DELFT UNIVERSITY OF TECHNOLOGY
Department of Electrical Engineering
Telecommunications and Traffic-Control Systems Group

Unslotted Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications

H.L.A. Le

Type: Thesis Report
Size: xiii + 147 + 33 pages
Date: September 5th, 1995

Graduation Professor:	Prof. Dr. R. Prasad
Mentors:	Prof. Dr. R. Prasad
	Ir. J.A.M. Nijhof
	Ir. H.R.R. van Roosmalen
Assignment number:	A-664
Period:	November 1994 - October 1995

Abstract

A hybrid Code Division Multiple Access / Inhibit Sense Multiple Access (CDMA/ISMA) protocol has been proposed as an effective multiple access scheme for Indoor Wireless Computer Communications. This new protocol combines the advantages of both CDMA and ISMA into one protocol. On the one hand the ISMA protocol introduces a limitation to the number of simultaneous accesses to the transmission channel. On the other hand the CDMA protocol introduces an improvement to the packet survival chance.

It is shown that the performance of the hybrid protocol is indeed better than CDMA only. In addition, code sharing can be applied to reduce hardware cost. The slotted hybrid CDMA/ISMA protocol using the *p-persistent* ISMA scheme has already been analysed. In this report, we investigate the hybrid protocol using the *unslotted non-persistent* ISMA scheme. The analysis is done for a star-connected multiple access wireless computer network. The performance comparison between the slotted and unslotted hybrid CDMA/ISMA protocol is evaluated in terms of throughput and delay using computer simulation and mathematical analysis.

Indexing Terms: Code Division Multiple Access (CDMA), Inhibit Sense Multiple Access (ISMA), Spread Spectrum, Data Communications, Rayleigh Fading.

SUMMARY

Compared to the fixed wired systems, Indoor Wireless Communication offers advantages such as time and cost saving, reduction or elimination of wiring and rewiring and increased flexibility and efficiency. However, the scarce electromagnetic spectrum and the hostile transmission medium oblige us to choose an appropriate multiple access protocol for the indoor wireless communication.

Herefore, a hybrid Code Division Multiple Access (CDMA) / Inhibit Sense Multiple Access (ISMA) protocol has been proposed. This protocol combines the advantages of both CDMA and ISMA into one protocol. The ISMA protocol limits the number of simultaneous transmissions of data packets while CDMA improves the survival chances of the transmitted packets. Furthermore, code sharing can be applied with a reasonable penalty in performance degradation.

We distinguish two types of hybrid protocols: slotted and unslotted. The slotted p-persistent hybrid CDMA/ISMA protocol has already been investigated and reported on. This work concentrates on the unslotted non-persistent hybrid CDMA/ISMA protocol.

The performance is evaluated in terms of throughput and delay in relationship to the offered traffic in a star-connected wireless computer network. The throughput is a measure for the efficiency of the protocol and the delay gives the average delay the data packets suffer before they are correctly received. We adopt two approaches to derive the performance of the hybrid protocol. First, we set up a computer simulation program and next, a mathematical model is derived. Then the results are compared.

As can be expected, comparison between the slotted and unslotted protocol shows that for low offered traffic the delay for the unslotted protocol is better than the slotted one. For high traffic it is the other way round. The hybrid protocol performs very well in case the propagation delay is not a concern. If the propagation delay is too high, the performance drops quickly. The results also show that code sharing is a good option for the hybrid protocol.

SAMENVATTING

Vergeleken met de kabelsystemen biedt binnenshuis draadloze communicatie voordelen zoals tijd- en kostenbesparing, reductie of eliminatie van bekabeling en herbekabeling en toename van flexibiliteit en efficiëntie. Echter, de schaarste van de elektromagnetische spectrum en de slechte transmissiemedium verplicht ons om een geschikte "multiple access" protocol voor de binnenhuis draadloze communicatie te vinden.

Hiervoor is een hybride Code Division Multiple Access (CDMA)/ Inhibit Sense Multiple Access (ISMA) protocol voorgesteld. Dit protocol combineert de voordelen van zowel CDMA als ISMA in één protocol. Het ISMA protocol beperkt het aantal gelijktijdige transmissies van datapakketten terwijl CDMA de overlevingskansen van de gezonden pakketten verbetert. Bovendien kan hergebruik van codes toegepast worden met een redelijke degradatie in de prestatie.

Wij onderscheiden twee types hybride protocollen: 'slotted' en 'unslotted'. De slotted p-persistent hybride CDMA/ISMA protocol was al onderzocht en gerapporteerd. Dit werk concentreert zich op de non-persistent hybride CDMA/ISMA protocol.

De prestatie is geëvalueerd in termen van doorstroming en vertraging in relatie tot het aangeboden verkeer in een draadloze sterverbindingsnetwerk. De doorstroming is een maat voor de efficiëntie van de protocol en de vertraging geeft de gemiddelde vertraging van de datapakketten voordat zij correct worden ontvangen. We gebruiken twee benaderingswijzen om de prestatie van de hybride protocol te achterhalen. We zetten eerst een computer simulatie programma op en daarna wordt een mathematische model afgeleid. Daarna worden de resultaten met elkaar vergeleken.

Zoals verwacht laat de vergelijking tussen de slotted en de unslotted protocol zien dat voor laag aangeboden verkeer, de vertraging voor de unslotted protocol beter is dan die van de slotted protocol. Voor hoog aangeboden verkeer is het net andersom. De hybride protocol presteert goed in het geval wanneer de propagatievertraging geen rol speelt. Als de propagatievertraging te hoog is, neemt de prestatie snel af. De resultaten laten ook zien dat hergebruik van codes een goede optie is voor hybride protocol.

PREFACE

This report is the results of my thesis work performed at the Department of Telecommunications and Traffic Control Systems, Faculty of Electrical Engineering, Delft University of Technology.

I would like to thank all my colleage students at the department for the friendly and nice atmosphere and the many joyable discussions during the breaks. Of course, this thesis work would not have been possible without the valuable advices, support and fruitful cooperation I have received from my mentors. This is why I specially like to thank R. Prasad, J.A.M. Nijhof and H.R.R. van Roosmalen.

I'd like to dedicate this report to my parents and Hong Hanh for their encouragements, motivation, support and guidance.

Huy Linh Anh Le

Delft, September 5th 1995

LIST OF SYMBOLS AND ABBREVIATIONS

a_k	Code waveform
A	Carrier signal amplitude
b_k	The *k*th user's information signal
BW_{RF}	RF bandwidth
c	Speed of light
C	Capacity in bits per second
d	Inhibit delay fraction
E_b	Energy per bit
f	Signal frequency
g_T	Transmitter gain
g_R	Receiver gain
G	Offered traffic
G_p	Processing gain
K	Number of users
K_a	Number of antennas
l	Distance between the transmitter and the receiver
L	Number of resolvable paths
L_p	Packet length
M	Order of diversity
$n(t)$	White Gaussian noise
N	Noise power
M_{max}	Maximum order of diversity
P_{tr}	Probability that a data packet is transmitted in case the channel is sensed free (only for slotted ISMA protocol)
P_R	Received power
P_T	Transmitted power
R_{data}	Data rate

$r(t)$	Received signal
$s_k(t)$	Transmitted signal
S	Signal power
t_c	Cycle time
T_b	Bit duration
T_c	Chip duration
T_{IA}	inter-arrival time
T_m	The rms delay spread
T_{pd}	Packet duration
W	Bandwidth in hertz
x	The number of arrivals during the inhibit delay fraction
α	Random delay parameter
β	Path gain
γ	Path phase
λ	Arrival rate
τ	Path delay
μ	Attenuation parameter
ω_c	Carrier frequency
AWGN	Additive White Gaussian Noise
CDF	Cumulative Distribution Function
CDMA	Code Division Multiple Access
CSMA	Carrier Sense Multiple Access
DPSK	Differential Phase Shift Keying
DS	Direct Sequence
FDMA	Frequency Division Multiple Access
FH	Frequency Hopping
ISMA	Inhibit Sense Multiple Access
IWC	Indoor Wireless Communications
LOS	Line Of Sight
PDF	Probability Density Function

PN	Pseudo Noise
PRMA	Packet Reservation Multiple Access
PSD	Program Structure Diagram
RF	Radio Frequency
SDL	Specification and Description Language
SIR	Signal to Interference Ratio
TDMA	Time Division Multiple Access
TH	Time Hopping
TP	Transmission Period
IP	Idle Period

CONTENTS

1. INTRODUCTION

Indoor Wireless Communication (IWC) offers companies a flexibility not available in the fixed wired systems. For example it is not necessary to bring the network down for several minutes to add, remove or move the terminals to other places.

Suppose the company employs a group of representatives, who are most of the time in 'the field' for negotiation with customers and armed with powerful portable computers. When the representatives are back in the office, they can join in the network any time they like it to. In case of the fixed wired systems, it is necessary to perform a routine exercise like plugging and unplugging which can cause network failures because of bad connections. If the computers can communicate with each other wirelessly, they can participate in the network without problems concerning network connections. Time-consuming activities like rearranging the office to account for network connections is history. We can conclude that IWC offer increased flexibility and efficiency, reduction of wiring and down time of services and time and cost saving. Before we can make use of these advantages, we have to solve the problems of the scarce electromagnetic spectrum and the hostile transmission medium first.

Because the electromagnetic spectrum is scarce, it must be used as efficiently as possible. Therefore, it is necessary to solve the problem of the scarce bandwidth. Universities, research laboratories and industries are putting forward enormous effort to investigate new system concepts that can improve the spectrum efficiency. A very important subject with respect to the spectrum efficiency is the choice of a multiple access technique.

Indoor wireless office communication is our main research field. The office system we focus on consists of a building in which users work together in groups. The participants generate terminal traffic. Terminals communicate with each other by radio transmission using a random access protocol. Terminals might not detect each other's transmission in radio communications. It can easily happen that two users are hidden from each other by some obstacle, in which case they are not aware of each other's transmission and collisions can occur. This is called the *hidden terminal problem*. This results in severe performance degradation. The introduction of a central base station can alleviate this problem by instructing it to send a busy tone to all participating terminals to forbid new transmissions when a transmission is going on. Still, a situation can occur in which two or more terminals simultaneously start their transmission, resulting in a collision. However, a great reduction in the number of simultaneous transmissions is achieved by the introduction of a central base station. This concept is called ISMA [1]-[4].

If we could somehow increase the survival chances of colliding packets, we could improve protocol performance. The near-far effect is one way to achieve this. A packet may 'capture' the receiver if its received power is much stronger than its competitors. This can happen when terminals use the same transmission power, but are located at different distances from the receiver. Because the 'near terminals' have better performance compared to the 'far terminals' (due to the better survival chance of the packets), the near-far effect introduces an unfair element among the terminals. Perfect power control, in which the transmitted power of the terminals are adjusted such that their received powers are all equal, can eliminate the near-far effect described above. To increase the survival chances of colliding packets, we use the CDMA concept. Herefore, each terminal multiplies a specific code sequence to the data sequence before transmission. The receiver uses the same code sequence to gain back the data sequence. The receiver can lock into the first incoming packet and receive this packet correctly even when collisions occur. The performance of CDMA has been reported in a number of publications, e.g. [5]-[10].

The hybrid CDMA/ISMA protocol combines the advantages of both CDMA and ISMA into one protocol. The advantages of CDMA and ISMA are the improvement of the survival chance of data packets and the limitation of contention in the channel. Code sharing can also be applied. This is an important aspect because the number of distinct useful transmission codes is limited, especially for short code length.

In [11] and [12], the hybrid protocol combines Direct Sequence CDMA with *slotted* p-persistent ISMA. In this report, we investigate the performance of the hybrid CDMA/ISMA protocol using the *unslotted* non-persistent ISMA scheme. It is expected that for low traffic, the delay of the unslotted protocol improves because when a terminal has a data packet to send, it does not have to wait until the start of the next time slot. In addition, even when data packets collide, there is a probability that the data is received correctly due to the use of CDMA. This is the strength of the hybrid CDMA/ISMA protocol. By using the *unslotted non-persistent* ISMA scheme, it is expected that we take more advantage of this strength.

Fading is a result of the propagation of the transmitted signal through several paths. In this report, we model the channel as a Rayleigh fading channel. This assumption is valid if the received signal is composed of faded paths with more or less equal power. This means that a line of sight path is absent. For signal modulation, we employ Differential Phase Shift Keying (DPSK).

The performance of the hybrid CDMA/ISMA protocol is measured in terms of throughput and delay in relationship with the offered traffic. To this end, we follow two ways to derive the protocol's performance. First, a computer simulation program is set up. Second, we develop closed form formulas by mathematical analysis. To facilitate the calculations, we adopt a symmetrical star-connected network for our analysis.

This report is organised as follows. Chapter 2 gives the system description. The derivation of the DPSK bit error rate in Rayleigh fading channel is then given in Chapter 3. Chapter 4 describes the performance analysis of the hybrid protocol by simulation. The mathematical analysis then follows in Chapter 5. Subsequently, in Chapter 6, the results will be shown. Finally, conclusions and recommendations are drawn in Chapter 7.

2. SYSTEM DESCRIPTION

This chapter introduces the system description as will be used in the analysis. Special attention will be given to CDMA and ISMA, the basic schemes for the hybrid CDMA/ISMA protocol. Also important assumptions with respect to the hybrid CDMA/ISMA protocol will be discussed. This chapter starts with a general discussion of the different types of multiple access protocols.

2.1 Multiple Access Protocols

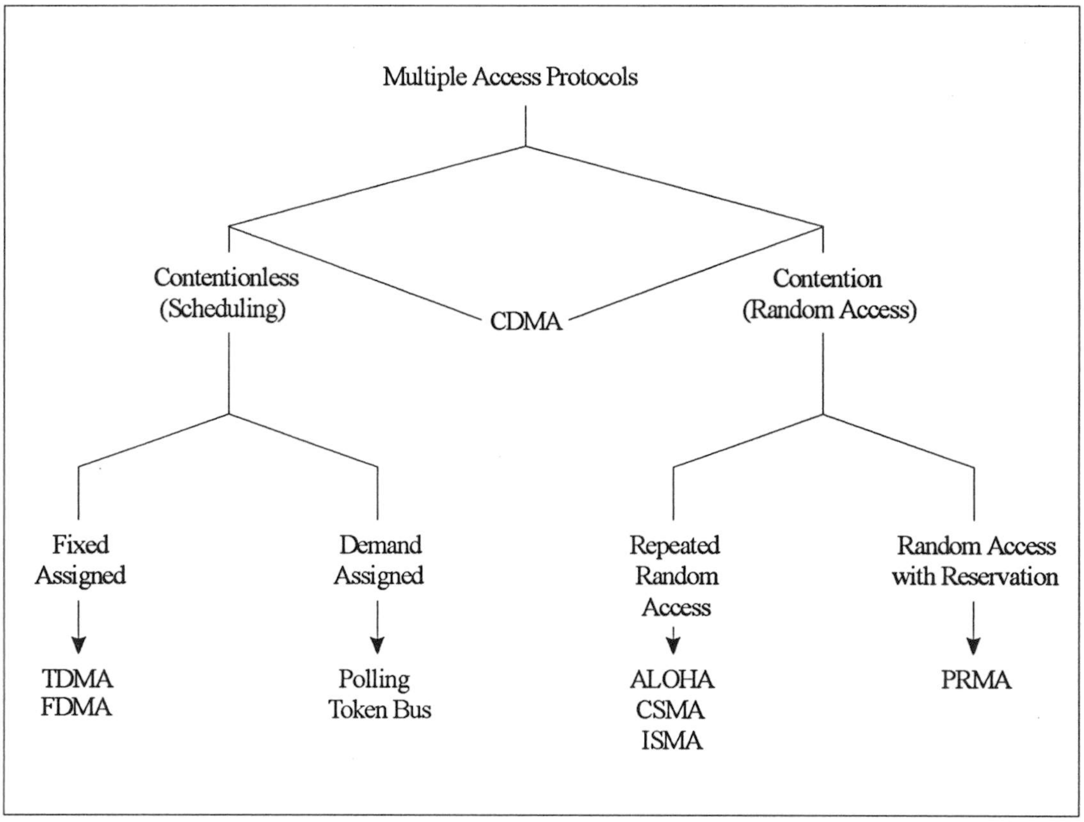

Fig. 2.1: Classification of the multiple access protocols

Fig. 2.1 depicts the classification of the multiple access protocols. Multiple access protocols can be divided in two classes, *contentionless* (or scheduling) and *contention* (or random access) [13]. The contentionless class of protocols do not allow contention in the channel. The users are scheduled to transmit in an orderly manner. On the other hand, the contention class does allow contention in the channel. For this class of protocol, collisions can occur because users may access the channel at the same time. CDMA is a protocol which belongs to both classes. This protocol will be described in section 2.3.

2.1.1 Contentionless Multiple Access Scheme

Among the class of contentionless we reckon protocols which avoid the simultaneous access of the channel by two or more users. There are two ways to do this. The first *fixed assigned* method assigns to each user a fix amount of the channel's capacity, independent of his activity. If a user is not active, then his capacity is wasted because it can not be used by others. Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA) belong to this method. *Demand assigned* is the second method in the class of contentionless multiple access protocols. This method only assigns channel's capacity to a user if he has something to transmit. We consider polling and token bus to be demand assigned methods.

2.1.2 Contention Multiple Access Scheme

In contrast to the contentionless multiple access scheme where transmissions are organised orderly, transmissions of the class of contention multiple access protocols are not scheduled in an orderly manner. A user does not know about the intention of other users. At most, an ongoing transmission can be detected. This class of protocols can be further subdivided into two groups, *repeated random access* and *random access with reservation*. The repeated random access method uses retransmissions to resolve conflicts. We consider ALOHA, Carrier Sense Multiple Access (CSMA) and Inhibit Sense Multiple Access (ISMA) to belong to this class of protocols. With the random access with reservation scheme, only the first transmission suffers from contention. Once a user has access to the channel, capacity is reserved to this user and he is the only one who may use this. If this user is idle for some time, then the capacity is reclaimed. Packet Reservation Multiple Access (PRMA) belongs to this type of protocol.

2.2 Fixed Assigned Contentionless Multiple Access Protocols

In this section, the fixed assigned contentionless multiple access protocols (TDMA and FDMA) will be discussed in more detail.

2.2.1 Frequency Division Multiple Access

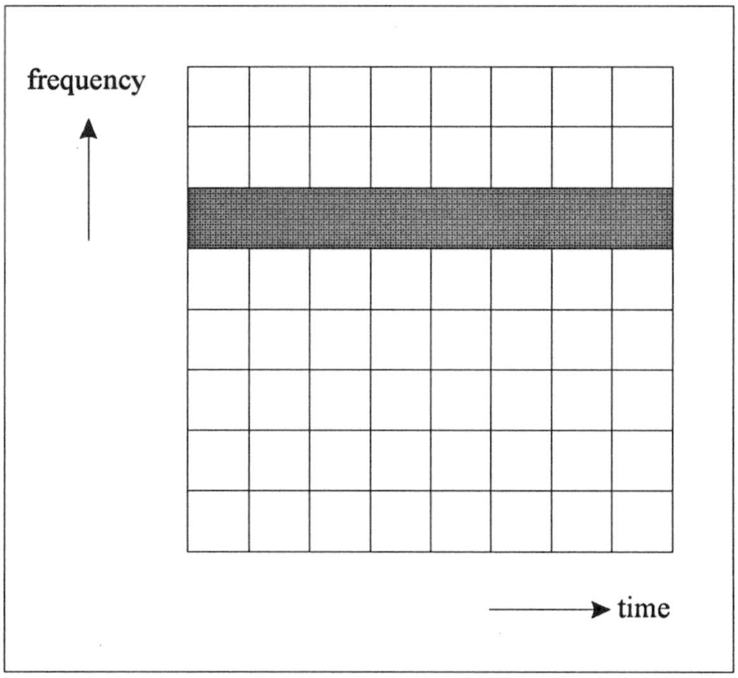

Fig. 2.2: Principle of FDMA

The classical method of providing multiple access capability is FDMA. In FDMA, each user is assigned a particular frequency band. An assigned frequency band can only be used by a particular user all the time and everybody else is excluded from this band (Fig. 2.2). When all frequency bands are occupied, the system has reached its capacity and no further users may be added.

2.2.2 Time Division Multiple Access

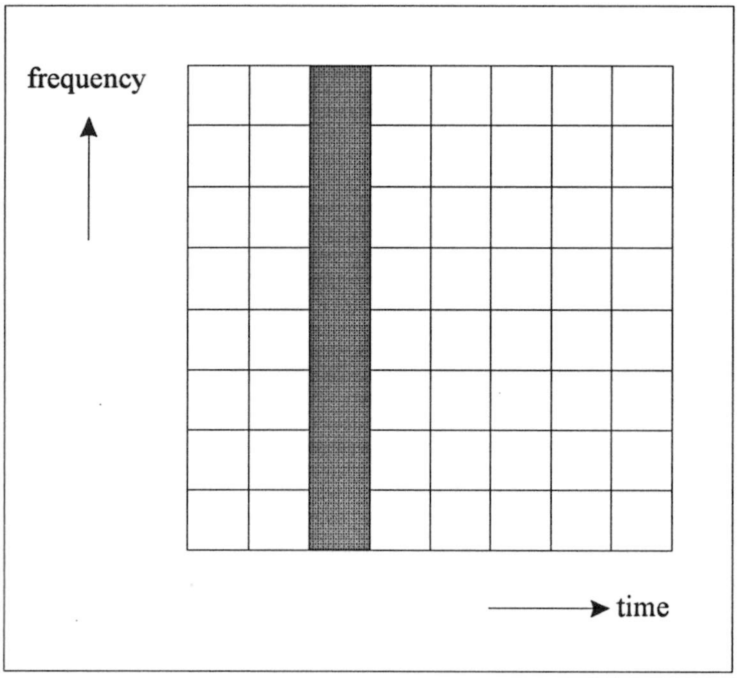

Fig. 2.3: Principle of TDMA

A more recent technique for providing multiple access is TDMA. In TDMA, the time is divided into equal frames and the frames on their turn are divided into equal time slots. Each user is assigned a particular time slot within a time frame, and during this time slot, transmits a portion of a message by any standard digital technique. Each user is the sole 'owner' of his particular assigned time slot (Fig. 2.3). Time-division multiple access is an important application for satellite communications as well as ground-based digital communication systems. Again, however, when all time slots are occupied, the system is operating at capacity and no more users can be added.

2.3 Code Division Multiple Access

Code division multiple access (CDMA) is always accomplished by means of spread spectrum. In this system each user is assigned a particular code. Each user employs his own code to spread the signal into a wide band signal. Fig. 2.4 shows the transformation of the original data signal into a spread spectrum signal. The horizontal axis represents the frequency while the vertical axis depicts the power spectral density. At the receiver side, the same code is used to transform the wide band signal back to the original signal. The cross correlation between users can be decreased by a proper choice of codes. If we manage to realise a low cross correlation, then we can bring down the interference between users [14]. In contrast to the previously mentioned methods of multiple access, code-division multiple access does not have any sharply defined system capacity. As the number of users increases, the signal-to-interference ratio (SIR) becomes smaller and there is a gradual degradation in performance until the SIR falls below a certain threshold. Thus the system can tolerate significant amounts of overload if the users are willing to tolerate poorer performance.

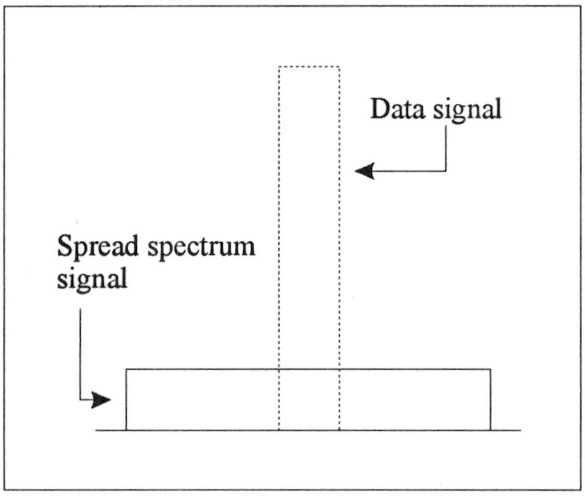

Fig. 2.4: Spreading of the data signal

An additional advantage of CDMA is that the messages intended for one user are not readily decodable by other users, because they may not know the proper codes or have the equipment for generating the appropriate reference signals. Thus there is a privacy feature that is not available in other multiple-access techniques.

The Shannon formula, which express the channel capacity, is the basis of spread spectrum technology:

$$C = W \cdot log_2\left(1+\frac{S}{N}\right) \tag{2.1}$$

where C = Capacity in bits per second,

W = Bandwidth in hertz,

S = Signal power,

N = Noise power.

Equation (2.1) shows the trade-off between the transmitted power and the signal bandwidth. Spread spectrum is based on the increase of the signal bandwidth, thereby allowing the signal to noise ratio to decrease to achieve a certain channel capacity.

2.3.1 Processing Gain

The most commonly used quantity in describing or specifying spread spectrum systems is that of processing gain (G_p):

$$G_p = \frac{BW_{RF}}{R_{data}} \tag{2.2}$$

Where the RF bandwidth (BW_{RF}) is the bandwidth of the transmitted spread spectrum signal and the information rate (R_{data}) is the data rate in the information baseband channel. So, the ratio between transmitted and original bandwidth is defined as the processing gain.

2.3.2 Properties of Spread Spectrum

An application of spread spectrum that is of particular interest in mobile communications is the ability of a wide-band signal to resist the effects of multipath fading. A property of multipath fading is that frequencies separated by only a few hundred kilohertz may fade essentially independently. Thus at any given time when the signal has a large bandwidth, only a small portion of the bandwidth will be in a fade. The average received signal power thus can be made more nearly constant than it would be for a narrow-band signal. This resistance to fading is an important consideration in the potential application of spread spectrum to mobile communication situations.

Another important element employed in the design of spread spectrum signals is pseudo-randomness. It makes the signals appear similar to random noise and difficult to demodulate by receivers other than the intended ones. A message may be hidden in the background noise by spreading its bandwidth with coding and transmitting the resultant signal at a low average power. Because of its low power level, the transmitted signal is said to be 'covert'. It has a low probability of being intercepted (detected) by a casual listener.

Message privacy may be obtained by superimposing a pseudo-random pattern on a transmitted message. The message can be demodulated by the intended receivers that know the pseudo-random pattern or key used at the transmitter but not by any other receivers that do not have knowledge of the key. Resuming the advantages spread spectrum offers we come to the following:

- Interference rejection;
- Anti-multipath fading (frequency diversity);
- Low probability of intercept;
- Secrecy;
- Multiple access capability.

2.3.3 CDMA Techniques

We can distinguish between three main types of spread spectrum techniques: Direct Sequence (DS), Frequency Hopping (FH) and Time Hopping (TH).

For the direct sequence CDMA method, a digital information stream is modulated by a binary code sequence (Fig. 2.5). The recovering of the signal at the receiver is done by correlating the received signal with the original code sequence used by the transmitter. In a frequency hopped spread spectrum communications system, the available channel bandwidth is subdivided into a large number of adjacent frequency bands. In any signalling interval, the transmitted signal occupies one or more of the available frequency bands. The selection of the frequency band(s) in each signalling interval is made pseudo-randomly according to the output from a pseudo noise (PN) generator. For the time hopping method, the time axis is divided into frames, which in turn are subdivided into time slots. The user transmits the data in a different time slot every frame, using all the available bandwidth.

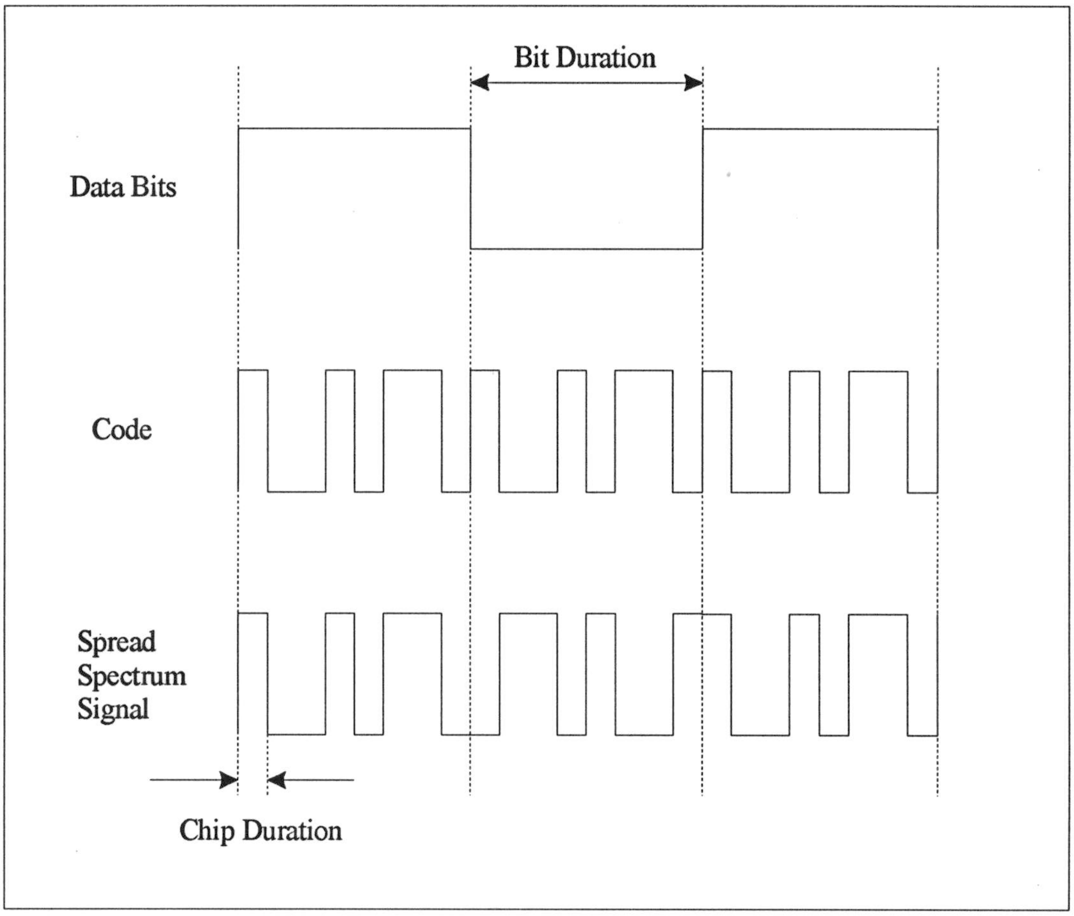

Fig. 2.5: Direct Sequence CDMA signal spreading

2.4 Repeated Random Access Protocols

ALOHA, CSMA and ISMA schemes belong to this type of repeated random access protocols. We only discuss the slotted p-persistent ISMA and the unslotted non-persistent ISMA protocol in this section.

2.4.1 The Slotted P-Persistent ISMA Protocol

The Specification and Description Language (SDL) diagram of the slotted p-persistent ISMA protocol is depicted in Fig. 2.6. Before transmitting the data packet to the receiver, the terminal first listens to the channel to detect whether there is a busy tone transmitted by the base station. The base station broadcasts a busy tone to indicate that the channel is busy because there is a transmission going on.

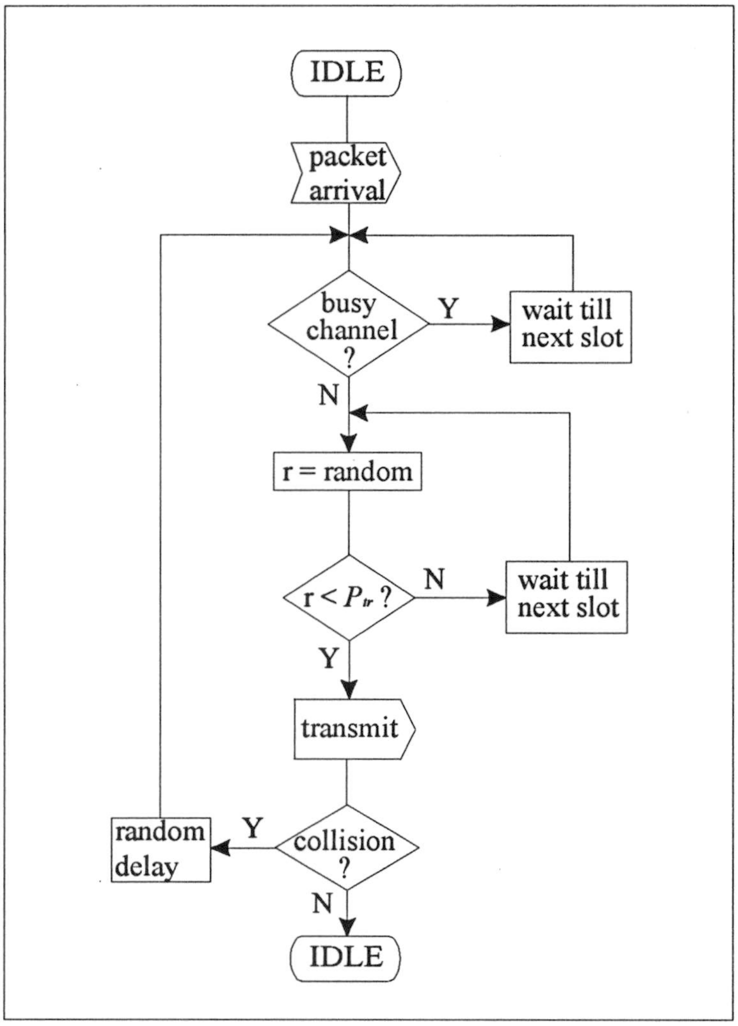

Fig. 2.6: Slotted p-persistent ISMA protocol

If there is a busy tone, indicating that the channel is busy, the terminal postpones its transmission to the next time slot. Otherwise, if the channel is free, the terminal draws a number to see if it is permitted to send its data packet. The probability that a terminal succeeds is P_{tr}. (If P_{tr}=0.1 then the protocol is called 0.1-persistent ISMA) . In case the terminal is unlucky, it waits till the next time slot and draws a number again to see if it's permitted to send. This mechanism is built in to reduce the number of simultaneous transmissions.

If the terminal is permitted to send, it transmits its data packet to the receiver. It is possible that two or more terminals transmit in the same time slot. In this case a collision occurs and the data packets are damaged. The terminals then wait a random delay and start all over again (see also Fig. 2.6).

2.4.2 The Unslotted Non-Persistent ISMA Protocol

The SDL-diagram of the unslotted non-persistent ISMA protocol is shown in Fig. 2.7. This ISMA protocol is unslotted. There are two major differences between this unslotted and the previously in section 2.4.1 described slotted protocol.

The first difference is that the terminal waits a random delay when it senses that the channel is busy. Because the time axis is not divided into time slots, a random delay has to be introduced. After the random delay, terminals are allowed to sense the channel again to see if it's free.

The second difference is related to the first one. In the unslotted scheme, the terminals immediately transmit if the channel is free. The terminals already wait a random delay in case the channel is busy. So, if the channel is free, the number of terminals attempting to transmit is already reduced. It is not necessary to let the terminals draw a random number again to see if they may transmit.

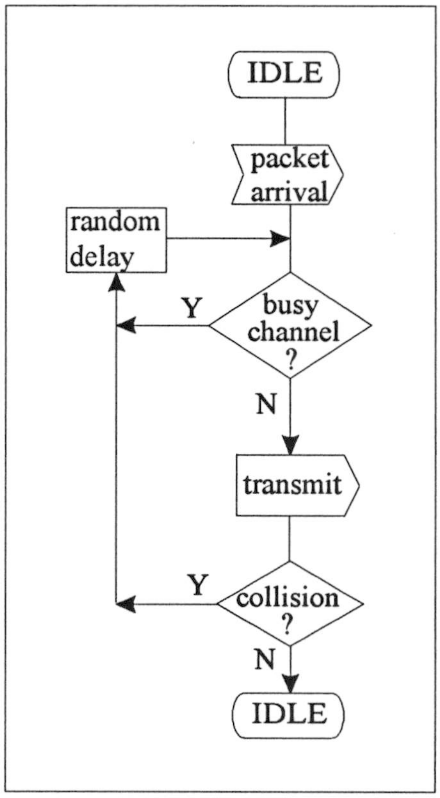

Fig. 2.7: Unslotted non-persistent ISMA

Although the number of collisions is greatly reduced with ISMA, collisions still can occur. In this unslotted ISMA scheme, collisions may occur due to multiple terminals transmitting data packets during an interval called the *inhibit delay fraction d*. This interval *d* is necessary to switch the busy tone from 'idle' to 'busy'. The reverse interval from 'busy' to 'idle' is denoted by *d'*. The inhibit delay fraction is normalised to the packet length, resulting in $0 \leq d < 1$ (Fig. 2.8). In this report we assume that $d' = d$.

The analysis of the hybrid protocol using slotted p-persistent ISMA has already been performed ([11] and [12]). We now concentrate on the hybrid protocol using the unslotted non-persistent ISMA scheme. In case of the unslotted protocol, we expect that for low traffic, the average packet delay can be reduced because terminals do not have to wait to the start of the new time slot.

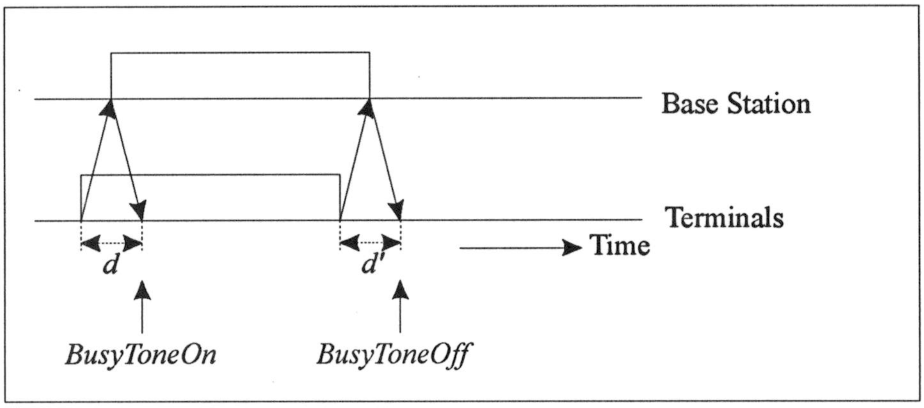

Fig. 2.8: Inhibit delay fraction *d*

2.5 Description of the Hybrid CDMA/ISMA Protocol

Users behind terminals generate data traffic, which is divided in data packets. The data packets arrive at terminals, who take care of the correct delivery. We assume that the arrivals of these packets are generated by a Poisson process. The state of the terminals can either be free or blocked. At the start, the terminals are in the free state. If a packet arrives at a free terminal, the terminal jumps into the blocked state. In the blocked state, the terminal takes care that the arriving data packet is serviced successfully. In the mean time, the blocked terminal ignores all incoming packets. This is a consequence of the assumption that the terminals don't have buffers for the incoming data packets. We assume that the layer above will handle this correctly. In the blocked state, the terminal uses the unslotted non-persistent ISMA protocol to service its data packets. If a data packet is received erroneously, then the ISMA scheme takes care that this problem is solved. Only when the data packet is received correctly, the terminal leaves the blocked state to jump into the free state. New data packets can then be handled.

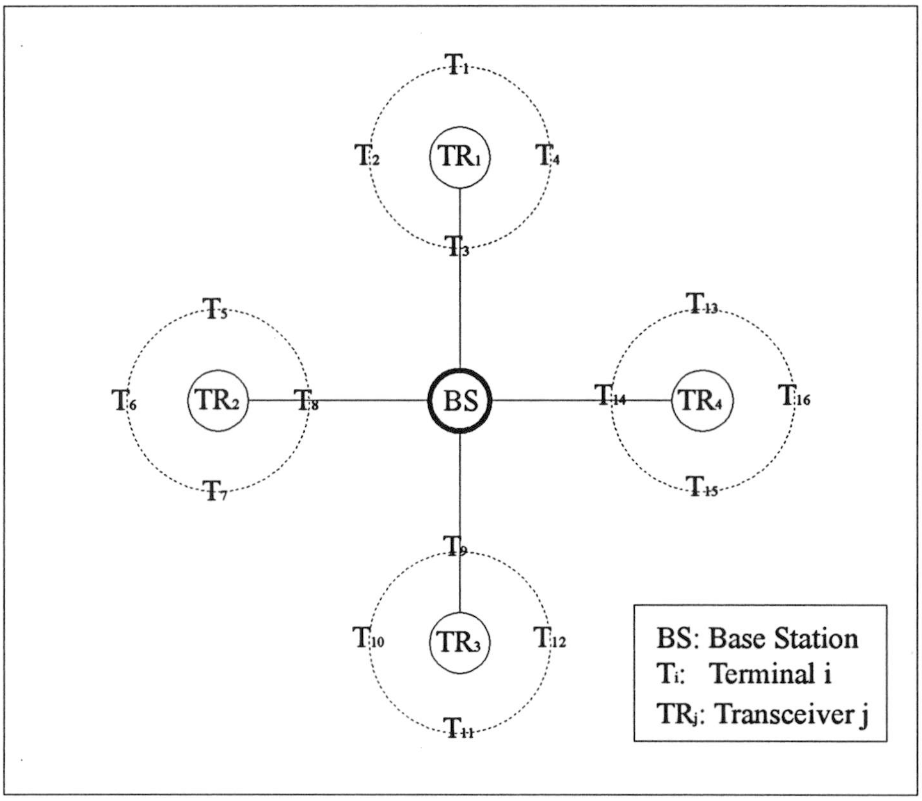

Fig. 2.9: 16-terminal network configuration

Fig. 2.9 depicts an example of the 16-terminal network configuration of the protocol. There is one central base station that is connected to several transceivers by wire. The base station controls the traffic flow with a busy tone that can be detected by all participated terminals. Because all terminals can detect the busy tone, the hidden terminal problem is solved. Several terminals together with one transceiver form a group. Wireless communication is considered to take place between the transceiver and the terminals. The terminals around each transceiver share the same code. In this way the number of codes can be reduced. When a data packet is ready for transmission, a terminal transmits the packet to its transceiver according to the chosen ISMA scheme.

After the arrival of the data packet, the transceiver then simply forwards this packet to the base station which forwards it to all terminals via the transceivers. The destination then receives the packet and decides whether the packet is errorless or not. In case the packet was erroneous, a retransmission must take place. The return channel is not included in our analysis because only the base station makes use of this return channel and therefore contention is not a concern.

2.6 Assumptions

It is impossible and mostly also not necessary to take all complexities of the real life system into account. That is why assumptions have to be made in order to be able to analyse a certain aspect of the system. It is also important to list the assumptions used to indicate under which conditions the generated result is valid. The assumptions we adopted in our work are listed here below. Unless stated otherwise, the values of the parameters we use to obtain the protocol's performance is also given in this section.

2.6.1 Arrival Process

The arrival of data packets at terminals is assumed to be generated by a Poisson process. This means that the inter-arrival times of the data packets at terminals are negative exponentially distributed with λ as parameter. Before a retransmission (caused by a collision or a busy channel) can take place, terminals have to undergo a random delay first (see Fig. 2.7). This random delay is also negative exponentially distributed with α as parameter. α is chosen to be 0.1 and the inhibit delay fraction d is 0.01. The negative exponential distribution will be discussed in Chapter 4.

2.6.2 Signal and Channel Characteristics

After Differential Phase Shift Keying (DPSK) modulation, the signal is spread with a Gold code. The length of this code is chosen to be 31 or 127. The signal is then transmitted, modulated on a 1.7 GHz carrier. The data rate is arbitrarily chosen to be 256·1024 b/s (0.26 Mb/s). The packet length is 64 bits. The delay spread is supposed to be 100 ns and the SNR is 20 dB. Perfect power control is assumed to assure that signals from terminals within the same group arrive at the transceiver with the same power.

The far field model [15] is chosen to describe the signal attenuation. Given the transmitted power P_T, the received power P_R can be expressed by the following equation:

$$P_R = \frac{g_T g_R}{\left(\frac{4\pi fl}{c}\right)^{\mu}} P_T \tag{2.3}$$

where g_T and g_R are respectively the transmitter and the receiver gains, f the signal frequency, l the distance between the transmitter and the receiver, c the speed of light and μ is the attenuation parameter. This attenuation parameter is chosen equal to 2, corresponding to free space propagation.

Fading is the result of the propagation of the transmitted signal through several paths. The channel is modelled as a Rayleigh fading channel. The Rayleigh fading channel model is valid when each path contributes the same amount of energy to the composite received signal. A Line of Sight (LOS) path is therefore assumed to be absent.

2.6.3 Network Configuration

An example of a 16-terminal network configuration that we adopted in this paper is shown in Fig. 2.9. The terminals are clustered around distributed transceivers at a fixed distance (5 m). The distributed transceivers are clustered around the base station at a fixed distance (30 m). Those distances are kept fixed throughout the analysis. The number of terminals within a transceiver group (also called group size) is chosen as a power of two. For a certain fixed number of participated terminals, we have to halve the group size if we want to double the number of codes (the number of codes also equals the number of transceivers). The comparison is fair in this manner, because when we want to investigate the effect of the number of codes on the performance, we have to keep the number of participating terminals fixed. The number of terminals is 32 unless stated otherwise.

3. BIT ERROR RATE IN RAYLEIGH FADING CHANNEL

The bit error rate for direct-sequence spread spectrum with Differential Phase Shift Keying (DPSK) modulation and diversity in Rayleigh fading channel is evaluated in this chapter. The transmitter, channel and receiver model will be presented in the subsequent sections and are similar to those in [16].

3.1 Transmitter Model

The number of active users in the indoor wireless communications system is denoted by K. Fig. 3.1 depicts the CDMA transmission model. User i applies code i to communicate with the base station.

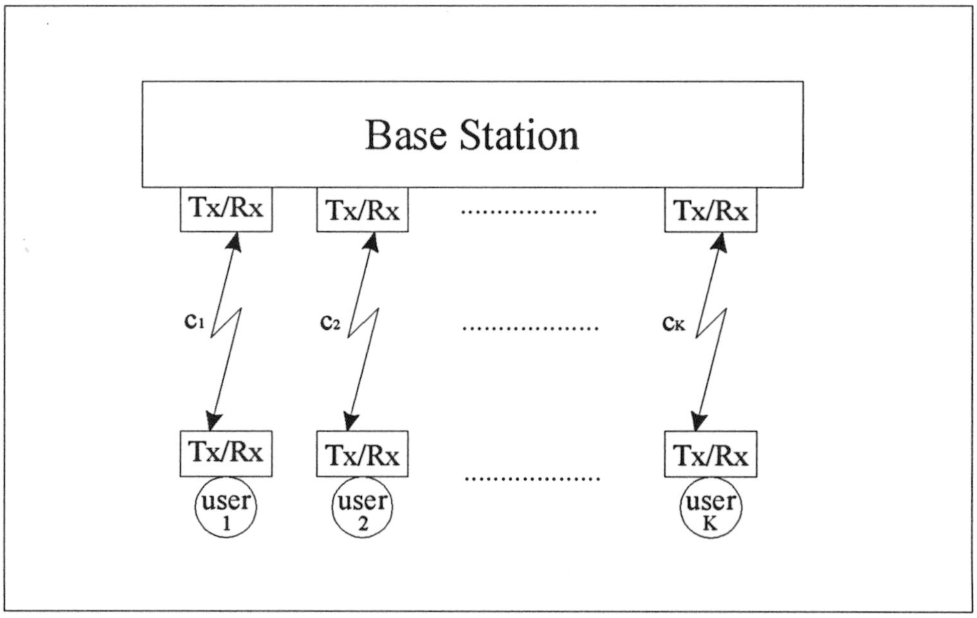

Fig. 3.1: CDMA transmission channel

The kth user's information signal b_k is the differentially encoded binary data sequence of rectangular pulses of width T_b. To spread each data bit, a code waveform a_k of N rectangular pulses of width T_c is used. In literature, a_k is also called the chip waveform. There is one period of code sequence per data bit resulting in $T_b = NT_c$. The chip and data waveforms for the kth user are denoted by $a_k(t)$ and $b_k(t)$ respectively and are given by:

$$a_k(t) = \sum_i a_k^i P_{T_c}(t - iT_c), \; a_k^i \in \{-1,1\} \tag{3.1}$$

$$b_k(t) = \sum_{j=-\infty}^{\infty} b_k^j P_{T_b}(t - jT_b), \; b_k^j \in \{-1,1\} \tag{3.2}$$

Here a_k^i represents the ith chip value of the kth user, b_k^j the kth user data bit at the jth timing interval and $a_k^i = a_k^{i+N}$ for all i; $P_W(t)$ is a rectangular pulse of unit height and width W.

Spreading is achieved by multiplying (or modulo-2 adding) the direct-sequence code to the data signal. The spread signal is then modulated onto the RF carrier signal $A\,cos(\omega_c t + \theta_k)$. The carrier frequency is the same for all users and is denoted by ω_c, while the carrier phase for the kth user is θ_k and A is the carrier level (the energy per bit E_b equals $A^2T_b/2$). It is further assumed that $\omega_c T_b = 2l\pi$, in which l is an integer. The transmitted signal for the kth user $s_k(t)$ is given by:

$$s_k(t) = Aa_k(t)b_k(t)\,cos(\omega_c t + \theta_k) \tag{3.3}$$

3.2 Channel Model

The emitted information reaches the destination not only via the line of sight (LOS) path but also via reflections by buildings and walls. In case the line of sight path dominates the other reflected signals, the sum of the signals at the receiver can be modeled by the Rician distribution. When all the signals are received with approximately the same power, the receiver power distribution can be modeled by the Rayleigh distribution (Fig. 3.2).

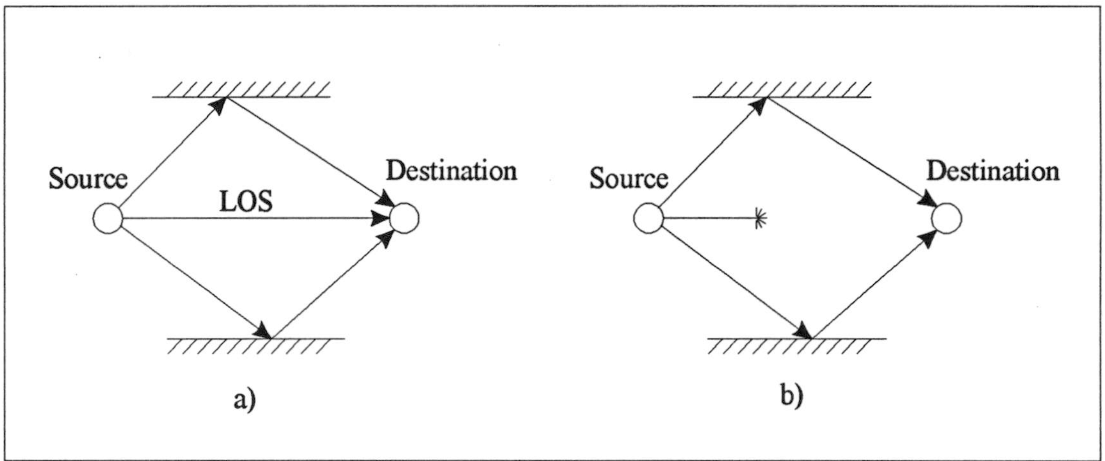

Fig 3.2: Fading channels a) Rice; b) Rayleigh

For our work we model our channel as a Rayleigh fading channel. The complex lowpass equivalent impulse response of the multipath Rayleigh fading channel for the link between the *k*th user and the base station (in our case it is the transceiver) is given by:

$$h_k(\tau) = \sum_{l=1}^{L} \beta_{lk} \delta(\tau - \tau_{lk}) e^{j\gamma lk} \tag{3.4}$$

where β is the path gain and is Rayleigh distributed, τ is the path delay and γ is the path phase. The index *lk* refers to the *l*th path of the *k*th user, and $j^2 = -1$. $\delta(.)$ is the Dirac delta function and L is the number of resolvable paths and is upper bounded by:

$$L = \left\lfloor \frac{T_m}{T_c} \right\rfloor + 1 \tag{3.5}$$

In which $\lfloor x \rfloor$ is the largest integer smaller than or equal to x [12], T_m is the rms delay spread and depends on the size and type of buildings. Reported values are between 20 and 50 ns for small and medium-size office buildings, between 30 and 300 ns for various factory environments, under 100 ns at several university buildings, less than 160 ns over 90% of the area in a shielded building, less than 80 ns in an office building, under 120 ns in a large office building, and up to 200 ns in other large office buildings [17]. For our work we adopted 100 ns as the value for T_m.

