efficient generation of correlated nakagami fading channels," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 2385 – 2392, Nov. 2000.

[34] I. A. Stegun and M. Abramowitz, *Handbook of Mathematical Functions.* New York : Dover, 1972.

[35] Chengshan Xiao, Zheng, Yahong Rosa and N. C. Bealieu, "Novel sum-of-sinusoids simulation models for rayleigh and rician fading channels," *IEEE Trans. on Wireless Communications*, vol. 5, pp. 3667 – 3679, Dec. 2006.

[36] Christos Komninakis, "Fast and accurate rayleigh fading simulator," *GLOBECOM '03, IEEE,* vol. 6, pp. 3306 – 3310, Dec. 2003.

[37] Kareem E. Baddour and Norman C. Bealieu, "Autoregressive Modeling for Fading Channel Simulation," *GLOBECOM '01, IEEE,* vol. 4, pp. 1650 – 1662, July 2005.

[38] David J. Young and N. C. Bealieu, "The generation of correlated rayleigh random variates by inverse discrete fourier transform," *IEEE Trans. on Communications,* vol. 48, pp. 1114 – 1127, July 2000.