We assume that the path delays τ_{lk} are independent random variables and uniformly distributed over [0, T_b]. Fading is the result of the propagation of the transmitted signal through several paths. The channel is modeled as a Rayleigh fading channel. The Rayleigh fading channel model is valid when each path contributes the same amount of energy to the composite received signal. A Line of Sight (LOS) path is therefore assumed to be absent (Fig. 3.2). So, the path gain β_{lk} is an independent Rayleigh random variable for each l and k. The Rayleigh distribution for β_{lk} is given as:

$$P_{\beta_{lk}}(r) = \frac{r}{\rho_{lk}} exp\left(-\frac{r^2}{2\rho_{lk}}\right), \; r \geq 0 \tag{3.6}$$

In which r is the signal amplitude and $\rho_{lk} = \frac{1}{2}E\left[\beta_{lk}^2\right]$ is the average scattered power.

3.3 Receiver Model

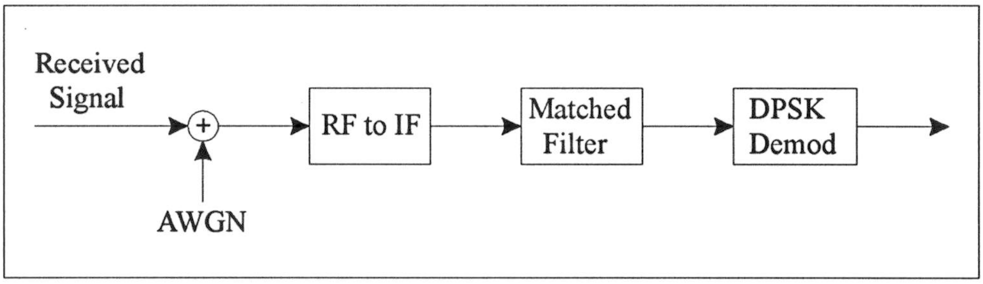

Fig. 3.3: Spread spectrum receiver using DPSK modulation

The receiver consists of a RF to IF converter, a matched filter and a DPSK demodulator (Fig. 3.3). We assume that the received signal *r(t)* is composed of the contributions of the different users, their different paths and additive white Gaussian noise (AWGN). The receiver input signal at the antenna for a certain user can be written as:

$$r(t)=\sum_{k=1}^{K}\sum_{l=1}^{L}A\beta_{lk}a_k(t-\tau_{lk})b_k(t-\tau_{lk})cos(\omega_c t+\phi_{lk})+n(t) \tag{3.7}$$

Here, *n(t)* is the white Gaussian noise with two-sided power spectral density $N_0/2$ [W/Hz]. We assume that the path phase ϕ_{lk}, given by $(\omega_c\tau_{lk}+\gamma_{lk}+\theta_k)$, is an independent random variable uniformly distributed over [0, 2π]. In terms of low-pass signals, *r(t)* and *n(t)* can also be expressed as:

$$r(t)=x(t)cos(\omega_c t)-y(t)sin(\omega_c t) \tag{3.8}$$

$$n(t)=n_c(t)cos(\omega_c t)-n_s(t)sin(\omega_c t) \tag{3.9}$$

Here, $n_c(t)$ and $n_s(t)$ are the lowpass equivalent components of $n(t)$ and

$$x(t)=\sum_{k=1}^{K}\sum_{l=1}^{L}A\beta_{lk}a_k(t-\tau_{lk})b_k(t-\tau_{lk})cos(\phi_{lk})+n_c(t) \tag{3.10}$$

$$y(t)=\sum_{k=1}^{K}\sum_{l=1}^{L}A\beta_{lk}a_k(t-\tau_{lk})b_k(t-\tau_{lk})sin(\phi_{lk})+n_s(t) \tag{3.11}$$

Selecting user 1 as the reference user, each component of the received signal is multiplied by the direct sequence code associated with user 1. The output (in-phase and quadrature component) of the matched filter of user 1 at the sampling instant $(t=T_b)$ is then given as:

$$g_x(T_b)=\sum_{k=1}^{K}\sum_{l=1}^{L}A\beta_{lk}\,cos(\phi_{lk})\left[b_k^{-1}R_{1k}(\tau_{lk})+b_k^{0}\hat{R}_{1k}(\tau_{lk})\right]+\eta \tag{3.12}$$

$$g_y(T_b)=\sum_{k=1}^{K}\sum_{l=1}^{L}A\beta_{lk}\,sin(\phi_{lk})\left[b_k^{-1}R_{1k}(\tau_{lk})+b_k^{0}\hat{R}_{1k}(\tau_{lk})\right]+\upsilon \tag{3.13}$$

Where g_x and g_y are the in-phase and the quadrature components, b_k^{-1} the previous and b_k^0 the current data bit. The noise samples η and υ are indepentdent, zero-mean Gaussian random variables with identical variance $\sigma_n^2 = N_0 T_b$. Further,

$$R_{1k}(\tau) = \int_0^{\tau} a_k(t-\tau) a_1(t) dt \tag{3.14}$$

$$\hat{R}_{1k}(\tau) = \int_{\tau}^{T_b} a_k(t-\tau) a_1(t) dt \tag{3.15}$$

Let us assume without loss of generality that the receiver synchronizes to the *j*th path of user 1, so that $\tau_{j1} = 0$ and $\phi_{j1} = 0$. All other delay paths constitute interference and are then defined as relative according to this reference. The complex envelope of the signal $z_1 = g_x(T_b) + j g_y(T_b)$ at the current sampling instant is:

$$\begin{aligned} z_1 = A\beta_{j1} T_b b_1^0 + A\sum_{k=1}^{K}\left(b_k^{-1} X_k + b_k^0 \hat{X}_k\right) \\ + jA\sum_{k=1}^{K}\left(b_k^{-1} Y_k + b_k^0 \hat{Y}_k\right) + (\eta_1 + j\upsilon_1) \end{aligned} \tag{3.16}$$

For the previous sampling instant, we have something similar

$$\begin{aligned} z_{-1} = A\beta_{j1} T_b b_1^{-1} + A\sum_{k=1}^{K}\left(b_k^{-2} X_k + b_k^{-1} \hat{X}_k\right) \\ + jA\sum_{k=1}^{K}\left(b_k^{-2} Y_k + b_k^{-1} \hat{Y}_k\right) + (\eta_2 + j\upsilon_2) \end{aligned} \tag{3.17}$$

in which b_k^{-2} is the *k*th user data bit transmitted 2 bits intervals prior to b_k^0 and η_1, η_2, υ_1 and υ_2 are the noise variables independent of one another. Further,

$$X_1 = \sum_{\substack{l=1 \\ l \neq j}}^{L} R_{11}(\tau_{l1}) \beta_{l1} \cos(\phi_{l1}) \tag{3.18}$$

$$\hat{X}_1 = \sum_{\substack{l=1 \\ l \neq j}}^{L} \hat{R}_{11}(\tau_{l1})\beta_{l1} cos(\phi_{l1}) \tag{3.19}$$

$$Y_1 = \sum_{\substack{l=1 \\ l \neq j}}^{L} R_{11}(\tau_{l1})\beta_{l1} sin(\phi_{l1}) \tag{3.20}$$

$$\hat{Y}_1 = \sum_{\substack{l=1 \\ l \neq j}}^{L} \hat{R}_{11}(\tau_{l1})\beta_{l1} sin(\phi_{l1}) \tag{3.21}$$

For $k \geq 2$,

$$X_k = \sum_{l=1}^{L} R_{1k}(\tau_{lk})\beta_{lk} cos(\phi_{lk}) \tag{3.22}$$

$$\hat{X}_k = \sum_{l=1}^{L} \hat{R}_{1k}(\tau_{lk})\beta_{lk} cos(\phi_{lk}) \tag{3.23}$$

$$Y_k = \sum_{l=1}^{L} R_{1k}(\tau_{lk})\beta_{lk} sin(\phi_{lk}) \tag{3.24}$$

$$\hat{Y}_k = \sum_{l=1}^{L} \hat{R}_{1k}(\tau_{lk})\beta_{lk} sin(\phi_{lk}) \tag{3.25}$$

The decision variable for DPSK demodulation is

$$\xi = Re\left[z_1 z_{-1}^*\right] \tag{3.26}$$

In which $Re[x]$ denotes the real part of x and * the complex conjugation. Errors occur if the channel attenuation is large. If we somehow provide the receiver with several independently fading signal paths, the probability that all the signal components will fade simultaneously is reduced considerably. This is the basis for diversity techniques. The selection diversity of order M is based on selecting the strongest of the decision variable:

$$\xi_{max}^{M} = max_{i=1,\dots,M}\left(\xi_i\right) \tag{3.27}$$

By using multiple antennas, the highest possible order of diversity, i.e. number of paths to choose from, can be increased to $M_{max}=K_aL$ where M_{max} is the maximum order of diversity and K_a is the number of antennas. Designate ξ_{max} as the decision variable obtained from demodulation of the strongest path. The bit error probability in case of selection diversity is defined as

$$P_{be} \underline{\Delta} P_r\left(\xi_{max} < 0 \middle| b_1^0 b_1^{-1} = 1\right) \underline{\Delta} P_r\left(\xi_{max} > 0 \middle| b_1^0 b_1^{-1} = -1\right) \tag{3.28}$$

If we assume that all path delays are given and β_{max} is correctly selected, the formula for the bit error probability is given by [16]:

$$P_{be|\beta_{max},\{\tau_{lk}\},L} = Q(a,b) - \frac{1}{2}\left(1 + \frac{\mu_{12}}{\sqrt{\mu_1\mu_2}}\right) I_0(a,b) exp\left(-\frac{a^2+b^2}{2}\right) \tag{3.29}$$

where

$Q(a,b)$ is the Marcum Q-function

$$a = \frac{m}{\sqrt{2}}\left|\frac{1}{\sqrt{\mu_1}} - \frac{1}{\sqrt{\mu_2}}\right| \tag{3.30}$$

$$b = \frac{m}{\sqrt{2}}\left|\frac{1}{\sqrt{\mu_1}} + \frac{1}{\sqrt{\mu_2}}\right| \tag{3.31}$$

$$m = A\beta_{max}T_b b_1^{\,0} = A\beta_{max}T_b b_1^{\,-1} \tag{3.32}$$

$$\mu_1 = A^2 E\left[\sum_{k=1}^{K} X_k{}^2 + \hat{X}_k{}^2 + Y_k{}^2 + \hat{Y}_k{}^2 \middle| \{\tau_{lk}\}, L\right]$$

$$+2A^2 E\left[X_1\hat{X}_1 + Y_1\hat{Y}_1 \middle| \{\tau_{lk}\}, L\right] + 2\sigma_n{}^2 \tag{3.33}$$

$$\mu_2 = A^2 E\left[\sum_{k=1}^{K} X_k{}^2 + \hat{X}_k{}^2 + Y_k{}^2 + \hat{Y}_k{}^2 \middle| \{\tau_{lk}\}, L\right] + 2\sigma_n{}^2 \tag{3.34}$$

$$\mu_{12} = A^2 E\left[\sum_{k=1}^{K} \left(X_k\hat{X}_k + Y_k\hat{Y}_k\right) + \hat{X}_1{}^2 + \hat{Y}_1^2 \middle| \{\tau_{lk}\}, L\right] \tag{3.35}$$

The conditional expectations in the above expressions can be evaluated as follows:

$$E\left[X_1^2 \middle| \{\tau_{lk}\}, L\right] = E\left[Y_1^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{\substack{l=1 \\ l \neq j}}^{L} R_{11}^2(\tau_{l1})\rho_{l1} \tag{3.36}$$

$$E\left[\hat{X}_1^2 \middle| \{\tau_{lk}\}, L\right] = E\left[\hat{Y}_1^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{\substack{l=1 \\ l \neq j}}^{L} \hat{R}_{11}^2(\tau_{l1})\rho_{l1} \tag{3.37}$$

$$E\left[X_1\hat{X}_1 \middle| \{\tau_{lk}\}, L\right] = E\left[Y_1\hat{Y}_1 \middle| \{\tau_{lk}\}, L\right] = \sum_{\substack{l=1 \\ l \neq j}}^{L} R_{11}(\tau_{l1})\hat{R}_{11}(\tau_{l1})\rho_{l1} \tag{3.38}$$

$$E\left[X_k^2 \middle| \{\tau_{lk}\}, L\right] = E\left[Y_k^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{l=1}^{L} R_{1k}^2(\tau_{lk})\rho_{lk} \tag{3.39}$$

$$E\left[\hat{X}_k^2 \middle| \{\tau_{lk}\}, L\right] = E\left[\hat{Y}_k^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{l=1}^{L} \hat{R}_{1k}^2(\tau_{lk})\rho_{lk} \qquad ;k \geq 2 \tag{3.40}$$

$$E\left[X_k\hat{X}_k \middle| \{\tau_{lk}\}, L\right] = E\left[Y_k\hat{Y}_k \middle| \{\tau_{lk}\}, L\right] = \sum_{l=1}^{L} R_{1k}(\tau_{lk})\hat{R}_{1k}(\tau_{lk})\rho_{lk} \qquad ;k \geq 2 \tag{3.41}$$

4. COMPUTER SIMULATION

A simulation program has been created to analyse the protocol behaviour. In this chapter we describe how the computer simulation program was set up to measure the performance of the unslotted hybrid CDMA/ISMA protocol. The primary goal of this simulation program is to see whether the performance of the hybrid protocol using the *slotted p-persistent* scheme on the one hand and the *unslotted non-persistent* scheme on the other, differ from each other, and if so in what way. Knowledge in this matter give us meaningful insight that we can use to design our protocol appropriately. This chapter also includes the generation of Poisson data traffic.

4.1 Simulation Techniques

In general, if we want to use a computer simulation to analyse a system, the following steps can be distinguished. The dynamic behaviour of the system is studied by tracing various system states as a function of time and then collecting and analysing the system statistics. The events that change the system state are generated at different points in time, and the passage of time is represented by an internal clock which is incremented and maintained by the simulation program.

The simulation time can be advanced in two ways. The first method is the *interval-oriented simulation* (or the uniform time increment method) where the clock is advanced from time t to $(t + \Delta t)$ where Δt is a uniform fixed time increment. Fig. 4.1a depicts this mechanism. The second method is the *event-oriented simulation* (or the variable time increment method) where the clock is incremented from time t to the next event time t', whatever may be the value of t'. The state changes are made at event time t, the next event time t', and this process is continuously repeated. Thus, only events are represented explicitly in a simulation model and the periods between events are treated as inactive or insignificant and therefore consume no time even though the inter-event activities do consume time in the real world (Fig. 4.1b).

Obviously, method 1 detects the events that occur during the interval (t, $t+\Delta t$) only at time ($t+\Delta t$), thereby introducing errors in simulation. Another drawback of this method is that if the interval between two events is very large compared to Δt, then the simulator goes through several unproductive clock increments (during periods of inactivity) and the associated computing effort which will not bring about any change in system states. This fixed time increment method is suitable for the simulation of continuous systems and in particular systems with large numbers of state variables.

The second method involves sorting of event activation times and maintaining an *event list*. In our work we will employ the event-oriented simulation to analyse the unslotted hybrid CDMA/ISMA protocol.

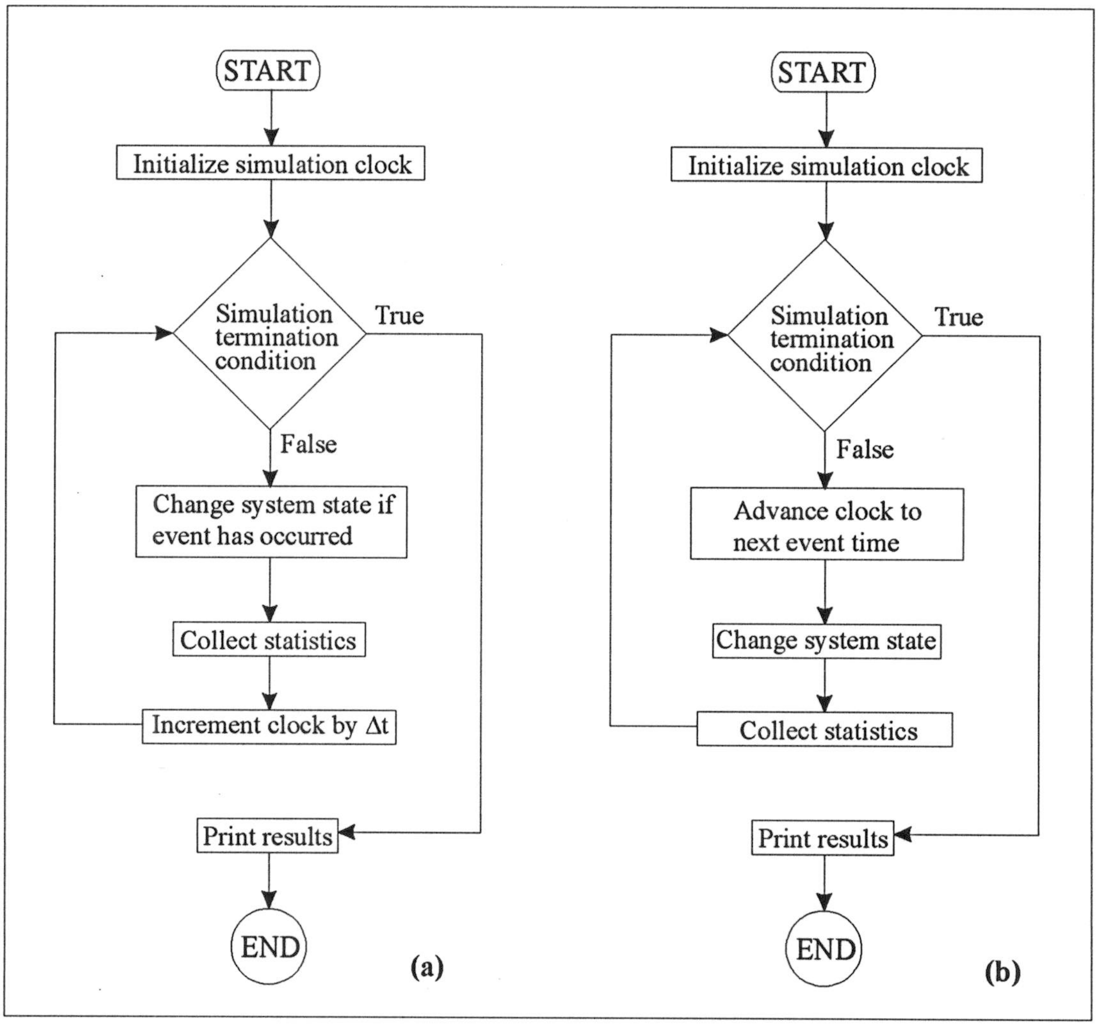

Fig. 4.1: (a) Interval-oriented simulation; (b) Event-oriented simulation

4.2 Simulation Program

This section describes the simulation program in terms of main program structure, event list and event types. The source code can be found in Appendix D.

4.2.1 Main Structure

The Program Structure Diagram (PSD) of the main program is depicted in Fig. 4.2. The simulation program simulates the working of the unslotted hybrid CDMA/ISMA protocol. First of all, the counters have to be reset to the proper values. *Clock* indicates the actual time while the simulation will last *EndTime*. While *Clock* has not reached *EndTime*, the simulation will loop. The loop is divided in four steps as shown in Fig. 4.2. In the first step, the program advances *Clock* to the next event time. This event is the first event to occur in future time. Because the event-oriented simulation is used, only events are represented explicitly in the simulation model. Depending on the event type, the program processes this event. This is the second step. The processing can generate other events or change the system state. In the third step, the generated events and the changed system states will be updated. Finally, statistics collection completes the loop. (The warm up procedure is not shown in Fig. 4.2).

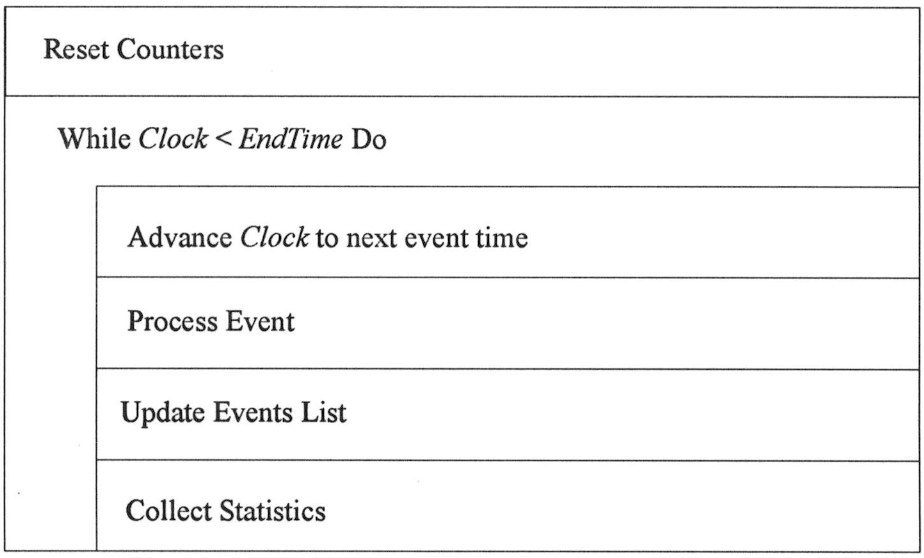

Fig. 4.2: PSD of main program

4.2.2 Event List

From Fig. 4.2 it emerges that events and event list play an important role in the main program structure. The whole program is based on the event list. The event list consists of events linked together in an organised manner. Here, organised means in an order in which the events occur. An example of an event list is shown in Fig. 4.3. The event list consists of nodes linked together to form a chain. In the event-oriented simulation this chain symbolises the time axis. Every node represents an event and consists of an event type, the time this event occur, other information and a tail. The tail consists of a pointer pointing to the next node. The last node points nowhere. This is illustrated with the ground symbol.

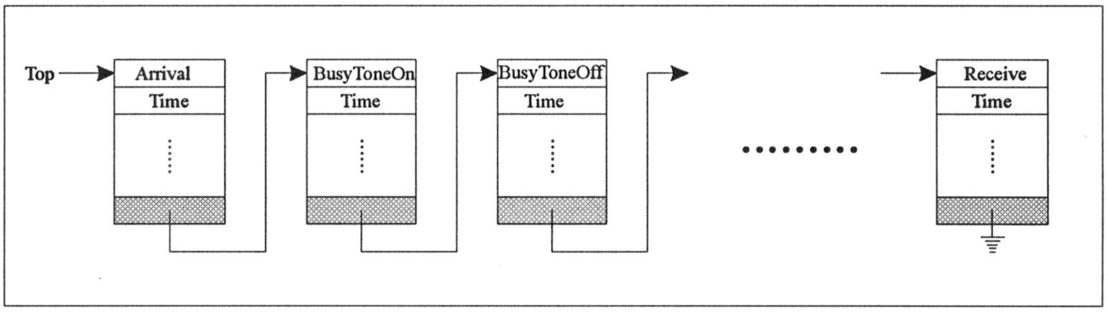

Fig. 4.3: Example of an event list

4.2.3 Event Types

There are five event types: *Arrival*, *Retransmission*, *BusyToneOn*, *BusyToneOff* and *Receive*. From Fig. 4.2 we see that after advancing the clock to the next event, the event will be processed depending on the event type. Those event types together with their corresponding processing are described here below.

Arrival - The event *Arrival* indicates the arrivals of newly arrived packets. This arrival process is modelled by the Poisson process. The inter-arrival times of the packets are negative exponentially distributed. Section 4.3 shows how the generation of this traffic stream is implemented in the simulation program.

The procedure *ProcessArrival* is illustrated in Fig. 4.4. If a new packet arrives, the next packet arrival is generated and put into the event list. This new *Arrival* event has to occur after the inter-arrival time T_{IA}, which depends on the implemented Poisson process with parameter λ. Because the terminals are assumed not be possess any buffers, a new packet only get serviced if this terminal is idle. Provided that this is the case, then the state of the terminal is set to BLOCKED, which means that newly arrived packets will be ignored. Then, collection of the arrival data is performed. *Clock* denotes the current simulation time of the internal clock of the program.

The program makes use of a global variable *BusyTone* that can takes on two values: ON or OFF. In case the *BusyTone* is ON, in which case the terminal is inhibited to send, then it has to wait a random delay before it can try again. Again, data collection must take place, this time it is the retransmission and delay data. If the *BusyTone* is OFF, then the terminal is allowed to send its data packet. First, the terminal state is altered to TRANSMIT. Then, after the inhibit delay fraction d, the *BusyTone* is set to ON. Subsequently, the event *Receive* is created. And finally, the delay data is collected for later processing.

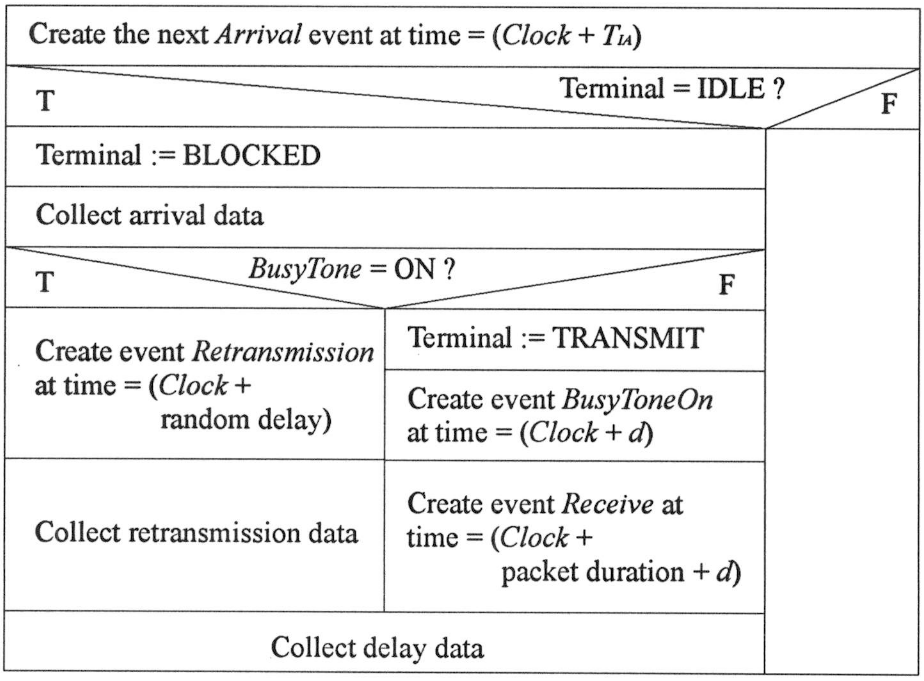

Fig. 4.4: PSD of procedure *ProcessArrival*

Retransmission - The event *Retransmission* indicates the arrivals of the retransmitted packets. The random delay the terminals have to wait before they are allowed to try again is also modelled by the Poisson process. The inter-arrival time of the packets is negative exponentially distributed with parameter α.

Fig. 4.5 depicts the procedure *ProcessRetransmission*. The terminal senses the channel upon the arrival of a retransmitted packet. If *BusyTone* is ON, then an another retransmission must be sent after a random delay. The collection of retransmission and delay data is then performed. If the *BusyTone* is OFF, then exactly the same steps as in *ProcessArrival* (after *BusyTone*=ON? is F) have to be taken.

T — *BusyTone* = ON ?	F
Create event *Retransmission* at time = (*Clock* + random delay)	Terminal := TRANSMIT
	Create event *BusyToneOn* at time = (*Clock* + *d*)
Collect retransmission data	Create event *Receive* at time = (*Clock* + packet duration + *d*)
Collect delay data	

Fig. 4.5: PSD of procedure *ProcessRetransmission*

BusyToneOn and *BusyToneOff* - The events *BusyToneOn* and *BusyToneOff* take care of the inhibit mechanism of the ISMA protocol. The global variable *BusyTone* contains the right state of the *BusyTone* at any clock instant, indicating whether the channel is free or not. Fig. 4.6 and 4.7 depict the procedures *ProcessBusyToneOn* and *ProcessBusyToneOff*.

BusyTone := ON

Fig. 4.6: *ProcessBusyToneOn*

BusyTone := OFF

Fig. 4.7: *ProcessBusyToneOff*

Receive - The event *Receive* indicates the moment that the terminal knows whether the transmitted packet is received correctly or it has to be retransmitted. The procedure *ProcessReceive* is shown in Fig. 4.8. Only the first incoming packet at the base station will be serviced. If the terminal is not the first sending terminal then it has to retransmit his packet again after a random delay because its packet is ignored by the base station. This is accomplished by the event *Retransmission*. Then retransmission and delay data is collected.

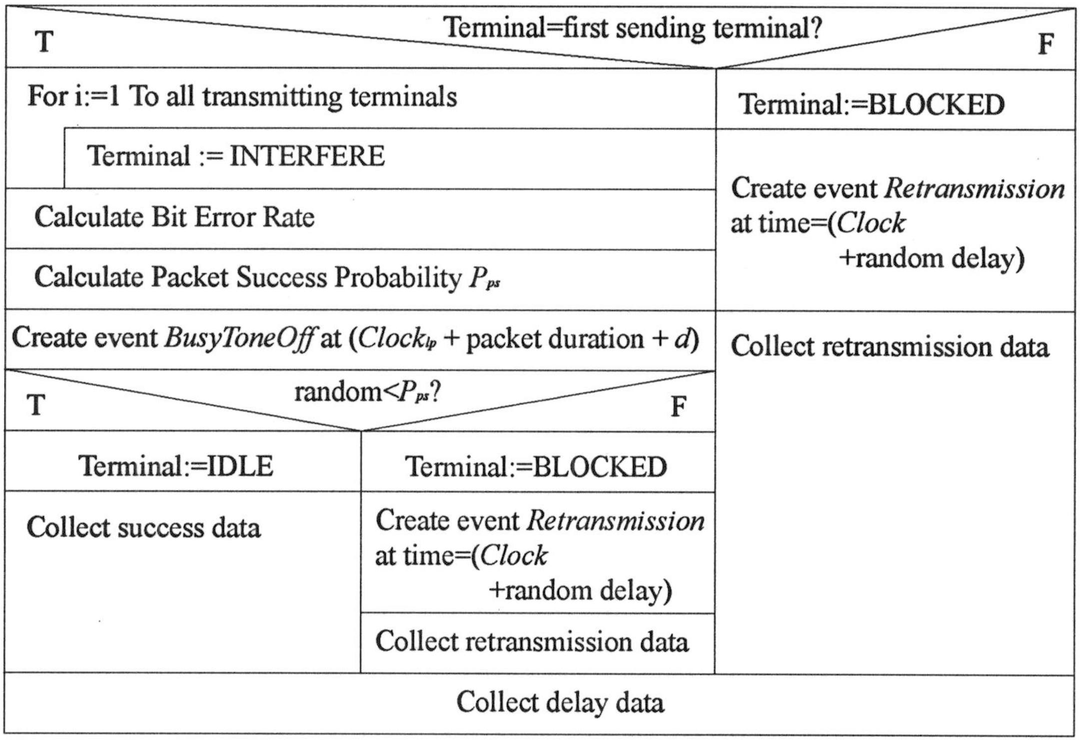

Fig. 4.8: *ProcessReceive*

If the terminal is the first sending terminal then its data packet is serviced by the base station. This does not mean that this packet is automatically received correctly by the destination. Due to the other terminals who transmit during the inhibit delay fraction (d), data packets collide which causes interference. Because the inhibit delay fraction is only a small fraction of the packet duration and the channel is stationary (=time invariant) (see also equation 3.4), we assume that the bit error rate does not alter during the data packet. Notice that this is the worst case situation we are calculating (Fig. 4.9). If the bit error rate is known, then the packet success probability is not difficult to calculate. Fig. 4.9 also shows where the *BusyTone* has to be turned OFF. This time instant is denoted in Fig. 4.8 with

($Clock_{lp}$ + packet duration + d). $Clock_{lp}$ is the time instant that the last packet is transmitted during the inhibit delay fraction. If the packet is received correctly, then the terminal state is set to IDLE to allow other newly arrived packets to get serviced. Then, the data is collected. On the other hand, if the packet is received erroneously, then the terminal state is set to BLOCKED again and a retransmission is generated. Data collection finally completes the discussion of the five event types and their processing procedures.

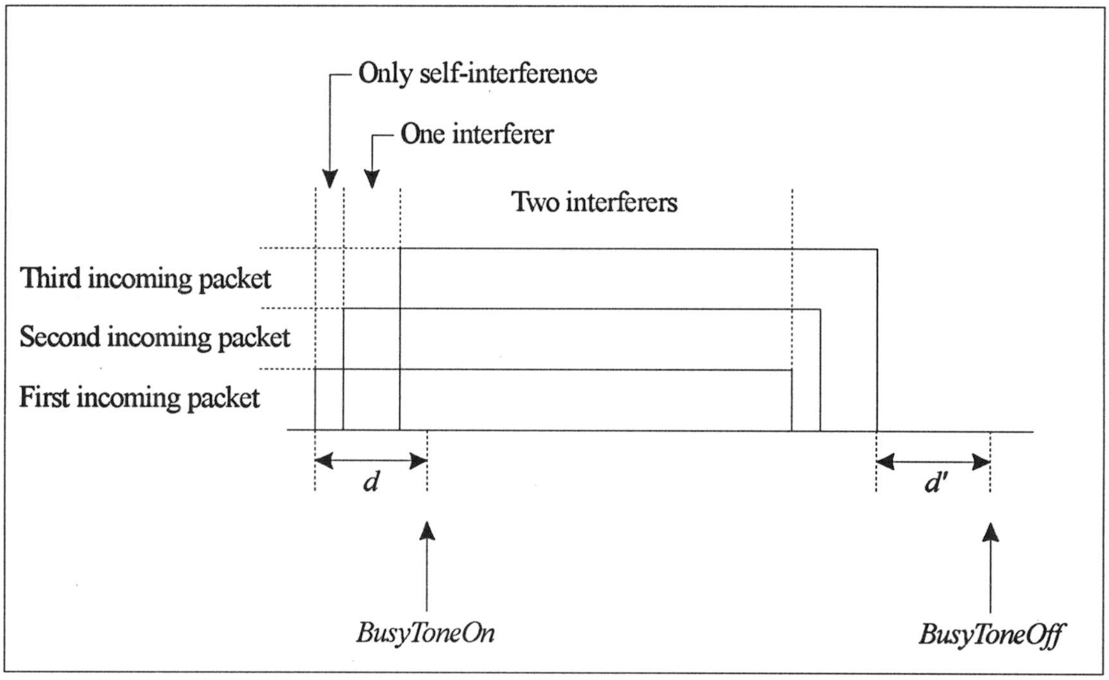

Fig. 4.9: Data packet collisions

4.3 The Negative Exponential Distribution

Arrivals of data packets at terminals are supposed to be generated by a Poisson process. The inter-arrival times of events of this Poisson process is negative exponentially distributed. The negative exponential distribution has a memoryless characteristic. The probability density function (PDF) and the cumulative distribution function (CDF) for the inter-arrival times are respectively given as:

$$f_T(t) = \begin{cases} \lambda \cdot e^{-\lambda t} & ;t \geq 0 \\ 0 & ;t < 0 \end{cases} \tag{4.2}$$

$$F_T(t) = \begin{cases} 1 - e^{-\lambda t} & ;t \geq 0 \\ 0 & ;t < 0 \end{cases} \tag{4.3}$$

In which t is the time it will passed till the next arrival of a packet at a terminal when on average there are λ arrivals per time unit [18]. The random variable T is then negative exponentially distributed with average and variance given as:

$$m_1 = \frac{1}{\lambda} \tag{4.4}$$

$$\sigma^2 = \frac{1}{\lambda^2} \tag{4.5}$$

Now we wish to generate a random variable T whose PDF and CDF are given in (4.2) and (4.3). Herefore, we make use of the so called 'Inverse Transform Method' [19]. This method functions as follows:

When the PDF $f_X(x)$ is given, we can calculate the CDF $F_X(x)$ by integration of $f_X(x)$. Otherwise, by differentiation of $F_X(x)$, we get $f_X(x)$. The value of $F_X(x)$ varies between zero and one. So, when we wish to generate a random variable whose PDF or CDF is given, we first have to generate a random number U between zero and one with a uniform distribution. This number U functions as a value for $F_X(x)$. We then have: $F_X(x) = U$. The random variable X we want to generate follows easily: $x = F_X{}^{-1}(U)$.

The above description is of course not a mathematical proof, but serves only as background for the inverse transform method. The mathematical proof follows now.

4.3.1 Proof of the Inverse Transform Method

Let X denote a random variable with CDF F_X and $F_X{}^{-1}$ the inverse of F_X such that:

$$F_X{}^{-1}(y) = inf\,imum\{x|\ F_X(x) \geq y\} \qquad ;0 < y < 1 \tag{4.6}$$

This means that $F_X{}^{-1}(y)$ is the *infimum* or smallest value of X for which $F_X(x) \geq y$ applies. First, we define the random variable Y as:

$$Y = F_X{}^{-1}(U) \tag{4.7}$$

Then,

$$\begin{aligned} Pr(Y \leq y) &= Pr\left[F_X{}^{-1}(U) \leq y\right] \\ &= Pr\left[U \leq F_X(y)\right] \\ &= F_X(y) = Pr(X \leq y) \end{aligned} \tag{4.8}$$

So we can conclude that X and Y have the same distribution. Therefore, applying $F_X{}^{-1}$ to an uniform distribution U produces an x from F_X. This result suggests the following computing algorithm:

Let U be uniformly distributed on $(0,1)$. Then,

1. Generate U;
2. $x := F_X{}^{-1}(U)$;
3. Deliver x

4.3.2 Generation of the Random Numbers for a Negative Exponential Distribution

Since the CDF of the negative exponential distribution exists explicitly as

$$F_T(t) = 1 - e^{-\lambda t} \tag{4.9}$$

and its inverse is easy to calculate

$$F_T^{-1}(t) = -\frac{1}{\lambda} \cdot ln(1-t) \tag{4.10}$$

we can use the inverse transform method to generate the random numbers for the negative exponential distribution.

Algorithm for random numbers generation of $F_T(t)$:

Let U be uniformly distributed on (0,1), then

1. Generate U;
2. $t = F_T^{-1}(U) = -\frac{1}{\lambda} \cdot ln(1-U)$
3. Deliver t.

5. MATHEMATICAL ANALYSIS

The performance analysis of the hybrid unslotted non-persistent CDMA/ISMA will be explained from Fig. 5.1. The time axis in Fig. 5.1 is normalised to the packet duration. If a packet arrives and the terminal senses the channel free, then the packet is sent immediately. Suppose that this time instant is t. It takes d, the inhibit delay fraction, before the busy tone is heard by all terminals. This moment occurs at the same time at all terminals because of the symmetric network configuration. Any other packet arriving between t and $t+d$ will sense the channel free (because the busy tone has not arrived yet) and will be transmitted resulting in a conflict. Because a CDMA scheme is also used, the conflict does not necessarily result in the loss of all packets. There is a probability that the first arriving packet can still be recovered successfully. If no other terminal transmits a packet during this period d, then no conflict occurs. We model the channel between the terminals and the corresponding receivers as a multipath Rayleigh-fading channel. So, when no conflicts occur, there is still a probability that the packet can not be recovered successfully.

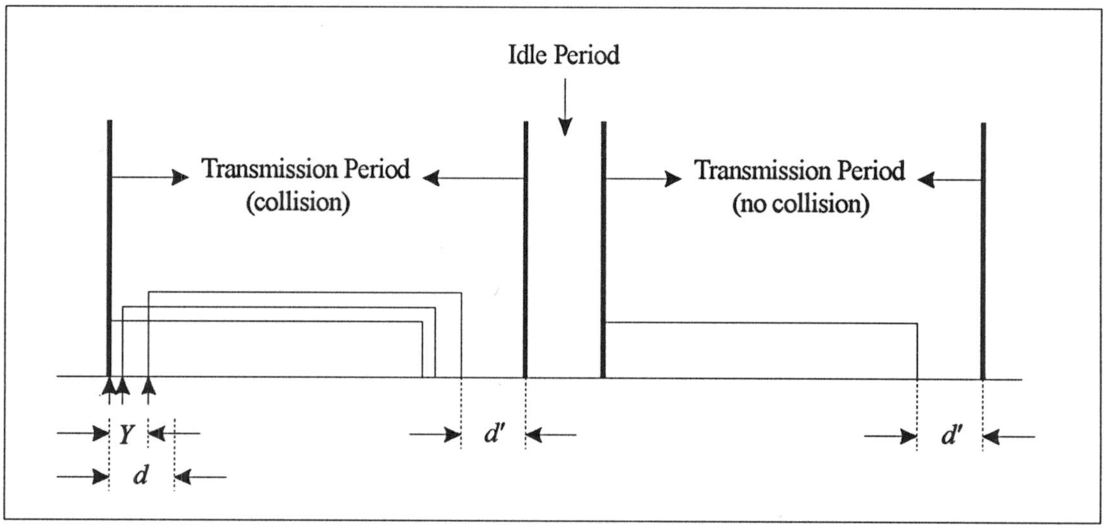

Fig. 5.1: Data cycle of the unslotted non-persistent ISMA protocol

5.1 Throughput

Let $t+Y$ be the time of occurrence of the last packet arriving between t and $t+d$. This obviously means that Y must be between zero (the first transmitted packet is the only packet in the transmission period) and d (the end of the vulnerable period). The transmission of all packets arriving in $(t, t+Y)$ will be completed at $t+Y+1$. As noted before, the channel is sensed unused only a period d later. So, any terminal becoming ready between $t+d$ and $t+Y+1+d$ will sense the channel busy and hence will reschedule its packet. The interval between t and $t+Y+1+d$ is called a *transmission period* (*TP*). There will be *at most* one successful transmission during a *TP*. The *idle period* (*IP*) is defined as the period of time between two consecutive *TP*'s. A transmission period plus the following idle period constitute a *cycle*. Let $\overline{TP}$ be the expected duration of the transmission period, $\overline{I}$ the expected duration of the idle period, and the average cycle time can be written as [20]:

$$\overline{t_c} = \overline{TP} + \overline{I} \tag{5.1}$$

Let U denote the time during a cycle in which the inbound channel (towards base station) is used to carry a successful packet transmission and $\overline{U}$ the corresponding average value, then we can write the throughput as:

$$S = \frac{\overline{U}}{\overline{t_c}} \tag{5.2}$$

The expected useful time $\overline{U}$ can be computed as follows. When a packet is successful, the channel carries useful information for a duration of T_{pd}, the packet duration. In the unsuccessful case, no useful information is carried at all or, in other words:

$$U = \begin{cases} T_{pd} & ;Successful \quad period \\ 0 & ;Unsuccessful \quad period \end{cases} \tag{5.3}$$

If $P_{success}$ denotes the probability that a transmitted packet is successful then

$$\overline{U} = E[U] = T_{pd} \cdot P_{success} + 0 \cdot (1 - P_{success}) = T_{pd} \cdot P_{success} \quad (5.4)$$

As noted before, the time is normalised to T_{pd} (the packet duration), and therefore T_{pd} equals one. So, this gives us:

$$\overline{U} = P_{success} \quad (5.5)$$

To calculate the average idle period we make the assumption that the total rate at which users schedule new and retransmitted packets forms a Poisson process with parameter G. So, new and rescheduled packets arrive at a rate of G packets per unit time. Not all arrivals do result in a transmission. If a packet arrives and the terminal finds the channel in a busy state then the packet is rescheduled for transmission at a later time instant. In literature, G is also called the offered channel traffic.

If the packet arrival times are independent and exponentially distributed with a mean arrival rate of G packets per second, the probability of k arrivals in an interval of duration t is then a Poisson process given by $P_k(t)$ [18], where

$$P_k(t) = \frac{(Gt)^k \cdot e^{-Gt}}{k!} \quad (5.6)$$

For our work, the time is normalised to the packet duration (T_{pd}). This means that G denotes the number of packet arrivals per *packet duration* and the time unit is also measured in *packet duration* instead of seconds. The probability of the idle time being greater than some value t is the probability that no packets are scheduled within a time interval of duration t and with the assumed Poisson packet scheduling process this probability becomes:

$$P(I \geq t) = P(\textit{no packet arrival in interval of duration } t) = e^{-Gt} \quad (5.7)$$

Therefore the average value of I can be expressed as:

$$\overline{I} = \frac{1}{G} \quad (5.8)$$

The average duration of a transmission period equals to:

$$\overline{TP} = 1 + \overline{Y} + d \tag{5.9}$$

where $\overline{Y}$ is the expected value of *Y*. Since *Y* denotes the time at which the last interfering packet is scheduled, the probability of *Y* being smaller than some time *y* is the probability that no other packets (either new or retransmissions) are scheduled for transmission in an interval of duration *d-y*. With Poisson arrivals, the distribution for *Y* is:

$$\begin{aligned} F_Y(y) \underline{\Delta} Pr\{Y \le y\} &= Pr\{\text{no arrival occurs in an interval of length } d\text{-}y\} \\ &= e^{-G(d-y)} \end{aligned} \tag{5.10}$$

The average of *Y* is therefore given by:

$$\overline{Y} = d - \frac{1}{G}\left(1 - e^{-Gd}\right) \tag{5.11}$$

Combining the equations (5.1), (5.8), (5.9) and (5.11), we get:

$$\overline{t}_c = \overline{TP} + \overline{I} = 1 + \overline{Y} + d + \overline{I} = 1 + \left[d - \frac{1}{G}\left(1 - e^{-Gd}\right)\right] + d + \frac{1}{G} \tag{5.12}$$

$$S = \frac{\overline{U}}{\overline{t}_c} = \frac{P_{success}}{1 + \left[d - \frac{1}{G}\left(1 - e^{-Gd}\right)\right] + d + \frac{1}{G}} = \frac{G \cdot P_{success}}{G \cdot (1 + 2d) + e^{-Gd}} \tag{5.13}$$

The term $P_{success}$ in (5.7) is the only term that needs to be specified. Herefore we distinguish between four types of transmissions during a transmission period:

1. Successful transmission without conflict
2. Unsuccessful transmission without conflict
3. Successful transmission with conflict

4. Unsuccessful transmission with conflict

If the channel is used for the delivering of a successful packet during a *TP* then this is denoted by a successful transmission (situation 1 or 3). The difference between a transmission with or without conflict can be found in the number of packets transmissions during a *TP*. In case there is more than one packet transmission during a *TP*, we speak of a transmission with conflict. If there is only one packet transmission then the transmission is conflict-free. Keeping this in mind, we can divide $P_{success}$ up into two parts:

$$P_{success} = P_{success|noconflict} + P_{success|conflict} \tag{5.14}$$

In which $P_{success|noconflict}$ corresponds to situation 1 and $P_{success|conflict}$ to situation 3.

$$P_{success|noconflict} = e^{-Gd} P_{ps} \tag{5.15}$$

This is equal to the probability that no terminal transmits during the inhibit delay fraction *d* multiplied by the packet success probability P_{ps}. The multiplication with P_{ps} is necessary because of the assumption of the Rayleigh fading channel.

$$P_{ps} = \left(1 - P_{be|x=0}\right)^{L_p} \tag{5.16}$$

$P_{be|x=0}$ is the bit error probability in Rayleigh fading channel using the CDMA scheme and is caused by self interference due to multipath. L_p denotes the packet length; in our model the packet is 64 bits long. The calculation of $P_{success|noconflict}$ is not very complicated. However, this is not the case for $P_{success|conflict}$ which is the probability that the first packet has arrived successfully given a collision (between two or more packets).

$$P_{success|conflict} = \sum_{x=1}^{K-1} P_x(d) \cdot P\{first \quad packet \quad OK|x\} \tag{5.17}$$

in which:

K is the number of active users in the system

$P_x(d)$ is the probability of x arrivals during d for a Poisson distribution

x denotes the number of arrivals during d

Terminals within one group are supposed to be identical in every aspect. They have the same arrival rate, the same code and the same distance to their own transceiver. Notice that the distance between a terminal and the base station does not play a role. This can be explained as follows. If a terminal is allowed to transmit its data packet then it will do so using radio communication to deliver its packet to its transceiver. The transceiver on its turn forwards the received packet to the base station by wire. The base station only process the first incoming packet arriving by wire (see also Fig. 5.2). All other packets will be ignored by the base station.

Suppose that the network configuration in Fig. 5.2 is deployed. T_i denotes terminal number i and TR_j the transceiver number j. The data packet transmitted by T_i is denoted by p_i. For example, we adopt the following scenario. T_1 starts sending its data packet p_1 at time instant t and T_7 a fraction later, at $(t+\Delta t)$ in which $\Delta t \leq d$. The data packet p_1 arrives Δt earlier at its transceiver TR_1 than its competitor p_7 at TR_2. Consequently, TR_1 forwards its received packet Δt earlier to the base station than TR_2. The base station notices that TR_1 is the first active receiver and processes this packet. TR_2, TR_3 and TR_4 are all ignored. Even though the data packet p_7 is ignored by the base station, this packet causes interference for p_1. The radio transmitted packet p_7 not only arrives at TR_2 (its own receiver), but also at other receivers (TR_1, TR_3 and TR_4). Consequently TR_1 not only receives p_1, but also the interfering packet p_7. This interference is less severe than from the interference caused by T_2, T_3 or T_4. The explanation herefore is twofold. First, T_2, T_3 and T_4 all have the same code as T_1 and second, the received interference power (at TR_1) of T_2, T_3 or T_4 are higher than T_7 because of the shorter distance to TR_1.

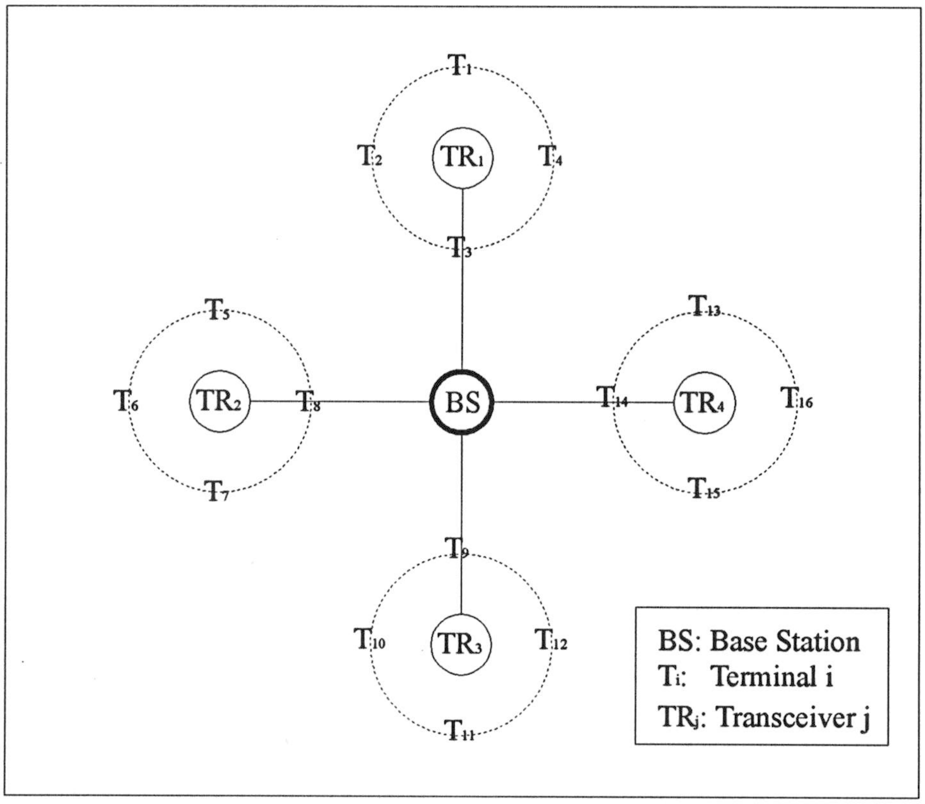

Fig. 5.2: Example of a 16-terminal network configuration

Without loss of generality, suppose that T_1 is the reference terminal. This terminal sends its data packet at an idle period first. Other terminals sending within the inhibit delay fraction interfere with the reception of the data packet p_1. The bit error rate depends on the configuration of transmitting terminals. $P\{first \quad packet \quad OK|x\}$ can then be evaluated as:

$$P\{first \quad packet \quad OK|x\} = E\left[1 - P_{be|\vec{x}}\right]^{L_p} \tag{5.18}$$

E[.] denotes the expectation value, and $\vec{x}$ a certain configuration of x. The total number of combinations for $\vec{x}$ equals $\binom{K-1}{x}$, in which $\binom{K-1}{x} = \frac{(K-1)!}{x!(K-1-x)!}$.

Because we assume an uniform distribution for all possible configurations we can write (K is the total number of active users in the system):

$$E\left[1-P_{be|\vec{x}}\right]^{L_p}=\frac{1}{\binom{K-1}{x}}\cdot\sum_{\vec{x}}\left(1-P_{be|\vec{x}}\right)^{L_p} \tag{5.19}$$

$$P_{success|conflict}=\sum_{x=1}^{K-1}P_x(d)\cdot\frac{1}{\binom{K-1}{x}}\cdot\sum_{\vec{x}}\left(1-P_{be|\vec{x}}\right)^{L_p} \tag{5.20}$$

$$P_{success}=e^{-Gd}\left(1-P_{be|x=0}\right)^{L_p}+\sum_{x=1}^{K-1}P_x(d)\cdot\frac{1}{\binom{K-1}{x}}\cdot\sum_{\vec{x}}\left(1-P_{be|\vec{x}}\right)^{L_p} \tag{5.21}$$

Combining the equations (5.13) and (5.21) finally gives:

$$S=\frac{G\left(e^{-Gd}\left(1-P_{be|x=0}\right)^{L_p}+\sum_{x=1}^{K-1}P_x(d)\cdot\frac{1}{\binom{K-1}{x}}\cdot\sum_{\vec{x}}\left(1-P_{be|\vec{x}}\right)^{Lp}\right)}{G(1+2d)+e^{-Gd}} \tag{5.22}$$

For large networks ($K \geq 32$), the calculation of the second term in the numerator can take long. If we take a closer look at this term we can distinguish two summations. For example, we take a 32 terminal network, and this simplified term looks like:

$$\sum_{x=1}^{31}\sum_{\binom{31}{x}}\cdots \tag{5.23}$$

We see that the value of x determines the duration of the calculations. Obviously, it is not possible to derive equation (5.23) for x = 1 up to 31. The total number of combinations will be over 4 billions! Fortunately, this is also not necessary. We can limit x to a certain extent. To explain this we have to return to equation (5.17). From (5.17) we see that $P_{success/conflict}$ depends on the multiplication of $P_x(d)$ with *P{first packet OK| x}*. Fig. 5.3 and 5.4 depict

$P_x(d)$ versus G with x and d as parameter respectively. $P_x(d)$ is derived from formula (5.6). For high values of x, $P_x(d)$ drops quickly. Because high values of x is identical to a high number of interferes, *P{first packet OK|x}* also drops quickly for increasing values of x. Therefore, the contribution in formula (5.17) for high values of x is negligible. Consequently, we can limit x to a certain value. There is a trade off between the calculation time and the accuracy of the calculation.

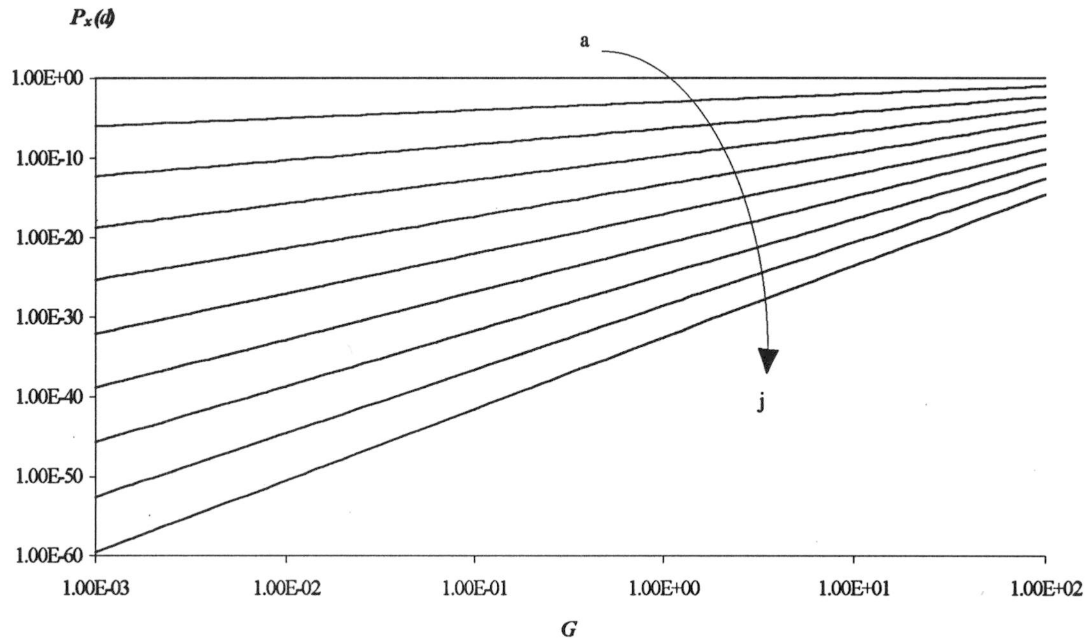

Fig. 5.3: $P_x(d)$ versus G with x as parameter; a..j: x=0, 1, 2, .., 8, 9; $d = 0.001$

We adopt the following strategy for the derivation of the throughput for the hybrid protocol. The throughput will be derived by formula (5.22) for increasing limit value of x. After a certain value, the throughput with x do not differ much from the throughput with (x-1) and the calculation time will last too long. Depending on the achieved results we can choose a reasonable and considered limit value of x. Section 5.3 deals with this aspect in more detail.

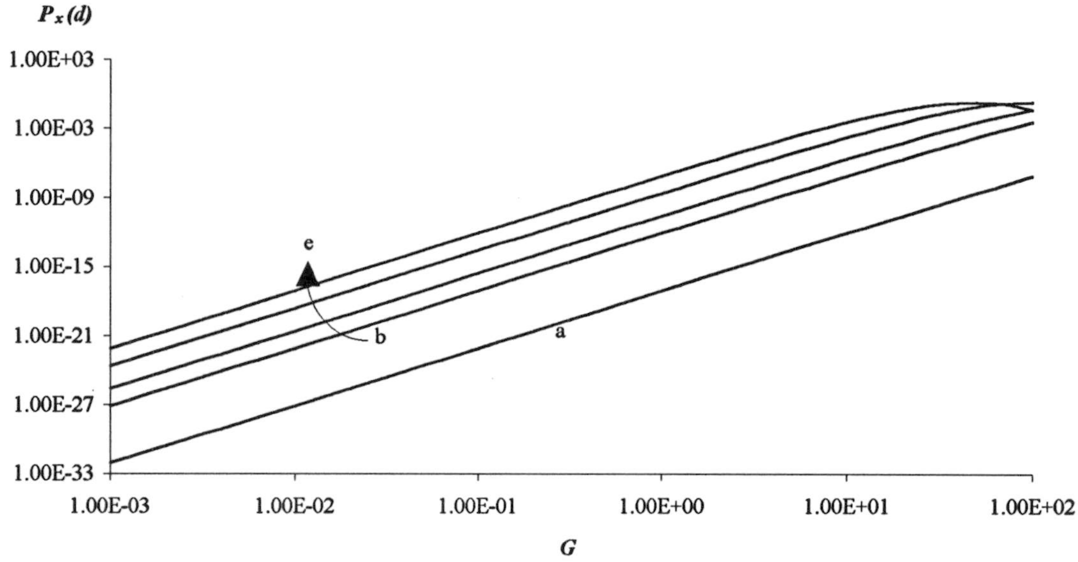

Fig. 5.4: $P_x(d)$ versus G with d as parameter; a..e: d=0.001, 0.01, 0.02, 0.05 and 0.1; $x = 5$

5.2 Delay

The average packet delay is defined as the duration between the transmission of the first bit till the correct reception of the last bit. For the calculation of the packet delay of the unslotted hybrid CDMA/ISMA protocol, we use the block diagram depicted in Fig. 5.5.

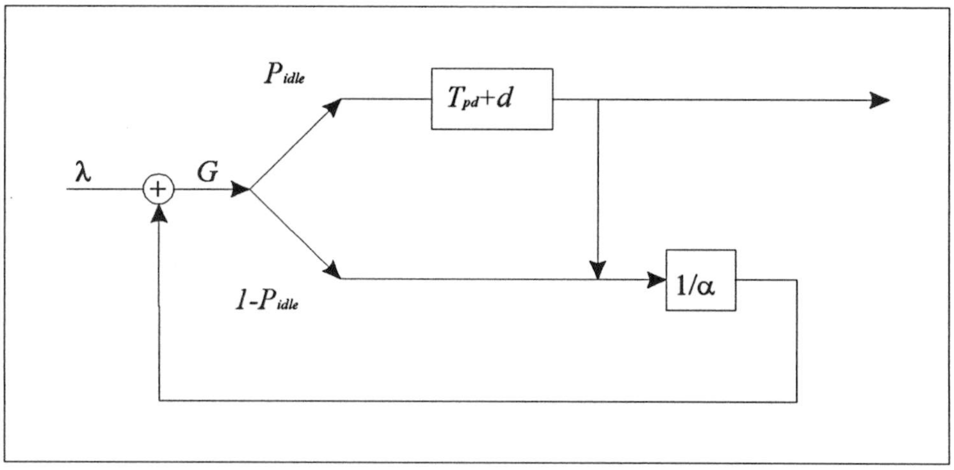

Fig. 5.5: Delay in the unslotted non-persistent hybrid CDMA/ISMA protocol

If a new packet arrives, the terminal immediately senses the channel to decide if it can transmit the packet. In case the channel is sensed busy (probability 1-P_{idle}), the packet is rescheduled for a later time instant. This random delay process is assumed to be negative exponentially distributed with parameter α. The average random delay time is then $1/\alpha$. The average number of times the packet has to suffer this delay is $G(1-P_{idle})/S$ (see Fig. 5.5). S is the throughput as given by (5.22).

On the other hand, if the channel is sensed idle (probability P_{idle}), the packet will be transmitted immediately. This transmission will take a packet duration T_{pd} plus the inhibit delay fraction d before the knowledge about the outcome of the transmission is available (see also Fig. 5.5). So, the total delay until the terminal knows whether the transmission was successful or not is $(T_{pd}+d)$. The transmission of a data packet does not necessarily result in a successful transmission; there are two possibilities: a successful or a failed transmission. If the transmission was successful, the packet leaves the system. The delay this packet suffers is $(T_{pd}+d)$. If the transmission was a failure, the packet again has to wait a random delay. In this case, the average delay is $\left(T_{pd}+d+\frac{1}{\alpha}\right)$ and the average number of schedulings is $\left(\frac{GP_{idle}}{S}-1\right)$.

Combining the above results, the average delay is finally given by equation (5.24).

$$D=\left(\frac{GP_{idle}}{S}-1\right)\left[T_{pd}+d+\frac{1}{\alpha}\right]+\frac{G(1-P_{idle})}{S}\cdot\frac{1}{\alpha}+\left(T_{pd}+d\right) \tag{5.24}$$

Again, because T_{pd} equals to 1, (5.24) can be rewritten as:

$$D=\left(\frac{GP_{idle}}{S}-1\right)\left[1+d+\frac{1}{\alpha}\right]+\frac{G(1-P_{idle})}{S}\cdot\frac{1}{\alpha}+(1+d) \tag{5.25}$$

S can be found in formula (5.22) and P_{idle} still has to be calculated. The average probability of sensing the channel idle is the average probability that the channel is idle and this probability equals to the average idle time ($\bar{I}$) divided by the average cycle length ($\bar{t}_c$).

$$P_{idle} = \frac{\bar{I}}{\bar{t}_c} = \frac{\frac{1}{G}}{\frac{1}{G} + \left\{1 + d - \frac{1}{G}\left(1 - e^{-Gd}\right) + d\right\}} = \frac{1}{G(1+2d) + e^{-Gd}} \tag{5.26}$$

5.3 Practical Implementation Aspect

Formula (5.22) can not be calculated exactly if we do not possess strong computing power. However, formula (5.22) allows us to approach the exact value very accurately in case we only have limited computing power.

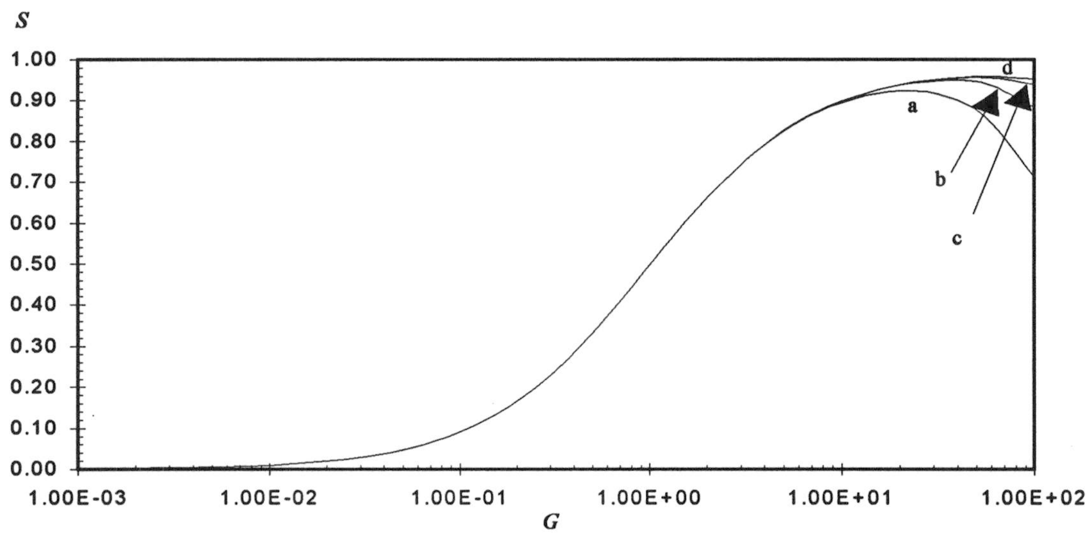

Fig. 5.6: Throughput versus offered traffic; a..d: x=1..4; α=0.1; d=0.01; code length=31; SNR=20 dB; number of codes=8; number of terminals=32

The influence of x in formula (5.22) on the results is depicted in Fig. 5.6 and 5.7. As explained from the previous section, x denotes the number of arrivals during the inhibit delay fraction d and determines for a great deal the calculation duration. From Fig. 5.6 and 5.7 in which the throughput and delay are depicted respectively, we can see that the protocol performance converges to a certain final level. (The delay time is normalised to the packet duration T_{pd}). The difference between x=3 and x=4 is hardly noticeable. We adopt x=4 as a reasonable value for our further analysis. However, for higher values of the offered traffic G, this difference increases. This can be explained by the fact that if the offered traffic increases, more packets arrive during the inhibit delay fraction.

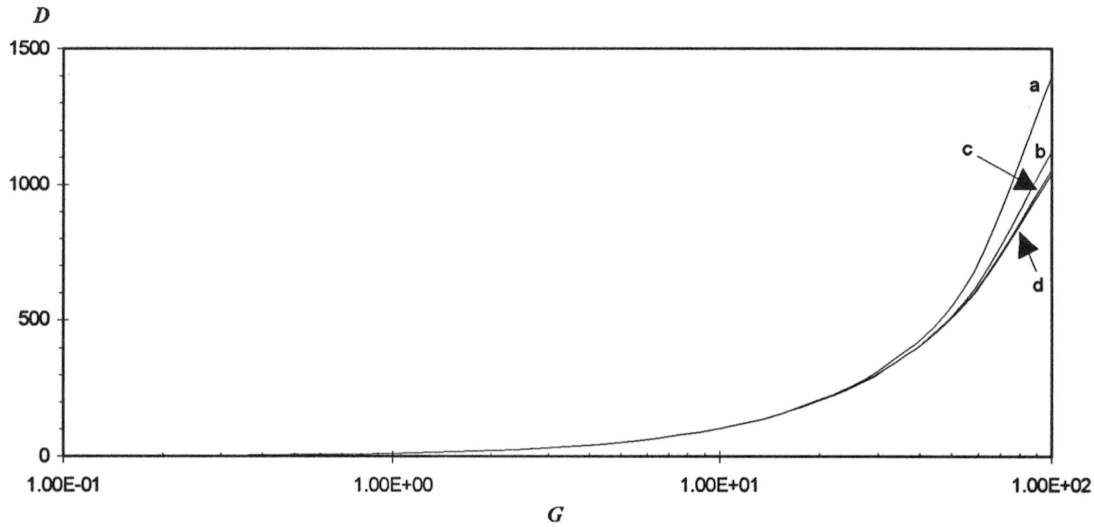

Fig. 5.7: Delay versus offered traffic; a..d: x=1..4; α=0.1; d=0.01; code length=31; SNR=20dB; number of codes=8; number of terminals=32

The mathematical analysis in this chapter is nice in the sense that it allows us to make a trade off between accuracy and computing time. If we do not have strong computing power or computing time, we are also able to obtain the performance result of the hybrid protocol. However, we have to live with a less accurate result. On the other hand, if accuracy if necessary, then we have to pay for strong computing power.

6. RESULTS

This chapter presents the results obtained from simulation and mathematical analysis as explained in the previous chapters. The protocol performance has been measured in terms of throughput S and delay D in relationship with the offered traffic G. The throughput is a measure of the efficiency of the protocol and is defined as the fraction of time in which correct data packets are received. The delay is the time between the arrival of the first bit of a data packet at a terminal and its arrival of the last bit at the destination. The delay is measured in terms of the packet duration.

In section 1, we first compare the performance between the slotted and unslotted protocol using simulation. Then, in section 2, we discuss the comparison between the simulation and mathematical model for the unslotted protocol. The conditions under which the results are valid can be found in section 2.6. If a parameter is changed, then it will be stated explicitly.

6.1 Comparison between Slotted and Unslotted Protocol

The performance shown in this section is generated by simulation. Fig. 6.1a and b compare the performance of the slotted and unslotted hybrid protocol using computer simulation. In these figures, we also show the influence of P_{tr} on the performance of the slotted protocol. P_{tr} is one of the most important parameters for the slotted protocol and is described in paragraph 2.4.1 as the probability that the packet will actually be transmitted in case the channel is sensed *free*. As we can observe, for increasing value of P_{tr}, the throughput improves. The declaration for this observation can be found by the use of direct sequence CDMA. If only pure ISMA is used, than it is to be expected that for high traffic, the increase in P_{tr} leads to a degradation of the protocol's performance. High values of P_{tr} means a high number of simultaneouly transmitted data packets. Collision between those packets results in corruption of all data packets. However, because CDMA is also used, there is still a probability that the first incoming packet can be received correctly.

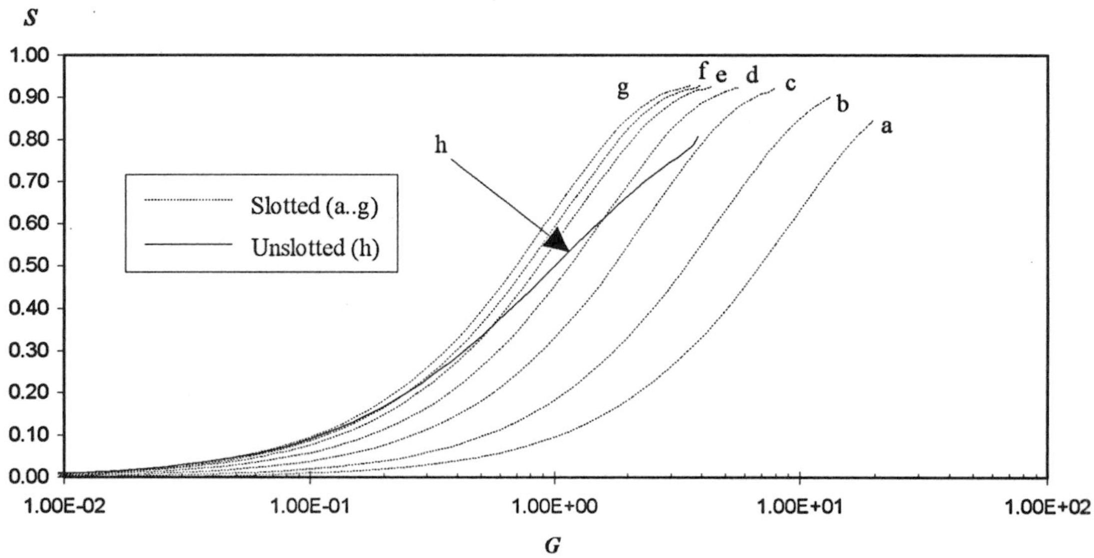

Fig. 6.1a: Throughput versus offered traffic using simulation of slotted (a..g: P_{tr} =0.1, 0.2, 0.4, 0.6, 0.8, 0.9 and 1) and unslotted (h: d=0.01) hybrid protocol; 8 codes

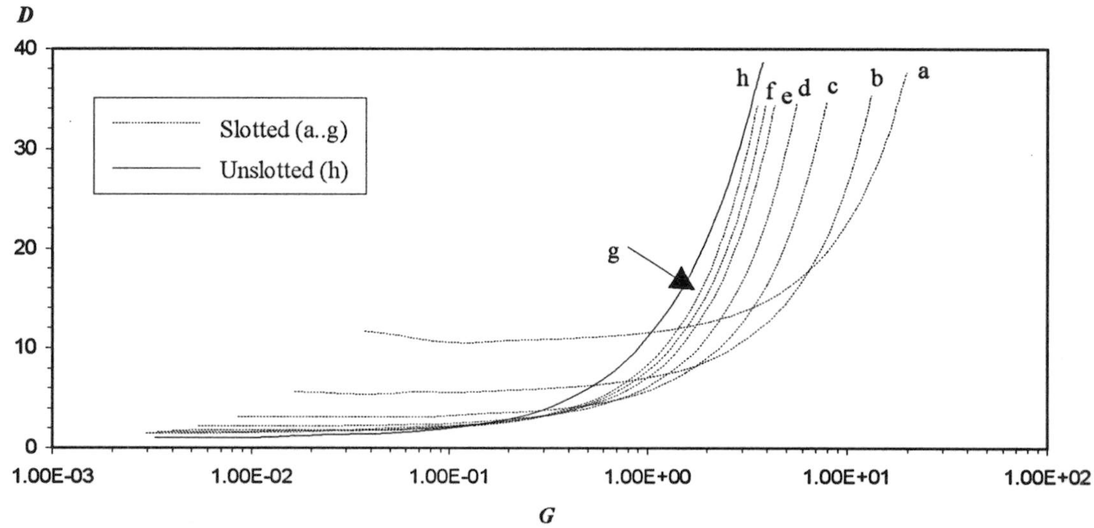

Fig. 6.1b: Delay versus offered traffic using simulation of slotted (a..g: P_{tr} =0.1, 0.2, 0.4, 0.6, 0.8, 0.9 and 1) and unslotted (h: d=0.01) hybrid protocol; 8 codes

For the delay, there is one interesting observation we can make. If we look at low offered traffic, then the delay for low P_{tr} is high. For example, the average delay for P_{tr}=0.1 is about 10 packet durations. This is not surprising because the probability of actually transmitting a packet if the channel is sensed idle equals to 0.1. So, on average, a packet has to wait 10 packet durations before it can be transmitted. On the other hand, if we look at high offered traffic, the delay is lower for low values of P_{tr}. For example, take a look at the delay curves of P_{tr}=0.1 and 0.2 (curves a and b of Fig. 6.1b). For an offered traffic below 8 packets/(packet duration), the delay for P_{tr}=0.1 is higher than for P_{tr}=0.2. For an offered traffic beyond 8 packets/(packet duration) it is the other way round. So, if we have to choose a slotted scheme, then the region of the offered traffic in which we are, decides of value for the parameter P_{tr}. If the offered traffic is low, than a slotted ISMA protocol with a high value of P_{tr} is better.

The throughput of the unslotted protocol is better than the throughput of the slotted protocol for low values of the offered traffic. For increasing values of the offered traffic, the throughput of the unslotted protocol starts to drop below the throughput of the slotted protocol. This is because for high offered traffic, the slotted protocol organises its data stream better than the unslotted protocol. Packets are only allowed to be transmitted at the start of a time slot. Furthermore, the slotted protocol with low values of P_{tr} handles high offered traffic quite well, because it limits of the contention in the channel more profoundly. For low traffic, the delay of the unslotted protocol is, as expected, better than the slotted protocol. For high traffic, the situation is the other way round.

Fig. 6.2 shows the influence of code sharing on the performance of the slotted and unslotted protocol. P_{tr} plays an important role in the slotted scheme. This is the same for d with respect to the unslotted scheme. The slotted protocol is simulated for P_{tr}=0.9. As already noted in section 2.6, the inhibit delay fraction d equals to 0.01 for the unslotted protocol.

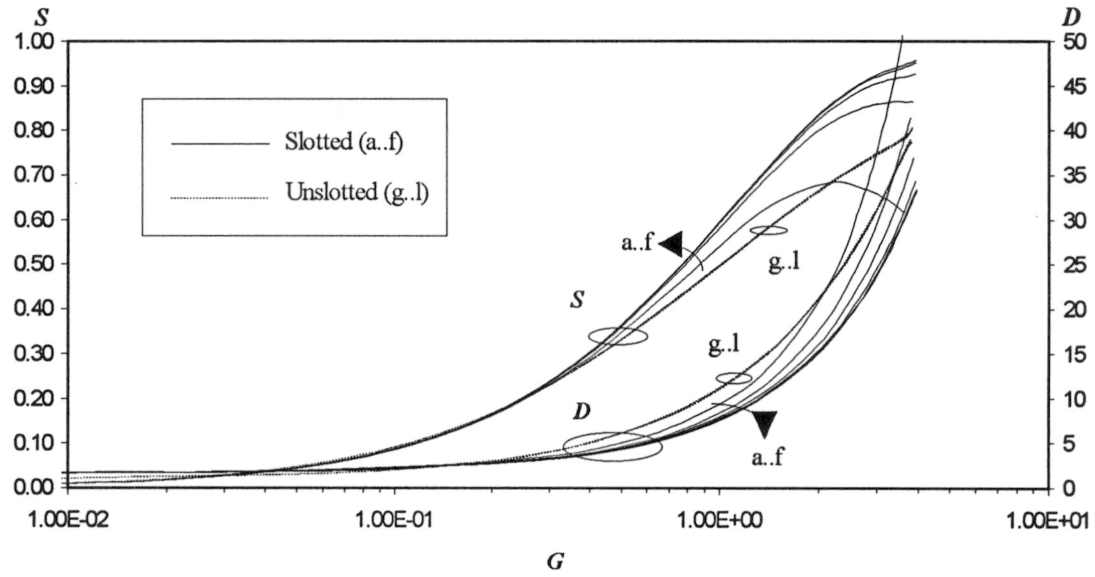

Fig. 6.2: Throughput and delay versus offered traffic using simulation for the slotted (P_{tr}=0.9) and unslotted (d=0.01) protocol, varying the number of codes a..f: g..l: number of codes: 1, 2, 4, 8, 16 and 32.

6.2 Comparison between Simulation and Mathematical Analysis

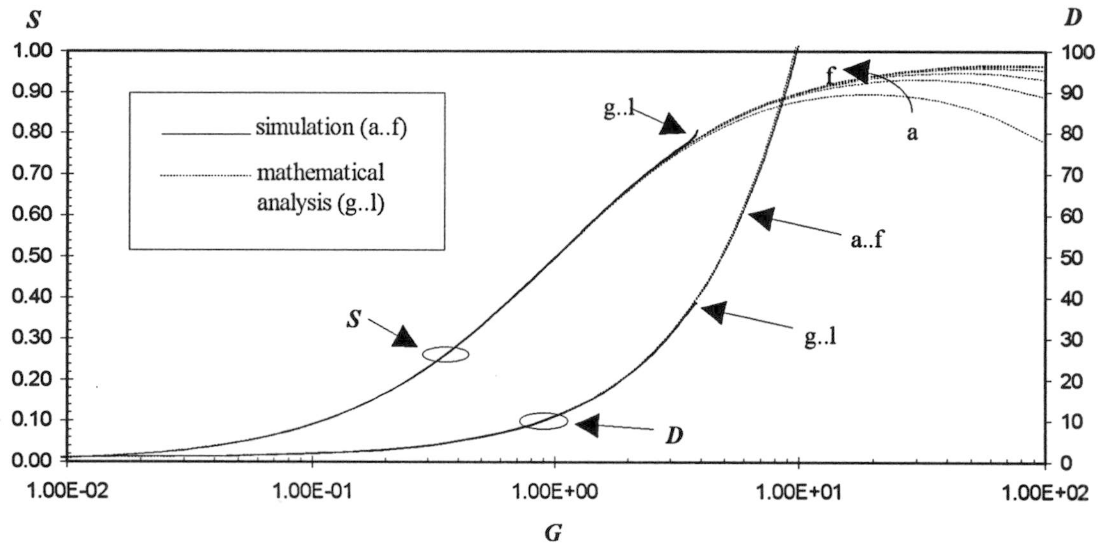

Fig. 6.3: Comparison between simulation and mathematical analysis for the unslotted protocol. a..f, g..l: 1, 2, 4, 8, 16, 32 codes

The comparison between the simulation and mathematical model for the unslotted protocol is given in Fig. 6.3. The performance of simulation does not exceed for $G \approx 4$ packets/(packet duration). This is because in case of simulation, the arrival rate at terminals is the input parameter and G is one of the output parameters. The results due to simulation and mathematical model are in good agreement. This graph also shows the effect of code sharing. For a 32-terminal network, it is sufficient to use only four codes. If we take a closer look at the simulation results, it seems like code sharing has no influence at all on the performance. Fig. 6.4a and b show the influence of the number of codes used on the performance of the unslotted protocol for poor parameters. In contrast to Fig. 6.3 where good parameters are chosen, Fig. 6.4a and b show that the number of codes used do influence protocol performance. The more codes, the better the protocol performs. However, code sharing is a very good option. The differences between 8 and 32 codes is much lower than the difference between 1 and 32 codes.

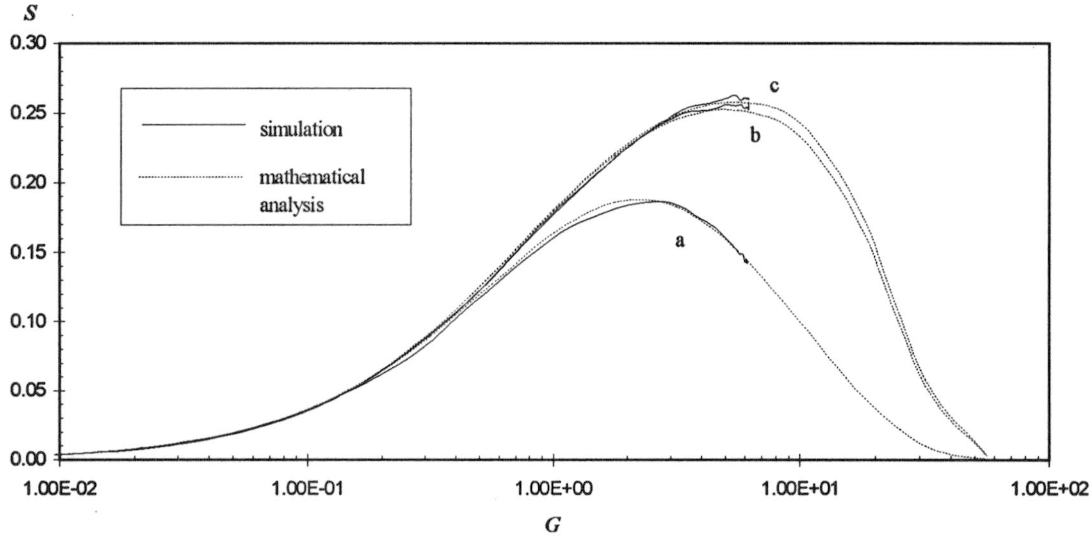

Fig. 6.4a: Throughput comparison between simulation and mathematical analysis for the unslotted protocol and varying the number of codes used; a..c: number of codes=1, 8 and 32; d=0.2; α=0.2; SNR=6 dB; code length = 31

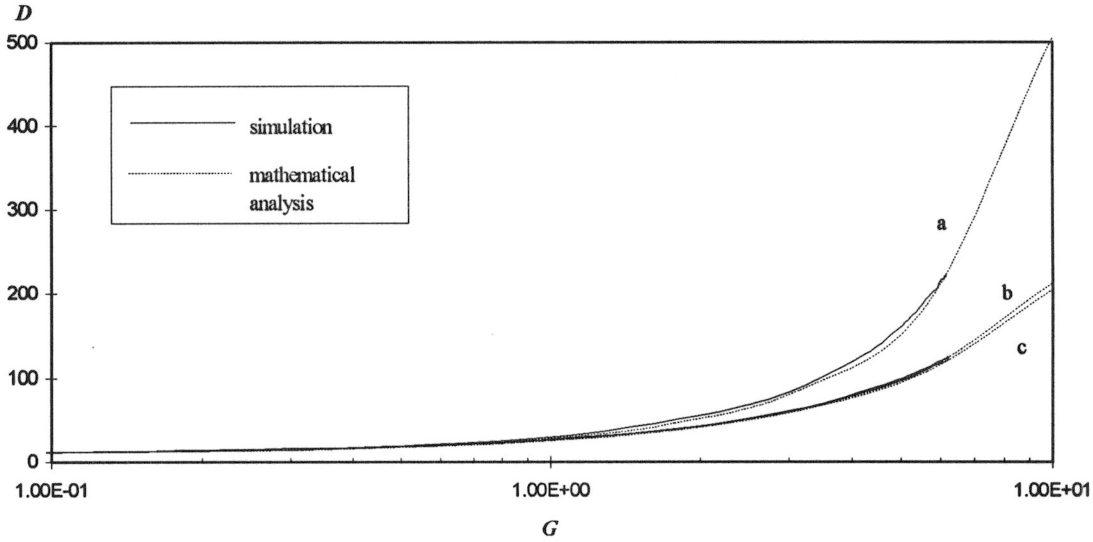

Fig. 6.4b: Delay comparison between simulation and mathematical analysis for the unslotted protocol and varying the number of codes used; a..c: number of codes=1, 8 and 32; d=0.2; α=0.2; SNR=6 dB; code length = 31

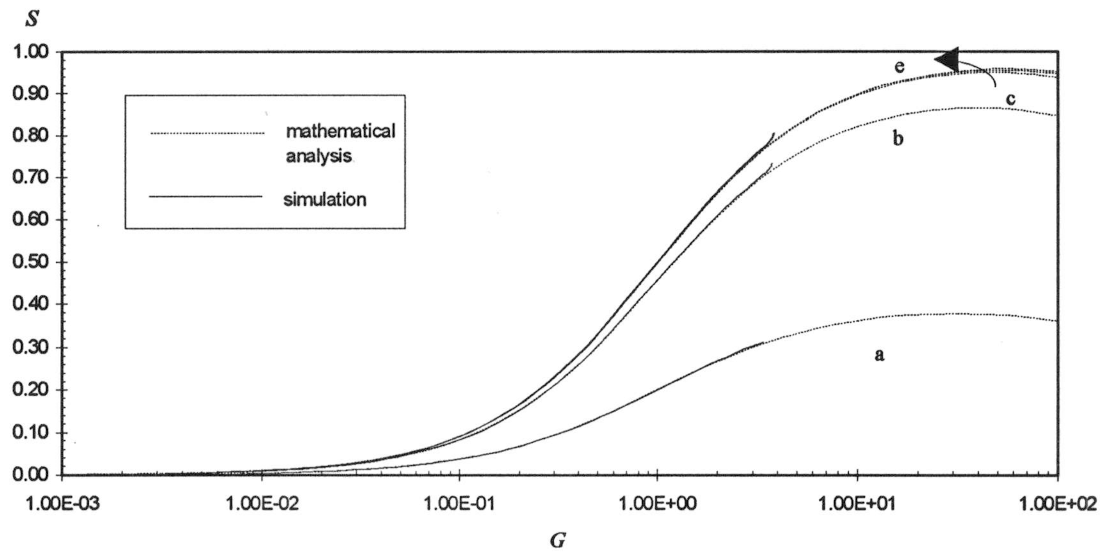

Fig. 6.5a: Throughput comparison between simulation and mathematical analysis for the unslotted protocol and varying SNR; a..e: SNR=6, 8, 10, 12 and 20 dB; number of codes=8

Fig. 6.5a and b show the effect of the signal to noise ratio (SNR) on the performance of the unslotted protocol. Of course, the throughput and delay improve for increasing SNR.

However, Fig. 6.5a and b also show that a SNR of 10 dB suffice. For values of SNR higher than 10 dB, the protocol performance also improves, but only for high values of the offered traffic and the improvement is only additional. Again, simulation and mathematical analysis agree very well with each other.

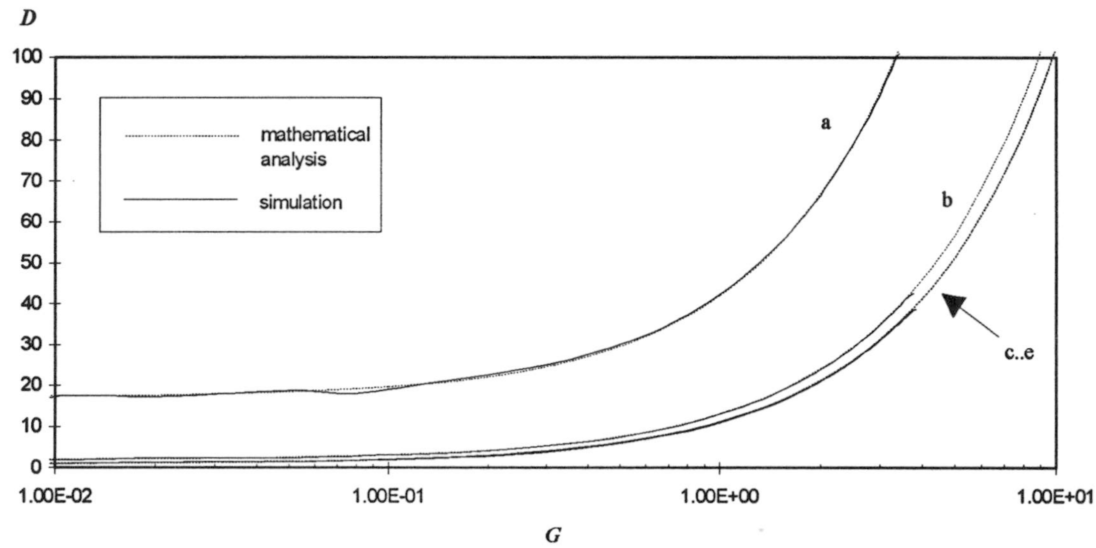

Fig. 6.5b: Delay comparison between simulation and mathematical analysis for the unslotted protocol and varying SNR; a..e: SNR=6, 8, 10, 12 and 20 dB; number of codes=8;

Fig. 6.6a and b show the effect of the inhibit delay fraction *d* on the performance of the unslotted protocol. For high values of *d*, the performance degrades tremendously. This is due to the increase of the number of data packets colliding during this period. This is why the ISMA protocol is not useful for environment in which the propagation delay is high.

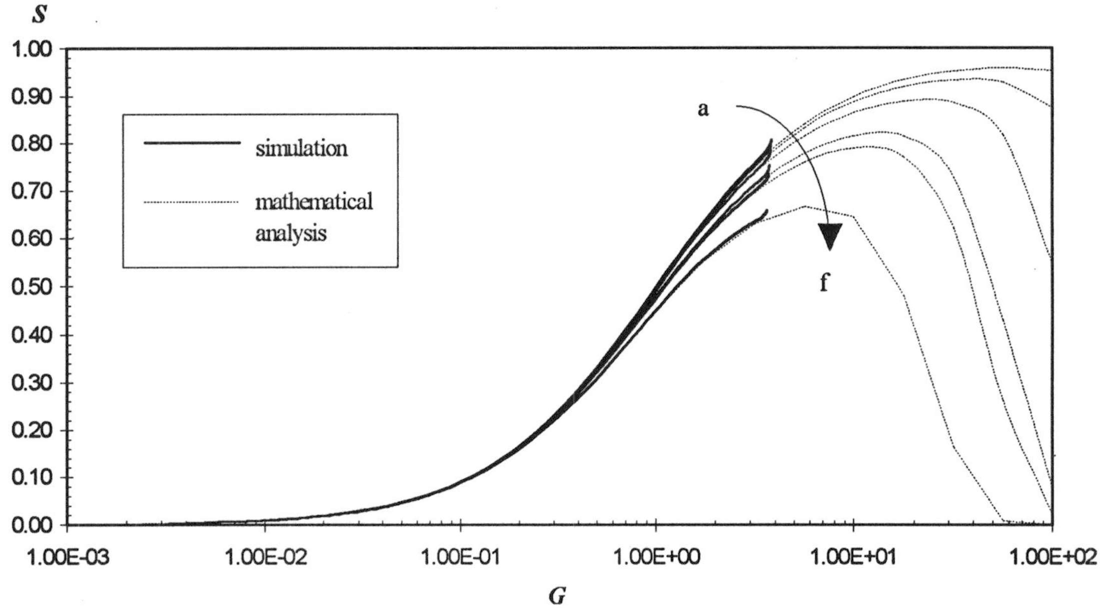

Fig. 6.6a: Throughput versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying the inhibit delay fraction *d*. a..f: *d*=0.01, 0.02, 0.04, 0.08, 0.1 and 0.2; 8 codes

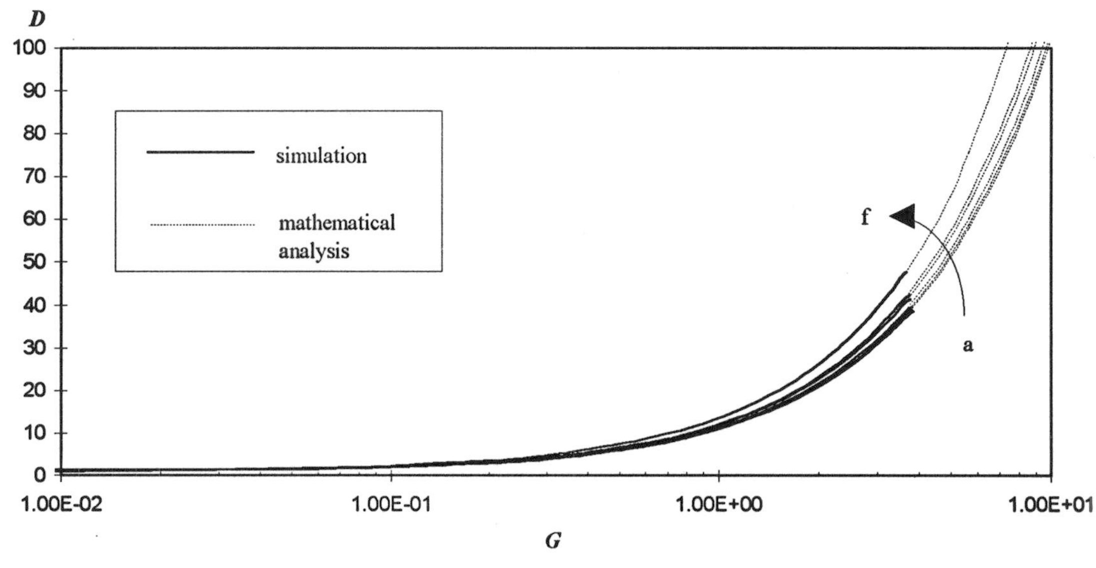

Fig. 6.6b: Delay versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying the inhibit delay fraction *d*; a..f: *d*=0.01, 0.02, 0.04, 0.08, 0.1 and 0.2; 8 codes

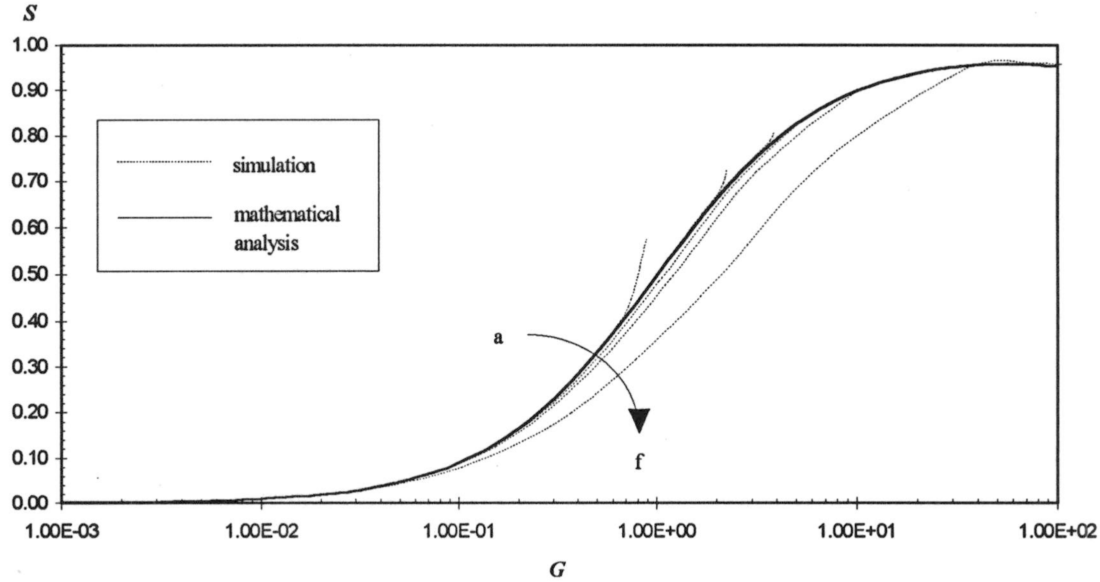

Fig. 6.7a: Throughput versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying α, the random delay parameter; a..f: α=0.01, 0.05, 0.1, 0.5, 1 and 5; 8 codes

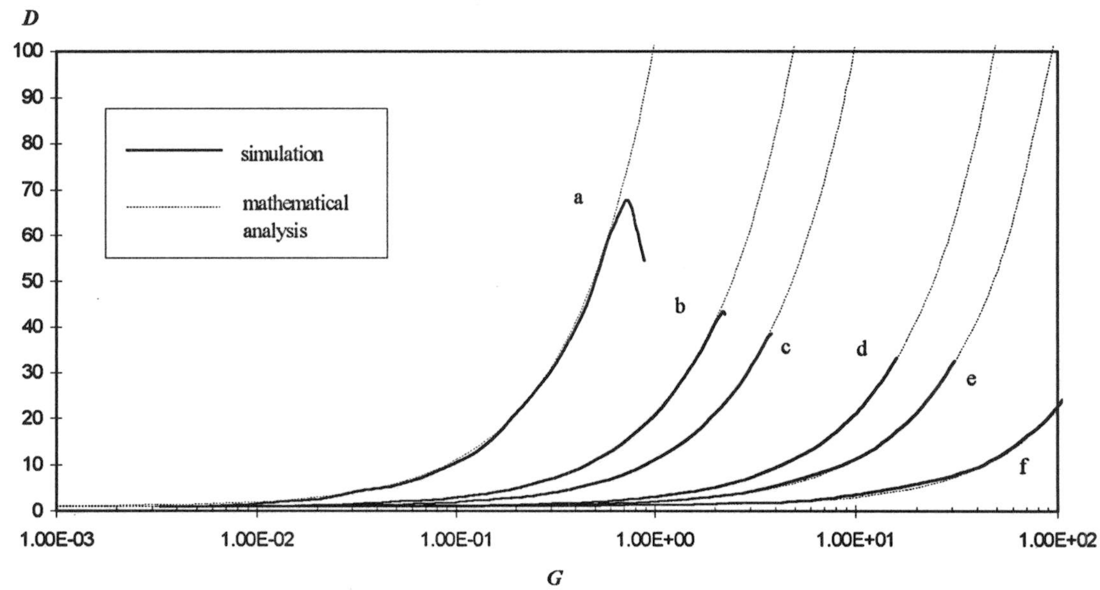

Fig. 6.7b: Delay versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying α, the random delay parameter; a..f: α=0.01, 0.05, 0.1, 0.5, 1 and 5; 8 codes

Fig. 6.7a and b show the effect of α, the parameter for the random delay mechanism, on the performance of the unslotted protocol. This parameter is explained in paragraph 4.2.3. If a packet has to be retransmitted, than on average it has to wait $1/\alpha$ before it can try again. So, for low values of α, the delay increases tremendously. This can be observed in Fig. 6.7b. In case of the mathematical model, the throughput in Fig. 6.7a does not show much variation if the parameter α is changed. Fig. 6.7a and b also show strange bendings at the end of simulations. We try to depict this phenemenon more clearly by showing the delay versus throughput in Fig. 6.8.

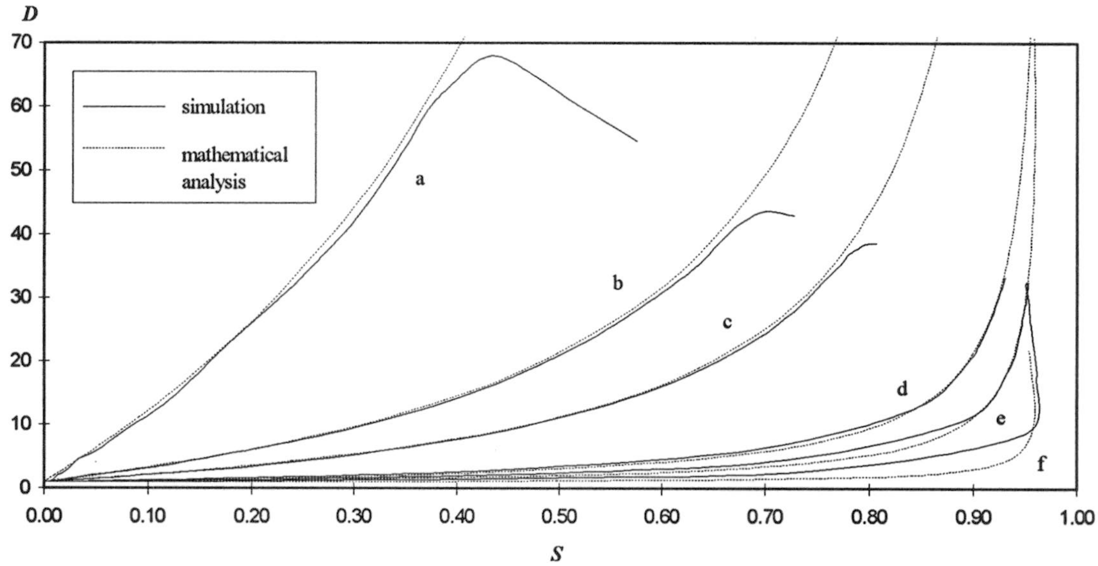

Fig. 6.8: Delay versus throughput using simulation and mathematical analysis of unslotted protocol, varying α, the random delay parameter; a..f: α=0.01, 0.05, 0.1, 0.5, 1 and 5; 8 codes

Fig. 6.8 depicts the delay-throughput characteristics of the unslotted protocol for varying values of α. If the throughput increases, then the delay also increases. For low values of α and especially at the end of the curves, the computer simulation does not agree with the mathematical analysis. Especially for curve a, where the delay even decreases tremendously at the end. It is very likely that this is caused by the abrupt program termination and the low value of α. A low value of α means that the random delay is large. This means that the collection of statistics has been done while most of the terminals still have to wait a random delay.

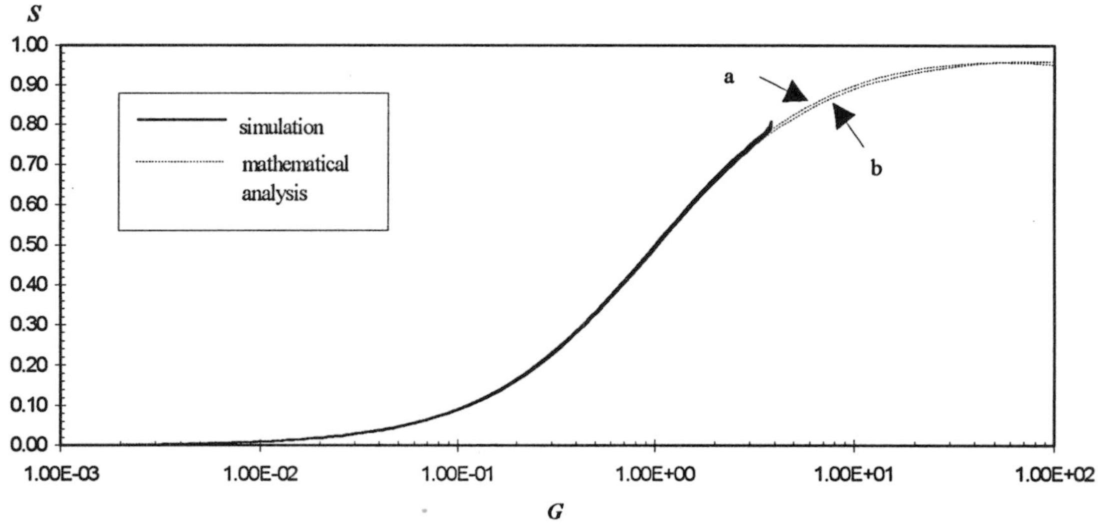

Fig. 6.9a: Throughput versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying the code length; a..b: code length = 31 and 127; 8 codes

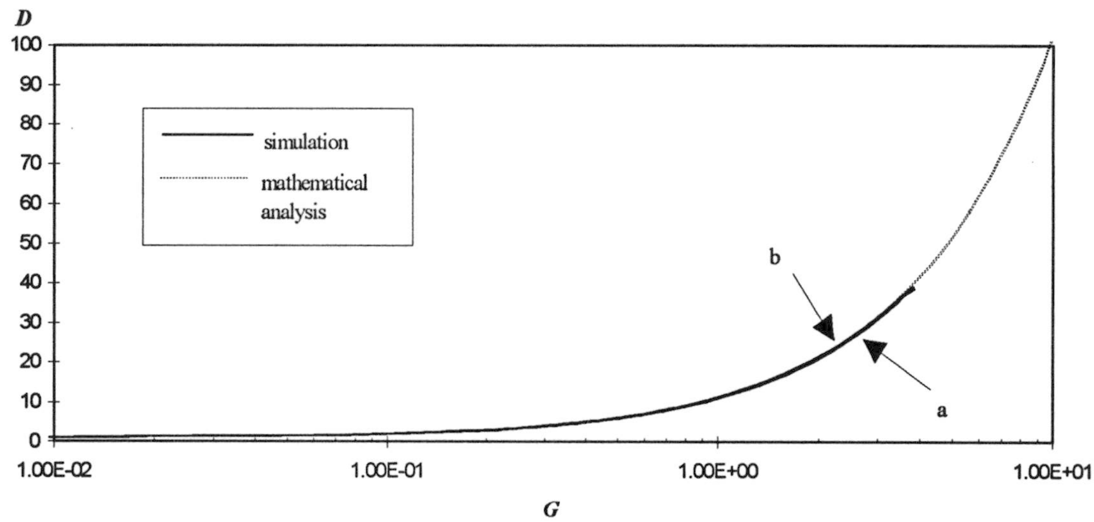

Fig. 6.9b: Delay versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying the code length; a..b: code length = 31 and 127; 8 codes

Fig. 6.9a and b show the effect of code length on the performance of the unslotted protocol. At first sight, it is surprisingly to notice that there is hardly a difference between the performance if the code length is altered. We would expect that the performance of a longer code is better than a shorter one. This is the case for good parameters.

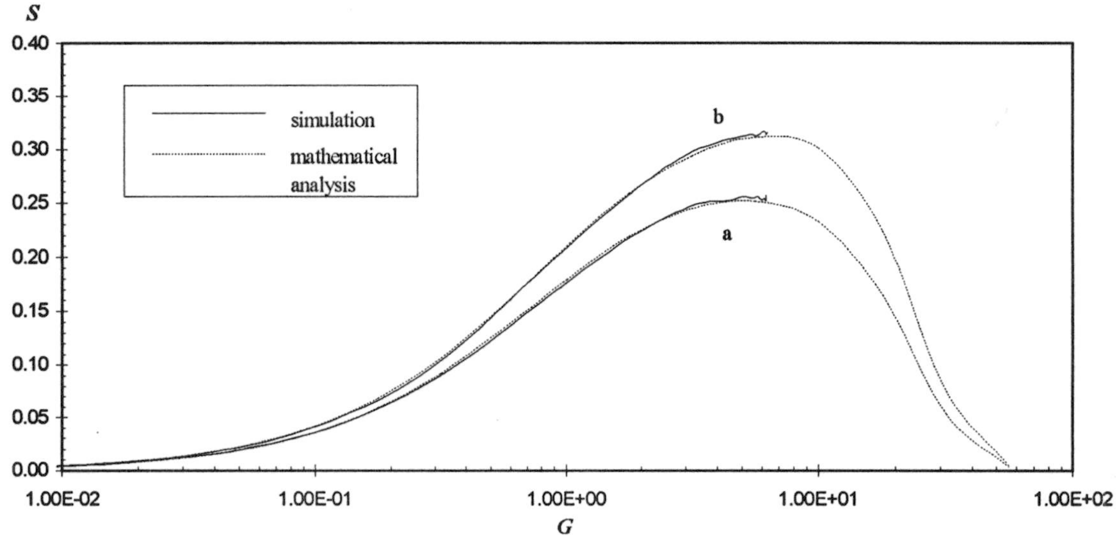

Fig. 6.10a: Throughput versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying the code length; a..b: code length = 31 and 127; 8 codes; d=0.2; α=0.2; SNR=6 dB

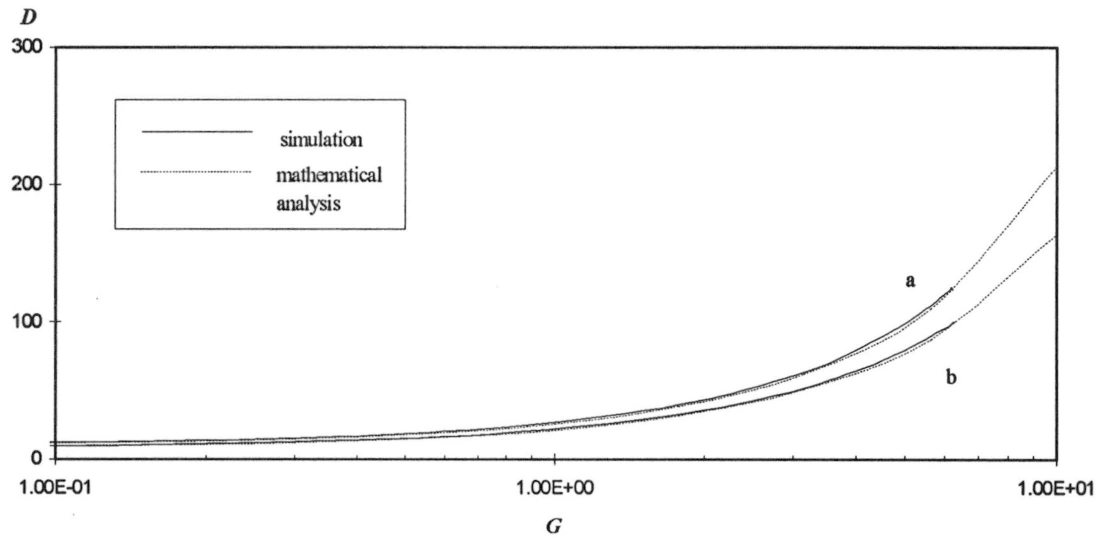

Fig. 6.10b: Delay versus offered traffic using simulation and mathematical analysis of unslotted protocol, varying the code length; a..b: code length = 31 and 127; 8 codes; d=0.2; α=0.2; SNR=6 dB

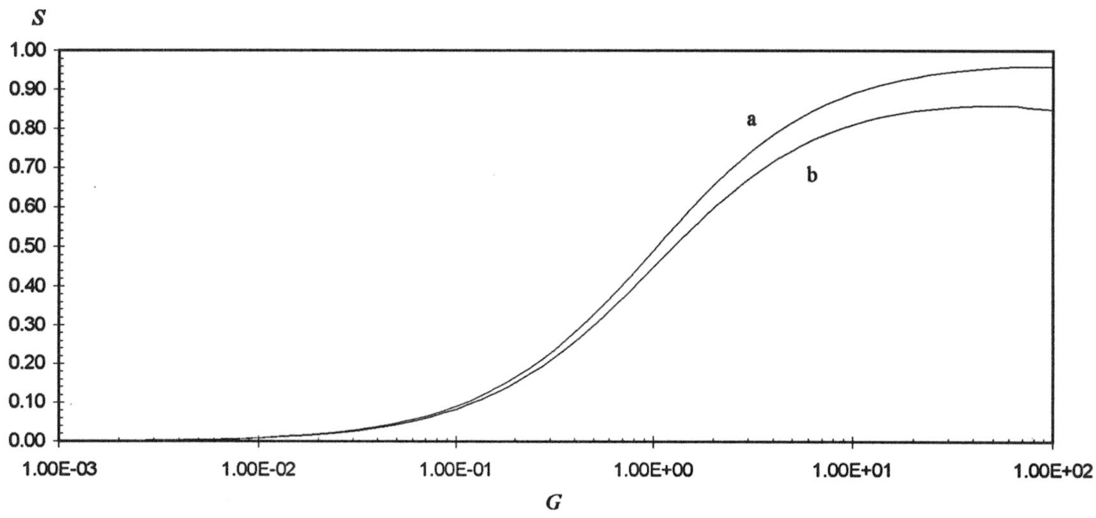

Fig. 6.11a: Throughput versus offered traffic using mathematical analysis of unslotted protocol, varying the code length; a: code length =127 and bit rate = 256*1024; b: code length = 31 and bit rate = 256*1024*4; 8 codes

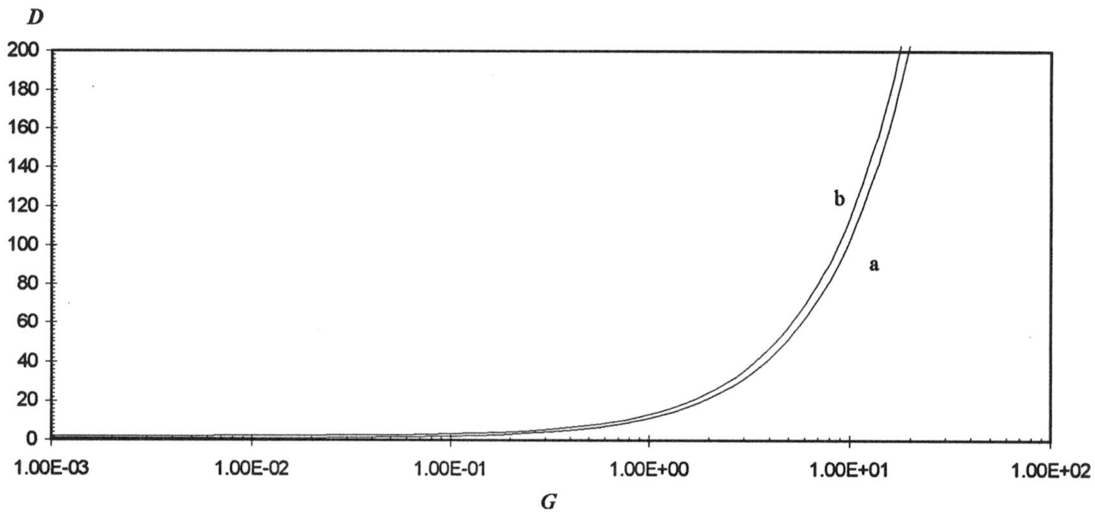

Fig. 6.11b: Delay versus offered traffic using mathematical analysis of unslotted protocol, varying the code length; a: code length =127 and bit rate = 256*1024; b: code length = 31 and bit rate = 256*1024*4; 8 codes

In case the parameters are poor, we do see differences in the performance if the code length is altered (Fig. 6.10a and b). Longer codes perform better than shorter codes because they have better cross correlations characteristics. However, longer codes need a larger bandwidth. If we keep the bandwidth fixed, and we increase the code length, than we have to decrease the bit rate. This is illustrated in Fig. 6.11a and b. Here, we see that the performanee is better in case the code length is longer.

7. CONCLUSIONS AND RECOMMENDATIONS

7.1 Conclusions

The unslotted hybrid CDMA/ISMA protocol has been investigated. The protocol performance is measured in terms of throughput S and delay D in relationship with the offered traffic G. Herefore, we have been able to derive a close form formula for the throughput and delay. Also a simulation program has been developed to compare the results. The following conclusions can be drawn from the previous chapters:

- First of all it can be concluded that the results from simulation and mathematical model agree very well with each other. The mathematical model is very suitable for analysing the hybrid protocol even when strong computing power is not available. This mathematical model allows a trade off between accuracy and computing power or computing time.

- Comparison between the slotted and unslotted protocol shows that for low offered traffic the delay for the unslotted protocol is better than the slotted protocol. For high traffic it is the other way round.

- The most important parameter for the slotted p-persistent scheme, P_{tr}, has a great influence on the protocol's performance. The choice of this parameter depends on the offered traffic the system generates.

- The hybrid protocol supports code sharing extremely well. For a 32 terminal network, only four codes suffice.

- Longer codes do not perform better than shorter codes if the system is under good conditions (the parameters are good). On the other hand, if the parameters are poor, then longer codes can indeed be used to improve the protocol's performance.

- The retransmission parameter α influences the average packet delay for a great deal.

- If the SNR increases, the performance also improves. However, beyond 10 dB, the performance hardly improves anymore.

- The inhibit delay fraction d plays an important role in the unslotted non-persistent ISMA scheme. It is concluded that the proposed unslotted hybrid protocol performs very well in case the propagation delay is not a concern (d is small). If the propagation delay is too high (d is large) the performance drops quickly.

7.2 Recommendations

- The system model should be extended with buffers. How does this feature influence the protocol's performance?

- Suppose the line of sight path is present. With other words, if the channel is modeled as a Rician fading channel, how does the protocol then perform? And does the Rayleigh fading channel describes our indoor channel sufficiently?

- The offered traffic is assumed to be a Poisson process. This is only valid if the number of terminals is large. The investigation of our system is only restricted to 32 terminals because we lack computer power and simulation time. Future research should pay attention to this matter.

- Error corrections can be included to improve the protocol's performance.

- The hybrid protocol composes of CDMA and ISMA. What about other combinations? How about CDMA and TDMA? We also can think of a protocol in which for low offered traffic, the protocol behaves like a random access protocol and for high offered traffic the protocol has a contentionless characteristic.

- Enhance the analysis of the protocol with the feedback channel and the acknowledgements to see how these aspects influence the protocol performance.

REFERENCES

[1] R. Prasad,
"Performance Analysis of Mobile Packet Radio Networks in Real Channels with Inhibit-Sense Multiple Access",
IEE Proc. I, Vol. 138, pp. 458-464, October 1991.

[2] I. Widipangestu, A.J. 't Jong and R. Prasad,
"Capture Probability and Throughput Analysis of Slotted ALOHA and Unslotted np-ISMA in a Rician/Rayleigh Environment",
IEEE Transactions on Vehicular Technology, Vol. 43, No. 3, pp. 457-465, August 1994.

[3] K.J. Zdunek, D.R. Ucci and J.L. Locicero,
"Throughput of Nonpersistent Inhibit Sense Multiple Access with Capture",
Electronics Letters, Vol. 25, No. 1, pp. 30-31, 5th January 1989.

[4] R. Prasad, and C.Y. Liu,
"Throughput Analysis of Some Mobile Packet Radio Protocols in Rician Fading Channels",
IEE Proceedings-I, Vol. 139, No. 3, pp. 297-302, June 1992.

[5] M.G. Jansen, and R. Prasad,
"Capacity, Throughput, and Delay Analysis of a Cellular DS CDMA System With Imperfect Power Control and Imperfect Sectorization",
IEEE Transactions on Vehicular Technology, Vol. 44, No. 1, pp. 67-75, February 1995.

[6] R.A. Scholtz,
"The Origins of Spread Spectrum Communications",

IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 822-854, May 1982.

[7] W.C.Y. Lee,
"Overview of Cellular CDMA",
IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 291-302, May 1991.

[8] M.B. Pursley,
"Performance Evaluation of Phase Coded Spread Spectrum Multiple Access Communication - Part I: System Analysis",
IEEE Transactions on Communications, Vol. COM-25, No. 8, pp. 795-799, August 1977.

[9] R.L. Pickholtz, D.L. Schilling and L.B. Milstein
"Theory of Spread Spectrum Communications - A Tutorial",
IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 855-884, May 1982.

[10] K.S. Gilhousen, I.M. Jacobs, R. Padovani, A.J. Viterbi, L.A. Weaver, Jr. and C.E.Wheatley III,
"On the Capacity of a Cellular CDMA System",
IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 303-312, May 1991.

[11] H. van Roosmalen, J. Nijhof and R. Prasad,
"Performance Analysis of a Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications",
IEEE Journal on Selected Areas in Communications, Vol. 12, No. 5, pp. 909-916, June 1994.

[12] H. van Roosmalen,
"Performance Analysis of a Hybrid ISMA/CDMA Protocol for Indoor Wireless Communication",
Thesis report, Delft University of Technology, the Netherlands, August 1992

[13] C. van den Broek,
"Multiple Access Techniques with a special interest in Code Division Multiple Access",
AIO report Delft University of Technology, the Netherlands, January 1992

[14] R. C. Dixon
Spread Spectrum Systems
John Wiley & Sons, 2nd edition, 1984

[15] A. B. Carlson,
Communication systems,
Mc. Graw Hill Book Company, 3rd edition, 1988, pp. 92.

[16] M. Kavehrad and B. Ramamurthi,
"Direct -Sequence Spread Spectrum with DPSK Modulation and Diversity for Indoor Wireless Communications",
IEEE Transactions on Communications, Vol. COM-35, No. 2, pp. 224-236, February 1987.

[17] H. Hashemi,
"The Indoor Radio Propagation Channel"
Proceedings of the IEEE, VOL. 81, No. 7, pp. 943-966, Juli 1993

[18] F. Schoute,
Prestatie-Analyse van Telecommunicatiesystemen,
Kluwer Technische Boeken B.V., 1989

[19] I. Mitrani,
Simulation Techniques for Discrete Event Systems,
Cambridge: Cambridge University Press, 1982

[20] L. Kleinrock and F.A. Tobagi,
"Packet Switching in Radio Channels: Part I- Carrier Sense Multiple Access and their Throughput-Delay Characteristics",
IEEE Trans. on Commun., Vol. COM-23, No. 12, pp. 1400-1416, December 1975.

APPENDIX A:

"Unslotted Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications"

Proceedings IEEE Third Symposium on Communications and Vehicular Technology in the Benelux, October 25-26 1995, Eindhoven, The Netherlands, pp. 68-75

Unslotted Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications

Huy Linh Anh Le, *Student Member, IEEE**
Huub van Roosmalen**
Jos Nijhof, *Member, IEEE**
Ramjee Prasad, *Senior Member, IEEE**

*Telecommunications and Traffic Control Systems Group
Delft University of Technology
P.O. Box 5031, 2600 GA Delft
The Netherlands
Tel.: +31 15-78.23.86 (before 10-10-'95)
+31 15-278.24.17 (after 10-10-'95)
Fax: +31 15-78.17.74
E-Mail: Huy@Octopus.et.tudelft.nl

**Holland Institute of Traffic Technology B.V.
P.O. Box 245
7300 AE Apeldoorn
The Netherlands
Fax: +31 55-432553

ABSTRACT **- A hybrid Code Division Multiple Access / Inhibit Sense Multiple Access (CDMA/ISMA) protocol has been proposed as an effective multiple access scheme for Indoor Wireless Computer Communications. This new protocol combines the advantages of both CDMA and ISMA into one protocol. On the one hand the ISMA protocol introduces a limitation to the number of simultaneous accesses to the transmission channel. On the other hand the CDMA protocol introduces an improvement to the packet survival chance. *Slotted* hybrid CDMA/ISMA protocol has been reported in [1]. It is shown that the performance of the hybrid protocol is indeed better than CDMA only. In addition, code sharing can be applied to reduce hardware cost. This paper presents the performance analysis of the *unslotted* CDMA/ISMA protocol in order to take more advantage of the strength of the hybrid protocol. The performance comparison between the slotted and unslotted hybrid CDMA/ISMA protocol is evaluated in terms of throughput and delay using computer simulation and mathematical analysis.**

I. INTRODUCTION

Indoor wireless office communication is our main research field. The office system we focus on consists of a building in which users work together in groups. The participants generate terminal traffic. Terminals communicate with each other by radio transmission using a random access protocol. Terminals might not detect each other's transmission in radio communications. It can easily happen that two users are hidden from each other by some obstacle, in which case severe performance degradation results. This is called the *hidden terminal problem.* The introduction of a central base station can alleviate this problem by instructing it to send a busy tone to all participating terminals to forbid new transmissions when a transmission is going on. Still, a situation can occur in which two or more terminals simultaneously start their transmission, resulting in a collision. However, a great reduction in the number of simultaneous transmissions is achieved by the introduction of a central base station. This concept is called ISMA [1]-[4].

If we could somehow increase the survival chances of colliding packets, we could improve protocol performance. The near-far effect is one way to achieve this. A packet may 'capture' the receiver if it is much stronger than its competitors. This can happen when terminals use the same transmission power, but at different distances from the receiver. Because the 'near terminals' have better performance compared to the 'far terminals' (due to the better survival chance of the packets), the near-far effect introduces an unfair element among the terminals. Perfect power control, in which the transmitted power of the terminals are adjusted such that their received powers are all equal, can eliminate the near-far effect described above. The performance of CDMA has been reported in a number of publications, e.g. [5]-[10].

The hybrid CDMA/ISMA protocol combines the advantages of both CDMA and ISMA into one protocol. The advantages of CDMA and ISMA are the improvement of the survival chance of packets and the limitation of contention in the channel. Code sharing can also be applied. This is an important aspect because the number of distinct useful transmission codes is limited, especially for short code length.

In [11] the hybrid protocol combines Direct Sequence CDMA with *slotted* p-persistent ISMA. In this paper, we have investigated the performance of the hybrid CDMA/ISMA protocol using the *unslotted* non-persistent ISMA scheme. It is expected that for low traffic, the delay of the unslotted protocol will improve because when a terminal has a data packet to send, it does not have to wait until the start of the next time slot. In addition, even when data packets collide, there is a probability that the data is received correctly due to the use of CDMA. This is the strength of the hybrid CDMA/ISMA protocol. By using the *unslotted non-persistent* ISMA scheme, it is expected that we take more advantage of this strength.

This paper is organized as follows. Section II gives the protocol description of the unslotted hybrid CDMA/ISMA protocol. The performance analysis is then given in Section III. Section IV shows the assumptions used. Results are discussed in Section V. Finally, conclusions can be found in Section VI.

II. PROTOCOL DESCRIPTION

Fig. 1 depicts an example of the 16-terminal network configuration of the protocol. There is one central base station that is connected to several transceivers by wire. The base station controls the traffic flow with a busy tone that can be detected by all participating terminals. Because all terminals can detect the busy tone, the hidden terminal problem is solved. Several terminals together with one transceiver form a group. Wireless communication is considered to take place between the transceiver and the terminals. The terminals around each transceiver share the same code. In this way the number of codes can be reduced. When a data packet is ready for transmission, a terminal transmits the packet to its transceiver according to the chosen ISMA scheme.

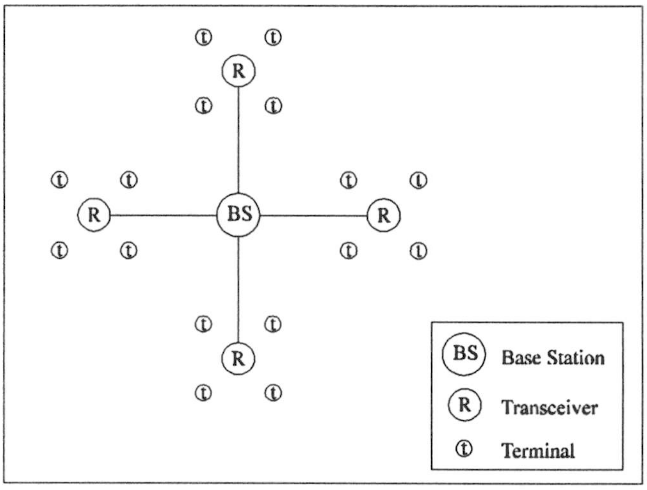

Fig. 1: 16-terminal network configuration

After the arrival of the data packet, the transceiver then simply forwards this packet to the base station which forwards it to all terminals via the transceivers. The destination then receives the packet and decides whether the packet is errorless or not. In case the packet was erroneous, a retransmission must take place. The return channel is not included in our analysis because only the base station makes use of this return channel and therefore contention is not a concern. The state of the terminals can either be free or blocked. At the beginning, the terminals are in the free state. If a packet arrives at a free terminal, the terminal jumps into the blocked state. In the blocked state, the terminal takes care that the arriving data packet is serviced successfully. In the mean time the blocked terminal ignores all incoming packets. This is a consequence of the assumption that the terminals do not have buffers for the incoming data packets.

The unslotted non-persistent ISMA protocol

Users behind terminals generate data packets. The data packets arrive at terminals, which will take care of the correct delivery of the packets. We assume that the arrivals of these packets are generated by a Poisson process. The SDL diagram of the unslotted non-persistent ISMA protocol is shown in Fig. 2. Before transmitting the data packet to the receiver, the terminal first listens to the channel to detect whether there is a busy tone going on. The base station broadcasts a busy tone to signal that the channel is busy because there is a transmission in progress. If the channel is busy, the terminal waits a random delay before it can try again. Otherwise, the packet will be transmitted immediately. If a collision occurs from the transmission, the collided packets have to wait a random delay before they are allowed to try again (Fig. 2).

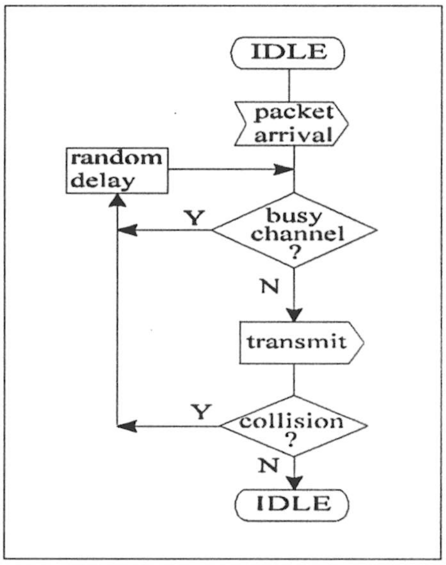

Fig. 2: Unslotted non-persistent ISMA

Although the number of collisions is greatly reduced with ISMA, collisions still can occur. In this unslotted ISMA scheme, collisions may occur due to multiple terminals transmitting packets during an interval called the *inhibit delay fraction d*. This interval d is necessary to switch from 'idle' to 'busy'. The reverse interval from 'busy' to 'idle' is denoted by d'. The inhibit delay fraction is normalized to the packet length, resulting in $0 \leq d < 1$ (Fig. 3). In this paper we assume $d = d'$.

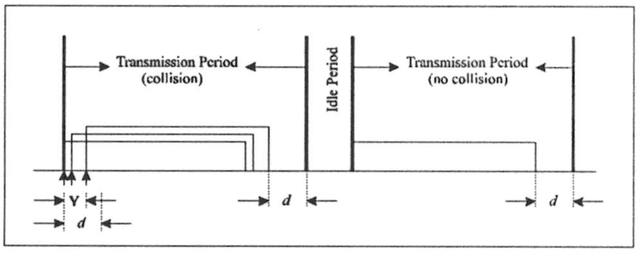

Fig. 3: Data Cycle

III. PERFORMANCE ANALYSIS

The protocol performance has been measured in terms of throughput S and delay D in relationship with the offered traffic G. The throughput measures the efficiency of the protocol and is defined as the fraction of time in which correct data packets are received. The delay is the time between the arrival of the first bit of a data packet at a terminal and its arrival of the last bit at the destination.

A. Throughput

The performance analysis of the hybrid unslotted non-persistent CDMA/ISMA will be explained from Fig. 3. The time axis in Fig. 3 is normalized to the packet duration. If a packet arrives and the terminal senses the channel free then the packet is sent immediately. Suppose that this time instant is t. It takes d, the inhibit delay fraction, before a busy tone is heard by all terminals. This moment occurs at the same time at all terminals because of the symmetric network configuration. Any other packet arriving between t and $t+d$ will sense the channel free (because the busy tone has not arrived yet) and will be transmitted resulting in a conflict. Because a CDMA scheme is also used, the conflict does not necessarily result in the loss of all packets. There is a probability that the first arriving packet can still be recovered successfully. If no other terminal transmits a packet during this period d, then no conflict occurs. We model the channel between the terminals and the corresponding transceivers as a multipath Rayleigh-fading channel. So, when no conflicts occur, there is still a probability that the packet *cannot* be recovered successfully.

Let $t+Y$ be the time of occurrence of the last packet arriving between t and $t+d$. This obviously means that Y must be between zero (the first transmitted packet is the only packet in the transmission period) and d (the end of the vulnerable period). The transmission of all packets arriving in $(t,\ t+Y)$ will be completed at $t+Y+1$. As noted before, the channel is sensed unused only a period d later. So, any terminal becoming ready between $t+d$ and $t+Y+1+d$ will sense the channel busy and hence will reschedule its packet. The interval between t and $t+Y+1+d$ is called a *transmission period* (*TP*). There will be *at most* one successful transmission during a *TP*. The *idle period* (*IP*) is defined as the period of time between two consecutive *TP*'s. A transmission period plus the following idle period constitute a *cycle*. Let $\overline{TP}$ be the expected duration of the transmission period, $\overline{I}$ the expected duration of the idle period, and the average cycle time can be written as:

$$\overline{t_c} = \overline{TP} + \overline{I} \tag{1}$$

Let U denote the time during a cycle in which the inbound channel (towards base station) is used to carry a successful packet transmission and $\overline{U}$ the corresponding average value, then we can write the throughput as [12]:

$$S = \frac{\overline{U}}{\overline{t_c}} \tag{2}$$

The expected useful time $\overline{U}$ can easily be computed. When a packet is successful, the channel carries useful information for a duration of T_{pd}, the packet duration. In the unsuccessful case no useful information is carried at all or, in other words:

$$U = \begin{cases} T_{pd} & \textit{Successful period} \\ 0 & \textit{Unsuccessfull period} \end{cases} \tag{3}$$

If $P_{success}$ denotes the probability that a transmitted packet is successful then

$$\overline{U} = T_{pd} \cdot P_{success} \tag{4}$$

As noted before, the time is normalized to T_{pd} (the packet duration), and therefore T_{pd} equals to one. So, this gives us:

$$\overline{U} = P_{success} \tag{5}$$

To calculate the average idle period we make the assumption that the total rate at which users schedule new and retransmitted packets forms a Poisson process with parameter G. So, new and rescheduled packets arrive at a rate of G packets per unit time. In literature, G is also called the offered channel traffic.

The probability of the idle time being greater than some value t is the probability that no packets are scheduled within a time interval of duration t and with the assumed Poisson packet scheduling process this probability becomes e^{-Gt}. Therefore the average value of I can be expressed as:

$$\overline{I} = \frac{1}{G} \tag{6}$$

The average duration of a transmission period equals to:

$$\overline{TP} = 1 + \overline{Y} + d \tag{7}$$

where $\overline{Y}$ is the expected value of Y. Since Y denotes the time at which the last interfering packet is scheduled, the probability of Y being smaller than some time y is the probability that no other packets (either new or retransmissions) are scheduled for transmission in an interval of duration d-y. With Poisson arrivals, the distribution for Y is $e^{-G(d-y)}$. The average of Y is therefore given by:

$$\overline{Y} = d - \frac{1}{G}\left(1 - e^{-Gd}\right) \tag{8}$$

Applying the formulas obtained above we get:

$$\begin{aligned} \overline{t_c} &= \overline{TP} + \overline{I} = 1 + \overline{Y} + d + \overline{I} \\ &= 1 + \left[d - \frac{1}{G}\left(1 - e^{-Gd}\right)\right] + d + \frac{1}{G} \end{aligned} \tag{9}$$

$$\begin{aligned} S = \frac{\overline{U}}{\overline{t_c}} &= \frac{P_{success}}{1 + \left[d - \frac{1}{G}\left(1 - e^{-Gd}\right)\right] + d + \frac{1}{G}} \\ &= \frac{G \cdot P_{success}}{G(1 + 2d) + e^{-Gd}} \end{aligned} \tag{10}$$

The term $P_{success}$ in (10) is the only term that needs to be specified. Herefore, we distinguish between four types of transmissions during a transmission period:

1. Successful transmission without conflict
2. Unsuccessful transmission without conflict
3. Successful transmission with conflict
4. Unsuccessful transmission with conflict

If the channel is used for the delivering of a successful packet during a TP then this is denoted by a successful transmission (situation 1 or 3). The difference between a transmission with or without conflict can be found in the number of packet transmissions during a TP. In case there is more than one packet transmission during a TP, we speak of a transmission with conflict. If there is only one packet transmission then the transmission is conflict-free. Keeping this in mind, we can divide $P_{success}$ into two parts:

$$P_{success} = P_{success|noconflict} + P_{success|conflict} \tag{11}$$

In which $P_{success|noconflict}$ corresponds to situation 1 and $P_{success|conflict}$ to situation 3.

$$P_{success|noconflict} = e^{-Gd} P_{ps} \tag{12}$$

This is equal to the probability that no terminal transmits during the inhibit delay fraction d multiplied by the packet success probability P_{ps}. The multiplication with P_{ps} is necessary because of the assumption of the Rayleigh fading channel.

$$P_{ps} = \left(1 - P_{be|x=0}\right)^{L_p} \tag{13}$$

$P_{be|x=0}$ is the bit error probability in Rayleigh fading channel using the CDMA scheme and is caused by self interference due to multipath. L_p denotes the packet length. We assume that during the packet duration, the bit error rate remains constant. The calculation of $P_{success|noconflict}$ is not very complicated. However, this is not the case for $P_{success|conflict}$ which is the probability that the first packet has arrived successfully given a collision (between two or more packets).

$$P_{success|conflict} = \sum_{x=1}^{K-1} P_x(d) \cdot P\{first\_packet\_OK|x\} \tag{14}$$

in which:
K is the number of active users in the system;
$P_x(d)$ is the probability of x arrivals during d for a Poisson distribution;
x denotes the number of arrivals during d.

Consider a reference terminal T_R. This terminal sends its data packet at an idle period first. Other terminals sending within the inhibit delay fraction interfere with the reception at this reference terminal T_R. The bit error rate depends on the configuration. So, if we try to take the different configurations into account, we have to calculate the bit error probability for all possible configurations and then average.

$$P\{first\_packet\_OK|x\} = E\left[1 - P_{be|\vec{x}}\right]^{L_p} \tag{15}$$

$E[.]$ denotes the expectation value, and $\vec{x}$ a certain configuration of x. The total number of combinations for $\vec{x}$ equals $\binom{K-1}{x}$. Because we assume an uniform distribution for all possible configurations we can write (K is the total number of active users in the system):

$$E\left[1 - P_{be|\vec{x}}\right]^{L_p} = \frac{1}{\binom{K-1}{x}} \cdot \sum_{\vec{x}} \left(1 - P_{be|\vec{x}}\right)^{L_p} \tag{16}$$

$$P_{success|conflict} = \sum_{x=1}^{K-1} P_x(d) \cdot \frac{1}{\binom{K-1}{x}} \cdot \sum_{\vec{x}} \left(1 - P_{be|\vec{x}}\right)^{L_p} \tag{17}$$

The formula for the bit error probability with DPSK modulation in Rayleigh fading channel is given by Kavehrad and Ramamurthi (formula 13a in [13]) in which the signal to noise ratio (SNR) is taken to be 20 dB. Combining the above results, the throughput can easily be calculated.

B. Delay

The average packet delay is defined as the duration between the transmission of the first bit till the correct reception of the last bit. For the calculation of the packet delay of the unslotted hybrid CDMA/ISMA protocol, we use the block diagram depicted in Fig. 4.

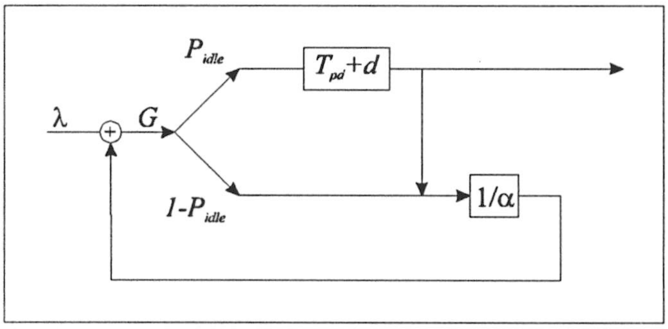

Fig. 4: Delay in the unslotted non-persistent hybrid CDMA/ISMA protocol

If a new packet arrives, the terminal immediately senses the channel to decide if it can transmit the packet. In case the channel is sensed busy (probability $1-P_{idle}$), the packet is rescheduled for a later time instant. This random delay process is assumed to be negative exponentially distributed with parameter α. The average random delay time is then $1/\alpha$. The average number of times the packet has to suffer this delay is $G(1-P_{idle})/S$. In which G, S and P_{idle} are defined as before.

On the other hand, if the channel is sensed idle (probability P_{idle}), the packet will be transmitted immediately. This transmission will take a packet duration T_{pd} plus the inhibit delay fraction d before the knowledge about the outcome of the transmission is available (see also Fig. 3). So, the total delay until the terminal knows if the transmission was successful or not is $\left(T_{pd} + d\right)$. The transmission of a data packet does not necessarily result in a successful transmission; there are two possibilities: a successful or a failed transmission. If the transmission was successful, the packet leaves the system. The delay this packet suffers is $\left(T_{pd} + d\right)$. If the transmission was a failure, the packet again has to wait a random delay. In this case, the average delay is $\left(T_{pd} + d + \frac{1}{\alpha}\right)$ and the average number of schedulings is $\left(\frac{GP_{idle}}{S} - 1\right)$. Combining the above results, the average delay is finally given by formula (18).

$$D = \left(\frac{GP_{idle}}{S} - 1\right)\left[T_{pd} + d + \frac{1}{\alpha}\right] + \frac{G(1-P_{idle})}{S}\cdot\frac{1}{\alpha} + \left(T_{pd} + d\right) \quad (18)$$

IV. ASSUMPTIONS

After DPSK modulation the signal is spread with a Gold code. The length of these codes are chosen to be 31. The signal is then transmitted, modulated on a 1.7 GHz carrier. The data rate is arbitrarily chosen to be 256·1024 b/s (0.26 Mb/s). The packet length is 64 bits. The delay spread is supposed to be 100 ns. Perfect power control is assumed to assure that signals from terminals within the same group arrive at the transceiver with the same power.

The far field model [14] is chosen to describe the signal attenuation. Given the transmitted power P_T, the received power P_R can be expressed by the following equation:

$$P_R = \frac{g_T g_R}{\left(\frac{4\pi f l}{c}\right)^{\alpha}} P_T \quad (19)$$

where g_T and g_R are respectively the transmitter and the receiver gains, f the signal frequency, l the distance between the transmitter and the receiver, c the speed of light and α is the attenuation parameter. This attenuation parameter is chosen equal to 2, corresponding to free space propagation.

Fading is the result of the propagation of the transmitted signal through several paths. The channel is modeled as a Rayleigh fading channel. The Rayleigh fading channel model is valid when each path contributes the same amount of energy to the composite received signal. A Line of Sight (LOS) path is therefore assumed to be absent.

An example of a 16-terminal network configuration that we adopted in this paper is shown in Fig. 2. The terminals are clustered around distributed transceivers at a fixed distance (5 m). The distributed transceivers are clustered around the base station at a fixed distance (30 m). Those distances are kept fixed for all simulations. The number of terminals within a transceiver group (also called group size) is chosen as a power of two. For a certain fixed number of participated terminals, we have to halve the group size if we want to double the number of codes (the number of codes is also the number of transceivers). The comparison is fair in this manner, because when we want to investigate the effect of the number of codes on the performance, we have to keep the number of participated terminals fixed. The number of terminals is 32 unless stated otherwise.

V. RESULTS

Fig. 5 a and b depict the simulation comparison between the slotted and unslotted hybrid protocol.

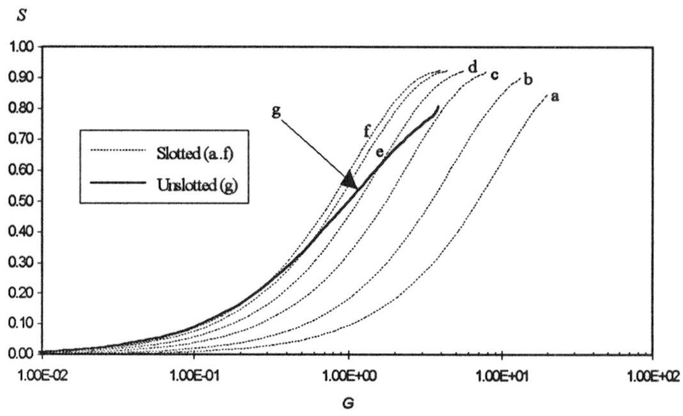

Fig. 5a: Throughput versus offered traffic using simulation of slotted (a..f: P_{tr} =0.1, 0.2, 0.4, 0.6, 0.8, 0.9) and unslotted (g: d=0.01) hybrid protocol; 8 codes; α=0.1

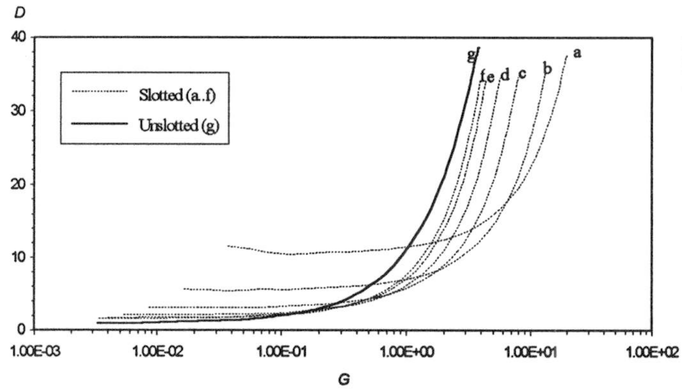

Fig. 5b: Delay versus offered traffic using simulation of slotted (a..f: P_{tr} =0.1, 0.2, 0.4, 0.6, 0.8, 0.9) and unslotted (g: d=0.01) hybrid protocol; 8 codes; α=0.1

In [11] P_{tr} is defined for the slotted protocol as the probability that the packet will actually be transmitted in case the channel is sensed *free*. For low traffic, the delay of the unslotted protocol is, as expected, better than the slotted protocol. For high traffic the situation is the other way round. The slotted protocol does not differ much from the unslotted protocol for high values of P_{tr},.

Fig. 6 represents the effect of d on the performance of the unslotted protocol. For high values of d, the performance degrades tremendously. This is due to the increase of the number of data packets colliding during this period. This is why the ISMA protocol is not useful for environment in which the propagation delay is high.

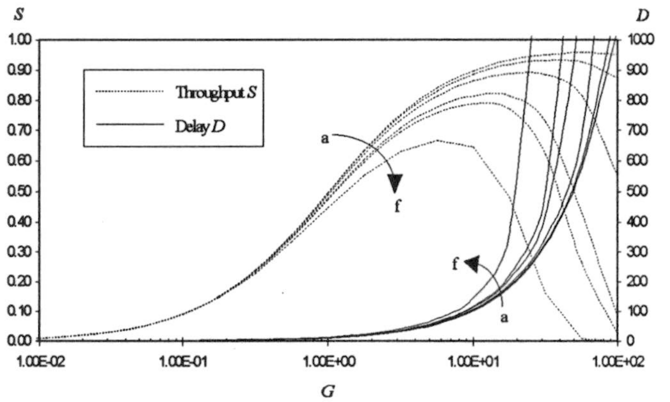

Fig. 6: Throughput and delay versus offered traffic using mathematical analysis of unslotted protocol, varying the inhibit delay fraction d. a..f: d=0.01, 0.02, 0.04, 0.08, 0.1 and 0.2; 8 codes; α=0.1

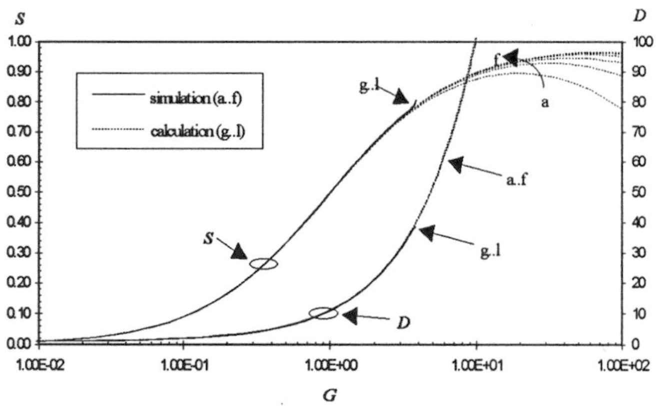

Fig. 7: Comparison between simulation and mathematical analysis for the unslotted protocol. a..f, g..l: 1, 2, 4, 8, 16, 32 codes; d=0.01; α=0.1

The comparison between the simulation and mathematical model for the unslotted protocol is given in Fig. 7. In case of simulation, the arrival rate at terminals is the input parameter and G is one of the output parameters. In addition, due to the assumption that terminals does not have buffers the performance of simulation does not exceed $G \approx 4$ packets/(packet duration). The simulation and mathematical model does quite agree with each other. This graph also shows the effect of code sharing. For a 32-terminal network, it is sufficient to use only four codes.

VI. CONCLUSIONS

The unslotted hybrid CDMA/ISMA protocol has been investigated. The protocol performance is measured in terms of throughput S and delay D in relationship with the offered traffic G. Herefore, we have been able to derive a close form formula for the throughput and delay. Also a simulation program has been developed to compare the results. The results from simulation and mathematical model does quite agree with each other.

Comparison between the slotted and unslotted protocol shows that for low offered traffic the delay for the unslotted protocol is better than the slotted protocol. For high traffic it is the other way round.

Furthermore it is concluded that the proposed hybrid protocol performs very well in case the propagation delay is not a concern. If the propagation delay is too high the performance drops quickly.

REFERENCES

[1] R. Prasad,
"Performance Analysis of Mobile Packet Radio Networks in Real Channels with Inhibit-Sense Multiple Access",
IEE Proc. I, Vol. 138, pp. 458-464, October 1991.

[2] I. Widipangestu, A.J. 't Jong and R. Prasad,
"Capture Probability and Throughput Analysis of Slotted ALOHA and Unslotted np-ISMA in a Rician/Rayleigh Environment",
IEEE Transactions on Vehicular Technology, Vol. 43, No. 3, pp. 457-465, August 1994.

[3] K.J. Zdunek, D.R. Ucci and J.L. Locicero,
"Throughput of Nonpersistent Inhibit Sense Multiple Access with Capture",
Electronics Letters, Vol. 25, No. 1, pp. 30-31, 5th January 1989.

[4] R. Prasad, and C.Y. Liu,
"Throughput Analysis of Some Mobile Packet Radio Protocols in Rician Fading Channels",
IEE Proceedings-I, Vol. 139, No. 3, pp. 297-302, June 1992.

[5] M.G. Jansen, and R. Prasad,
"Capacity, Throughput, and Delay Analysis of a Cellular DS CDMA System With Imperfect Power Control and Imperfect Sectorization",
IEEE Transactions on Vehicular Technology, Vol. 44, No. 1, pp. 67-75, February 1995.

[6] R.A. Scholtz,
"The Origins of Spread Spectrum Communications",
IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 822-854, May 1982.

[7] W.C.Y. Lee,
"Overview of Cellular CDMA",
IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 291-302, May 1991.

[8] M.B. Pursley,
"Performance Evaluation of Phase Coded Spread Spectrum Multiple Access Communication - Part I: System Analysis",
IEEE Transactions on Communications, Vol. COM-25, No. 8, pp. 795-799, August 1977.

[9] R.L. Pickholtz, D.L. Schilling and L.B. Milstein
"Theory of Spread Spectrum Communications - A Tutorial",
IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 855-884, May 1982.

[10] K.S. Gilhousen, I.M. Jacobs, R. Padovani, A.J. Viterbi, L.A. Weaver, Jr. and C.E.Wheatley III,
"On the Capacity of a Cellular CDMA System",
IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 303-312, May 1991.

[11] H. van Roosmalen, J. Nijhof and R. Prasad,
"Performance Analysis of a Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications",
IEEE Journal on Selected Areas in Communications, Vol. 12, No. 5, pp. 909-916, June 1994.

[12] L. Kleinrock and F.A. Tobagi,
"Packet Switching in Radio Channels: Part I- Carrier Sense Multiple Access and their Throughput-Delay Characteristics",
IEEE Trans. on Commun., Vol. COM-23, No. 12, pp. 1400-1416, December 1975.

[13] M. Kavehrad and B. Ramamurthi,
"Direct -Sequence Spread Spectrum with DPSK Modulation and Diversity for Indoor Wireless Communications",
IEEE Transactions on Communications, Vol. COM-35, No. 2, pp. 224-236, February 1987.

[14] A. B. Carlson,
Communication systems,
Mc. Graw Hill Book Company, 3rd edition, 1988, pp. 92.

APPENDIX B:

“Indoor Wireless Computer Communications using Unslotted Hybrid CDMA/ISMA Protocol”

Submitted for participation in the International Conference on Communications (ICC ’96).

Indoor Wireless Computer Communications using Unslotted Hybrid CDMA/ISMA Protocol

Ramjee Prasad, *Senior Member, IEEE*
Huy Linh Anh Le, *Student Member, IEEE*
Huub van Roosmalen
Jos Nijhof, *Member, IEEE*

Correspondence address:
Telecommunications and Traffic Control Systems Group
Delft University of Technology
Prof. Dr. R. Prasad
P.O. Box 5031, 2600 GA Delft
The Netherlands
Tel.: +31 15-78.24.17(before 10-10-'95) +31 15-278.24.17(after 10-10-'95)
Fax: +31 15-78.17.74(before 10-10-'95) +31 15-278.17.74(after 10-10-'95)

Presenter
Should the paper be accepted, then Prof. Dr. R. Prasad will attend the conference and also present the paper.

Technical Subject Areas:
- Computer Communications
- Data Communications
- Broadband Access and Delivery Systems

Abstract
A hybrid Code Division Multiple Access / Inhibit Sense Multiple Access (CDMA/ISMA) protocol has been proposed as an effective multiple access scheme for Indoor Wireless Computer Communications. This new protocol combines the advantages of both CDMA and ISMA into one protocol. On the one hand the ISMA protocol introduces a limitation to the number of simultaneous accesses to the transmission channel. On the other hand the CDMA protocol introduces an improvement to the packet survival chance. It is shown that the performance of the hybrid protocol is superior to the performance of CDMA only. In addition, code sharing can be applied to reduce hardware cost. To take more advantage of the strength of the hybrid protocol, we have investigated the *unslotted non-persistent* ISMA scheme instead of the *slotted p-persistent* ISMA scheme. The analysis is done for a star-connected multiple access radio network. The performance comparison between the slotted and unslotted hybrid CDMA/ISMA protocol is evaluated in terms of throughput and delay using computer simulation and mathematical analysis.

I. INTRODUCTION

Indoor wireless communication has proved to be a very attractive means of computer/data communication in an office environment. The office system we focus on consists of a building in which users work together in groups. The participants generate terminal traffic. Terminals communicate with each other by radio transmission using a random access protocol. Terminals might not detect each other's transmission in radio communications. It can easily happen that two users are hidden from each other by some obstacle, in which case severe performance degradation results. This is called the *hidden terminal problem*. The introduction of a central base station can alleviate this problem by instructing it to send a busy tone to all participating terminals to forbid new transmissions when a transmission is going on. Still, a situation can occur in which two or more terminals simultaneously start their transmission, resulting in a collision. However, a great reduction in the number of simultaneous transmissions is achieved by the introduction of a central base station. This concept is called ISMA [1]-[4].

If we could somehow increase the survival chances of colliding packets, we could improve protocol performance. The near-far effect is one way to achieve this. A packet may 'capture' the receiver if it is much stronger than its competitors. This can happen when terminals use the same transmission power, but at different distances from the receiver. Because the 'near terminals' have better performance compared to the 'far terminals' (due to the better survival chance of the packets), the near-far effect introduces an unfair element among the terminals. Perfect power control, in which the transmitted power of the terminals are adjusted such that their received powers are all equal, can eliminate the near-far effect described above. The performance of CDMA has been reported in a number of publications, e.g. [5]-[10].

The hybrid CDMA/ISMA protocol combines the advantages of both CDMA and ISMA into one protocol. The advantages of CDMA and ISMA are the improvement of the survival chance of packets and the limitation of contention in the channel. Code sharing can also be applied. This is an important aspect because the number of distinct useful transmission codes is limited, especially for short code length.

In [11] the hybrid protocol combines Direct Sequence CDMA with *slotted* p-persistent ISMA. In this paper, we have investigated the performance of the hybrid CDMA/ISMA protocol using the *unslotted* non-persistent ISMA scheme. It is expected that for low traffic, the delay of the unslotted protocol will improve because when a terminal has a data packet to send, it does not have to wait until the start of the next time slot. In addition, even when data packets collide, there is a probability that the data is received correctly due to the use of CDMA. This is the strength of the hybrid CDMA/ISMA protocol. By using the *unslotted non-persistent* ISMA scheme, it is expected that we take more advantage of this strength.

This paper is organized as follows. Section II gives the protocol description of the unslotted hybrid CDMA/ISMA protocol. The performance analysis is then given in Section III. Section IV shows the assumptions used. Results are discussed in Section V. Finally, conclusions are given in Section VI.

II. PROTOCOL DESCRIPTION

Fig. 1 depicts an example of the 16-terminal network configuration of the protocol. There is one central base station that is connected to several transceivers by wire. The base station controls the traffic flow with a busy tone that can be detected by all participating terminals. Because all terminals can detect the busy tone, the hidden terminal problem is solved. Several terminals together with one transceiver form a group. Wireless communication is considered to take place between the transceiver and the terminals. The terminals around each transceiver share the same code. In this way the number of codes can be reduced. When a data packet is ready for transmission, a terminal transmits the packet to its transceiver according to the chosen ISMA scheme.

After the arrival of the data packet, the transceiver then simply forwards this packet to the base station which forwards it to all terminals via the transceivers. The destination then receives the packet and decides whether the packet is errorless or not. In case the packet was erroneous, a retransmission must take place. The return channel is not included in our analysis because only the base station makes use of this return channel and therefore contention is not a concern. The state of the terminals can either be free or blocked. At the beginning, the terminals are in the free state. If a packet arrives at a free terminal, the terminal jumps into the blocked state. In the blocked state, the terminal takes care that the arriving data packet is serviced successfully. In the mean time the blocked terminal ignores all incoming packets. This is a consequence of the assumption that the terminals do not have buffers for the incoming data packets.

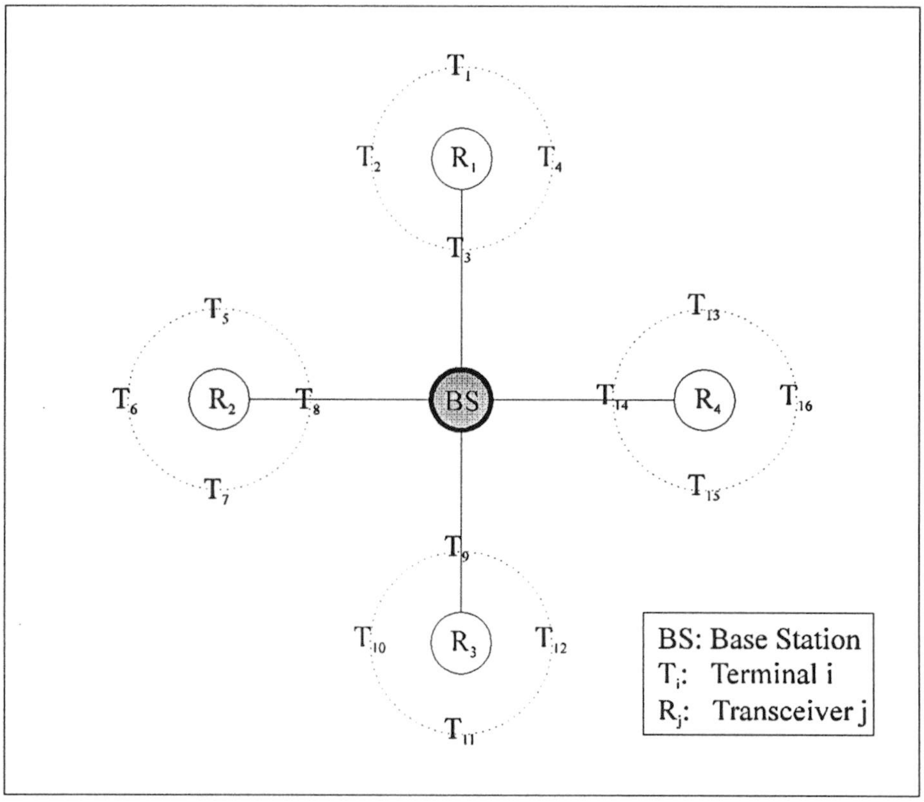

Fig. 1: 16-terminal network configuration

The unslotted non-persistent ISMA protocol

Users behind terminals generate data packets. The data packets arrive at terminals, which will take care of the correct delivery of the packets. We assume that the arrivals of these packets are generated by a Poisson process. The Specification and Description Language (SDL) diagram of the unslotted non-persistent ISMA protocol is shown in Fig. 2. Before transmitting the data packet to the receiver, the terminal first listens to the channel to detect whether there is a busy tone going on. The base station broadcasts a busy tone to signal that the channel is busy because there is a transmission in progress. If the channel is busy, the terminal waits a random delay before it can try again. Otherwise, the packet will be transmitted immediately. If a collision occurs from the transmission, the collided packets have to wait a random delay before they are allowed to try again (Fig. 2).

Although the number of collisions is greatly reduced with ISMA, collisions still can occur. In this unslotted ISMA scheme, collisions may occur due to multiple terminals transmitting packets during an interval called the *inhibit delay fraction* d. This interval d is necessary to switch from 'idle' to 'busy'. The reverse interval from 'busy' to 'idle' is denoted by d'. The inhibit delay fraction is normalized to the packet length, resulting in $0 \leq d < 1$ (Fig. 3). In this paper we assume $d' = d$.

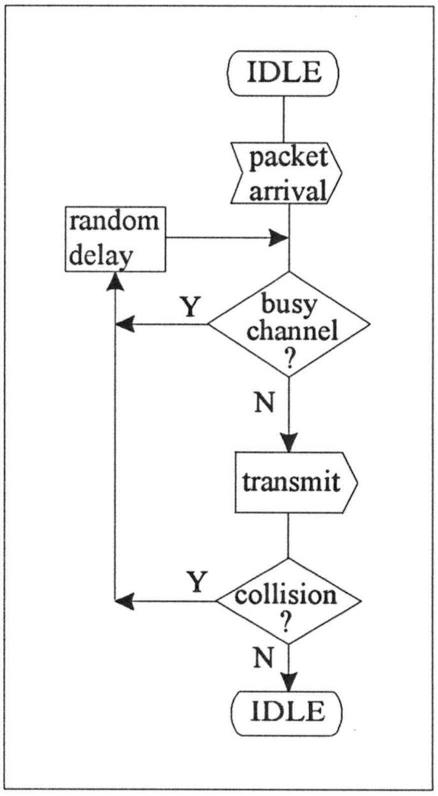

Fig. 2: Unslotted non-persistent ISMA

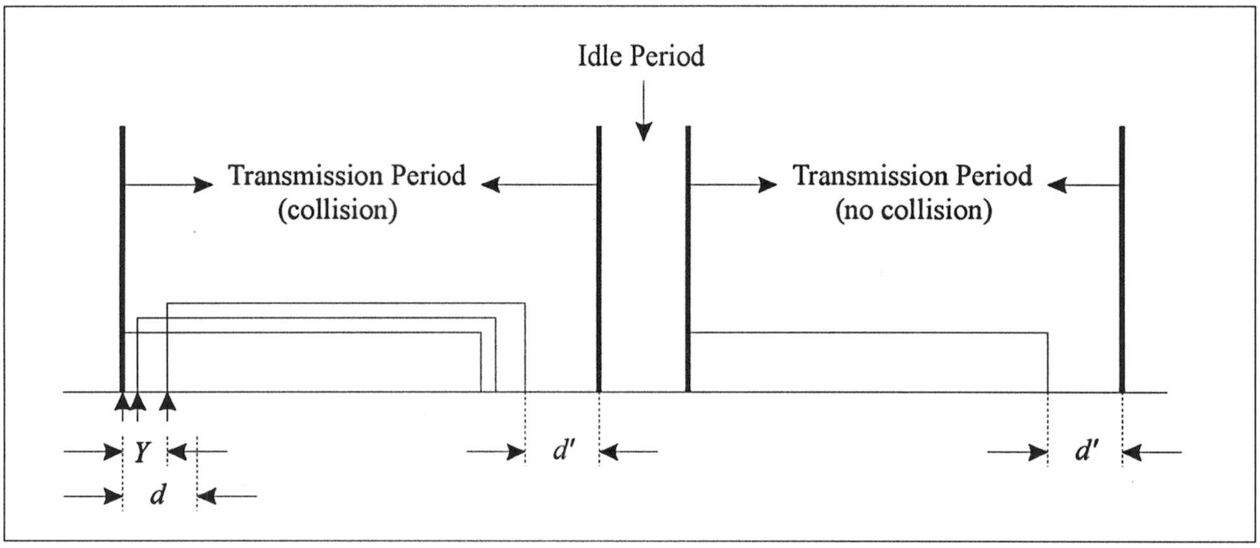

Fig. 3: Data Cycle

III. PERFORMANCE ANALYSIS

The protocol performance has been measured in terms of throughput S and delay D in relationship with the offered traffic G. The throughput measures the efficiency of the protocol and is defined as the fraction of time in which correct data packets are received. The delay is the time between the arrival of the first bit of a data packet at a terminal and its arrival of the last bit at the destination.

A. Throughput

The performance analysis of the hybrid unslotted non-persistent CDMA/ISMA will be explained from Fig. 3. The time axis in Fig. 3 is normalized to the packet duration. If a packet arrives and the terminal senses the channel free, then the packet is sent immediately. Suppose that this time instant is t. It takes d, the inhibit delay fraction, before a busy tone is heard by all terminals. This moment occurs at the same time at all terminals because of the symmetric network configuration. Any other packet arriving between t and $t+d$ will sense the channel free (because the busy tone has not arrived yet) and will be transmitted resulting in a conflict. Because a CDMA scheme is also used, the conflict does not necessarily result in the loss of all packets. There is a probability that the first arriving packet can still be recovered successfully. If no other terminal transmits a packet during this period *d*, then no conflict occurs. We model the channel between the terminals and the corresponding transceivers as a multipath Rayleigh-fading channel. So, when no conflicts occur, there is still a probability that the packet *cannot* be recovered successfully.

Let $t+Y$ be the time of occurrence of the last packet arriving between t and $t+d$. This obviously means that Y must be between zero (the first transmitted packet is the only packet in the transmission period) and d (the end of the vulnerable period). The transmission of all packets arriving in $(t, t+Y)$ will be completed at $t+Y+1$. As noted before, the channel is sensed unused only a period d later. So, any terminal becoming ready between $t+d$ and $t+Y+1+d$ will sense the channel busy and hence will reschedule its packet. The interval between t and $t+Y+1+d$ is called a *transmission period* (*TP*). There will be *at most* one successful transmission during a *TP*. The *idle period* (*IP*) is defined as the period of time between two consecutive *TP*'s. A transmission period plus the following idle period constitute a *cycle*. Let $\overline{TP}$ be the expected duration of the transmission period, $\overline{I}$ the expected duration of the idle period, and the average cycle time can be written as:

$$\overline{t_c} = \overline{TP} + \overline{I} \tag{1}$$

Let U denote the time during a cycle in which the inbound channel (towards base station) is used to carry a successful packet transmission and $\overline{U}$ the corresponding average value, then we can write the throughput as [12]:

$$S = \frac{\overline{U}}{\overline{t_c}} \tag{2}$$

The expected useful time $\overline{U}$ can easily be computed. When a packet is successful, the channel carries useful information for a duration of T_{pd}, the packet duration. In the unsuccessful case no useful information is carried at all or, in other words:

$$U = \begin{cases} T_{pd} & \textit{Successful} \quad \textit{period} \\ 0 & \textit{Unsuccessfull} \quad \textit{period} \end{cases} \tag{3}$$

If $P_{success}$ denotes the probability that a transmitted packet is successful then

$$\overline{U} = T_{pd} \cdot P_{success} \tag{4}$$

As noted before, the time is normalized to T_{pd} (the packet duration), and therefore T_{pd} equals to one. So, this gives us:

$$\overline{U} = P_{success} \tag{5}$$

To calculate the average idle period we make the assumption that the total rate at which users schedule new and retransmitted packets forms a Poisson process with parameter G. So, new and rescheduled packets arrive at a rate of G packets per unit time. In literature, G is also called the offered channel traffic. If the packet arrival times are independent and exponentially distributed with a mean arrival rate of G packets per second, the probability of k arrivals in an interval of duration t is then a Poisson process given by $P_k(t)$ where

$$P_k(t) = \frac{(Gt)^k \cdot e^{-Gt}}{k!} \tag{6}$$

The probability of the idle time being greater than some value t is the probability that no packets are scheduled within a time interval of duration t and with the assumed Poisson packet scheduling process this probability becomes e^{-Gt}. Therefore the average value of I can be expressed as:

$$\bar{I} = \frac{1}{G} \tag{7}$$

The average duration of a transmission period equals to:

$$\overline{TP} = 1 + \bar{Y} + d \tag{8}$$

where $\bar{Y}$ is the expected value of Y. Since Y denotes the time at which the last interfering packet is scheduled, the probability of Y being smaller than some time y is the probability that no other packets (either new or retransmissions) are scheduled for transmission in an interval of duration d-y. With Poisson arrivals, the distribution for Y is $e^{-G(d-y)}$. The average of Y is therefore given by:

$$\bar{Y} = d - \frac{1}{G}\left(1 - e^{-Gd}\right) \tag{9}$$

Applying the equations (8) and (9), we get:

$$\bar{t}_c = \overline{TP} + \bar{I} = 1 + \bar{Y} + d + \bar{I} = 1 + \left[d - \frac{1}{G}\left(1 - e^{-Gd}\right)\right] + d + \frac{1}{G} \tag{10}$$

Using (5) and (10), the throughput can be given as:

$$S = \frac{\overline{U}}{\overline{t}_c} = \frac{P_{success}}{1 + \left[d - \frac{1}{G}\left(1 - e^{-Gd}\right) \right] + d + \frac{1}{G}} = \frac{G \cdot P_{success}}{G(1+2d) + e^{-Gd}} \tag{11}$$

The term $P_{success}$ in (10) is the only term that needs to be specified. Herefore, we distinguish between four types of transmissions during a transmission period:

1. Successful transmission without conflict
2. Unsuccessful transmission without conflict
3. Successful transmission with conflict
4. Unsuccessful transmission with conflict

If the channel is used for the delivering of a successful packet during a *TP* then this is denoted by a successful transmission (situation 1 or 3). The difference between a transmission with or without conflict can be found in the number of packet transmissions during a *TP*. In case there is more than one packet transmission during a *TP*, we speak of a transmission with conflict. If there is only one packet transmission then the transmission is conflict-free. Keeping this in mind, we can divide $P_{success}$ into two parts:

$$P_{success} = P_{success|noconflict} + P_{success|conflict} \tag{12}$$

In which $P_{success|noconflict}$ corresponds to situation 1 and $P_{success|conflict}$ to situation 3.

$$P_{success|noconflict} = e^{-Gd} P_{ps} \tag{13}$$

This is equal to the probability that no terminal transmits during the inhibit delay fraction d multiplied by the packet success probability P_{ps}. The multiplication with P_{ps} is necessary because of the assumption of the Rayleigh fading channel.

$$P_{ps} = \left(1 - P_{be|x=0}\right)^{L_p} \tag{14}$$

$P_{be|x=0}$ is the bit error probability in Rayleigh fading channel using the CDMA scheme and is caused by self interference due to multipath. L_p denotes the packet length. We assume that during the packet duration, the bit error rate remains constant. The calculation of $P_{success|noconflict}$ is not very complicated. However, this is not the case for $P_{success|conflict}$ which is the probability that the first packet has arrived successfully given a collision (between two or more packets).

$$P_{success|conflict} = \sum_{x=1}^{K-1} P_x(d) \cdot P\{first\_packet\_OK|x\} \tag{15}$$

in which:

K is the number of active users in the system;

x denotes the number of arrivals during d;

$P_x(d)$ is the probability of x arrivals during d for a Poisson distribution and is given by (6).

Consider a reference terminal T_R. This terminal sends its data packet at an idle period first. Other terminals sending within the inhibit delay fraction interfere with the reception at this reference terminal T_R. The bit error rate depends on the configuration. So, if we try to take the different configurations into account, we have to calculate the bit error probability for all possible configurations and then average.

$$P\{first\_packet\_OK|x\} = E\left[1 - P_{be|\vec{x}}\right]^{L_p} \tag{16}$$

$E[.]$ denotes the expectation value, and $\vec{x}$ a certain configuration of x. The total number of combinations for $\vec{x}$ equals $\binom{K-1}{x}$.

Because we assume an uniform distribution for all possible configurations we can write (K is the total number of active users in the system):

$$E\left[1 - P_{be|\vec{x}}\right]^{L_p} = \frac{1}{\binom{K-1}{x}} \cdot \sum_{\vec{x}} \left(1 - P_{be|\vec{x}}\right)^{L_p} \tag{17}$$

$$P_{success|conflict} = \sum_{x=1}^{K-1} P_x(d) \cdot \frac{1}{\binom{K-1}{x}} \cdot \sum_{\vec{x}} \left(1 - P_{be|\vec{x}}\right)^{L_p} \tag{18}$$

$$P_{success} = e^{-Gd}\left(1 - P_{be|x=0}\right)^{L_p} + \sum_{x=1}^{K-1} P_x(d) \cdot \frac{1}{\binom{K-1}{x}} \cdot \sum_{\vec{x}} \left(1 - P_{be|\vec{x}}\right)^{L_p} \tag{19}$$

The formula for the bit error probability with DPSK modulation in Rayleigh fading channel is given by Kavehrad and Ramamurthi (formula 13a in [13]) in which the signal to noise ratio (SNR) is taken to be 20 dB. Combining the above results, the throughput can easily be calculated as:

$$S = \frac{G\left(e^{-Gd}\left(1 - P_{be|x=0}\right)^{L_p} + \sum_{x=1}^{K-1} P_x(d) \cdot \frac{1}{\binom{K-1}{x}} \cdot \sum_{\vec{x}} \left(1 - P_{be|\vec{x}}\right)^{Lp}\right)}{G(1+2d) + e^{-Gd}} \tag{20}$$

B. Delay

The average packet delay is defined as the duration between the transmission of the first bit till the correct reception of the last bit. For the calculation of the packet delay of the unslotted hybrid CDMA/ISMA protocol, we use the block diagram depicted in Fig. 4.

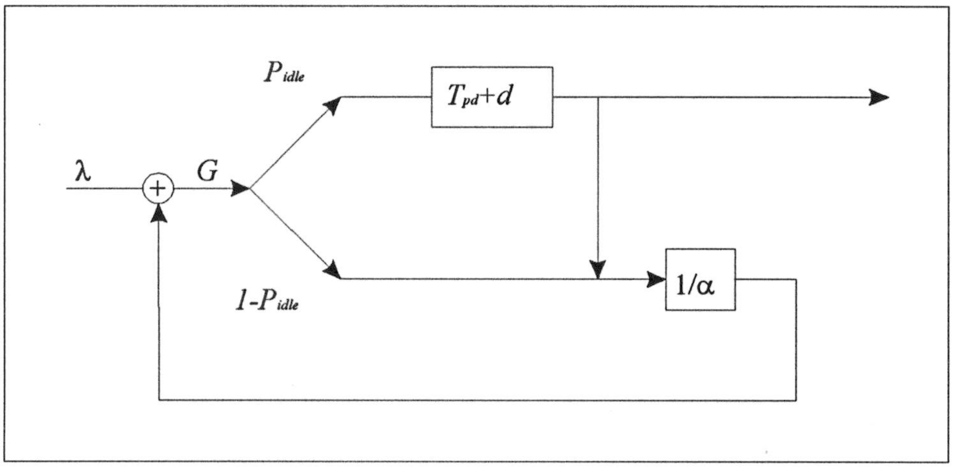

Fig. 4: Delay in the unslotted non-persistent hybrid CDMA/ISMA protocol

If a new packet arrives, the terminal immediately senses the channel to decide if it can transmit the packet. In case the channel is sensed busy (probability $1-P_{idle}$), the packet is rescheduled for a later time instant. This random delay process is assumed to be negative exponentially distributed with parameter α. The average random delay time is then $1/\alpha$. The average number of times the packet has to suffer this delay is $G(1-P_{idle})/S$.

On the other hand, if the channel is sensed idle (probability P_{idle}), the packet will be transmitted immediately. This transmission will take a packet duration T_{pd} plus the inhibit delay fraction d before the knowledge about the outcome of the transmission is available (see also Fig. 3). So, the total delay until the terminal knows if the transmission was successful or not is $(T_{pd}+d)$. The transmission of a data packet does not necessarily result in a successful transmission; there are two possibilities: a successful or a failed transmission. If the transmission was successful, the packet leaves the system. The delay this packet suffers is $\left(T_{pd}+d\right)$. If the transmission was a failure, the packet again has to wait a random delay. In this case, the average delay is $\left(T_{pd}+d+\frac{1}{\alpha}\right)$ and the average number of schedulings is $\left(\frac{GP_{idle}}{S}-1\right)$. Combining the above results, the average delay is finally given by:

$$D=\left(\frac{GP_{idle}}{S}-1\right)\left[T_{pd}+d+\frac{1}{\alpha}\right]+\frac{G\left(1-P_{idle}\right)}{S}\cdot\frac{1}{\alpha}+\left(T_{pd}+d\right) \tag{21}$$

S can be found in formula (20) and P_{idle} still has to be calculated. The average probability of sensing the channel idle is the average probability that the channel is idle and this probability is equal to the average idle time ($\bar{I}$) divided by the average cycle length ($\bar{t}_c$).

$$P_{idle}=\frac{\bar{I}}{\bar{t}_c}=\frac{\frac{1}{G}}{\frac{1}{G}+\left\{1+d-\frac{1}{G}\left(1-e^{-Gd}\right)+d\right\}}=\frac{1}{G\left(1+2d\right)+e^{-Gd}} \tag{22}$$

IV. COMPUTER SIMULATION

In this section we describe how the computer simulation program was set up to measure the performance of the unslotted hybrid CDMA/ISMA protocol.

A. Representation of Time

In general, if we want to use a computer simulation to analyse a system, the following steps can be distinguished. The dynamic behaviour of the system is studied by tracing various system states as a function of time and then collecting and analysing the system statistics. The events that change the system state are generated at different points in time, and the passage of time is represented by an internal clock which is incremented and maintained by the simulation program.

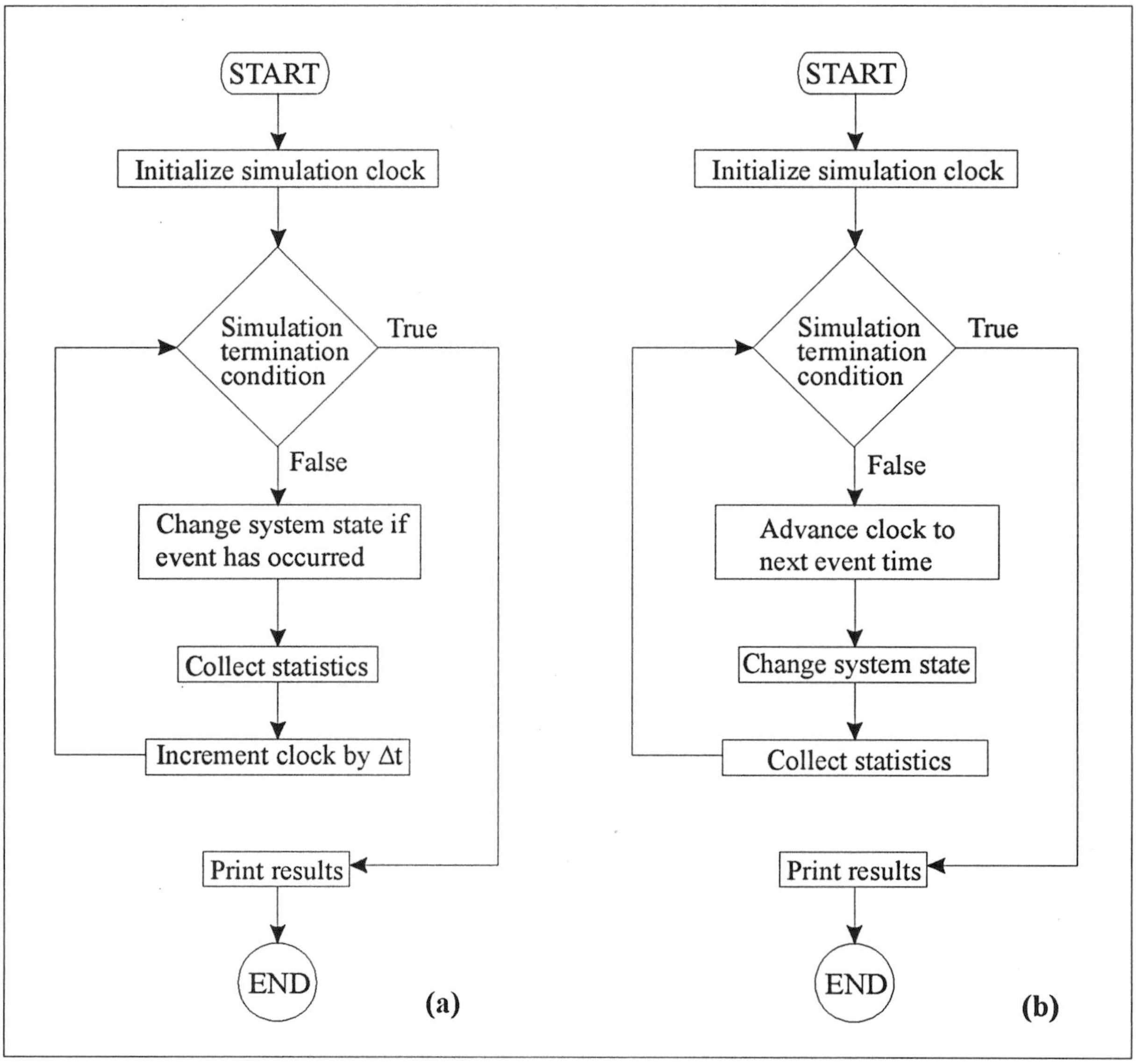

Fig. 5: (a) Interval-oriented simulation; (b) Event-oriented simulation

The simulation time can be advanced in two ways. The first method is the *interval-oriented simulation* (or the uniform time increment method) where the clock is advanced from time t to $(t + \Delta t)$ where Δt is a uniform fixed time increment. Fig. 5a depicts this mechanism. The second method is the *event-oriented simulation* (or the variable time increment method) where the clock is incremented from time t

to the next event time t', whatever may be the value of t'. The state changes are made at event time t, the next event time t', and this process is continuously repeated. Thus, only events are represented explicitly in a simulation model and the periods between events are treated as inactive or insignificant and therefore consume no time even though the interevent activites do consume time in the real world (Fig. 5b).

Obviously, method 1 detects the events that occur during the interval $(t, t+\Delta t)$ only at time $(t+\Delta t)$, thereby introducing errors in simulation. Another drawback of this method is that if the interval between two events is very large compared to Δt, then the simulator goes through several unproductive clock increments (during periods of inactivity) and the associated computing effort which will not bring about any change in system states. This fixed time increment method is suitable for the simulation of continuous systems and in particular systems with large numbers of state variables.

The second method involves sorting of event activation times and maintaining *events list*. In our work we will use the event-oriented simulation (Figure 5b) to simulate the unslotted hybrid CDMA/ISMA protocol.

B. Main Structure

The Program Structure Diagram (PSD) of the main program is depicted in Fig. 6. The simulation program simulates the working of the unslotted hybrid CDMA/ISMA protocol. First of all, the counters have to be reset to the proper values. *Clock* indicates the actual time while the simulation will last *EndTime*. While *Clock* has not achieved *EndTime*, the simulation will loop. The loop is divided in four steps as shown in Fig. 6. In the first step, the program advances *Clock* to the next event time. This event is the first event to occur in future time. Because the event-oriented simulation is used, only events are represented explicitly in the simulation model. Depending on the event type, the program processes this event. This is the second step. The processing can generate other events or change the system state. In the third step, the generated events and the changed system state will be updated. Finally, statistics collection completes the loop.

Fig. 6: PSD of main program

C. Event Lists

From Fig. 6 it emerges that events and event list play an important role in the main program structure.

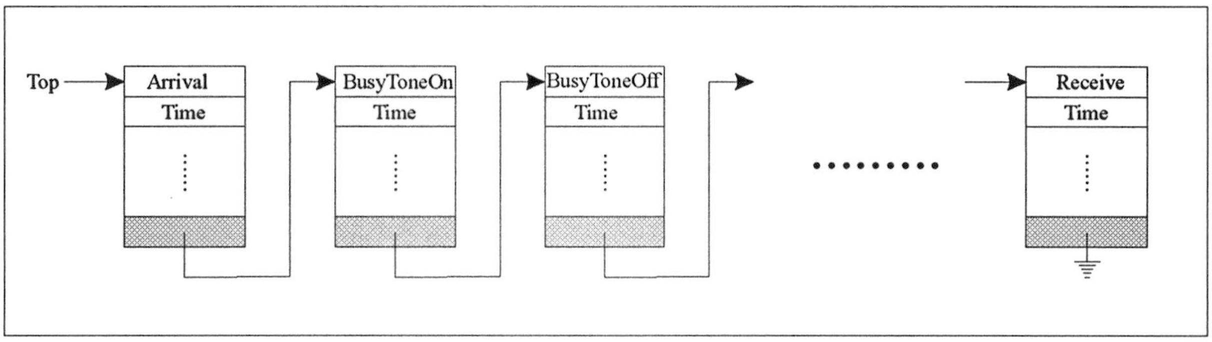

Fig. 7: Example of an event list

The whole program is based on the event list. The event list consists of events linked together in an organised manner. Here, organised means in an order in which the events occur. An example of an event list is shown in Fig. 7. The event list consists of nodes linked together to form a chain. In the event oriented simulation this chain symbolizes the time axis. Every node represents an event and consists of an *EventType*, the time this event occur, a tail and other information. The tail consists of a pointer pointing to the next node. The last node points to nowhere. This is illustrated with the ground symbol.

IV. ASSUMPTIONS

After DPSK modulation the signal is spread with a Gold code. The length of these codes are chosen to be 31. The signal is then transmitted, modulated on a 1.7 GHz carrier. The data rate is arbitrarily chosen to be 256·1024 b/s (0.26 Mb/s). The packet length is 64 bits. The delay spread is supposed to be 100 ns. Perfect power control is assumed to assure that signals from terminals within the same group arrive at the transceiver with the same power.

The far field model [14] is chosen to describe the signal attenuation. Given the transmitted power P_T, the received power P_R can be expressed by the following equation:

$$P_R = \frac{g_T g_R}{\left(\frac{4\pi f l}{c}\right)^{\alpha}} P_T \tag{19}$$

where g_T and g_R are respectively the transmitter and the receiver gains, f the signal frequency, l the distance between the transmitter and the receiver, c the speed of light and α is the attenuation parameter. This attenuation parameter is chosen equal to 2, corresponding to free space propagation.

Fading is the result of the propagation of the transmitted signal through several paths. The channel is modeled as a Rayleigh fading channel. The Rayleigh fading channel model is valid when each path contributes the same amount of energy to the composite received signal. A Line of Sight (LOS) path is therefore assumed to be absent.

An example of a 16-terminal network configuration that we adopted in this paper is shown in Fig. 2. The terminals are clustered around distributed transceivers at a fixed distance (5 m). The distributed transceivers are clustered around the base station at a fixed distance (30 m). Those distances are kept fixed for all simulations. The number of terminals within a transceiver group (also called group size) is chosen as a power of two. For a certain fixed number of participated terminals, we have to halve the group size if we want to double the number of codes (the number of codes is also the number of transceivers). The comparison is fair in this manner, because when we want to investigate the effect of the number of codes on the performance, we have to keep the number of participated terminals fixed. The number of terminals is 32 unless stated otherwise.

V. RESULTS

Fig. 5 a and b compare the performance of the slotted and unslotted hybrid protocol using computer simulation.

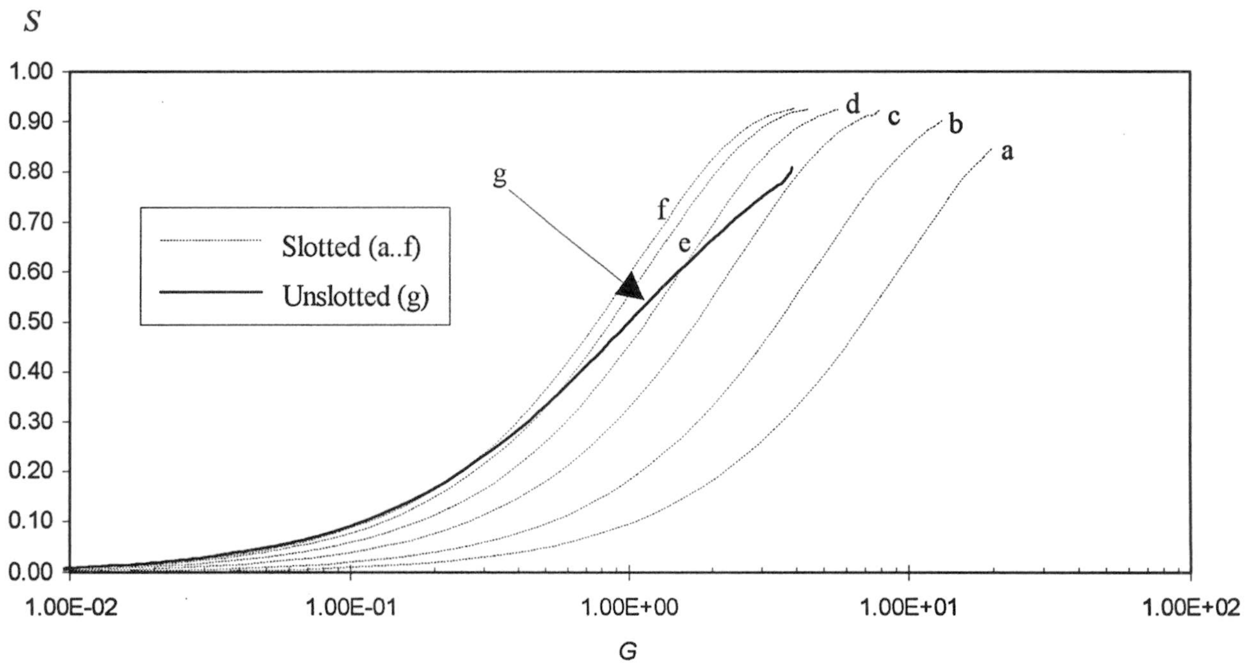

Fig. 5a: Throughput versus offered traffic using simulation of slotted (a..f: P_{tr} =0.1, 0.2, 0.4, 0.6, 0.8, 0.9) and unslotted (g: d=0.01) hybrid protocol; 8 codes; α=0.1

In [1] P_{tr} is defined for the slotted protocol as the probability that the packet will actually be transmitted in case the channel is sensed *free*. For low traffic, the delay of the unslotted protocol is, as expected, better than the slotted protocol. For high traffic the situation is the other way round. The slotted protocol does not differ much from the unslotted protocol for high values of P_{tr},.

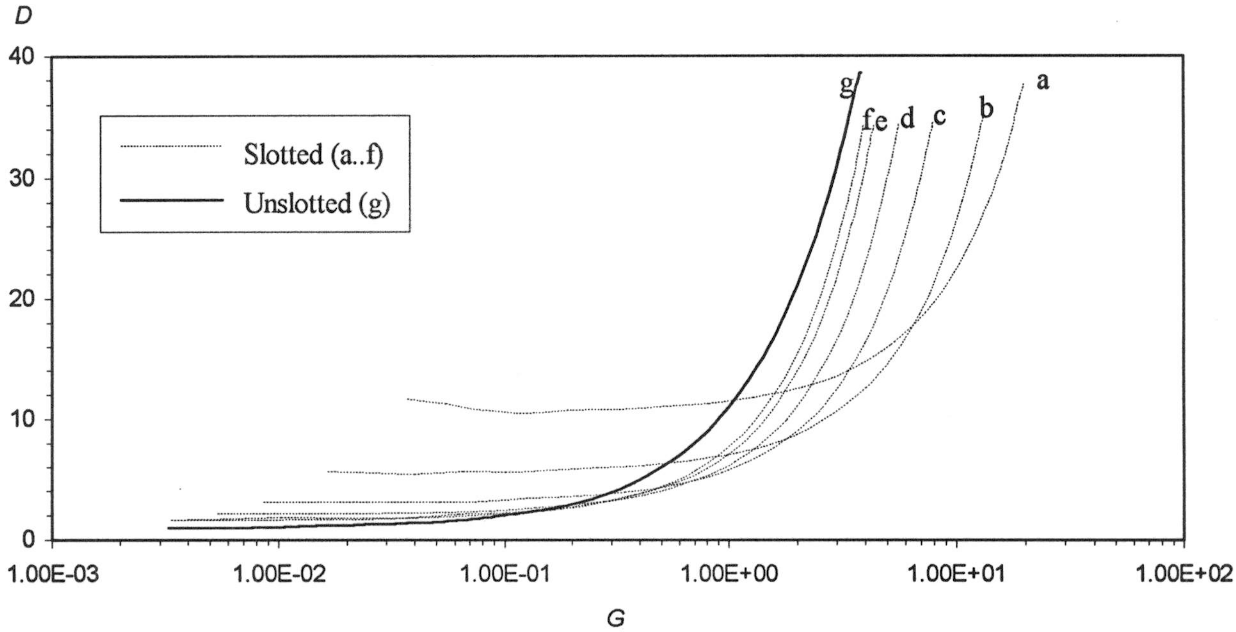

Fig. 5b: Delay versus offered traffic using simulation of slotted (a..f: P_{tr} =0.1, 0.2, 0.4, 0.6, 0.8, 0.9) and unslotted (g: d=0.01) hybrid protocol; 8 codes; α=0.1

Fig. 6 shows the effect of the inhibit delay fraction d on the performance of the unslotted protocol. For high values of d, the performance degrades tremendously. This is due to the increase of the number of data packets colliding during this period. This is why the ISMA protocol is not useful for environment in which the propagation delay is high.

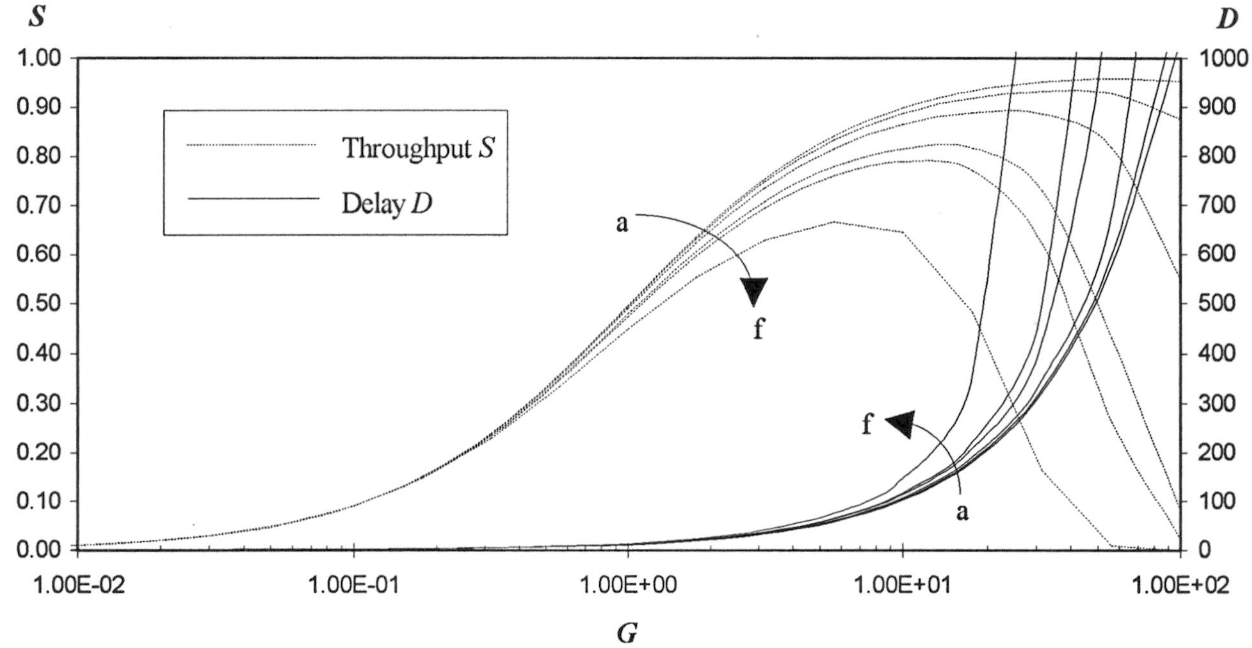

Fig. 6: Throughput and delay versus offered traffic using mathematical analysis of unslotted protocol, varying the inhibit delay fraction d. a..f: d=0.01, 0.02, 0.04, 0.08, 0.1 and 0.2; 8 codes; α=0.1

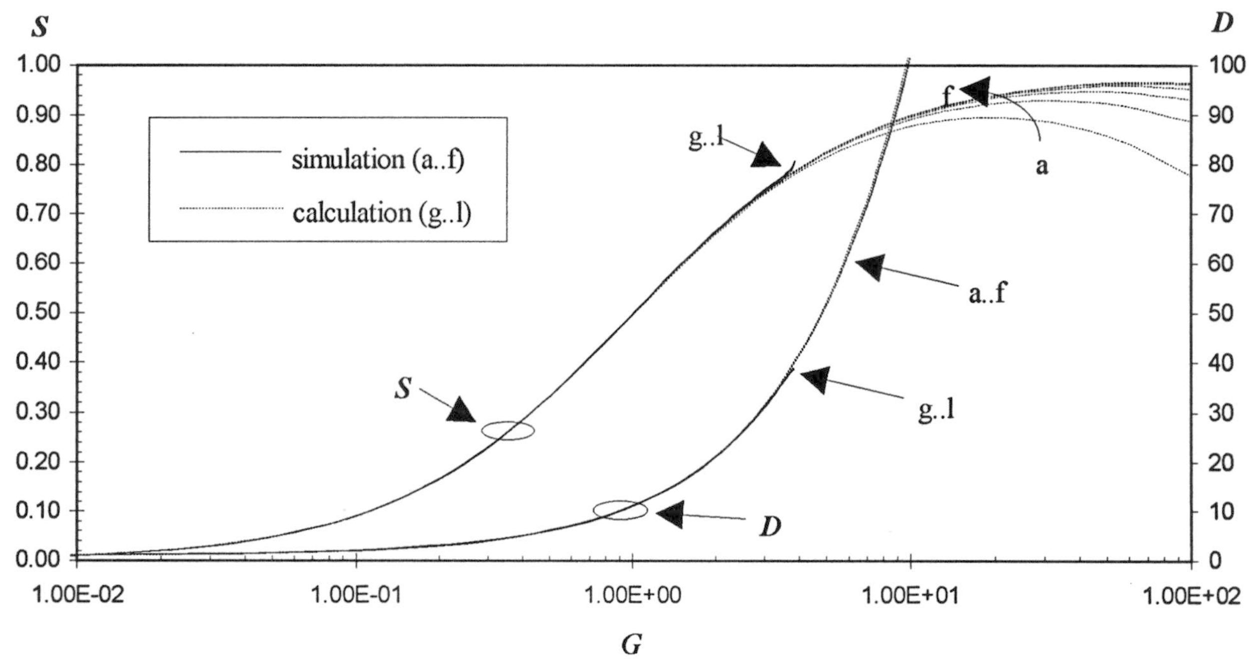

Fig. 7: Comparison between simulation and mathematical analysis for the unslotted protocol. a..f, g..l: 1, 2, 4, 8, 16, 32 codes; d=0.01; α=0.1

The comparison between the simulation and mathematical model for the unslotted protocol is given in Fig. 7. In case of simulation, the arrival rate at terminals is the input parameter and G is one of the output parameters. In addition, due to the assumption that terminals do not have buffers the performance of simulation does not exceed for $G \approx 4$ packets/(packet duration). The results due to simulation and mathematical model are in good agreement. This graph also shows the effect of code sharing. For a 32-terminal network, it is sufficient to use only four codes.

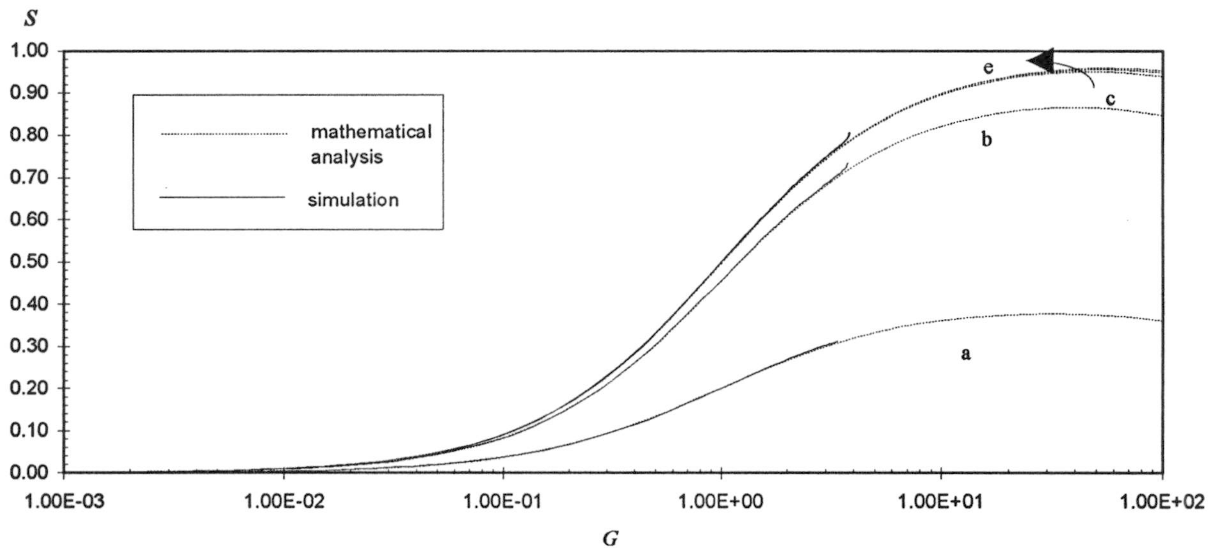

Fig. 8: Comparison between simulation and mathematical analysis for the unslotted protocol and varying SNR; a..e: SNR=6, 8, 10, 12 and 20 dB; number of codes=8; d=0.01; α=0.1

Fig. 8 shows the effect of the signal to noise ratio (SNR) on the throughput of the unslotted protocol. Again, simulation and mathematical analysis agree very well with each other. Of course, the throughput improves for increasing SNR. However, Fig. 8 also shows that a SNR of 12 dB suffice. For values of SNR higher than 12 dB, the protocol performance also improves, but only for high values of the offered traffic and the improvement is only additional.

VI. CONCLUSIONS

The unslotted hybrid CDMA/ISMA protocol has been investigated. The protocol performance is measured in terms of throughput S and delay D in relationship with the offered traffic G. Herefore, we have been able to derive a close form formula for the throughput and delay. Also a simulation program has been developed to compare the results. The results from simulation and mathematical agree very well with each other.

Comparison between the slotted and unslotted protocol shows that for low offered traffic the delay for the unslotted protocol is better than the slotted protocol. For high traffic it is the other way round. Furthermore it is concluded that the proposed hybrid protocol performs very well in case the propagation delay is not a concern. If the propagation delay is too high the performance drops quickly. The results also show that the hybrid protocol supports code sharing extremely good. For a 32 terminal network, only four codes suffice.

Thus it can be concluded that unslotted hybrid CDMA/ISMA protocol will be highly suitable for the indoor wireless computer communication.

REFERENCES

[1] R. Prasad,

"Performance Analysis of Mobile Packet Radio Networks in Real Channels with Inhibit-Sense Multiple Access",

IEE Proc. I, Vol. 138, pp. 458-464, October 1991.

[2] I. Widipangestu, A.J. 't Jong and R. Prasad,
"Capture Probability and Throughput Analysis of Slotted ALOHA and Unslotted np-ISMA in a Rician/Rayleigh Environment",
IEEE Transactions on Vehicular Technology, Vol. 43, No. 3, pp. 457-465, August 1994.

[3] K.J. Zdunek, D.R. Ucci and J.L. Locicero,
"Throughput of Nonpersistent Inhibit Sense Multiple Access with Capture",
Electronics Letters, Vol. 25, No. 1, pp. 30-31, 5th January 1989.

[4] R. Prasad, and C.Y. Liu,
"Throughput Analysis of Some Mobile Packet Radio Protocols in Rician Fading Channels",
IEE Proceedings-I, Vol. 139, No. 3, pp. 297-302, June 1992.

[5] M.G. Jansen, and R. Prasad,
"Capacity, Throughput, and Delay Analysis of a Cellular DS CDMA System With Imperfect Power Control and Imperfect Sectorization",
IEEE Transactions on Vehicular Technology, Vol. 44, No. 1, pp. 67-75, February 1995.

[6] R.A. Scholtz,
"The Origins of Spread-Spectrum Communications",
IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 822-854, May 1982.

[7] W.C.Y. Lee,
"Overview of Cellular CDMA",
IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 291-302, May 1991.

[8] M.B. Pursley,

"Performance Evaluation of Phase Coded Spread Spectrum Multiple Access Communication - Part I: System Analysis",

IEEE Transactions on Communications, Vol. COM-25, No. 8, pp. 795-799, August 1977.

[9] R.L. Pickholtz, D.L. Schilling and L.B. Milstein

"Theory of Spread Spectrum Communications - A Tutorial",

IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 855-884, May 1982.

[10] K.S. Gilhousen, I.M. Jacobs, R. Padovani, A.J. Viterbi, L.A. Weaver, Jr. and C.E.Wheatley III,

"On the Capacity of a Cellular CDMA System",

IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 303-312, May 1991.

[11] H. van Roosmalen, J. Nijhof and R. Prasad,

"Performance Analysis of a Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications",

IEEE Journal on Selected Areas in Communications, Vol. 12, No. 5, pp. 909-916, June 1994.

[12] L. Kleinrock and F.A. Tobagi,

"Packet Switching in Radio Channels: Part I- Carrier Sense Multiple Access and their Throughput-Delay Characteristics",

IEEE Trans. on Commun., Vol. COM-23, No. 12, pp. 1400-1416, December 1975.

[13] M. Kavehrad and B. Ramamurthi,

"Direct -Sequence Spread Spectrum with DPSK Modulation and Diversity for Indoor Wireless Communications", IEEE Transactions on Communications, Vol. COM-35, No. 2, pp. 224-236, February 1987.

[14] A. B. Carlson,

Communication Systems,

Mc. Graw Hill Book Company, 3rd edition, 1988, pp. 92.

APPENDIX C:

"Unslotted Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications"

Submitted for publication in IEEE Journal on Selected Areas of Communications.

Unslotted Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications

Huy Linh Anh Le, *Student Member, IEEE*

Huub van Roosmalen

Jos Nijhof, *Member, IEEE*

Ramjee Prasad, *Senior Member, IEEE*

Telecommunications and Traffic Control Systems Group

Delft University of Technology

P.O. Box 5031, 2600 GA Delft

The Netherlands

Abstract

A hybrid Code Division Multiple Access / Inhibit Sense Multiple Access (CDMA/ISMA) protocol has been proposed as an effective protocol for Indoor Wireless Computer Communications. This new protocol combines the advantages of both CDMA and ISMA into one protocol. On the one hand the ISMA protocol introduces a limitation to the number of simultaneous accesses to the transmission channel. On the other hand the CDMA protocol introduces an improvement to the packet survival chance. It is shown that the performance of the hybrid protocol is indeed better than CDMA only. In addition, code sharing can be applied to reduce hardware cost. To take more advantage of the strength of the hybrid protocol, we have investigated the *unslotted non-persistent* ISMA scheme instead of the *slotted p-persistent* ISMA scheme. The performance comparison between the slotted and unslotted hybrid CDMA/ISMA protocol is evaluated in terms of throughput and delay using computer simulation.

I. INTRODUCTION

On the one hand the Inhibit Sense Multiple Access (ISMA) protocol introduces a limitation to the number of simultaneous accesses to the transmission channel. On the other hand the Code Division Multiple Access (CDMA) protocol introduces an improvement to the packet survival chance. It is therefore to be expected that the combination of the strengths of the two protocols could improve performance. This concept is described below.

Indoor wireless office communication is our main research field. The office system we focus on consists of a building in which users work together in groups. The participants generate terminal traffic. Terminals communicate with each other by radio transmission using a random access protocol. The information is lost when two or more terminals transmit simultaneously, i.e. a collision occurs. Terminals might not detect each other's transmission in radio communications. It can easily happen that two users are hidden from each other by some obstacle, in which case severe performance degradation results. This is called the *hidden terminal problem*. The introduction of a central base station can alleviate this problem by instructing it to send a busy tone to all participating terminals to forbid new transmissions when a transmission is going on. Still, a situation can occur in which two or more terminals simultaneously start their transmission, resulting in a collision. However, a great reduction in the number of simultaneous transmissions is achieved by the introduction of a central base station. This concept is called the Inhibit Sense Multiple Access (ISMA) [1]-[4].

If we could somehow increase the survival chances of colliding packets, we could improve protocol performance. The near-far effect is one way to achieve this. A packet may 'capture' the receiver if it is much stronger than its competitors. This can happen when terminals use the same transmission power, but at different distances from the receiver. Because the 'near terminals' have better performance compared to the 'far terminals' (due to the better survival chance of the packets), the near-far effect introduces an unfair element among the terminals. Perfect power control, in which the transmitted power of the terminals are adjusted such that their received powers are all equal, can eliminate the near-far effect described above. The performance of CDMA has been reported in a number of publications, e.g. [5]-[10].

A hybrid CDMA/ISMA protocol has been proposed in [11]. This new protocol combines the advantages of both CDMA and ISMA into one protocol. The advantages of CDMA and ISMA are the improvement of the survival chance of packets and the limitation of contention in the channel. A less complex receiver structure is needed. More importantly, the number of distinct useful transmission codes is limited, especially for short code length.

In [11] the hybrid protocol combines Direct Sequence CDMA with *slotted* p-persistent ISMA. In this paper, we have investigated the performance of the hybrid CDMA/ISMA protocol using the *unslotted* non-persistent ISMA scheme. It is expected that for low traffic, the delay of the unslotted protocol will improve because when a terminal has a data packet to send, it does not have to wait until the start of the next time slot. In addition, even when data packets collide, there is a probability that the data is received correctly due to the use of CDMA. This is the strength of the hybrid CDMA/ISMA protocol. By using the *unslotted non-persistent* ISMA scheme, it is expected that we take more advantage of this strength.

This paper is organized as follows. Section II gives the protocol description of both the slotted and unslotted hybrid ISMA/CDMA protocol. Assumptions are given in Section III. Computer simulations and results are discussed in Section IV. Finally, conclusions can be found in Section V.

II. PROTOCOL DESCRIPTION

Figure 1 depicts the network structure of the protocol. There is one central base station that is connected to several receivers by wire. The base station controls the traffic flow with a busy tone that can be detected by all participating terminals. Because all terminals can detect the busy tone, the hidden terminal problem is solved. Several terminals together with one receiver form a group. Wireless communication is considered to take place between the receiver and the terminals. The terminals around each receiver share the same code. In this way the number of codes can be reduced. In this paper we assume that the several groups have the same groupsize; the distance from the central base station to the several receivers is the same and the distance between the terminals and the receiver within one group is also the same. When a data packet is ready for transmission, a terminal transmits the packet to its receiver according to the chosen ISMA scheme. Two ISMA schemes are important for us and have been discussed later on: *slotted* p-persistent ISMA protocol and *unslotted* non-persistent ISMA protocol.

After the arrival of the data packet, the receiver then simply forwards this packet to the base station which broadcasts it to all terminals. The destination then receives the packet and decides whether the packet is errorless or not. In case the packet was erroneous, a retransmission must take place. The return channel is not included in our analysis because only the base station makes use of this return channel and therefore contention is not a concern. The state of the terminals can either be free or blocked. At the beginning, the terminals are in the free state. If a packet arrives at a free terminal, the terminal jumps into the blocked state. In the blocked state, the terminal takes care that the arriving data packet is serviced successfully. In the mean time the blocked terminal ignores all incoming packets. This is a consequence of the assumption that the terminals do not have buffers for the incoming data packets. The arrivals of data packets at terminals are assumed to be generated by a Poisson process. This means that the inter-arrival times of data packets at terminals are negative exponentially distributed with P_{Arr} as parameter (P_{Arr} indicates the average number of arrivals per terminal per time unit).

A. The slotted p-persistent ISMA protocol

Users behind terminals generate data packets. The data packets arrive at terminals, which will take care of the correct delivery of the packets. We assume that the arrivals of these packets are generated by a Poisson process. The Specification and Description Language (SDL) like diagram of the slotted p-persistent ISMA protocol is depicted in Figure 2. In this case, the terminal uses the slotted p-persistent ISMA protocol to service its data packet when it is in the blocked state. Before transmitting the data packet to the receiver, the terminal first listens to the channel to hear whether there is a busy tone transmitted by the base station. The base station broadcasts a busy tone to signal that the channel is busy because there is a transmission in progress. If there is a busy tone, indicating that the channel is busy, the terminal postpones its transmission to the next time slot. Otherwise, if the channel is free, the terminal performs a random binary experiment to determine whether it will send its data packet. If the terminal decides not to transmit, it waits until the next time slot and draws a number again to see if it is permitted to send. This mechanism is built in to reduce the number of simultaneous transmissions. Simultaneous transmissions cause collisions and must be reduced as much as possible.

If the terminal is permitted to send, it transmits its data packet to the receiver. It is possible that two or more terminals transmit in the same time slot. In this case a collision occurs. Because we also make use of CDMA, a collision does not necessarily result in the loss of the data packets. There is a probability that the receiver can lock onto one of the data streams and this data packet can be received properly. In case the data packet is received erroneously, the terminal then waits for a random period and starts all over again (see also Figure 2). On the other hand, if no collision occurs, the terminal assumes that the packet has been received correctly and leaves the blocked state to jump into the free state. The random delay for retransmissions is set to zero in our experiment because the random binary experiment already introduces a delay after a collision.

B. The unslotted non-persistent ISMA protocol

The SDL-like diagram of the unslotted non-persistent ISMA protocol is shown in Figure 3. This ISMA protocol is unslotted. There are two major differences between this unslotted and the previously described slotted protocol. The first difference is that the terminal waits a random period when it senses that the channel is busy. Because the time axis is not divided into time slots, terminals have to wait a random period. After the random delay, the terminals are allowed to sense the channel again to see if it is free. The random period is modeled to be negative exponentially distributed. The second difference is related to the first one. In the unslotted scheme, the terminals immediately transmit if the channel is free. As a consequence of the first difference mentioned above, the terminals already wait a random period in case the channel is busy. So, if the channel is free the number of terminals attempting to transmit is already reduced. It is not necessary to let the terminals draw a random number to see if they may transmit. Although the number of collisions is greatly reduced with ISMA, collisions can still occur. In this unslotted ISMA scheme, collisions may occur due to multiple terminals transmitting packets during an interval called the *inhibit delay fraction* (d). This interval d is necessary to switch from 'idle' to 'busy'. The reverse interval from 'busy' to 'idle' is denoted by d'. The inhibit delay fraction is normalized to the packet length, resulting in $0 \leq d < 1$ (Figure 4). In this paper we assume $d = d'$.

III. ASSUMPTIONS

A. Transmitter and Receiver

After DPSK modulation the signal is spread with a Gold code. The length of these codes are chosen to be 31 and 127 to evaluate the influence of code length. The signal is then transmitted, modulated on a 1.7 GHz carrier. The data rate is arbitrarily chosen to be 256·1024 b/s (0.26 Mb/s). The packet length is 64 bits. Perfect power control is assumed to assure that signals from terminals within the same group arrive at the receiver with the same power.

We consider one antenna for each receiver. So, only spread spectrum diversity is taken into account. The number of resolvable paths L can be found with the following formula [12]:

$$L = \left\lfloor \frac{T_m}{T_c} \right\rfloor + 1 \qquad (1)$$

where T_m is the RMS delay spread, T_c is the chip duration and $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x. The relationship between T_c the chip duration and T_b the bit duration is given by the code length N:

$$T_c = \frac{T_b}{N} \qquad (2)$$

We have a number of resolvable paths due to spread spectrum. The maximum value for T_m has been measured by Saleh and Valenzuela and is 100 ns [13]. For our work we assume T_m to be 100 ns. For $N = 31$ this gives one resolvable path, and for $N = 127$ this gives us four resolvable paths.

B. Signal Attenuation Model

The far field model is chosen to describe the signal attenuation [14]. Given the transmitted power P_T, the received power P_R can be expressed by the following equation:

$$P_R = \frac{g_T g_R}{\left(\frac{4\pi f l}{c}\right)^{\alpha}} P_T \tag{3}$$

where g_T and g_R are respectively the transmitter and the receiver gains, f the signal frequency, l the distance between the transmitter and the receiver, c the speed of light and α is the attenuation parameter. This attenuation parameter is chosen equal to 2, corresponding to free space propagation.

C. Rayleigh Fading Channel

Fading is the result of the propagation of the transmitted signal through several paths. The channel is modeled as a Rayleigh fading channel. The Rayleigh fading channel model is valid when each path contributes the same amount of energy to the composite received signal. A Line of Sight (LOS) path is therefore assumed to be absent. The formula for the bit error probability is given by Kavehrad and Ramamurthi [15]:

$$P_{be|\beta_{\max},\{\tau_{lk}\},L} = Q(a,b) - \frac{1}{2}\left(1 + \frac{\mu_{12}}{\sqrt{\mu_1 \mu_2}}\right) I_0(a,b) \exp\left(-\frac{a^2+b^2}{2}\right) \tag{4.1}$$

where

$Q(a,b)$ is the Marcum Q-function

$$a = \frac{m}{\sqrt{2}}\left|\frac{1}{\sqrt{\mu_1}} - \frac{1}{\sqrt{\mu_2}}\right|, \text{ and } \quad b = \frac{m}{\sqrt{2}}\left|\frac{1}{\sqrt{\mu_1}} + \frac{1}{\sqrt{\mu_2}}\right| \tag{4.2}$$

$$m = A\beta_{\max} T_b b_1^{\ 0} = A\beta_{\max} T_b b_1^{\ -1} \tag{4.3}$$

$$\mu_1 = A^2 E\left[\sum_{k=1}^{Nt} X_k^{\ 2} + \hat{X}_k^{\ 2} + Y_k^{\ 2} + \hat{Y}_k^{\ 2} \middle| \{\tau_{lk}\}, L\right]$$
$$+2A^2 E\left[X_1\hat{X}_1 + Y_1\hat{Y}_1 \middle| \{\tau_{lk}\}, L\right] + 2\sigma_n^{\ 2} \tag{4.4}$$

$$\mu_2 = A^2 E\left[\sum_{k=1}^{Nt} X_k^{\ 2} + \hat{X}_k^{\ 2} + Y_k^{\ 2} + \hat{Y}_k^{\ 2} \middle| \{\tau_{lk}\}, L\right] + 2\sigma_n^{\ 2} \tag{4.5}$$

$$\mu_{12} = A^2 E\left[\sum_{k=1}^{Nt} \left(X_k\hat{X}_k + Y_k\hat{Y}_k\right) + \hat{X}_1^{\ 2} + \hat{Y}_1^{\ 2} \middle| \{\tau_{lk}\}, L\right] \tag{4.6}$$

In which $\sigma_n^{\ 2}$ is the thermal noise power; A is the transmitted signal amplitude; Nt denotes the number of active users; L is the number of paths; β_{lk} and τ_{lk} are the lth path gain and delay respectively, for the kth user; β_{max} denotes the largest value of β_{l1}. The conditional expectations in the above expressions can be evaluated as follows:

$$E\left[X_1^2 \middle| \{\tau_{lk}\}, L\right] = E\left[Y_1^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{\substack{l=1 \\ l\neq j}}^{L} R_{11}^2(\tau_{l1})\rho_{l1} \tag{5.1}$$

$$E\left[\hat{X}_1^2 \middle| \{\tau_{lk}\}, L\right] = E\left[\hat{Y}_1^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{\substack{l=1 \\ l\neq j}}^{L} \hat{R}_{11}^2(\tau_{l1})\rho_{l1} \tag{5.2}$$

$$E\left[X_1\hat{X}_1 \middle| \{\tau_{lk}\}, L\right] = E\left[Y_1\hat{Y}_1 \middle| \{\tau_{lk}\}, L\right] = \sum_{\substack{l=1 \\ l\neq j}}^{L} R_{11}(\tau_{l1})\hat{R}_{11}(\tau_{l1})\rho_{l1} \tag{5.3}$$

$$E\left[X_k^2 \middle| \{\tau_{lk}\}, L\right] = E\left[Y_k^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{l=1}^{L} R_{1k}^2(\tau_{lk})\rho_{lk} \tag{5.4}$$

$$E\left[\hat{X}_k^2 \middle| \{\tau_{lk}\}, L\right] = E\left[\hat{Y}_k^2 \middle| \{\tau_{lk}\}, L\right] = \sum_{l=1}^{L} \hat{R}_{1k}^2(\tau_{lk})\rho_{lk} \qquad ;k \geq 2 \tag{5.5}$$

$$E\left[X_k \hat{X}_k \middle| \{\tau_{lk}\}, L\right] = E\left[Y_k \hat{Y}_k \middle| \{\tau_{lk}\}, L\right] = \sum_{l=1}^{L} R_{1k}(\tau_{lk}) \hat{R}_{1k}(\tau_{lk}) \rho_{lk} \qquad ;k \geq 2 \tag{5.6}$$

In which

$$R_{1k}(\tau) = \int_0^{\tau} a_k(t-\tau) a_1(t) dt \tag{5.7}$$

$$\hat{R}_{1k}(\tau) = \int_{\tau}^{T} a_k(t-\tau) a_1(t) dt \tag{5.8}$$

$$a_k(t) = \sum_i a_k^i P_{T_c}(t - iT_c) \tag{5.9}$$

$$b_k(t) = \sum_{j=-\infty}^{\infty} b_k^j P_T(t - jT_b) \tag{5.10}$$

and $$\rho_{lk} = \frac{1}{2} E\left[\beta_{lk}^2\right] \tag{5.11}$$

D. Network Configuration

An example of a 16-terminal network configuration that we adopted in this paper is shown in Figure 5. The terminals are clustered around distributed receivers at a fixed distance (5 m). The distributed receivers are clustered around the base station at a fixed distance (30 m). Those distances are kept fixed for all simulations. The number of terminals within a receiver group (also called group size) is chosen as a power of two. For a certain fixed number of participated terminals, we have to halve the group size if we want to double the number of codes (the number of codes is also the number of receivers). The comparison is fair in this manner, because when we want to investigate the effect of the number of codes on the performance, we have to keep the number of participated terminals fixed.

IV. COMPUTER SIMULATION AND RESULTS

In this section the performance of the slotted p-persistent and the unslotted non-persistent ISMA scheme is given. The performance analysis for the slotted scheme has been evaluated by computer simulation and Markov model in [11]. In this paper we concentrate on the performance comparison between the slotted and unslotted hybrid CDMA/ISMA protocol through computer simulation.

A1. Representation of Time

As a first approach to analyse the *unslotted* hybrid CDMA/ISMA protocol, a computer simulation program will be set up. The dynamic behaviour of the system is studied by tracing various system states as a function of time and then collecting and analysing the system statistics. The events that change the system state are generated at different points in time, and the passage of time is represented by an internal clock which is incremented and maintained by the simulation program.

The simulation time can be advanced in two ways. The first method is the *interval-oriented simulation* (or the uniform time increment method) where the clock is advanced from time t to $(t + \Delta t)$ where Δt is a uniform fixed time increment. Figure 6a depicts this mechanism. The second method is the *event-oriented simulation* (or the variable time increment method) where the clock is incremented from time t to the next event time t', whatever may be the value of t'. The state changes are made at event time t, the next event time t', and this process is continuously repeated. Thus, only events are represented explicitly in a simulation model and the periods between events are treated as inactive or insignificant and therefore consume no time even though the interevent activites do consume time in the real world (Figure 6b).

Obviously, method 1 detects the events that occur during the interval $(t, t + \Delta t)$ only at time $(t + \Delta t)$, thereby introducing errors in simulation. Another drawback of this method is that if the interval between two events is very large compared to Δt, then the simulator goes through several unproductive clock increments (during periods of inactivity) and the associated computing effort which will not bring about any change in system states. This fixed time increment method is suitable for the simulation of continuous systems and in particular systems with large numbers of state variables.

The second method involves sorting of event activation times and maintaining *events list*. In our work we will use the event-oriented simulation (Figure 6b) to simulate the hybrid unslotted CDMA/ISMA protocol.

A2. Main Structure

The Program Structure Diagram (PSD) of the main program is depicted in Figure 7. The simulation program simulates the working of the hybrid unslotted CDMA/ISMA protocol. First of all, the counters have to be reset to the proper values. *Clock* indicates the actual time while the simulation will last *EndTime*. While *Clock* has not achieved *EndTime*, the simulation will loop. The loop is divided in four steps as shown in Figure 7. In the first step, the program advances *Clock* to the next event time. This event is the first event to occur in future time. Because the event-oriented simulation is used, only events are represented explicitly in the simulation model. Depending on the event type, the program processes this event. This is the second step. The processing can generate other events or change the system state. In the third step, the generated events and the changed system state will be updated. Finally, statistics collection completes the loop.

A3. Event Lists

From Figure 7 it emerges that events and event lists play an important role in the main program structure. The whole program is based on the event lists. The event lists consist of events linked together in an organised manner. Here, organised means in an order in which the events occur. An example of the event list is shown in Figure 8. The event lists consist of nodes linked together to form a chain. In the event oriented simulation this chain symbolizes the time axis. Every node represents an event and consists of an *EventType*, the time this event occur, a tail and other information. The tail consists of a pointer pointing to the next node. The last node points to nowhere. This is illustrated with the ground symbol.

B. Performance Measurements

The protocol performance has been measured in terms of throughput S and delay D in relationship with the arrival rate at terminals P_{Arr}. Note that in [11] the physical traffic is used instead of P_{Arr} (see also Figures 2 and 3 for the physical traffic reference). The physical traffic indicates the number of terminals that actually transmit data packets to the receiver.

The throughput S measures the efficiency of the protocol and is defined as the fraction of time in which correct data packets are received. The delay D is the time between the arrival of the first bit of a data packet at a terminal and its arrival of the last bit at the destination.

First, the performance inherent to the unslotted non-persistent scheme will be presented. After that, we present the performance comparison between the slotted and unslotted scheme.

C. Performance of the unslotted scheme

As discussed before, the inhibit delay fraction (*d*) is one of the parameters that influences the protocol behavior. By varying this inhibit delay fraction and measuring the performance, Figure 9 and Figure 10 are obtained. Figure 9 depicts the throughput of the unslotted scheme as a function of the arrival rate at terminals for different values of the inhibit delay fraction. This inhibit delay fraction *d* is varied between 0.01 to 0.2 (note that *d* is normalized and can vary between zero and one). For low arrival rates, the throughput curves hardly differ. This can be explained by the fact that for low arrival rates, the probability that more than one packet arrives within the inhibit delay fraction is very low. On the other hand, for high traffic, the throughput drops for high value of *d*. The throughput degradation is due to the collisions of data packets. Because for high traffic, more packets arrive within the inhibit delay fraction if *d* is too large.

The same explanation applies to the delay curves of the unslotted scheme which is shown in Figure 10. In this figure, the delay is depicted versus the arrival rate at terminals for different values of the inhibit delay fraction. The delay is normalized in the number of packet durations for comparison purposes with the slotted protocol later on. As can be expected from Figure 9, the delays for high traffic get worse for high values of the inhibit delay fraction *d*.

Another parameter which influences the performance of the unslotted hybrid protocol is the random delay that can be found in Figure 3. Before a retransmission (caused by a collision or a busy channel) can take place, terminals have to undergo a random delay first. This is to prevent too many terminals from retransmitting their data packets at the same time. We model this random process to be negative exponentially distributed with λ as parameter. λ represents the mean number of arrivals per packet duration (the time is normalized in packet duration for comparison purposes) and influences the period the data packet has to wait before it can be retransmitted. The average time a packet has to wait before it can be retransmitted is then given by: E[random delay]=$1/\lambda$ [16]. As a result of varying λ, we obtain the performance shown in Figures 11 and 12.

For different values of λ, Figure 11 depicts the throughput of the unslotted scheme as a function of the arrival rate at terminals. $\lambda=0.1$ means that on average 0.1 packets per packet duration will become available for retransmission at each blocked terminal (the time is normalized in packet durations). This results in an average delay of 10 packet durations before a retransmission takes place. For low values of λ, the throughput is also low. This is because erroneous packets have to wait a long period before they may be retransmitted. The same explanation also applies to Figure 12 in which the delay is depicted versus the arrival rate.

C. Comparison between the slotted and unslotted scheme

The compasison between the slotted and unslotted hybrid CDMA/ISMA protocol is presented in Figures 13 and 14. The throughput comparison between the slotted and unslotted hybrid protocol is shown in Figure 13. The number of users is kept constant to 32. The parameter we vary in those curves is the number of codes used (this is also the number receivers or groups in the network). The range in which we vary this parameter is from 1 to 32. For low number of codes used, the performance is low for both slotted and unslotted scheme. Low number of codes used means a high number of users shares the same code because the number of users is kept constant. Here we see the effect of code sharing. For low arrival rates, code sharing has not much influence because the probability that the terminals in the same group start sending at the same time is very low.

For high arrival rates, the throughput performance of the unslotted scheme is better than the slotted one. The explanation can be found in the characteristics of the retransmission mechanism for both schemes. For the slotted scheme this is a random binary experiment and for the unslotted scheme this is the Poisson process with the the negative exponential distribution. In addition, for the unslotted scheme, data packets can be sent immediately if terminals are allowed to and do not have to wait to the start of a slots as in the slotted scheme. The delay characteristics of both the slotted and unslotted schemes are depicted in Figure 14.

V. CONCLUSIONS

The protocol performance is measured in terms of throughput S and delay D in relationship with the arrival rate at terminals P_{Arr}. The performance of the unslotted non-persistent hybrid CDMA/ISMA protocol has been investigated by computer simulation. Two decisive parameters which influences the protocol performance have been analyzed: the inhibit delay fraction and the average time a packet has to wait to be retransmitted. The increase of these parameters have a negative impact on both throughput and delay for high arrival rates.

The performance comparison between the slotted p-persistent and the unslotted non-persistent hybrid CDMA/ISMA protocol is also considered. For high traffic, the performance of the unslotted protocol is significantly better than that of the slotted protocol. This is caused by differences in the retransmission algorithms of both protocols. The unslotted protocol also shows superiority compared to the slotted protocol in terms of packets delay. This is because for the unslotted scheme, data packets can be sent immediately if terminals are allowed to and do not have to wait for the start of a slot as in the slotted scheme.

Computer simulations show that the unslotted scheme is the scheme of choice in the hybrid CDMA/ISMA protocol. The unslotted hybrid protocol will be investigated using a Markov model in future work.

REFERENCES

[1] R. Prasad,
"Performance Analysis of Mobile Packet Radio Networks in Real Channels with Inhibit-Sense Multiple Access",
IEE Proc. I, Vol. 138, pp. 458-464, October 1991.

[2] I. Widipangestu, A.J. 't Jong and R. Prasad,
"Capture Probability and Throughput Analysis of Slotted ALOHA and Unslotted np-ISMA in a Rician/Rayleigh Environment",
IEEE Transactions on Vehicular Technology, Vol. 43, No. 3, pp. 457-465, August 1994.

[3] K.J. Zdunek, D.R. Ucci and J.L. Locicero,
"Throughput of Nonpersistent Inhibit Sense Multiple Access with Capture",
Electronics Letters, Vol. 25, No. 1, pp. 30-31, 5th January 1989.

[4] R. Prasad, and C.Y. Liu,
"Throughput Analysis of Some Mobile Packet Radio Protocols in Rician Fading Channels",
IEE Proceedings-I, Vol. 139, No. 3, pp. 297-302, June 1992.

[5] M.G. Jansen, and R. Prasad,
"Capacity, Throughput, and Delay Analysis of a Cellular DS CDMA System With Imperfect Power Control and Imperfect Sectorization",
IEEE Transactions on Vehicular Technology, Vol. 44, No. 1, pp. 67-75, February 1995.

[6] R.A. Scholtz,
"The Origins of Spread-Spectrum Communications",
IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 822-854, May 1982.

[7] W.C.Y. Lee,
"Overview of Cellular CDMA",
IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 291-302, May 1991.

[8] M.B. Pursley,
"Performance Evaluation of Phase Coded Spread Spectrum Multiple Access Communication - Part I: System Analysis",
IEEE Transactions on Communications, Vol. COM-25, No. 8, pp. 795-799, August 1977.

[9] R.L. Pickholtz, D.L. Schilling and L.B. Milstein
"Theory of Spread Spectrum Communications - A Tutorial",
IEEE Transactions on Communications, Vol. COM-30, No. 5, pp. 855-884, May 1982.

[10] K.S. Gilhousen, I.M. Jacobs, R. Padovani, A.J. Viterbi, L.A. Weaver, Jr. and C.E.Wheatley III,
"On the Capacity of a Cellular CDMA System",
IEEE Transactions on Vehicular Technology, Vol. 40, No. 2, pp. 303-312, May 1991.

[11] H. van Roosmalen, J. Nijhof and R. Prasad,
"Performance Analysis of a Hybrid CDMA/ISMA Protocol for Indoor Wireless Computer Communications",
IEEE Journal on Selected Areas in Communications, Vol. 12, No. 5, pp. 909-916, June 1994.

[12] C.A.F.J. Wijffels, H.S. Misser, and R. Prasad,
"A Micro-Cellular CDMA System over Slow and Fast Rician Fading Channels with Forward Error Correcring Coding and Diversity",
IEEE Transactions on Vehicular Technology, Vol. 42, No. 4, pp.570-580, November 1993.

[13] A.A.M. Saleh and R.A. Valenzuela,
"A Statistical Model for Indoor Multipath Propagation",
IEEE Journal on Selected Areas on Communications, Vol. 5, pp. 138-146, February 1987.

[14] A.B. Carlson,
<u>Communication Systems,</u>
McGraw-Hill International Editions, 1986

[15] M. Kavehrad and B. Ramamurthi,
"Direct -Sequence Spread Spectrum with DPSK Modulation and Diversity for Indoor Wireless Communications",
IEEE Transactions on Communications, Vol. COM-35, No. 2, pp. 224-236, February 1987.

[16] F.C. Schoute,
Prestatie-Analyse van Telecommunicatiesystemen,
Kluwer Technische Boeken B.V., 1989

List of Figure Captions

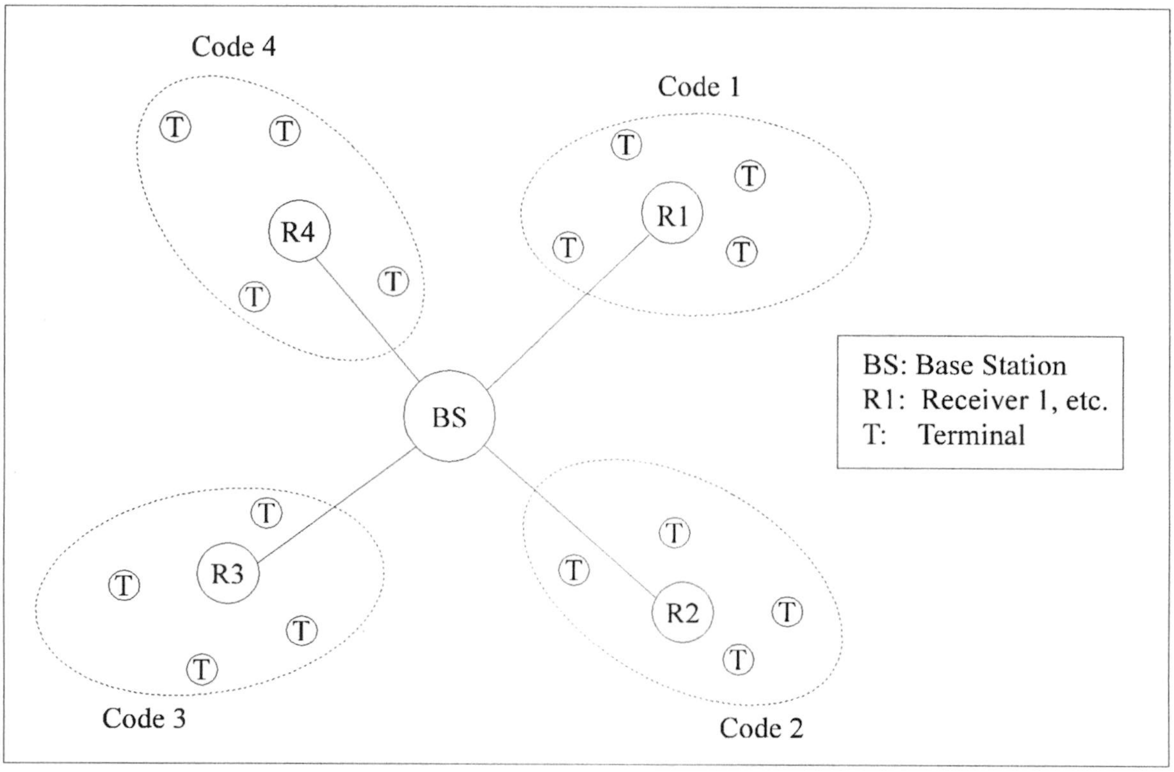

Figure 1: Network structure

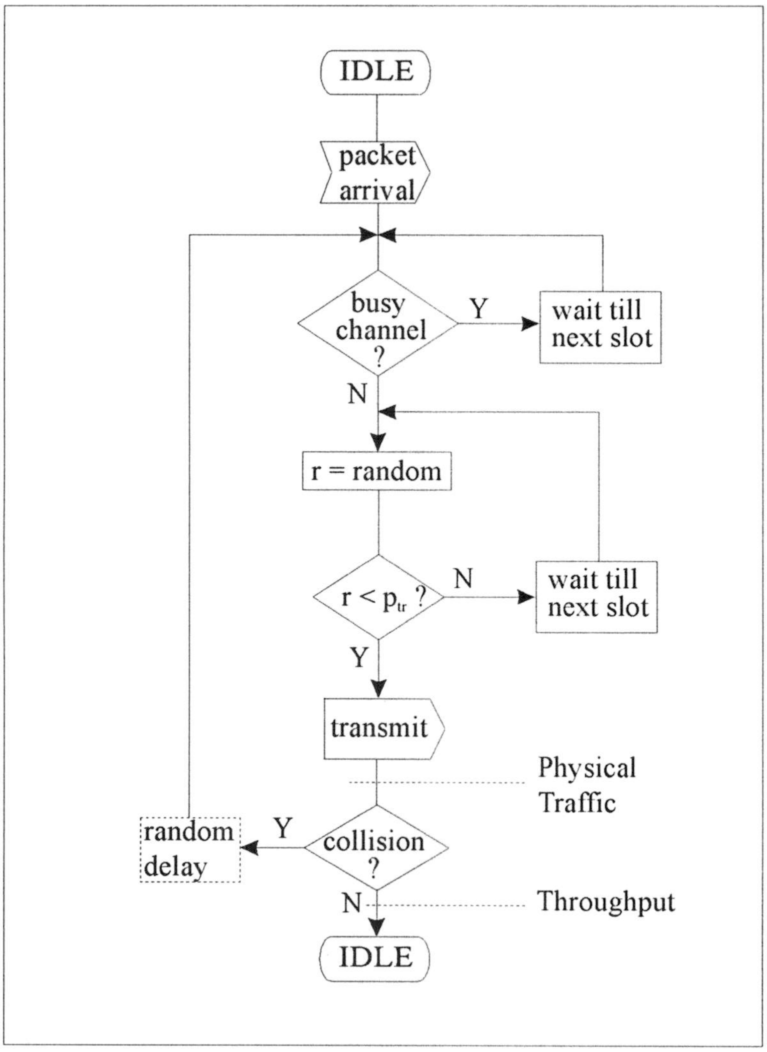

Figure 2: Slotted p-persistent ISMA scheme

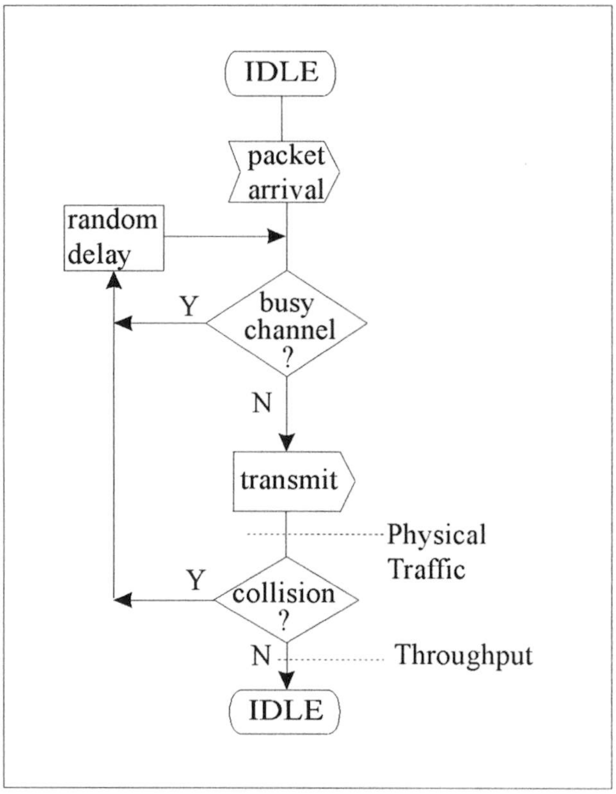

Figure 3: Unslotted non-persistent ISMA scheme

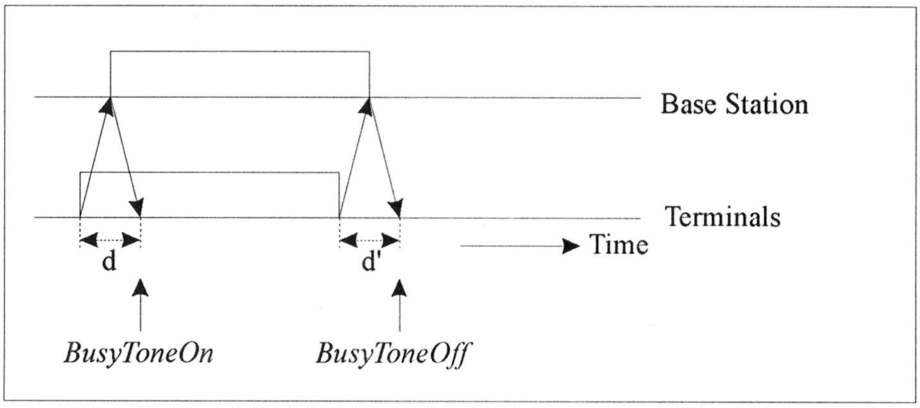

Figure 4: Inhibit delay fraction

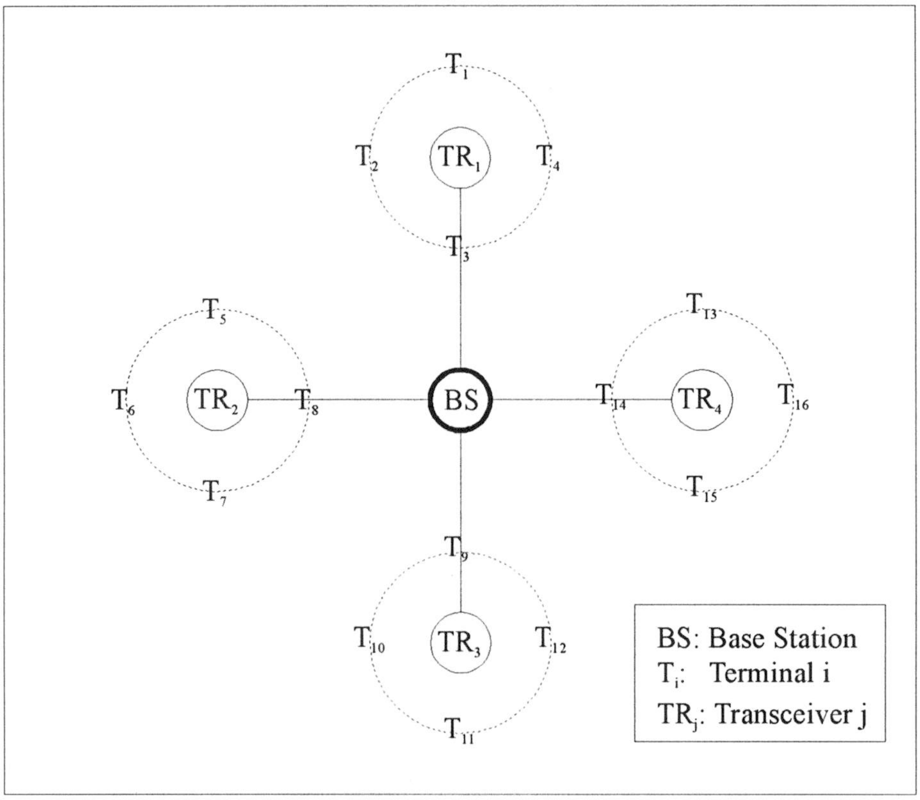

Figure 5: 16-terminal network configuration

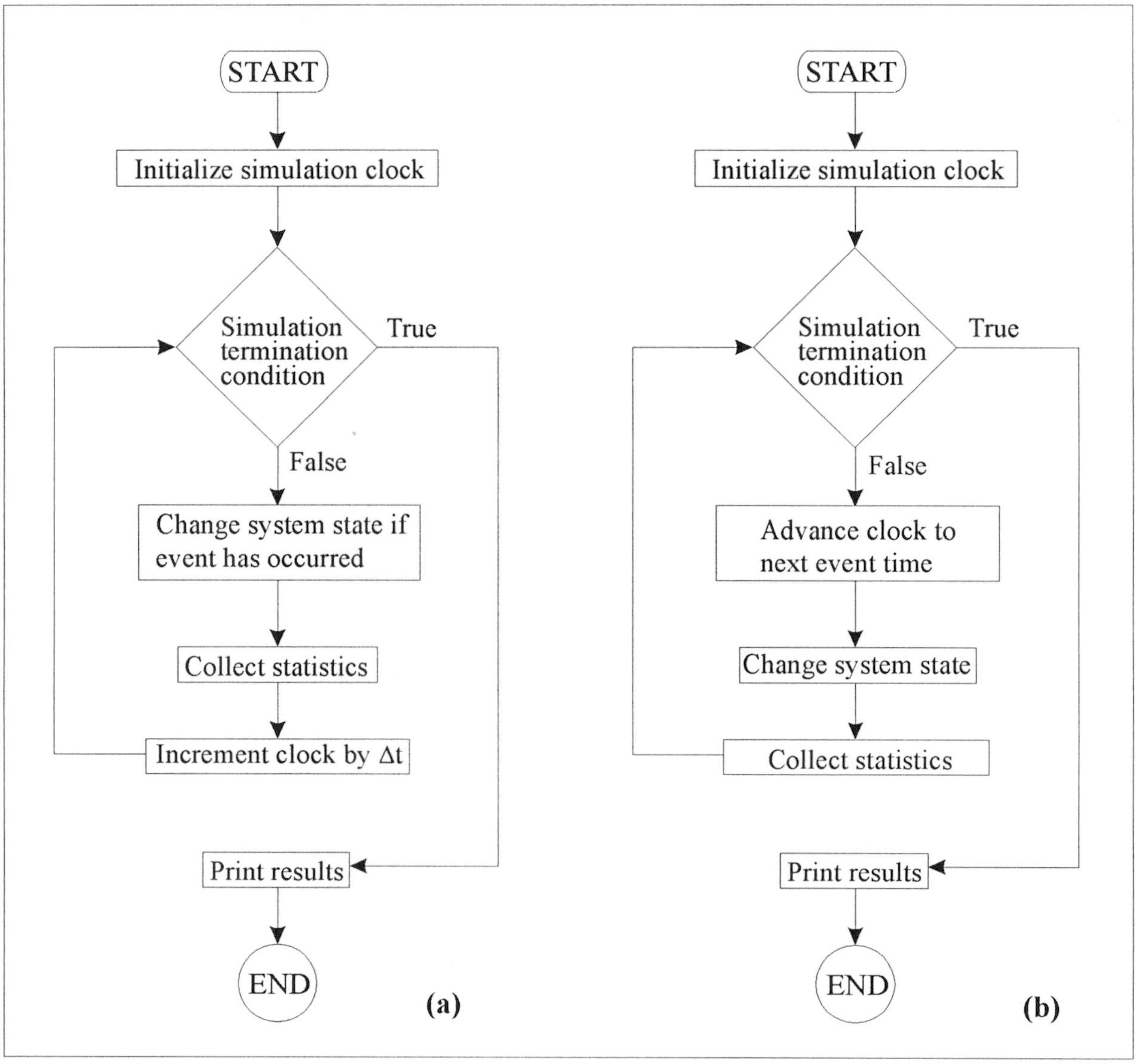

Figure 6: (a) Interval-oriented simulation; (b) Event-oriented simulation

Reset Counters

While *Clock* < *EndTime* Do

- Advance *Clock* to next event time
- Process Event
- Update Events List
- Collect Statistics

Figure 7: PSD of main program

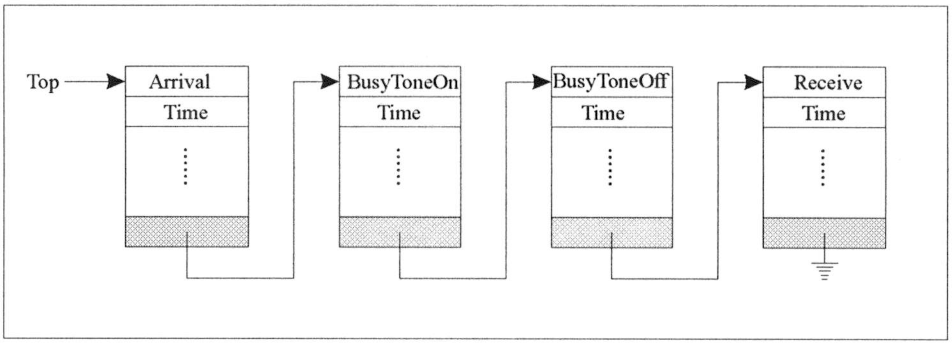

Figure 8: Example of an event list

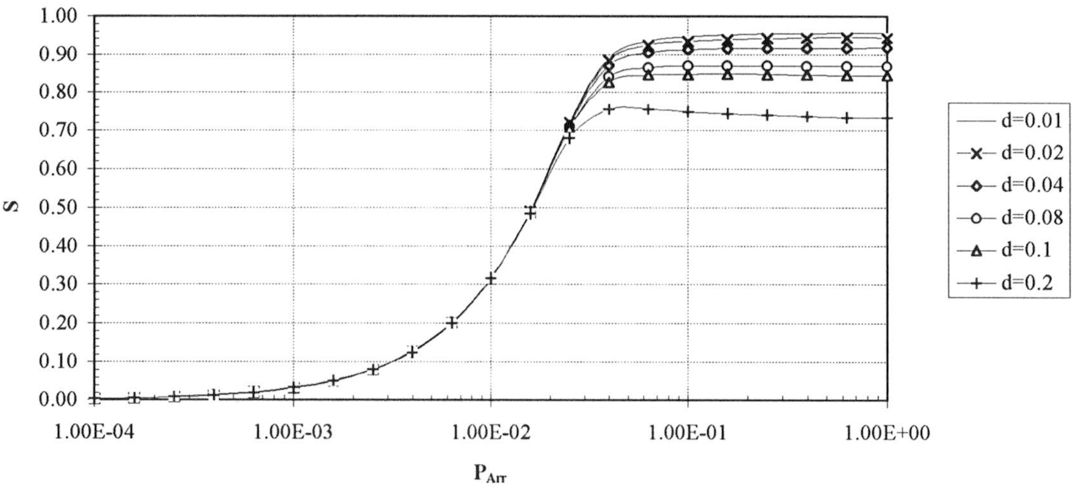

Figure 9: Throughput S versus arrival rate P_{Arr} for different values of the inhibit delay fraction *d* parameter for the unslotted hybrid CDMA/ISMA protocol; Code length = 31; SNR = 20 dB; Number of groups = 8; Group size = 4.

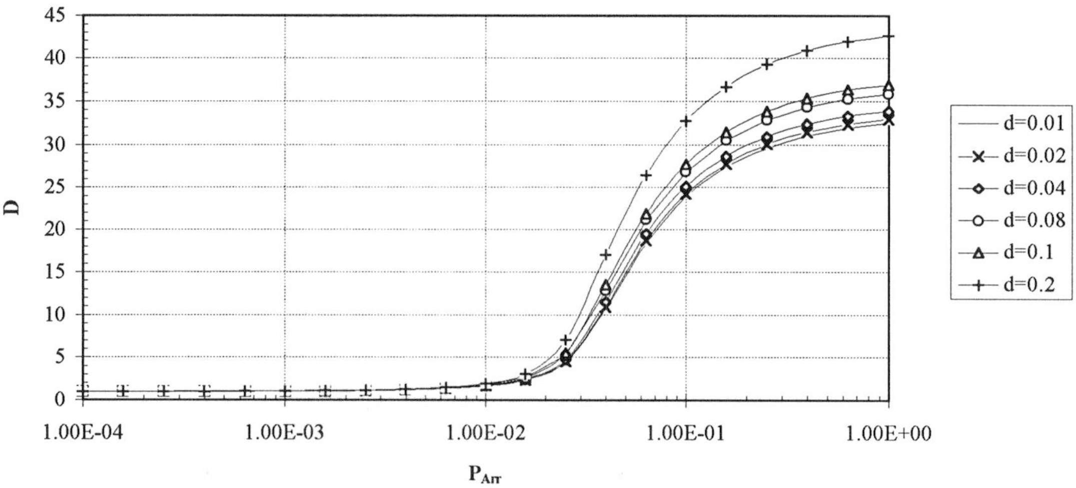

Figure 10: Delay D versus arrival rate P_{Arr} for different values of the inhibit delay fraction *d* parameter for the unslotted hybrid CDMA/ISMA protocol; Code length = 31; SNR = 20 dB; Number of groups = 8; Group size = 4.

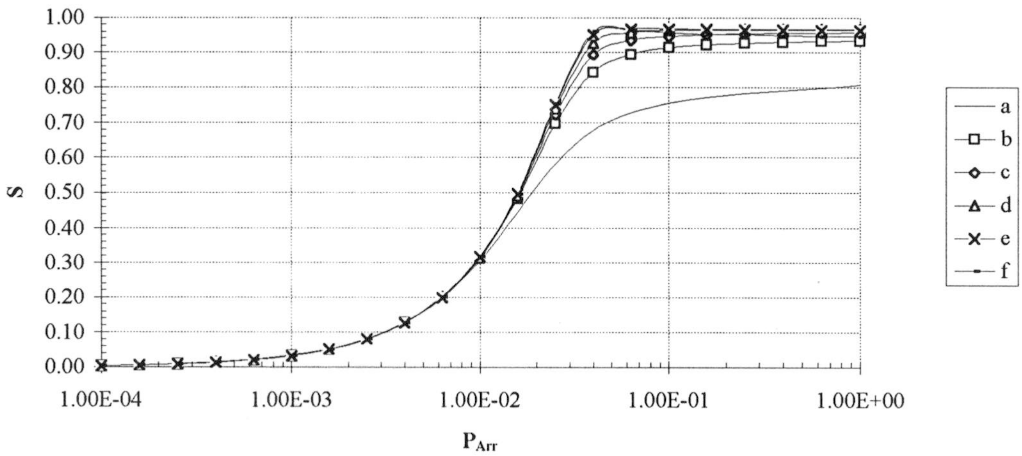

Figure 11: Throughput S versus arrival rate P_{Arr} for different values of λ for the unslotted hybrid CDMA/ISMA protocol; Code length = 31; SNR = 20 dB; a..f : 0.1, 0.5, 1, 2, 5, 10 for λ.

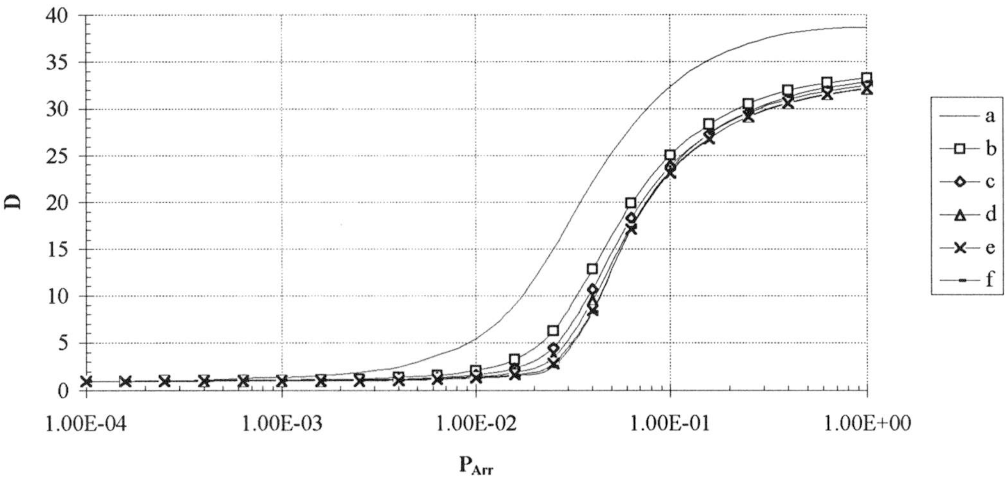

Figure 12: Delay D against arrival rate P_{Arr} for different values of λ for the unslotted hybrid CDMA/ISMA protocol; Code length = 31; SNR = 20 dB; Number of groups = 8; Group size = 4; a..f : 0.1, 0.5, 1, 2, 5, 10 for λ.

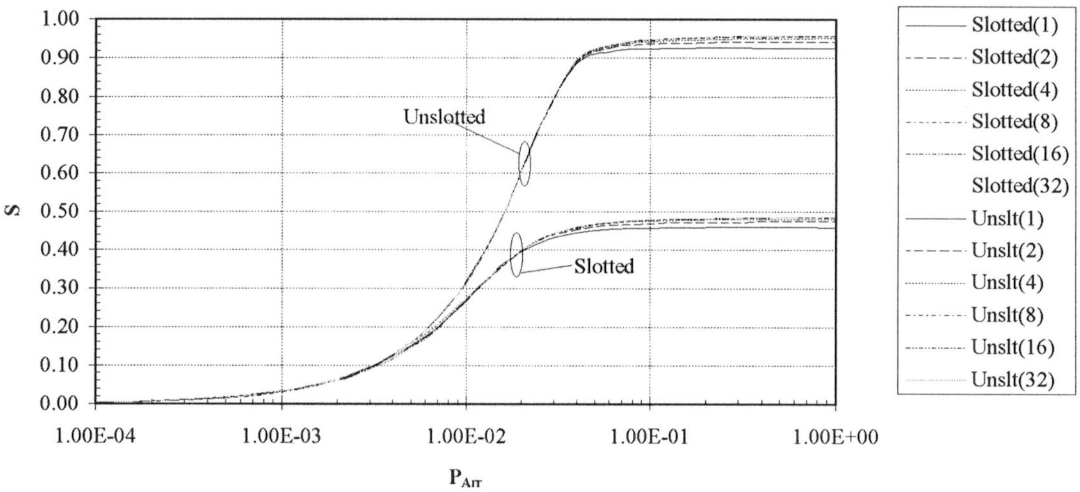

Figure 13: Throughput S versus arrival rate P_{Arr} for comparison between slotted and unslotted hybrid CDMA/ISMA protocol; Code length = 31; $\lambda=1$; d=0.01; Number of users = 32. The number of codes used vary from 1 to 32. Slotted(1) means the performance curve with 1 code for the slotted scheme, etc.

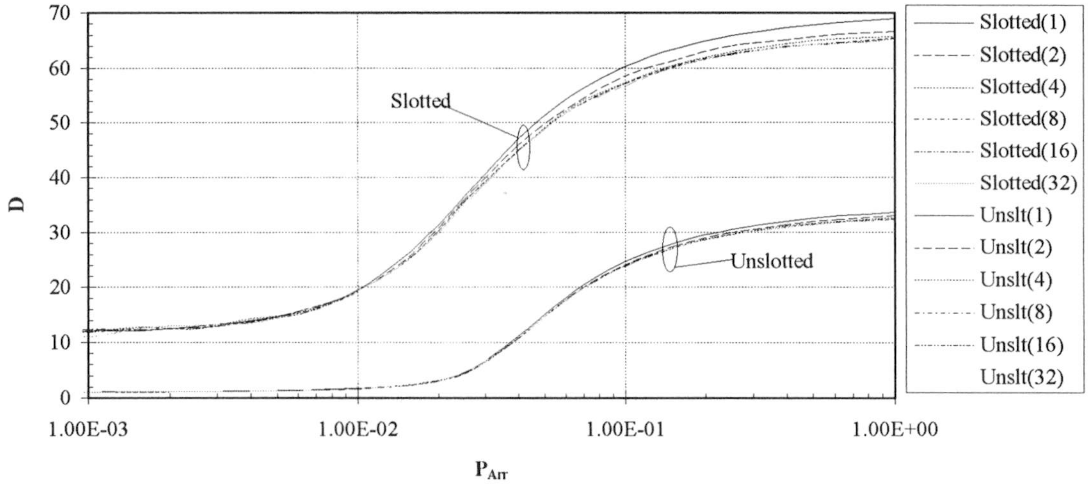

Figure 14: Delay D versus arrival rate P_{Arr} for comparison between slotted and unslotted hybrid CDMA/ISMA protocol; Code length = 31, λ=1 and d=0.01; Number of users = 32; The number of codes used varies from 1 to 32. Slotted(1) means the performance curve with 1 code, etc.

APPENDIX D:

Listings of Computer Programs

Three computer programs are included in this Appendix and are listed in the following order:

1. **SimSlotted**

   This is a computer simulation program which simulates the behaviour of the slotted hybrid CDMA/ISMA protocol.

2. **SimUnslotted**

   This is a computer simulation program which simulates the behaviour of the unslotted hybrid CDMA/ISMA protocol.

3. **CalcUnslotted**

   This is a computer program which calculates the performance of the unslotted hybrid CDMA/ISMA protocol.

```
{*****************************************************************************}
{*  SimSlotted is a program designed with the goal to simulate the           *}
{*  'Slotted Hybrid CDMA/ISMA protocol'. This program calculates the         *}
{*  throughput and delay of a network of terminals by simulation.            *}
{*                                                                           *}
{*  Most of the constants, variables, functions and procedures used here     *}
{*  originates from Huub van Roosmalen.                                      *}
{*                                                                           *}
{*  The terminals are clustered around distributed receivers at a fixed      *}
{*  distance. The distributed receivers are clustered around the base        *}
{*  station at a fixed distance. The packets from the terminals are          *}
{*  received by the distributed receivers and sent to the base station by    *}
{*  wire or lines. These lines can be seen as primitive concentrators        *}
{*                                                                           *}
{*  Huy Linh Anh Le, August 10 1995                                          *}
{*****************************************************************************}
Program SimSlotted(Input, Output);

Uses    Dos, Crt, Ultrax;

Const   MaxTerm        =            32; { Maximum number of terminals allowed  }
        PacketLength   =            64; { Number of bits per packet            }

        Pt             =             1; { Transmitter power in Watts           }
        LossExp        =             2; { Attenuation parameter for the ...    }
                                        {  ...far field model                  }
        c              =     299792458; { Speed of light in m/s                }
        f              =    1700000000; { Transmitter center frequency         }
        lambda         =           c/f; { Signal wavelength                    }

        twoPi          =        2 * Pi; { Two times Pi                         }
        Pi2            =       Pi * Pi; { Pi squared                           }

        Ptr            = 0.99;                   { Transmission probability   }
        Alpha          = 0.1;                   { The number of Rearrivals per
                                                   ...PacketDuration          }
        bitrate        = 256*1024;              { Bits transmitted per sec.  }
        PacketDuration= 64/bitrate;             {Time needed to send a Packet}

        maxcodelength = 127;                    { Maximum code size           }
        codelength    =  31;                    { Length of code              }
        Tb             = 1/bitrate;             { Bit duration                }
        Tc             = 1/(codelength*bitrate); { Chip duration              }
        Tm             = 100E-9;                { Maximum delay spread        }
        MaxL           = 8;                     { Maximum number of paths     }
        L              = Trunc(Tm/Tc)+1;        { Number of paths             }

        WarmTime       =         21000; { Warm up time in sec.                 }
        Interval       =           500; { Time between screen updates          }
        EndTime        =        125000; { Simulation time in sec.              }

        NumSim         =            20; { Number of activity levels simulated  }
        LRangeMin      =             4; { Terminal activity from 10^-LRangeMin }
        LRangeMax      =             0; { ...to 10^-LRangeMax                  }

Type    TermState = (IDLE, BLOCKED, TRANSMIT, INTERFERE);
        OnOff      = (ON, OFF);
        Matrix     = Array[1..MaxTerm, 1..MaxTerm]          Of Extended;
        PosArr     = Array[1..MaxTerm]                      Of Extended;
        RArr       = Array[1..MaxTerm, 1..MaxL]             Of Extended;
        StringArr = Array[1..MaxTerm]                       Of String;
        TermArr    = Array[1..MaxTerm]                      Of TermState;

{-----------------------------------------------------------------------------}
        p2CRec     = ^CRec;

        CRec       = Record
                        cc :Array[-maxcodelength..maxcodelength] Of ShortInt;
                     End;


        CArr       = Array [1..MaxTerm, 1..MaxTerm] Of p2CRec;

{-----------------------------------------------------------------------------}
        QueueEntry= RECORD
                       TerminalNo      :Integer;
                       Time            :Extended;
                    END;


        QueueType = RECORD
                       Count,
                       Front,
                       Rear: 0..Maxterm;
                       Entry: ARRAY [1..Maxterm] OF QueueEntry;
                    END;

{-----------------------------------------------------------------------------}
        EventType = (Arrival, ReArrival, BusyToneOn, BusyToneOff, Receive);

        ListPointer = ^ListNode;
        ListNode    = RECORD
                         Event          :EventType;
                         Time           :Extended;
                         TerminalNo     :Integer;
                         NextNode       :ListPointer
                      END;

        LinkedList  = RECORD
                          Head          :ListPointer
                      END;

{-----------------------------------------------------------------------------}
Var     Count,
        Front,
        Rear             : Integer;  { Counters for QueueType                  }

        CCat             : CArr;     { Catalog of code cross correlations      }

        PArrH,                       { Packet Arrival Probability Slotted      }
```

```
       PArrL,                   { Packet Arrival Probability Unslotted     }
       Rreceive,                { Receiver radius around base station      }
       Rterm,                   { Terminal radius around receivers         }
       SNR          :Extended;  { Signal to white Noise Ratio at receiver }

       Nt,                      { Number of terminals                      }
       GroupSize,               { Number of terminals per group            }
       NumberOfLines :Integer;  { Number of groups in the system. This is }
                                { ... also the number of codes used.       }

       DelayArray   :PosArr;    { Delay administration for each terminal  }
       DelayCount   :Extended;  { Total delay time in Tpd                  }

       SuccessCount,            { Number of successful packets             }
       ArrivalCount,            { Number of accepted packets               }
       GCount       :LongInt;   { Number of collisions occured             }

       S            :Matrix;    { Power loss ratio matrix                  }

       OldClock,                { Clock maintained for screen updates      }
       Clock        :Extended;  { Clock maintained by main program         }

       EventList    :LinkedList;{ Nodes of events linked together          }
       ReceiveQueue :QueueType; { Queue for received packets               }

       BusyTone     :OnOff;     { Indicates wheather the channel is free  }
                                { or not.                                  }
       Terminal     :TermArr;   { Terminal state information               }

       i            :Integer;   {Loopvariable for terminal activity levels}

       ResultFile   :Text;      { Result file with original values         }

       Throughput,              { Throughput: fraction of the time
                                  ...carrying a successful transmission   }
       D,                       { Mean Delay normalized in slots           }
       G            :Extended;  { Mean Delay: time between the arrival of
                                  ...the first bit till the receive of the
                                  ...last bit in sec.                      }
{****************************************************************************}
{  Power calculates a raised to the power of b                               }
{                                                                            }
{  Depends on: none                                                          }
{  Calls      : none                                                         }
{  Modifies   : none                                                         }
{                                                                            }
{  Pre : a >= 0 And b >=0                                                    }
{  Post: Power = a^b                                                         }
{****************************************************************************}
Function Power(a,b:Extended):Extended;
Begin
    If a<>0
    Then
        power := Exp(b*Ln(a))
    Else
        If b<>0
        Then
            power := 0                                  { power(0,b) = 0 }
        Else
            power := 1;                                 { power(0,0) = 1 }
End;{Power}

{****************************************************************************}
{  ShowState shows the terminal states on screen                             }
{                                                                            }
{  Depends on: config, numberOfLines, terminal                               }
{  Calls      : none                                                         }
{  Modifies   : none                                                         }
{****************************************************************************}
Procedure ShowState;
Var
    i : Integer;
    c : Char;
Begin
    GotoXY(12, 22);
    Write('                  1                   2                   3    ');
    GotoXY(12, 23);
    Write('1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 3 2 4 5 6 7 8 9 0 1 2');
    For i := 1 To Nt Do
    Begin
        GotoXY(10+2*i,24);
        Case Terminal[i] Of
            IDLE        : Write('i');
            BLOCKED     : Write('b');
            TRANSMIT    : Write('t');
            INTERFERE   : Write('f');
        End;{Case}
    End;{For i := 1 TO Nt}
End;{ShowState}

{****************************************************************************}
{  R calculates a partial cross correlation                                  }
{                                                                            }
{  Depends on: config, CCat, Tc                                              }
{  Calls      : none                                                         }
{  Modifies   : none                                                         }
{****************************************************************************}
Function R(i,j:Integer; tau:Extended):Extended;
Var
    l,                          { Chip number belonging to tau              }
    Cl,                         { Discrete correlation function for chip l  }
    Cl1      :Integer;          { Discrete correlation function for chip l+1 }
Begin
    l   := Trunc(tau/Tc);
    Cl1 := CCat[i,j]^.cc[l+1-codelength];
    If l >= 1
    Then
    Begin
        Cl  := CCat[i,j]^.cc[l-codelength];
        R   := Cl*Tc+(Cl1-Cl)*(tau-l*Tc);
```

```
    End
    Else
        R := tau*Cl1;
End;{R}

{****************************************************************************}
{  Rhat calculates a partial cross correlation                               }
{                                                                            }
{  Depends on: config, CCat, Tc                                              }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{****************************************************************************}
Function Rhat(i,j:Integer; tau:Extended):Extended;
Var
   l,                           { Chip number belonging to tau           }
   Cl,                          { Discrete correlation function for chip l   }
   Cl1      :Integer;           { Discrete correlation function for chip l+1 }
Begin
    l   := Trunc(tau/Tc);
    Cl  := CCat[i,j]^.cc[l];
    If l < codelength-1
    Then
    Begin
         Cl1  := CCat[i,j]^.cc[l+1];
         Rhat := Cl*Tc+(Cl1-Cl)*(tau-l*Tc);
    End
    Else
        Rhat := ((l+1)*Tc-tau)*Cl;
End;{Rhat}

{****************************************************************************}
{  InitCCat sets up a catalog of discrete aperiodic cross correlation        }
{  functions for a set of (Gold) codes passed by gc. The catalog consists of }
{  a two dimensional array of pointers. These pointers refer to              }
{  one-dimensional arrays containing the cross correlation information. This }
{  information is in essence the number of simmilarities minus the numbers   }
{  of dissimilarities between two codes for a given discrete phase shift     }
{                                                                            }
{  Depends on: config, p2CRec                                                }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{****************************************************************************}
Procedure InitCCat(Var CCat:CArr; gc:StringArr);
Var
   i,                                       { Loop variable              }
   j,                                       { Loop variable              }
   k,                                       { Loop variable              }
   l,                                       { Discrete phase shift       }
   t       :Integer;                        { Correlation result for shift l }
Begin
    Write('Calculating ', Nt, ' by ', Nt,' correlations');
    For i := 1 To Nt Do                     { Double loop over all codes     }
        For j := 1 To Nt Do
        Begin
             GotoXY(70,1);                  { Show which codes are being done }
```

```
             Write(i:4,j:4);
             CCat[i,j] := New(p2CRec);    { Allocate space for new array    }

             { Calculate cross-correlation }
             For l := -codelength To codelength Do
             Begin
                  t := 0;                  { Reset correlation value         }
                  If l < 0                 { Different formulas for pos and  }
                  Then                     { neg phase shifts                }
                      For k := 0 To codelength-1+l Do
                          If gc[i][1+((k-l) Mod codelength)] = gc[j][k+1]
                          Then
                              Inc(t)
                          Else
                              Dec(t)
                  Else
                      For k := 0 To codelength-l-1 Do
                         If gc[i][1+(k Mod codelength)] =
                           gc[j][1+((k+l) Mod codelength)]
                         Then
                             Inc(t)
                         Else
                             Dec(t);

                  CCat[i,j]^.cc[l] := t;
             End;
        End;
End;{InitCCat}

{****************************************************************************}
{  DisposeCCat frees memory space taken up by the correlation catalog       }
{                                                                            }
{  Depends on: config                                                        }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{****************************************************************************}
Procedure DisposeCCat(CCat:CArr);
Var
   i,                                                     { Loop variable }
   j      :Integer;                                       { Loop variable }
Begin
    For i := 1 To Nt Do                      { Loop over all catalog entries }
        For j := 1 To Nt Do
            Dispose(CCat[i,j]);              { Free memory of element i,j    }
End;{DisposeCCat}

{****************************************************************************}
{ InitTerm initializes all terminals as free                                 }
{                                                                            }
{  Depends on: config                                                        }
{  Calls     : none                                                          }
{  Modifies  : pfn                                                           }
{****************************************************************************}
Procedure InitTerm(Var Terminal:TermArr);
Var
```

```
   i :Integer; { Loop variable }
Begin
    For i:= 1 To Nt Do
        terminal[i] := IDLE;
End;{InitTerm}

{****************************************************************************}
PROCEDURE InitDelayArray(VAR DelayArray:PosArr);
VAR
   i : Integer; { Loop variable }
BEGIN
    FOR i := 1 TO MaxTerm DO
        DelayArray[i] := 0;
END;{InitDelayArray}

{****************************************************************************}
PROCEDURE InitBusyTone(VAR BusyTone:OnOff);
BEGIN
    BusyTone := Off;
END;{InitBusyTone}

{****************************************************************************}
{  Initcode reads Gold codes from a file and has the cross correlation      }
{  catalog setup                                                            }
{                                                                           }
{  Depends on: config                                                       }
{  Calls     : initCCat                                                     }
{  Modifies  : pfn                                                          }
{****************************************************************************}
Procedure InitCodes(Var CCat:CArr; groupSize, numCodes:Integer);
Var
   i,                                  { Loop variable                      }
   j           :Integer;               { Loop variable                      }
   f           :text;                  { File containing the Gold codes     }
   codeCount,                          { Length of transmission code        }
   cfile       :String;                { Codefile name                      }
   gc          :StringArr;             { Gold codes as read from file       }
Begin
    ClrScr;
    Str(codelength, codeCount);  { Assign appropriate codefile name        }
    cfile := 'codes\'+codeCount+'chip.dat';
    Assign(f, cfile);            { Open code file                          }
    Reset(f);
    For i := 1 To Nt Do          { Read the codes from file                }
        ReadLn(f,gc[i]);
        Close(f);
        If numCodes <> Nt        { Reassign codes according to reuse scheme }
        Then                     { e.g. 12345678 -> 11223344 for 2 term/grp }
            For i := numCodes DownTo 1 Do
                For j := groupSize DownTo 1 Do
                    gc[(i-1)*groupSize+j] := gc[i];
                    initCCat(CCat, gc);
                                 { Initialize catalog of cross correlations }
End;{InitCodes}

{****************************************************************************}
{  InitPathLoss sets up a table of transmitter to receiver path gains from  }
{  all transmitters to all receivers. All gains are between 0 and 1. When   }
{  groups are larger than one terminal duplicate entries will exist in the  }
{  table because there are less receivers than terminals then.             }
{                                                                           }
{  Depends on: config, lambda                                               }
{  Calls     : Power                                                        }
{  Modifies  : none                                                         }
{****************************************************************************}
Procedure InitPathGain(Var S:Matrix; groupSize, numberOfLines :Integer;
                                    Rreceive,  Rterm, lossexp:Extended );
Var
   gamma,                              { Angle of receiver relative to x-axis    }
   theta    :Extended;                 { Angle of transmitter relative to x-axis }
   xt, yt,                             { Transmitter coordinates                 }
   xr, yr   :PosArr;                   { Receiver coordinates                    }
   i,                                  { Loop variable                           }
   j,                                  { Loop variable                           }
   k        :Integer;                  { Index variable                          }
Begin
     For i := 1 To numberOfLines Do{ Loop over all receivers                 }
         For j := 1 To groupSize Do{ Loop over all group terminals            }
         Begin
           k := (i-1) * groupSize + j;       {Determine index current term. }
           gamma := 2*Pi * (i-1)/numberOfLines;  {Calc. receiver angle      }
           xr[k] := Rreceive * Cos(gamma);      {Calc. receiver x-coord.   }
           yr[k] := Rreceive * Sin(gamma);      {Calc. receiver y-coord.   }
           theta := 2*Pi*(j-1)/groupSize+gamma+Pi;{Calc. transmitter angle  }
           xt[k] := xr[k] + Rterm * Cos(theta);  {Calc. transmitter x-coord.}
           yt[k] := yr[k] + Rterm * Sin(theta);  {Calc. transmitter y-coord.}
         End;
         For i := 1 To Nt Do                  {Double loop over all terminals}
             For j := 1 To Nt Do
             Begin
                  { Calculate path gains according to far field model }
                  S[i,j] := Power(lambda/(4*Pi*Sqrt(Sqr(xt[i]-xr[j])
                                  +Sqr(yt[i]-yr[j]))),lossexp);
             End
End;{InitPathGain}

{****************************************************************************}
{  Rayleigh randomly draws a value according to the rayleigh distribution   }
{                                                                           }
{  Depends on: none                                                         }
{  Calls     : duni                                                         }
{  Modifies  : pfn                                                          }
{****************************************************************************}
Function Rayleigh(ave:Extended):Extended;
Var
   r,
   u,
   v,
   x,
   y       :Extended;
```

```
Begin
     Repeat
           u := 2*duni-1;
           v := 2*duni-1;
           r := sqr(u)+sqr(v);
     Until (r<1.0) and (r>0.0);
     r := Sqrt(-2*Ln(r)/r);
     x := u*r;
     y := v*r;
     Rayleigh:=Sqrt(Sqr(x)+Sqr(y))*ave*Sqrt(2/Pi);
End;

{****************************************************************************}
{  MarcumQ calculates the Marcum Q-function                                  }
{                                                                            }
{  Depends on: none                                                          }
{  Calls      : none                                                         }
{  Modifies   : pfn                                                          }
{****************************************************************************}
Function MarcumQ(a,b:Extended):Extended;
Var
   aa,
   ab,
   ac,
   bc,
   bb,
   ba,
   dd,
   d1       :Extended;
   n        :Integer;
Begin
     If a = 0
     Then
         If b < 30
         Then
             MarcumQ := Exp(-Sqr(b)/2)
         Else
             MarcumQ := 0
     Else
     Begin
          If a < b
          Then
          Begin
               ab := 1;
               dd  := a/b;
          End
          Else
          Begin
               ab := 0;
               dd  := b/a;
          End;
          aa := 0  ;
          bb := 0.5;
          ba := 0  ;
          d1 := dd;
          n  := 0;
          Repeat
                n  := n + 1;
                ac := d1+2*n*ab/(a*b)+aa;
                bc := 1+2*n*bb/(a*b)+ba;
                d1 := dd*d1;
                aa := ab;
                ab := ac;
                ba := bb;
                bb := bc;
          Until bc > 1e12;
          If a < b
          Then
              MarcumQ := (ac/(2*bc))*Exp(-Sqr(a-b)/2)
          Else
              MarcumQ := 1-(ac/(2*bc))*Exp(-Sqr(a-b)/2);
     End;
End;

{****************************************************************************}
{  NegExpI0 approximates the modified Bessel function of the first kind      }
{  using the recurrence relation I     (z) = I     (z) + 2n/z I   (z)         }
{                                 0,n-1       0,n+1            0,n           }
{                                                                            }
{  and then multiplies the result with Exp(-(a^2+b^2)/2).                    }
{                                                                            }
{  Depends on: none                                                          }
{  Calls      : none                                                         }
{  Modifies   : pfn                                                          }
{****************************************************************************}
Function NegExpI0(a,b:Extended):Extended;
Var
   n    :Integer;                                        { Loop variable       }
   x,
   Ja,                                                   { I0,n-1              }
   Jb,                                                   { I0,n                }
   Jc,                                                   { I0,n+1              }
   tmp,
   t    :Extended;                                       { Intermediate result }
Begin
     tmp := (Sqr(a)+Sqr(b))/2;
     x   := a*b;
     If x = 0
     Then
         If tmp < 150
         Then
             NegExpI0 := Exp(-tmp)/2
         Else
             negExpI0 := 0
     Else
     Begin
          Jc := 0;
          Jb := 1;
          t  := 0;
          For n := 20 DownTo 1 Do
```

```
        Begin
            Ja := (2*n/x)*Jb+Jc;
            Jc := Jb;
            Jb := Ja;
            t  := t + Ja;
        End;
        NegExpI0 := Exp(x-(Sqr(a)+Sqr(b))/2)*Ja/(2*t-Ja);
    End;
End;

{*****************************************************************************}
{ CalcMu1 calculates the Mu1 parameter for the bit error formula in           }
{ Rayleigh fading environments                                                }
{                                                                             }
{ Depends on: L                                                               }
{ Calls      : R, Rhat                                                        }
{ Modifies   : none                                                           }
{*****************************************************************************}
Function CalcMu1(u, h:Integer; tau:RArr; mu2:Extended):Extended;
Var
   tmp :Extended;                                   { Intermediate result }
   i   :Integer;                                    { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To L Do
        If i <> h
        Then
            tmp := tmp + R(u,u,tau[u,i])*Rhat(u,u,tau[u,i])*s[u,u];

    CalcMu1 := 4*2*Pt*tmp + mu2
End;

{*****************************************************************************}
{ CalcMu2 calculates the Mu2 parameter for the bit error formula in           }
{ Rayleigh fading environments                                                }
{                                                                             }
{ Depends on: config, L, Pt, terminal                                         }
{ Calls      : R, Rhat                                                        }
{ Modifies   : none                                                           }
{*****************************************************************************}
Function CalcMu2(u, h:Integer; tau:RArr; sigma_n2:Extended):Extended;
Var
   tmp :Extended;                                   { Intermediate result }
   i,                                               { Loop variable       }
   j   :Integer;                                    { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To Nt Do
       If (terminal[i] = INTERFERE) And (i <> u)
       Then
           For j := 1 To L Do
               tmp := tmp + (Sqr(R(u,i,tau[i,j]))
                          +  Sqr(Rhat(u,i,tau[i,j])))*s[i,u];

    For j := 1 To L Do
       If j <> h
       Then
           tmp := tmp + (Sqr(R(u,u,tau[u,j]))
                      +  Sqr(Rhat(u,u,tau[u,j])))*s[u,u];

    CalcMu2 := 2*(2*Pt*tmp+sigma_n2);
End;

{*****************************************************************************}
{ CalcMu12 calculates the Mu12 parameter for the bit error formula in         }
{ Rayleigh fading environments                                                }
{                                                                             }
{ Depends on: config, L, Pt, terminal                                         }
{ Calls      : R, Rhat                                                        }
{ Modifies   : none                                                           }
{*****************************************************************************}
Function CalcMu12(u,h:Integer;tau:RArr):Extended;
Var
   tmp :Extended;                                   { Intermediate result }
   i,                                               { Loop variable       }
   j   :Integer;                                    { Loop variable       }
Begin
     tmp := 0;
     For i := 1 To Nt Do
        If (terminal[i] = INTERFERE) And (i<>u)
        Then
            For j := 1 To L Do
               tmp := tmp + R(u,i,tau[i,j])*Rhat(u,i,tau[i,j])*s[i,u];

     For j := 1 To L Do
        If j <> h
        Then
            tmp := tmp + (Sqr(Rhat(u,u,tau[u,j]))
                       + R(u,u,tau[u,j])*Rhat(u,u,tau[u,j]))*S[u,u];

     CalcMu12 := 2*2*Pt*tmp;
End;

{*****************************************************************************}
{ CalculatePbe computes the bit error probability for a terminal, given       }
{ which other terminals transmit                                              }
{                                                                             }
{ Depends on: config, L, Pt, S, Tb, terminal                                  }
{ Calls      : duni, I0, MarcumQ, Rayleigh                                    }
{ Modifies   : none                                                           }
{*****************************************************************************}
Function CalculatePbe(u:Integer):Extended;
Var
   mu1,
   mu2,
   mu12,
   a,
   b,
   bmax,                          { Maximum path gain                         }
   tmp,
```

```
   m,
   mq,
   mb,
   sigma_n2:Extended;              { Noise power                        }
   beta,                           { Rayleigh distributed path gains    }
   tau     :RArr;                  { Uniformly [0,Tb] distributed path delays }
   h,                              { Number of path with largest gain   }
   i,                              { Loop variable                      }
   j       :Integer;               { Loop variable                      }
Begin
     { Assign random delays and path gains to transmitting terminals }
     For i := 1 To Nt Do
        If terminal[i] = INTERFERE
        Then
           For j := 1 To L Do
           Begin
                tau[i,j]  := Tb*duni;
                beta[i,j] := Rayleigh(Sqrt(S[i,u]));
           End
        Else
           For j := 1 To L Do
           Begin
                tau[i,j]  := 0;
                beta[i,j] := 0;
           End;

     { Determine what the largest gain is }
     bmax := 0;
     h    := 1;
     For i := 1 To L Do
        If beta[u,i]>bmax
        Then
        Begin
             bmax := beta[u,i];
             h    := i;
        End;

     { Determine noise power relative to largest incoming signal }
     sigma_n2 := power(10, -snr/10)*Sqr(bmax)*Pt*Tb*Tb;

     { Calculate bit error probability parameters }
     mu2  := CalcMu2(u, h, tau, sigma_n2);
     mu1  := CalcMu1(u, h, tau, mu2);
     mu12 := CalcMu12(u, h ,tau);
     m    := Sqrt(2*Pt)*bmax*Tb;
     a    := m*Abs(1/Sqrt(mu1)-1/Sqrt(mu2))/Sqrt(2);
     b    := m*(1/Sqrt(mu1)+1/Sqrt(mu2))/Sqrt(2);
     mq   := MarcumQ(a,b);
     mb   := NegExpI0(a,b);

     { Calculate the bit error probability }
     tmp := ( (1 + mu12/Sqrt(mu1*mu2))*mb )/2;
     CalculatePbe := mq-tmp
End;

{****************************************************************************}
{                                  Q U E U E S                               }
{****************************************************************************}


{****************************************************************************}
{  CreateQueue                                                               }
{  Pre : None                                                                }
{  Post: The queue Q has been created and is initialized to be empty.        }
{****************************************************************************}
PROCEDURE CreateQueue(VAR Q:QueueType);
BEGIN
     WITH Q DO
          BEGIN
               Count := 0;
               Front := 1;
               Rear := 0
          END
END;{CreateQueue}

{****************************************************************************}
PROCEDURE Append (TerminalNo:Integer; Time:Extended; VAR Q:QueueType);
VAR
   X : QueueEntry;
BEGIN
     WITH Q DO
          IF Count = MaxTerm THEN
             WRITELN ('Error: Attempt to append an entry to a full queue')
          ELSE
               BEGIN
                    X.TerminalNo := TerminalNo;
                    X.Time       := Time;
                    Count := Count + 1;
                    Rear := (Rear MOD MaxTerm) + 1;
                    Entry [Rear] := X;
               END
END;{Append}

{****************************************************************************}
PROCEDURE Remove (VAR X: QueueEntry; VAR Q:QueueType);
BEGIN
     WITH Q DO
          IF Count = 0 THEN
             WRITELN ('Error: Attempt to remove an entry from an empty queue')
          ELSE
               BEGIN
                    Count := Count - 1;
                    X := Entry [Front];
                    Front := (Front MOD MaxTerm) + 1
               END
END;{Remove}

{****************************************************************************}
FUNCTION QueueSize (VAR Q: QueueType): INTEGER;
BEGIN
```

```
     QueueSize := Q.Count
END;{QueueSize}

{*******************************************************************************}
FUNCTION QueueEmpty (VAR Q: QueueType): BOOLEAN;
BEGIN
     QueueEmpty := (Q.Count = 0)
END;{QueueEmpty}

{*******************************************************************************}
FUNCTION QueueFull (VAR Q: QueueType): BOOLEAN;
BEGIN
     QueueFull := (Q.Count = MaxTerm)
END;{QueueFull}

{*******************************************************************************}
PROCEDURE ShowQueue (VAR Q : QueueType);
VAR
   x, y : INTEGER;
   X1   : QueueEntry;
BEGIN
     {WRITELN('De inhoud van de queue is:');}
     x := QueueSize (Q);
     WITH Q DO
          y := front;
          WHILE x > 0 DO
          BEGIN
               X1 := Q.Entry[y];
               WRITELN('TN: ',X1.TerminalNo,'Time: ',X1.Time);
               x := x-1;
               y := (y MOD MaxTerm) + 1
          END;
          WRITELN;
END;{ShowQueue}


{*******************************************************************************}
{                            L I N K E D - L I S T                              }
{*******************************************************************************}


{*******************************************************************************}
{*  CreateList initializes linked list LL to be empty                          *}
{*                                                                             *}
{*  Pre:  None                                                                 *}
{*  Post: The list LL has been created and is initialized to be empty          *}
{*******************************************************************************}
PROCEDURE CreateList(VAR LL: LinkedList);
BEGIN
  LL.head := NIL
END;{CreateList}

{*******************************************************************************}
PROCEDURE InsertEvent(NewNode:ListPointer; T:Extended; VAR LL:LinkedList);
VAR
   Current,
   Trailing : ListPointer;
BEGIN
     Current  := LL.head;
     Trailing := NIL;
     WHILE (Current^.Time <= T) AND (Current^.NextNode <> NIL) DO
     BEGIN
          Trailing := Current;
          Current  := Current^.NextNode;
     END;
     IF (Current^.time <= T) AND (Current^.NextNode = NIL)
     THEN
     BEGIN
          Current^.NextNode := NewNode;
          NewNode^.NextNode := NIL;
     END
     ELSE IF (Current^.Time > T) AND (Trailing = NIL)
          THEN
          BEGIN
               NewNode^.NextNode   := LL.head;
               LL.head             := NewNode;
          END
          ELSE
          BEGIN
               NewNode^.NextNode  := Current;
               Trailing^.NextNode := NewNode;
          END;
END;{InsertEvent}

{*******************************************************************************}
PROCEDURE CreateEvent(E:EventType; T:Extended; TN:Integer; VAR LL:LinkedList);

VAR
   NewNode : ListPointer;
BEGIN
     New(NewNode);
     NewNode^.Event       := E;
     NewNode^.Time        := T;
     NewNode^.TerminalNo  := TN;
     IF LL.head = NIL
     THEN
     BEGIN
          NewNode^.NextNode := NIL;
          LL.head           := NewNode;
     END
     ELSE
     BEGIN
          InsertEvent(NewNode, T, LL);
     END;
END;{CreateEvent}

{*******************************************************************************}
PROCEDURE ShowList(VAR LL:LinkedList);
VAR
   Current : ListPointer;
```

```
   i        : Integer;
   Event    : String;
BEGIN
     i        := 1;
     Current := LL.head;
     ClrScr;
     WriteLn('The list consists of: (Press ENTER for the nodes to show up)');
     WriteLn;
     IF Current = NIL THEN
        BEGIN
             WRITELN('EventList is empty! (WriteList)');
        END
     ELSE
     WHILE (Current <> NIL) DO
     BEGIN
          CASE Current^.Event OF
               Arrival      : Event := 'Arrival';
               ReArrival    : Event := 'ReArrival';
               BusyToneOn   : Event := 'BusyToneOn';
               BusyToneOFF  : Event := 'BusyToneOff';
               Receive      : Event := 'Receive';
          ELSE WRITELN('ERROR in EventList (ShowList)!!!!!');
          END;
          WRITELN('Node:',i,'      Event:',Event:12,'      Time:',
                   Current^.Time:4,'      Terminal:',Current^.TerminalNo);
     i := i+1;
     Current := Current^.NextNode;
     ReadLn;
     END;
     WriteLn('End of list');
     ReadLn;
END;{ShowList}

{****************************************************************************}
PROCEDURE ProcessArrival(FirstNode:ListPointer; VAR BusyTone:OnOff;
                         VAR Terminal:TermArr; VAR LL:LinkedList;
                         VAR Q:QueueType);
VAR
   RandomDelay,
   Time,
   TInterArr : Extended;
   TN        : Integer;
BEGIN
     Clock     := FirstNode^.Time;
     TN        := FirstNode^.TerminalNo;
     Time      := FirstNode^.Time;
     TInterArr := Trunc(-(ln(1-DUNI))/PArrH)+1;
     CreateEvent(Arrival, Time + TInterArr, TN, LL);
     IF Terminal[TN] = IDLE
     THEN
     BEGIN
          Terminal[TN] := BLOCKED;
          Inc(ArrivalCount);
          IF (DUNI > Ptr)
          THEN
          BEGIN
               CreateEvent(ReArrival, Time + 1, TN, LL);
               DelayArray[TN] := DelayArray[TN] + 1;
               Inc(GCount);
          END
          ELSE
          BEGIN
               Terminal[TN] := TRANSMIT;
               Append(TN, Time, Q);
               CreateEvent(Receive, Time+ 0.95, TN, LL);
               DelayArray[TN] := DelayArray[TN] + 1;
          END
     END
END;{ProcessArrival}

{****************************************************************************}
PROCEDURE ProcessReArrival(FirstNode:ListPointer; VAR BusyTone:OnOff;
                           VAR Terminal:TermArr; VAR LL:LinkedList;
                           VAR Q:QueueType);
VAR
   RandomDelay,
   Time,
   TInterArr : Extended;
   TN        : Integer;
BEGIN
     Clock   := FirstNode^.Time;
     TN      := FirstNode^.TerminalNo;
     Time    := FirstNode^.Time;
     IF DUNI > Ptr
     THEN
     BEGIN
          CreateEvent (ReArrival, Time + 1, TN, LL);
          DelayArray[TN] := DelayArray[TN] + 1;
          Inc(GCount);
     END
     ELSE
     BEGIN
          Terminal[TN] := TRANSMIT;
          Append(TN, Time, Q);
          CreateEvent(Receive, Time+0.95, TN, LL);
          DelayArray[TN] := DelayArray[TN] + 1;
     END
END;{ProcessReArrival}

{****************************************************************************}
PROCEDURE ProcessReceive(FirstNode:ListPointer; VAR BusyTone:OnOff;
                         VAR Terminal:TermArr; VAR LL:LinkedList;
                         VAR Q:QueueType);
VAR
   X2          : QueueEntry;

   TN,
   TN2         : Integer;

   RandomDelay,
```

```
  Time,
  Time2,
  Pbe,
  Psp         : Extended;

BEGIN
    Clock := FirstNode^.Time+0.05;
    Time  := FirstNode^.Time+0.05;
    IF QueueEmpty(Q) = TRUE
    THEN
    BEGIN
         TN           := FirstNode^.TerminalNo;
         RandomDelay  := Trunc(-(ln(1-DUNI))/Alpha)+1;
         Terminal[TN] := BLOCKED;
         CreateEvent(ReArrival, Time+RandomDelay, TN, LL);
         DelayArray[TN]   := DelayArray[TN] + RandomDelay;
         Inc(GCount);
    END
    ELSE
    BEGIN
         TN           := FirstNode^.TerminalNo;
         Terminal[TN] := INTERFERE;
         WHILE NOT QueueEmpty(Q) DO
         BEGIN
              Remove(X2, Q);
              TN2 := X2.TerminalNo;
              Terminal[TN2] := INTERFERE;
         END;
         Pbe := CalculatePbe(TN);
         Psp := Power(1-Pbe, PacketLength);

         IF DUNI < Psp
         THEN
         BEGIN
              Inc(SuccessCount);
              Terminal[TN]   := IDLE;
              DelayCount     := DelayCount + DelayArray[TN] + 0.5;
              DelayArray[TN] := 0;
         END
         ELSE
         BEGIN
              Terminal[TN] := BLOCKED;
              RandomDelay  := Trunc(-(ln(1-DUNI))/Alpha)+1;
              CreateEvent(ReArrival, Time+RandomDelay, TN, LL);
              DelayArray[TN] := DelayArray[TN] + RandomDelay;
              Inc(GCount);
         END
    END
END;{ProcessReceive}

{*****************************************************************************}
PROCEDURE ProcessFirstNode(FirstNode:ListPointer; VAR LL:LinkedList;
                           VAR Q:QueueType);

BEGIN
    CASE FirstNode^.Event OF
         Arrival     : ProcessArrival(FirstNode, BusyTone, Terminal, LL, Q);
         ReArrival   : ProcessReArrival(FirstNode, BusyTone, Terminal, LL, Q);
         Receive     : ProcessReceive(FirstNode, BusyTone, Terminal, LL, Q);
    ELSE WRITELN('ERROR in EventList (ProcessFirstNode)!!!!!');
    END;

END;{ProcessFirstNode}

{*****************************************************************************}
PROCEDURE AdvanceNextEvent(VAR LL:LinkedList; VAR Q:QueueType);
VAR
   FirstNode : ListPointer;
BEGIN
    IF LL.head = NIL
    THEN
    BEGIN
         WRITELN('Error: EventList is empty (AdvanceNextEvent)');
    END
    ELSE IF LL.head^.NextNode = NIL
         THEN
         BEGIN
              FirstNode := LL.head;
              LL.head   := NIL;
              ProcessFirstNode(FirstNode, LL, Q);
         END
         ELSE
         BEGIN
              FirstNode := LL.head;
              LL.head   := LL.head^.NextNode;
              ProcessFirstNode(FirstNode, LL, Q);
         END;
    Dispose(FirstNode);
END;{AdvanceNextEvent}

{*****************************************************************************}
PROCEDURE InitEventList(VAR LL:LinkedList);
VAR
   i    : Integer;
   Time : Extended;
BEGIN
    FOR i := 1 TO Maxterm DO
    BEGIN
         Time := Trunc(-(ln(1-DUNI))/PArrH) + 1;
         CreateEvent(Arrival, Time, i, LL);
    END
END;{InitEventList}

{*****************************************************************************}
FUNCTION CountList(VAR LL:LinkedList):Integer;
VAR
   i       : Integer;
   Current : ListPointer;
BEGIN
    i := 0;
```

```
     Current := LL.head;
     WHILE Current <> NIL DO
     BEGIN
          Current := Current^.NextNode;
          i       := i + 1;
     END;
     CountList := i;
END;{CountList}

{*****************************************************************************}
PROCEDURE RemoveList(VAR LL:LinkedList);
VAR
   Current : ListPointer;
BEGIN
     Current := LL.head;
     WHILE Current <> NIL DO
     BEGIN
          Current := Current^.NextNode;
          Dispose(LL.head);
          LL.head := Current;
     END;
END;{RemoveList}

{*****************************************************************************}
PROCEDURE OpenResultFile(VAR OutFile:Text);
VAR
   GS,
   NL,
   FileName : String;
BEGIN
     Str(GroupSize, GS);
     Str(NumberOfLines, NL);
     FileName := 'V1SLT2P7.DAT';
     Assign(OutFile, FileName);

     ReWrite(OutFile);
{     WriteLn(OutFile,'RESULTS FROM V1SLOT.PAS:');
     WriteLn(OutFile);
     WriteLn(OutFile,'Size of group   : ', GroupSize);
     WriteLn(OutFile,'Number of groups: ', NumberOfLines);
     WriteLn(OutFile);
     WriteLn(OutFile,'Ptr             : ', Ptr);
     WriteLn(OutFile,'Alpha           : ', Alpha);
     WriteLn(OutFile,'SNR             : ', SNR);
     WriteLn(OutFile);
     WriteLn(OutFile,'Warm up time    : ', WarmTime);
     WriteLn(OutFile,'Simulation time: ', EndTime-WarmTime);
     WriteLn(OutFile);
     WriteLn(OutFile);
}     WriteLn(OutFile, ' G                 ':17,
                       'S                 ':16,
                       'D                 ':16);
END;{OpenResultFile}

{*****************************************************************************}
{  WarnBeep emits a half second warning tone and then waits 4.5 seconds       }
{                                                                             }
{  Depends on: none                                                           }
{  Calls     : none                                                           }
{  Modifies  : pfn                                                            }
{*****************************************************************************}
Procedure WarnBeep;
VAR
   i : Integer;
BEGIN
     FOR i := 1 TO 5 DO
     BEGIN
          Sound(500+i*20);
          Delay(300);
     END;
     NoSound;
     Delay(1000);
End;{WarnBeep}

{*****************************************************************************}
{                              M A I N   L I N E                              }
{*****************************************************************************}

Begin
     ClrScr;
     WriteLn('Output Testprogramma:');

     {Initialize variables}
     Rreceive      := 30;
     Rterm         :=  5;
     SNR           := 20;

     Nt            := 32;
     NumberOfLines :=  8;
     GroupSize     :=  4;

     OpenResultFile(ResultFile);
     InitCodes(CCat, GroupSize, NumberOfLines);
     InitPathGain(S, GroupSize, NumberOfLines, Rreceive, Rterm, LossExp);

     FOR i := 0 TO NumSim DO           { Loop over terminal activity levels  }
     BEGIN
          {Set terminal activity}
          PArrH := Power(10, -(LRangeMin + i*(LRangeMax-LRangeMin)/NumSim));
          rInit(3647523, 65321);       {Initialize FSU Ultra random generator}
          InitTerm(Terminal);          {Reset terminal states                }
          InitBusyTone(BusyTone);      {Set BusyTone to OFF                  }
          CreateList(EventList);
          CreateQueue(ReceiveQueue);
          InitEventList(EventList);    {Initialize EventList                 }
          InitDelayArray(DelayArray);  {Reset all terminal delays to zero    }

          {Show configuration to user}
          ClrScr;
          WriteLn('Receiver Radius: ', Rreceive:7:2);
```

```
          WriteLn('Terminal Radius: ', Rterm:7:2);
          WriteLn('Number of users: ', Nt:4);
          WriteLn('Size of group  : ', GroupSize:4);
          WriteLn('Number of codes: ', NumberOfLines:4);
          WriteLn('Code length    : ', codelength:4);
          WriteLn('Packet length  : ', Packetlength:4);
          WriteLn;
          WriteLn('Terminal activity (PArrH) : ',PArrH:8:5);

          {Reset event counters}
          ArrivalCount    := 0;
          SuccessCount    := 0;
          DelayCount      := 0;
          GCount          := 0;

          {Warm up the network}
          Clock    := 0;
          OldClock := 0;
          While Clock < WarmTime DO
          BEGIN
               AdvanceNextEvent(EventList, ReceiveQueue);
               IF (Clock - OldClock) > Interval
               THEN
               BEGIN
                    GoToXY(1,12);
                    WriteLn('Arrivals :', ArrivalCount:8);
                    WriteLn('Success  :', SuccessCount:8);
                    WriteLn;
                    WriteLn('Delay    :', DelayCount);
                    WriteLn('Clock    :', Clock);
                    ShowState;
                    OldClock := Clock;
               END;
          GoToXY(40,4);
          WriteLn('SIMULATING !!!!');
          GoTOXY(40,6);
          Write('STATUS: ',i,'/',NumSim);
          END;{Warm up loop}

          {Reset event counters}
          ArrivalCount    := 0;
          SuccessCount    := 0;
          GCount          := 0;
          DelayCount      := 0;

          {Main simulation Loop}
          While Clock < EndTime DO
          BEGIN
               AdvanceNextEvent(EventList, ReceiveQueue);
               IF (Clock - OldClock) > Interval
               THEN
               BEGIN
                    GoToXY(1,12);
                    WriteLn('Arrivals :', ArrivalCount:8);
                    WriteLn('Success  :', SuccessCount:8);
                    WriteLn;
                    WriteLn('Delay    :', DelayCount);
                    WriteLn('Clock    :', Clock);
                    ShowState;
                    OldClock := Clock;
               END;
          GoToXY(40,4);
          WriteLn('SIMULATING !!!!');
          GoTOXY(40,6);
          Write('STATUS: ',i,'/',NumSim);
          END;{Simulation Loop}

          {Calculate values for Physical Traffic, Throughput and Delay}
          Throughput := SuccessCount/(EndTime-WarmTime);
          D          := DelayCount/SuccessCount;
          G          := (ArrivalCount+GCount)/(EndTime-WarmTime);

          {Write simulation results in files for processing-purposes later on}
          WriteLn(ResultFile  , G          :16,
                                Throughput :16,
                                D          :16);

          RemoveList(EventList);

     END;{Terminal activity level loop}

     Close(ResultFile);

     {WarnBeep;}
     GoToXY(30,16);
     Write('I AM READY MISTER LE !!!!!!');
     GoToXY(30,17);
     Write('Press ENTER to return to program.');
     {ReadLn;}
END.
```

```
{*****************************************************************************}
{*  SimUnslotted is a program designed with the goal to simulate the        *}
{*  'Unslotted Hybrid CDMA/ISMA protocol'. This program calculates the      *}
{*  throughput and delay of a network of terminals by simulation.           *}
{*                                                                          *}
{*  Most of the constants, variables, functions and procedures used here    *}
{*  originates from Huub van Roosmalen.                                     *}
{*                                                                          *}
{*  The terminals are clustered around distributed receivers at a fixed     *}
{*  distance. The distributed receivers are clustered around the base       *}
{*  station at a fixed distance. The packets from the terminals are         *}
{*  received by the distributed receivers and sent to the base station by   *}
{*  wire or lines. These lines can be seen as primitive concentrators       *}
{*                                                                          *}
{*  Huy Linh Anh Le, August 10 1995                                         *}
{*****************************************************************************}
Program SimUnslotted(Input, Output);

Uses    Dos, Crt, Ultrax;

Const   MaxTerm       =            32; { Maximum number of terminals allowed  }
        PacketLength  =            64; { Number of bits per packet            }

        Pt            =             1; { Transmitter power in Watts           }
        LossExp       =             2; { Attenuation parameter for the ...    }
                                       {  ...far field model                  }
        c             =     299792458; { Speed of light in m/s                }
        f             =    1700000000; { Transmitter center frequency         }
        lambda        =           c/f; { Signal wavelength                    }

        twoPi         =        2 * Pi; { Two times Pi                         }
        Pi2           =       Pi * Pi; { Pi squared                           }

        Alpha         = 0.2;                  { The number of Rearrivals per
                                                 ...PacketDuration            }
        bitrate       = 256*1024;             { Bits transmitted per sec.     }
        PacketDuration= 64/bitrate;           {Time needed to send a Packet}
        Fraction      = 0.2;                  { Inhibit delay fraction        }

        maxcodelength = 127;                  { Maximum code size             }
        codelength    =  31;                  { Length of code                }
        Tb            = 1/bitrate;            { Bit duration                  }
        Tc            = 1/(codelength*bitrate); { Chip duration               }
        Tm            = 100E-9;               { Maximum delay spread          }
        MaxL          = 8;                    { Maximum number of paths       }
        L             = Trunc(Tm/Tc)+1;       { Number of paths               }

        WarmTime      =         21000; { Warm up time in Tpd                  }
        Interval      =           500; { Time between screen updates          }
        EndTime       =        125000; { Simulation time in Tpd               }

        NumSim        =            20; { Number of activity levels simulated  }
        LRangeMin     =             4; { Terminal activity from 10^-LRangeMin }
        LRangeMax     =             0; { ...to 10^-LRangeMax                  }

Type    TermState = (IDLE, BLOCKED, TRANSMIT, INTERFERE);
        OnOff     = (ON, OFF);
        Matrix    = Array[1..MaxTerm, 1..MaxTerm]          Of Extended;
        PosArr    = Array[1..MaxTerm]                      Of Extended;
        RArr      = Array[1..MaxTerm, 1..MaxL]             Of Extended;
        StringArr = Array[1..MaxTerm]                      Of String;
        TermArr   = Array[1..MaxTerm]                      Of TermState;

{-----------------------------------------------------------------------------}
        p2CRec    = ^CRec;

        CRec      = Record
                       cc :Array[-maxcodelength..maxcodelength] Of ShortInt;
                    End;


        CArr      = Array [1..MaxTerm, 1..MaxTerm] Of p2CRec;

{-----------------------------------------------------------------------------}
        QueueEntry= RECORD
                      TerminalNo      :Integer;
                      Time            :Extended;
                    END;


        QueueType = RECORD
                      Count,
                      Front,
                      Rear: 0..Maxterm;
                      Entry: ARRAY [1..Maxterm] OF QueueEntry;
                    END;

{-----------------------------------------------------------------------------}
        EventType = (Arrival, ReArrival, BusyToneOn, BusyToneOff, Receive);

        ListPointer = ^ListNode;
        ListNode    = RECORD
                        Event          :EventType;
                        Time           :Extended;
                        TerminalNo     :Integer;
                        NextNode       :ListPointer
                      END;

        LinkedList  = RECORD
                         Head          :ListPointer
                      END;

{-----------------------------------------------------------------------------}
Var     Count,
        Front,
        Rear            : Integer;  { Counters for QueueType                  }

        CCat            : CArr;     { Catalog of code cross correlations      }

        PArrH,                      { Packet Arrival Probability Slotted      }
```

```
       PArrL,                     { Packet Arrival Probability Unslotted     }
       Rreceive,                  { Receiver radius around base station      }
       Rterm,                     { Terminal radius around receivers         }
       SNR           :Extended;   { Signal to white Noise Ratio at receiver  }

       Nt,                        { Number of terminals                      }
       GroupSize,                 { Number of terminals per group            }
       NumberOfLines :Integer;    { Number of groups in the system. This is  }
                                  { ... also the number of codes used.       }

       DelayArray    :PosArr;     { Delay administration for each terminal   }
       DelayCount    :Extended;   { Total delay time in Tpd                  }

       SuccessCount,              { Number of successful packets             }
       ArrivalCount,              { Number of accepted packets               }
       GCount        :LongInt;    { Number of collisions occured             }

       S             :Matrix;     { Power loss ratio matrix                  }

       OldClock,                  { Clock maintained for screen updates      }
       Clock         :Extended;   { Clock maintained by main program         }

       EventList     :LinkedList;{ Nodes of events linked together           }
       ReceiveQueue  :QueueType;  { Queue for received packets               }

       BusyTone      :OnOff;      { Indicates wheather the channel is free   }
                                  { or not.                                  }
       Terminal      :TermArr;    { Terminal state information               }

       i             :Integer;    {Loopvariable for terminal activity levels}

       ResultFile    :Text;       { Result file with original values         }

       Throughput,                { Throughput: fraction of the time
                                    ...carrying a successful transmission    }
       D,                         { Mean Delay normalized in slots           }
       G             :Extended;   { Mean Delay: time between the arrival of
                                    ...the first bit till the receive of the
                                    ...last bit in sec.                      }
{****************************************************************************}
{  Power calculates a raised to the power of b                               }
{                                                                            }
{  Depends on: none                                                          }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{                                                                            }
{  Pre : a >= 0 And b >=0                                                    }
{  Post: Power = a^b                                                         }
{****************************************************************************}
Function Power(a,b:Extended):Extended;
Begin
    If a<>0
    Then
        power := Exp(b*Ln(a))
    Else
        If b<>0
        Then
            power := 0                                      { power(0,b) = 0 }
        Else
            power := 1;                                     { power(0,0) = 1 }
End;{Power}

{****************************************************************************}
{  ShowState shows the terminal states on screen                             }
{                                                                            }
{  Depends on: config, numberOfLines, terminal                               }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{****************************************************************************}
Procedure ShowState;
Var
   i : Integer;
   c : Char;
Begin
     GotoXY(12, 22);
     Write('                   1                   2                   3    ');
     GotoXY(12, 23);
     Write('1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 3 2 4 5 6 7 8 9 0 1 2');
     For i := 1 To Nt Do
     Begin
          GotoXY(10+2*i,24);
          Case Terminal[i] Of
              IDLE       : Write('i');
              BLOCKED    : Write('b');
              TRANSMIT   : Write('t');
              INTERFERE  : Write('f');
          End;{Case}
     End;{For i := 1 TO Nt}
End;{ShowState}

{****************************************************************************}
{  R calculates a partial cross correlation                                  }
{                                                                            }
{  Depends on: config, CCat, Tc                                              }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{****************************************************************************}
Function R(i,j:Integer; tau:Extended):Extended;
Var
   l,                            { Chip number belonging to tau              }
   Cl,                           { Discrete correlation function for chip l  }
   Cl1      :Integer;            { Discrete correlation function for chip l+1 }
Begin
     l   := Trunc(tau/Tc);
     Cl1 := CCat[i,j]^.cc[l+1-codelength];
     If l >= 1
     Then
     Begin
          Cl := CCat[i,j]^.cc[l-codelength];
          R  := Cl*Tc+(Cl1-Cl)*(tau-l*Tc);
```

```
    End
    Else
        R := tau*Cl1;
End;{R}

{*****************************************************************************}
{ Rhat calculates a partial cross correlation                                 }
{                                                                             }
{ Depends on: config, CCat, Tc                                                }
{ Calls     : none                                                            }
{ Modifies  : none                                                            }
{*****************************************************************************}
Function Rhat(i,j:Integer; tau:Extended):Extended;
Var
    l,                        { Chip number belonging to tau                  }
    Cl,                       { Discrete correlation function for chip l      }
    Cl1       :Integer;       { Discrete correlation function for chip l+1    }
Begin
    l   := Trunc(tau/Tc);
    Cl  := CCat[i,j]^.cc[l];
    If l < codelength-1
    Then
    Begin
        Cl1  := CCat[i,j]^.cc[l+1];
        Rhat := Cl*Tc+(Cl1-Cl)*(tau-l*Tc);
    End
    Else
        Rhat := ((l+1)*Tc-tau)*Cl;
End;{Rhat}

{*****************************************************************************}
{ InitCCat sets up a catalog of discrete aperiodic cross correlation          }
{ functions for a set of (Gold) codes passed by gc. The catalog consists of   }
{ a two dimensional array of pointers. These pointers refer to                }
{ one-dimensional arrays containing the cross correlation information. This   }
{ information is in essence the number of simmilarities minus the numbers     }
{ of dissimilarities between two codes for a given discrete phase shift       }
{                                                                             }
{ Depends on: config, p2CRec                                                  }
{ Calls     : none                                                            }
{ Modifies  : none                                                            }
{*****************************************************************************}
Procedure InitCCat(Var CCat:CArr; gc:StringArr);
Var
    i,                                    { Loop variable                     }
    j,                                    { Loop variable                     }
    k,                                    { Loop variable                     }
    l,                                    { Discrete phase shift              }
    t      :Integer;                      { Correlation result for shift l    }
Begin
    Write('Calculating ', Nt, ' by ', Nt,' correlations');
    For i := 1 To Nt Do                   { Double loop over all codes        }
        For j := 1 To Nt Do
        Begin
            GotoXY(70,1);                 { Show which codes are being done   }
            Write(i:4,j:4);
            CCat[i,j] := New(p2CRec);     { Allocate space for new array      }

            { Calculate cross-correlation }
            For l := -codelength To codelength Do
            Begin
                t := 0;                   { Reset correlation value           }
                If l < 0                  { Different formulas for pos and    }
                Then                      { neg phase shifts                  }
                    For k := 0 To codelength-1+l Do
                        If gc[i][1+((k-l) Mod codelength)] = gc[j][k+1]
                        Then
                            Inc(t)
                        Else
                            Dec(t)
                Else
                    For k := 0 To codelength-l-1 Do
                        If gc[i][1+(k Mod codelength)] =
                          gc[j][1+((k+l) Mod codelength)]
                        Then
                            Inc(t)
                        Else
                            Dec(t);

                CCat[i,j]^.cc[l] := t;
            End;
        End;
End;{InitCCat}

{*****************************************************************************}
{ DisposeCCat frees memory space taken up by the correlation catalog          }
{                                                                             }
{ Depends on: config                                                          }
{ Calls     : none                                                            }
{ Modifies  : none                                                            }
{*****************************************************************************}
Procedure DisposeCCat(CCat:CArr);
Var
    i,                                                      { Loop variable }
    j      :Integer;                                        { Loop variable }
Begin
    For i := 1 To Nt Do                     { Loop over all catalog entries }
        For j := 1 To Nt Do
            Dispose(CCat[i,j]);             { Free memory of element i,j    }
End;{DisposeCCat}

{*****************************************************************************}
{ InitTerm initializes all terminals as free                                  }
{                                                                             }
{ Depends on: config                                                          }
{ Calls     : none                                                            }
{ Modifies  : pfn                                                             }
{*****************************************************************************}
Procedure InitTerm(Var Terminal:TermArr);
Var
```

```
    i :Integer; { Loop variable }
Begin
    For i:= 1 To Nt Do
        terminal[i] := IDLE;
End;{InitTerm}

{****************************************************************************}
PROCEDURE InitDelayArray(VAR DelayArray:PosArr);
VAR
    i : Integer; { Loop variable }
BEGIN
    FOR i := 1 TO MaxTerm DO
        DelayArray[i] := 0;
END;{InitDelayArray}

{****************************************************************************}
PROCEDURE InitBusyTone(VAR BusyTone:OnOff);
BEGIN
    BusyTone := Off;
END;{InitBusyTone}

{****************************************************************************}
{  Initcode reads Gold codes from a file and has the cross correlation      }
{  catalog setup                                                            }
{                                                                           }
{  Depends on: config                                                       }
{  Calls      : initCCat                                                    }
{  Modifies   : pfn                                                         }
{****************************************************************************}
Procedure InitCodes(Var CCat:CArr; groupSize, numCodes:Integer);
Var
    i,                           { Loop variable                            }
    j           :Integer;        { Loop variable                            }
    f           :text;           { File containing the Gold codes           }
    codeCount,                   { Length of transmission code              }
    cfile       :String;         { Codefile name                            }
    gc          :StringArr;      { Gold codes as read from file             }
Begin
    ClrScr;
    Str(codelength, codeCount);  { Assign appropriate codefile name         }
    cfile := 'codes\'+codeCount+'chip.dat';
    Assign(f, cfile);            { Open code file                           }
    Reset(f);
    For i := 1 To Nt Do          { Read the codes from file                 }
        ReadLn(f,gc[i]);
        Close(f);
        If numCodes <> Nt        { Reassign codes according to reuse scheme }
        Then                     { e.g. 12345678 -> 11223344 for 2 term/grp }
            For i := numCodes DownTo 1 Do
                For j := groupSize DownTo 1 Do
                    gc[(i-1)*groupSize+j] := gc[i];
                    initCCat(CCat, gc);
                                 { Initialize catalog of cross correlations }
End;{InitCodes}

{****************************************************************************}
{  InitPathLoss sets up a table of transmitter to receiver path gains from  }
{  all transmitters to all receivers. All gains are between 0 and 1. When   }
{  groups are larger than one terminal duplicate entries will exist in the  }
{  table because there are less receivers than terminals then.              }
{                                                                           }
{  Depends on: config, lambda                                               }
{  Calls      : Power                                                       }
{  Modifies   : none                                                        }
{****************************************************************************}
Procedure InitPathGain(Var S:Matrix; groupSize, numberOfLines :Integer;
                                     Rreceive,  Rterm, lossexp:Extended );
Var
    gamma,                          { Angle of receiver relative to x-axis    }
    theta   :Extended;              { Angle of transmitter relative to x-axis }
    xt, yt,                         { Transmitter coordinates                 }
    xr, yr  :PosArr;                { Receiver coordinates                    }
    i,                              { Loop variable                           }
    j,                              { Loop variable                           }
    k       :Integer;               { Index variable                          }
Begin
    For i := 1 To numberOfLines Do{ Loop over all receivers                   }
        For j := 1 To groupSize Do{ Loop over all group terminals             }
        Begin
          k := (i-1) * groupSize + j;         {Determine index current term. }
          gamma := 2*Pi * (i-1)/numberOfLines;   {Calc. receiver angle       }
          xr[k] := Rreceive * Cos(gamma);        {Calc. receiver x-coord.    }
          yr[k] := Rreceive * Sin(gamma);        {Calc. receiver y-coord.    }
          theta := 2*Pi*(j-1)/groupSize+gamma+Pi;{Calc. transmitter angle    }
          xt[k] := xr[k] + Rterm * Cos(theta);   {Calc. transmitter x-coord.}
          yt[k] := yr[k] + Rterm * Sin(theta);   {Calc. transmitter y-coord.}
        End;
        For i := 1 To Nt Do                   {Double loop over all terminals}
            For j := 1 To Nt Do
            Begin
                { Calculate path gains according to far field model }
                S[i,j] := Power(lambda/(4*Pi*Sqrt(Sqr(xt[i]-xr[j])
                          +Sqr(yt[i]-yr[j]))),lossexp);
            End
End;{InitPathGain}

{****************************************************************************}
{  Rayleigh randomly draws a value according to the rayleigh distribution   }
{                                                                           }
{  Depends on: none                                                         }
{  Calls      : duni                                                        }
{  Modifies   : pfn                                                         }
{****************************************************************************}
Function Rayleigh(ave:Extended):Extended;
Var
    r,
    u,
    v,
    x,
    y       :Extended;
```

```
Begin
     Repeat
           u := 2*duni-1;
           v := 2*duni-1;
           r := sqr(u)+sqr(v);
     Until (r<1.0) and (r>0.0);
     r := Sqrt(-2*Ln(r)/r);
     x := u*r;
     y := v*r;
     Rayleigh:=Sqrt(Sqr(x)+Sqr(y))*ave*Sqrt(2/Pi);
End;

{*****************************************************************************}
{  MarcumQ calculates the Marcum Q-function                                   }
{                                                                             }
{  Depends on: none                                                           }
{  Calls     : none                                                           }
{  Modifies  : pfn                                                            }
{*****************************************************************************}
Function MarcumQ(a,b:Extended):Extended;
Var
   aa,
   ab,
   ac,
   bc,
   bb,
   ba,
   dd,
   d1      :Extended;
   n       :Integer;
Begin
     If a = 0
     Then
         If b < 30
         Then
             MarcumQ := Exp(-Sqr(b)/2)
         Else
             MarcumQ := 0
     Else
     Begin
          If a < b
          Then
          Begin
               ab := 1;
               dd  := a/b;
          End
          Else
          Begin
               ab := 0;
               dd  := b/a;
          End;
          aa := 0  ;
          bb := 0.5;
          ba := 0  ;
          d1 := dd;
          n  := 0;
          Repeat
                n  := n + 1;
                ac := d1+2*n*ab/(a*b)+aa;
                bc := 1+2*n*bb/(a*b)+ba;
                d1 := dd*d1;
                aa := ab;
                ab := ac;
                ba := bb;
                bb := bc;
          Until bc > 1e12;
          If a < b
          Then
              MarcumQ := (ac/(2*bc))*Exp(-Sqr(a-b)/2)
          Else
              MarcumQ := 1-(ac/(2*bc))*Exp(-Sqr(a-b)/2);
     End;
End;

{*****************************************************************************}
{  NegExpI0 approximates the modified Bessel function of the first kind       }
{  using the recurrence relation I     (z) = I     (z) + 2n/z I   (z)          }
{                                  0,n-1       0,n+1           0,n             }
{                                                                             }
{  and then multiplies the result with Exp(-(a^2+b^2)/2).                     }
{                                                                             }
{  Depends on: none                                                           }
{  Calls     : none                                                           }
{  Modifies  : pfn                                                            }
{*****************************************************************************}
Function NegExpI0(a,b:Extended):Extended;
Var
   n    :Integer;                                            { Loop variable      }
   x,
   Ja,                                                       { I0,n-1             }
   Jb,                                                       { I0,n               }
   Jc,                                                       { I0,n+1             }
   tmp,
   t    :Extended;                                           { Intermediate result }
Begin
     tmp := (Sqr(a)+Sqr(b))/2;
     x   := a*b;
     If x = 0
     Then
         If tmp < 150
         Then
             NegExpI0 := Exp(-tmp)/2
         Else
             negExpI0 := 0
     Else
     Begin
          Jc := 0;
          Jb := 1;
          t  := 0;
          For n := 20 DownTo 1 Do
```

```
            Begin
                Ja := (2*n/x)*Jb+Jc;
                Jc := Jb;
                Jb := Ja;
                t  := t + Ja;
            End;
            NegExpI0 := Exp(x-(Sqr(a)+Sqr(b))/2)*Ja/(2*t-Ja);
        End;
End;

{****************************************************************************}
{  CalcMu1 calculates the Mu1 parameter for the bit error formula in        }
{  Rayleigh fading environments                                             }
{                                                                           }
{  Depends on: L                                                            }
{  Calls      : R, Rhat                                                     }
{  Modifies   : none                                                        }
{****************************************************************************}
Function CalcMu1(u, h:Integer; tau:RArr; mu2:Extended):Extended;
Var
   tmp :Extended;                                  { Intermediate result }
   i   :Integer;                                   { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To L Do
        If i <> h
        Then
            tmp := tmp + R(u,u,tau[u,i])*Rhat(u,u,tau[u,i])*s[u,u];

    CalcMu1 := 4*2*Pt*tmp + mu2
End;

{****************************************************************************}
{  CalcMu2 calculates the Mu2 parameter for the bit error formula in        }
{  Rayleigh fading environments                                             }
{                                                                           }
{  Depends on: config, L, Pt, terminal                                      }
{  Calls      : R, Rhat                                                     }
{  Modifies   : none                                                        }
{****************************************************************************}
Function CalcMu2(u, h:Integer; tau:RArr; sigma_n2:Extended):Extended;
Var
   tmp :Extended;                                  { Intermediate result }
   i,                                              { Loop variable       }
   j   :Integer;                                   { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To Nt Do
       If (terminal[i] = INTERFERE) And (i <> u)
       Then
           For j := 1 To L Do
               tmp := tmp + (Sqr(R(u,i,tau[i,j]))
                           +  Sqr(Rhat(u,i,tau[i,j])))*s[i,u];

    For j := 1 To L Do
       If j <> h
       Then
           tmp := tmp + (Sqr(R(u,u,tau[u,j]))
                       +  Sqr(Rhat(u,u,tau[u,j])))*s[u,u];

    CalcMu2 := 2*(2*Pt*tmp+sigma_n2);
End;

{****************************************************************************}
{  CalcMu12 calculates the Mu12 parameter for the bit error formula in      }
{  Rayleigh fading environments                                             }
{                                                                           }
{  Depends on: config, L, Pt, terminal                                      }
{  Calls      : R, Rhat                                                     }
{  Modifies   : none                                                        }
{****************************************************************************}
Function CalcMu12(u,h:Integer;tau:RArr):Extended;
Var
   tmp :Extended;                                  { Intermediate result }
   i,                                              { Loop variable       }
   j   :Integer;                                   { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To Nt Do
       If (terminal[i] = INTERFERE) And (i<>u)
       Then
           For j := 1 To L Do
              tmp := tmp + R(u,i,tau[i,j])*Rhat(u,i,tau[i,j])*s[i,u];

    For j := 1 To L Do
       If j <> h
       Then
           tmp := tmp + (Sqr(Rhat(u,u,tau[u,j]))
                       + R(u,u,tau[u,j])*Rhat(u,u,tau[u,j]))*S[u,u];

    CalcMu12 := 2*2*Pt*tmp;
End;

{****************************************************************************}
{  CalculatePbe computes the bit error probability for a terminal, given    }
{  which other terminals transmit                                           }
{                                                                           }
{  Depends on: config, L, Pt, S, Tb, terminal                               }
{  Calls      : duni, I0, MarcumQ, Rayleigh                                 }
{  Modifies   : none                                                        }
{****************************************************************************}
Function CalculatePbe(u:Integer):Extended;
Var
   mu1,
   mu2,
   mu12,
   a,
   b,
   bmax,                          { Maximum path gain                  }
   tmp,
```

```
   m,
   mq,
   mb,
   sigma_n2:Extended;             { Noise power                              }
   beta,                          { Rayleigh distributed path gains          }
   tau     :RArr;                 { Uniformly [0,Tb] distributed path delays }
   h,                             { Number of path with largest gain         }
   i,                             { Loop variable                            }
   j       :Integer;              { Loop variable                            }
Begin
    { Assign random delays and path gains to transmitting terminals }
    For i := 1 To Nt Do
       If terminal[i] = INTERFERE
       Then
          For j := 1 To L Do
          Begin
             tau[i,j]  := Tb*duni;
             beta[i,j] := Rayleigh(Sqrt(S[i,u]));
          End
       Else
          For j := 1 To L Do
          Begin
             tau[i,j]  := 0;
             beta[i,j] := 0;
          End;

    { Determine what the largest gain is }
    bmax := 0;
    h    := 1;
    For i := 1 To L Do
       If beta[u,i]>bmax
       Then
       Begin
          bmax := beta[u,i];
          h    := i;
       End;

    { Determine noise power relative to largest incoming signal }
    sigma_n2 := power(10, -snr/10)*Sqr(bmax)*Pt*Tb*Tb;

    { Calculate bit error probability parameters }
    mu2  := CalcMu2(u, h, tau, sigma_n2);
    mu1  := CalcMu1(u, h, tau, mu2);
    mu12 := CalcMu12(u, h ,tau);
    m    := Sqrt(2*Pt)*bmax*Tb;
    a    := m*Abs(1/Sqrt(mu1)-1/Sqrt(mu2))/Sqrt(2);
    b    := m*(1/Sqrt(mu1)+1/Sqrt(mu2))/Sqrt(2);
    mq   := MarcumQ(a,b);
    mb   := NegExpI0(a,b);

    { Calculate the bit error probability }
    tmp := ( (1 + mu12/Sqrt(mu1*mu2))*mb )/2;
    CalculatePbe := mq-tmp
End;
```

```
{****************************************************************************}
{                               Q U E U E S                                  }
{****************************************************************************}


{****************************************************************************}
{  CreateQueue                                                               }
{  Pre : None                                                                }
{  Post: The queue Q has been created and is initialized to be empty.        }
{****************************************************************************}
PROCEDURE CreateQueue(VAR Q:QueueType);
BEGIN
     WITH Q DO
          BEGIN
               Count := 0;
               Front := 1;
               Rear := 0
          END
END;{CreateQueue}

{****************************************************************************}
PROCEDURE Append (TerminalNo:Integer; Time:Extended; VAR Q:QueueType);
VAR
   X : QueueEntry;
BEGIN
     WITH Q DO
          IF Count = MaxTerm THEN
             WRITELN ('Error: Attempt to append an entry to a full queue')
          ELSE
             BEGIN
                  X.TerminalNo := TerminalNo;
                  X.Time       := Time;
                  Count := Count + 1;
                  Rear := (Rear MOD MaxTerm) + 1;
                  Entry [Rear] := X;
             END
END;{Append}

{****************************************************************************}
PROCEDURE Remove (VAR X: QueueEntry; VAR Q:QueueType);
BEGIN
     WITH Q DO
          IF Count = 0 THEN
             WRITELN ('Error: Attempt to remove an entry from an empty queue')
          ELSE
             BEGIN
                  Count := Count - 1;
                  X := Entry [Front];
                  Front := (Front MOD MaxTerm) + 1
             END
END;{Remove}

{****************************************************************************}
FUNCTION QueueSize (VAR Q: QueueType): INTEGER;
BEGIN
```

```
     QueueSize := Q.Count
END;{QueueSize}

{******************************************************************************}
FUNCTION QueueEmpty (VAR Q: QueueType): BOOLEAN;
BEGIN
     QueueEmpty := (Q.Count = 0)
END;{QueueEmpty}

{******************************************************************************}
FUNCTION QueueFull (VAR Q: QueueType): BOOLEAN;
BEGIN
     QueueFull := (Q.Count = MaxTerm)
END;{QueueFull}

{******************************************************************************}
PROCEDURE ShowQueue (VAR Q : QueueType);
VAR
   x, y : INTEGER;
   X1   : QueueEntry;
BEGIN
     {WRITELN('De inhoud van de queue is:');}
     x := QueueSize (Q);
     WITH Q DO
          y := front;
          WHILE x > 0 DO
          BEGIN
               X1 := Q.Entry[y];
               WRITELN('TN: ',X1.TerminalNo,'Time: ',X1.Time);
               x := x-1;
               y := (y MOD MaxTerm) + 1
          END;
          WRITELN;
END;{ShowQueue}


{******************************************************************************}
{                              L I N K E D - L I S T                           }
{******************************************************************************}


{******************************************************************************}
{*  CreateList initializes linked list LL to be empty                        *}
{*                                                                           *}
{*  Pre:  None                                                               *}
{*  Post: The list LL has been created and is initialized to be empty        *}
{******************************************************************************}
PROCEDURE CreateList(VAR LL: LinkedList);
BEGIN
  LL.head := NIL
END;{CreateList}

{******************************************************************************}
PROCEDURE InsertEvent(NewNode:ListPointer; T:Extended; VAR LL:LinkedList);
VAR
```

```
   Current,
   Trailing : ListPointer;
BEGIN
     Current  := LL.head;
     Trailing := NIL;
     WHILE (Current^.Time <= T) AND (Current^.NextNode <> NIL) DO
     BEGIN
          Trailing := Current;
          Current  := Current^.NextNode;
     END;
     IF (Current^.time <= T) AND (Current^.NextNode = NIL)
     THEN
     BEGIN
          Current^.NextNode := NewNode;
          NewNode^.NextNode := NIL;
     END
     ELSE IF (Current^.Time > T) AND (Trailing = NIL)
          THEN
          BEGIN
               NewNode^.NextNode   := LL.head;
               LL.head             := NewNode;
          END
          ELSE
          BEGIN
               NewNode^.NextNode  := Current;
               Trailing^.NextNode := NewNode;
          END;
END;{InsertEvent}

{******************************************************************************}
PROCEDURE CreateEvent(E:EventType; T:Extended; TN:Integer; VAR LL:LinkedList);

VAR
   NewNode : ListPointer;
BEGIN
     New(NewNode);
     NewNode^.Event       := E;
     NewNode^.Time        := T;
     NewNode^.TerminalNo  := TN;
     IF LL.head = NIL
     THEN
     BEGIN
          NewNode^.NextNode := NIL;
          LL.head           := NewNode;
     END
     ELSE
     BEGIN
          InsertEvent(NewNode, T, LL);
     END;
END;{CreateEvent}

{******************************************************************************}
PROCEDURE ShowList(VAR LL:LinkedList);
VAR
   Current : ListPointer;
```

```
   i        : Integer;
   Event    : String;
BEGIN
     i        := 1;
     Current := LL.head;
     ClrScr;
     WriteLn('The list consists of: (Press ENTER for the nodes to show up)');
     WriteLn;
     IF Current = NIL THEN
        BEGIN
             WRITELN('EventList is empty! (WriteList)');
        END
     ELSE
     WHILE (Current <> NIL) DO
     BEGIN
          CASE Current^.Event OF
               Arrival     : Event := 'Arrival';
               ReArrival   : Event := 'ReArrival';
               BusyToneOn  : Event := 'BusyToneOn';
               BusyToneOFF : Event := 'BusyToneOff';
               Receive     : Event := 'Receive';
          ELSE WRITELN('ERROR in EventList (ShowList)!!!!!');
          END;
          WRITELN('Node:',i,'      Event:',Event:12,'      Time:',
                   Current^.Time:4,'      Terminal:',Current^.TerminalNo);
     i := i+1;
     Current := Current^.NextNode;
     ReadLn;
     END;
     WriteLn('End of list');
     ReadLn;
END;{ShowList}

{****************************************************************************}
PROCEDURE ProcessArrival(FirstNode:ListPointer; VAR BusyTone:OnOff;
                         VAR Terminal:TermArr; VAR LL:LinkedList;
                         VAR Q:QueueType);
VAR
   RandomDelay,
   Time,
   TInterArr : Extended;
   TN        : Integer;
BEGIN
     Clock     := FirstNode^.Time;
     TN        := FirstNode^.TerminalNo;
     Time      := FirstNode^.Time;
     TInterArr := -(ln(1-DUNI))/PArrH;
     CreateEvent(Arrival, Time + TInterArr, TN, LL);
     IF Terminal[TN] = IDLE
     THEN
     BEGIN
          Terminal[TN] := BLOCKED;
          Inc(ArrivalCount);
          IF BusyTone = ON
          THEN
          BEGIN
               RandomDelay := -(ln(1-DUNI))/Alpha;
               CreateEvent(ReArrival, Time + RandomDelay, TN, LL);
               DelayArray[TN] := DelayArray[TN] + RandomDelay;
               Inc(GCount);
          END
          ELSE
          BEGIN
               Terminal[TN] := TRANSMIT;
               Append(TN, Time, Q);
               CreateEvent(BusyToneOn, Time+Fraction, 0, LL);
               CreateEvent(Receive, Time+1+Fraction, TN, LL);
               DelayArray[TN] := DelayArray[TN] + 1 + Fraction;
          END
     END
END;{ProcessArrival}

{****************************************************************************}
PROCEDURE ProcessReArrival(FirstNode:ListPointer; VAR BusyTone:OnOff;
                           VAR Terminal:TermArr; VAR LL:LinkedList;
                           VAR Q:QueueType);
VAR
   RandomDelay,
   Time,
   TInterArr : Extended;
   TN        : Integer;
BEGIN
     Clock   := FirstNode^.Time;
     TN      := FirstNode^.TerminalNo;
     Time    := FirstNode^.Time;
     IF BusyTone = ON
     THEN
     BEGIN
          RandomDelay := -(ln(1-DUNI))/Alpha;
          CreateEvent (ReArrival, Time + RandomDelay, TN, LL);
          DelayArray[TN] := DelayArray[TN] + RandomDelay;
          Inc(GCount);
     END
     ELSE
     BEGIN
          Terminal[TN] := TRANSMIT;
          Append(TN, Time, Q);
          CreateEvent(BusyToneOn, Time+Fraction, 0, LL);
          CreateEvent(Receive, Time+1+Fraction, TN, LL);
          DelayArray[TN] := DelayArray[TN] + 1 + Fraction;
     END
END;{ProcessReArrival}

{****************************************************************************}
PROCEDURE ProcessBusyToneOn(FirstNode:ListPointer; VAR BusyTone:OnOff;
                            VAR LL:LinkedList);
BEGIN
     Clock := FirstNode^.Time;
     BusyTone := ON;
END;{ProcessBusyToneOn}
```

```pascal

{*****************************************************************************}
PROCEDURE ProcessBusyToneOff (FirstNode:ListPointer; VAR BusyTone:OnOff);
BEGIN
     Clock := FirstNode^.Time;
     BusyTone := OFF;
END;{ProcessBusyToneOff}

{*****************************************************************************}
PROCEDURE ProcessReceive(FirstNode:ListPointer; VAR BusyTone:OnOff;
                         VAR Terminal:TermArr; VAR LL:LinkedList;
                         VAR Q:QueueType);
VAR
   X2            : QueueEntry;

   TN,
   TN2           : Integer;

   RandomDelay,
   Time,
   Time2,
   Pbe,
   Psp           : Extended;

BEGIN
     Clock := FirstNode^.Time;
     Time  := FirstNode^.Time;
     IF QueueEmpty(Q) = TRUE
     THEN
     BEGIN
          TN             := FirstNode^.TerminalNo;
          RandomDelay  := -(ln(1-DUNI))/Alpha;
          Terminal[TN] := BLOCKED;
          CreateEvent(ReArrival, Time+RandomDelay, TN, LL);
          DelayArray[TN]   := DelayArray[TN] + RandomDelay;
          Inc(GCount);
     END
     ELSE
     BEGIN
          TN             := FirstNode^.TerminalNo;
          Terminal[TN] := INTERFERE;
          WHILE NOT QueueEmpty(Q) DO
          BEGIN
               Remove(X2, Q);
               TN2 := X2.TerminalNo;
               Time2 := X2.Time;
               Terminal[TN2] := INTERFERE;
          END;
          Pbe := CalculatePbe(TN);
          Psp := Power(1-Pbe, PacketLength);
          CreateEvent(BusyToneOff, Time2+1+Fraction, 0, LL);

          IF DUNI < Psp
          THEN
          BEGIN
               Inc(SuccessCount);
               Terminal[TN]   := IDLE;
               DelayCount     := DelayCount + DelayArray[TN];
               DelayArray[TN] := 0;
          END
          ELSE
          BEGIN
               Terminal[TN] := BLOCKED;
               RandomDelay  := -(ln(1-DUNI))/Alpha;
               CreateEvent(ReArrival, Time+RandomDelay, TN, LL);
               DelayArray[TN] := DelayArray[TN] + RandomDelay;
               Inc(GCount);
          END
     END
END;{ProcessReceive}

{*****************************************************************************}
PROCEDURE ProcessFirstNode(FirstNode:ListPointer; VAR LL:LinkedList;
                           VAR Q:QueueType);

BEGIN
     CASE FirstNode^.Event OF
          Arrival     : ProcessArrival(FirstNode, BusyTone, Terminal, LL, Q);
          ReArrival   : ProcessReArrival(FirstNode, BusyTone, Terminal, LL, Q);
          BusyToneOn  : ProcessBusyToneOn(FirstNode, BusyTone, LL);
          BusyToneOff : ProcessBusyToneOff(FirstNode, BusyTone);
          Receive     : ProcessReceive(FirstNode, BusyTone, Terminal, LL, Q);
     ELSE WRITELN('ERROR in EventList (ProcessFirstNode)!!!!!');
     END;

END;{ProcessFirstNode}

{*****************************************************************************}
PROCEDURE AdvanceNextEvent(VAR LL:LinkedList; VAR Q:QueueType);
VAR
   FirstNode : ListPointer;
BEGIN
     IF LL.head = NIL
     THEN
     BEGIN
          WRITELN('Error: EventList is empty (AdvanceNextEvent)');
     END
     ELSE IF LL.head^.NextNode = NIL
          THEN
          BEGIN
               FirstNode := LL.head;
               LL.head   := NIL;
               ProcessFirstNode(FirstNode, LL, Q);
          END
          ELSE
          BEGIN
               FirstNode := LL.head;
               LL.head   := LL.head^.NextNode;
               ProcessFirstNode(FirstNode, LL, Q);
          END;
```

```
    Dispose(FirstNode);
END;{AdvanceNextEvent}

{*****************************************************************************}
PROCEDURE InitEventList(VAR LL:LinkedList);
VAR
   i    : Integer;
   Time : Extended;
BEGIN
    FOR i := 1 TO Maxterm DO
    BEGIN
         Time := -(ln(1-DUNI))/PArrH;
         CreateEvent(Arrival, Time, i, LL);
    END
END;{InitEventList}

{*****************************************************************************}
FUNCTION CountList(VAR LL:LinkedList):Integer;
VAR
   i       : Integer;
   Current : ListPointer;
BEGIN
    i := 0;
    Current := LL.head;
    WHILE Current <> NIL DO
    BEGIN
         Current := Current^.NextNode;
         i       := i + 1;
    END;
    CountList := i;
END;{CountList}

{*****************************************************************************}
PROCEDURE RemoveList(VAR LL:LinkedList);
VAR
   Current : ListPointer;
BEGIN
    Current := LL.head;
    WHILE Current <> NIL DO
    BEGIN
         Current := Current^.NextNode;
         Dispose(LL.head);
         LL.head := Current;
    END;
END;{RemoveList}

{*****************************************************************************}
PROCEDURE OpenResultFile(VAR OutFile:Text);
VAR
   GS,
   NL,
   FileName : String;
BEGIN
    Str(GroupSize, GS);
    Str(NumberOfLines, NL);
    FileName := 'V2H8W3.DAT';
    Assign(OutFile, FileName);

    ReWrite(OutFile);
{    WriteLn(OutFile,'Size of group   : ', GroupSize);
    WriteLn(OutFile,'Number of groups: ', NumberOfLines);
    WriteLn(OutFile);
    WriteLn(OutFile,'Inhibit delay fraction: ', Fraction);
    WriteLn(OutFile,'Alpha           : ',Alpha);
    WriteLn(OutFile,'SNR             : ', SNR);
    WriteLn(OutFile);
}
    WriteLn(OutFile, ' G               ':17,
                     'S               ':16,
                     'D               ':16);
END;{OpenResultFile}

{*****************************************************************************}
{  WarnBeep emits a half second warning tone and then waits 4.5 seconds      }
{                                                                             }
{  Depends on: none                                                           }
{  Calls     : none                                                           }
{  Modifies  : pfn                                                            }
{*****************************************************************************}
Procedure WarnBeep;
VAR
   i : Integer;
BEGIN
    FOR i := 1 TO 5 DO
    BEGIN
         Sound(500+i*20);
         Delay(300);
    END;
    NoSound;
    Delay(1000);
End;{WarnBeep}

{*****************************************************************************}
{                          M A I N   L I N E                                  }
{*****************************************************************************}

Begin
    ClrScr;
    WriteLn('Output Testprogramma:');

    {Initialize variables}
    Rreceive      := 30;
    Rterm         :=  5;
    SNR           :=  6;

    Nt            := 32;
    NumberOfLines :=  4;
    GroupSize     :=  8;

    OpenResultFile(ResultFile);
```

```
    InitCodes(CCat, GroupSize, NumberOfLines);
    InitPathGain(S, GroupSize, NumberOfLines, Rreceive, Rterm, LossExp);

    FOR i := 0 TO NumSim DO              { Loop over terminal activity levels  }
    BEGIN
        {Set terminal activity}
        PArrH := Power(10, -(LRangeMin + i*(LRangeMax-LRangeMin)/NumSim));
        rInit(3647523, 65321);          {Initialize FSU Ultra random generator}
        InitTerm(Terminal);             {Reset terminal states                }
        InitBusyTone(BusyTone);         {Set BusyTone to OFF                  }
        CreateList(EventList);
        CreateQueue(ReceiveQueue);
        InitEventList(EventList);       {Initialize EventList                 }
        InitDelayArray(DelayArray);     {Reset all terminal delays to zero    }

        {Show configuration to user}
        ClrScr;
        WriteLn('Receiver Radius: ', Rreceive:7:2);
        WriteLn('Terminal Radius: ', Rterm:7:2);
        WriteLn('Number of users: ', Nt:4);
        WriteLn('Size of group  : ', GroupSize:4);
        WriteLn('Number of codes: ', NumberOfLines:4);
        WriteLn('Code length    : ', codelength:4);
        WriteLn('Packet length  : ', Packetlength:4);
        WriteLn;
        WriteLn('Terminal activity (PArrH) : ',PArrH:8:5);

        {Reset event counters}
        ArrivalCount    := 0;
        SuccessCount    := 0;
        DelayCount      := 0;
        GCount          := 0;

        {Warm up the network}
        Clock    := 0;
        OldClock := 0;
        While Clock < WarmTime DO
        BEGIN
            AdvanceNextEvent(EventList, ReceiveQueue);
            IF (Clock - OldClock) > Interval
            THEN
            BEGIN
                GoToXY(1,12);
                WriteLn('Arrivals :', ArrivalCount:8);
                WriteLn('Success  :', SuccessCount:8);
                WriteLn;
                WriteLn('Delay    :', DelayCount);
                WriteLn('Clock    :', Clock);
                ShowState;
                OldClock := Clock;
            END;
        GoToXY(40,4);
        WriteLn('SIMULATING !!!!');
        GoTOXY(40,6);
        Write('STATUS: ',i,'/',NumSim);
        END;{Warm up loop}

        {Reset event counters}
        ArrivalCount    := 0;
        SuccessCount    := 0;
        GCount          := 0;
        DelayCount      := 0;

        {Main simulation Loop}
        While Clock < EndTime DO
        BEGIN
            AdvanceNextEvent(EventList, ReceiveQueue);
            IF (Clock - OldClock) > Interval
            THEN
            BEGIN
                GoToXY(1,12);
                WriteLn('Arrivals :', ArrivalCount:8);
                WriteLn('Success  :', SuccessCount:8);
                WriteLn;
                WriteLn('Delay    :', DelayCount);
                WriteLn('Clock    :', Clock);
                ShowState;
                OldClock := Clock;
            END;
        GoToXY(40,4);
        WriteLn('SIMULATING !!!!');
        GoTOXY(40,6);
        Write('STATUS: ',i,'/',NumSim);
        END;{Simulation Loop}

        {Calculate values for Physical Traffic, Throughput and Delay}
        Throughput := SuccessCount/(EndTime-WarmTime);
        D          := DelayCount/SuccessCount;
        G          := (ArrivalCount+GCount)/(EndTime-WarmTime);

        {Write simulation results in files for processing-purposes later on}
        WriteLn(ResultFile  , G          :16,
                              Throughput :16,
                              D          :16);

        RemoveList(EventList);

    END;{Terminal activity level loop}

    Close(ResultFile);

    {WarnBeep;}
    GoToXY(30,16);
    Write('I AM READY MISTER LE !!!!!!');
    GoToXY(30,17);
    Write('Press ENTER to return to program.');
    {ReadLn;}
END.
```

```
{*****************************************************************************}
{*  V2_CALC1.pas is a program designed with the goal to calculate the        *}
{*  performance of the 'Unslotted Hybrid CDMA/ISMA protocol'. This program   *}
{*  calculates the throughput and delay of a network of terminals by         *}
{*  a mathematical formula.                                                  *}
{*                                                                           *}
{*  Most of the constants, variables, functions and procedures used here     *}
{*  originates from Huub van Roosmalen.                                      *}
{*                                                                           *}
{*  The terminals are clustered around distributed receivers at a fixed      *}
{*  distance. The distributed receivers are clustered around the base        *}
{*  station at a fixed distance. The packets from the terminals are          *}
{*  received by the distributed receivers and sent to the base station by    *}
{*  wire or lines. These lines can be seen as primitive concentrators.       *}
{*                                                                           *}
{*  Most time parameters are normalized to the PacketDuration. To have a     *}
{*  time parameter in seconds one needs to multiply by the PacketDuration    *}
{*                                                                           *}
{*  Huy Linh Anh Le, July 16 1994                                            *}
{*****************************************************************************}

Program CalcUnslotted(Input, Output);

Uses    Dos, Crt, Ultrax;

Const   MonteCarlo    =          1000;
        MaxTerm       =            32; { Maximum number of terminals allowed  }
        XMax          =             4; { Maximum number of interference pckts }
        PacketLength  =            64; { Number of bits per packet            }

        Pt            =             1; { Transmitter power in Watts           }
        LossExp       =             2; { Attenuation parameter for the ...    }
                                       {  ...far field model                  }
        c             =     299792458; { Speed of light in m/s                }
        f             =    1700000000; { Transmitter center frequency         }
        lambda        =           c/f; { Signal wavelength                    }

        twoPi         =        2 * Pi; { Two times Pi                         }
        Pi2           =       Pi * Pi; { Pi squared                           }

        Alpha         = 0.1;           { x retransmission per PacketDuration  }
        Fraction      = 0.01;                    { Inhibit delay fraction (d) }

        bitrate       = 256*1024;                { Bits transmitted per sec.  }
        PacketDuration= 64/bitrate;              {Time needed to send a Packet}
        maxcodelength = 127;                     { Maximum code size          }
        CodeLength    =  31;                     { Length of code             }
        Tb            = 1/bitrate;               { Bit duration               }
        Tc            = 1/(codelength*bitrate);  { Chip duration              }
        Tm            = 100E-9;                  { Maximum delay spread       }
        MaxL          = 8;                       { Maximum number of paths    }
        L             = Trunc(Tm/Tc)+1;          { Number of paths            }

        NumSim        = 20;
        LRangeMin     = 3;
        LRangeMax     = -2;

Type    TermState = (IDLE, BLOCKED, TRANSMIT, INTERFERE);
        Matrix     = Array[1..MaxTerm, 1..MaxTerm]        Of Extended;
        PosArr     = Array[1..MaxTerm]                    Of Extended;
        RArr       = Array[1..MaxTerm, 1..MaxL]           Of Extended;
        StringArr  = Array[1..MaxTerm]                    Of String;
        TermArr    = Array[1..MaxTerm]                    Of TermState;
        ConfigArr  = Array[1..XMax]                       Of Integer;
        PpsArr     = Array[1..XMax]                       Of Extended;
        ExtArr     = Array[0..XMax]                       Of Extended;
        SArr       = Array[0..NumSim]                     Of Extended;
{-----------------------------------------------------------------------------}
        p2CRec     = ^CRec;

        CRec       = Record
                        cc :Array[-maxcodelength..maxcodelength] Of ShortInt;
                     End;


        CArr       = Array [1..MaxTerm, 1..MaxTerm] Of p2CRec;


Var     CCat            : CArr;      { Catalog of code cross correlations      }

        Rreceive,                    { Receiver radius around base station     }
        Rterm,                       { Terminal radius around receivers        }
        SNR             :Extended;   { Signal to white Noise Ratio at receiver }

        Nt,                          { Number of terminals                     }
        GroupSize,                   { Number of terminals per group           }
        NumberOfLines   :Integer;    { Number of groups in the system. This is }
                                     { ... also the number of codes used.      }

        S               :Matrix;     { Power loss ratio matrix                 }

        Terminal        :TermArr;    { Terminal state information              }

        X,
        i,
        j               :Integer;    {Loopvariable for terminal activity levels}

        CountConfig     :LongInt;
        StartConfig,
        MaxConfig       :ConfigArr;
        PpsAveXArr      :PpsArr;
        SArray          :SArr;

        ResultFile      :Text;

        PpsAveX,
        PpsXZero,
        PConfl,
        PConflTemp,
        PNoConfl,
```

```
          ExpGd,
          Poisson,
          TCycle,
          Pidle,
          Psuccess,
          Throughput,
          D,
          G              : Extended;

          Faculteit      : ExtArr;

{*****************************************************************************}
{  Power calculates a raised to the power of b                                }
{                                                                             }
{  Depends on: none                                                           }
{  Calls     : none                                                           }
{  Modifies  : none                                                           }
{                                                                             }
{  Pre : a >= 0 And b >=0                                                     }
{  Post: Power = a^b                                                          }
{*****************************************************************************}
Function Power(a,b:Extended):Extended;
Begin
     If a<>0
     Then
         power := Exp(b*Ln(a))
     Else
         If b<>0
         Then
             power := 0                                      { power(0,b) = 0 }
         Else
             power := 1;                                     { power(0,0) = 1 }
End;{Power}

{*****************************************************************************}
{  InitFact                                                                   }
{*****************************************************************************}
Procedure InitFact (Var fact:ExtArr);
Var
   i : Integer;
Begin
     fact[0] := 1;
     For i := 1 TO XMax DO
         fact[i] := i*Fact[i-1];
End;{InitFact}

{*****************************************************************************}
{  ShowState shows the terminal states on screen                              }
{                                                                             }
{  Depends on: config, numberOfLines, terminal                                }
{  Calls     : none                                                           }
{  Modifies  : none                                                           }
{*****************************************************************************}
Procedure ShowState;
Var
   i : Integer;
   c : Char;
Begin
     GotoXY(12, 22);
     Write('                  1                   2                   3    ');
     GotoXY(12, 23);
     Write('1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 3 2 4 5 6 7 8 9 0 1 2');
     For i := 1 To Nt Do
     Begin
          GotoXY(10+2*i,24);
          Case Terminal[i] Of
              IDLE        : Write('i');
              BLOCKED     : Write('b');
              TRANSMIT    : Write('t');
              INTERFERE   : Write('f');
          End;{Case}
     End;{For i := 1 TO Nt}
End;{ShowState}

{*****************************************************************************}
{  R calculates a partial cross correlation                                   }
{                                                                             }
{  Depends on: config, CCat, Tc                                               }
{  Calls     : none                                                           }
{  Modifies  : none                                                           }
{*****************************************************************************}
Function R(i,j:Integer; tau:Extended):Extended;
Var
   l,                            { Chip number belonging to tau               }
   Cl,                           { Discrete correlation function for chip l   }
   Cl1       :Integer;           { Discrete correlation function for chip l+1 }
Begin
     l   := Trunc(tau/Tc);
     Cl1 := CCat[i,j]^.cc[l+1-codelength];
     If l >= 1
     Then
     Begin
          Cl  := CCat[i,j]^.cc[l-codelength];
          R   := Cl*Tc+(Cl1-Cl)*(tau-l*Tc);
     End
     Else
         R := tau*Cl1;
End;{R}

{*****************************************************************************}
{  Rhat calculates a partial cross correlation                                }
{                                                                             }
{  Depends on: config, CCat, Tc                                               }
{  Calls     : none                                                           }
{  Modifies  : none                                                           }
{*****************************************************************************}
Function Rhat(i,j:Integer; tau:Extended):Extended;
Var
   l,                            { Chip number belonging to tau               }
   Cl,                           { Discrete correlation function for chip l   }
```

```
   Cl1      :Integer;            { Discrete correlation function for chip l+1 }
Begin
    l   := Trunc(tau/Tc);
    Cl  := CCat[i,j]^.cc[l];
    If l < codelength-1
    Then
    Begin
        Cl1  := CCat[i,j]^.cc[l+1];
        Rhat := Cl*Tc+(Cl1-Cl)*(tau-l*Tc);
    End
    Else
        Rhat := ((l+1)*Tc-tau)*Cl;
End;{Rhat}

{****************************************************************************}
{  InitCCat sets up a catalog of discrete aperiodic cross correlation        }
{  functions for a set of (Gold) codes passed by gc. The catalog consists of }
{  a two dimensional array of pointers. These pointers refer to              }
{  one-dimensional arrays containing the cross correlation information. This }
{  information is in essence the number of simmilarities minus the numbers   }
{  of dissimilarities between two codes for a given discrete phase shift     }
{                                                                            }
{  Depends on: config, p2CRec                                                }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{****************************************************************************}
Procedure InitCCat(Var CCat:CArr; gc:StringArr);
Var
   i,                                   { Loop variable                  }
   j,                                   { Loop variable                  }
   k,                                   { Loop variable                  }
   l,                                   { Discrete phase shift           }
   t     :Integer;                      { Correlation result for shift l }
Begin
    Write('Calculating ', Nt, ' by ', Nt,' correlations');
    For i := 1 To Nt Do                 { Double loop over all codes     }
        For j := 1 To Nt Do
        Begin
            GotoXY(70,1);               { Show which codes are being done }
            Write(i:4,j:4);
            CCat[i,j] := New(p2CRec);   { Allocate space for new array   }

            { Calculate cross-correlation }
            For l := -codelength To codelength Do
            Begin
                t := 0;                 { Reset correlation value        }
                If l < 0                { Different formulas for pos and }
                Then                    { neg phase shifts               }
                    For k := 0 To codelength-1+l Do
                        If gc[i][1+((k-l) Mod codelength)] = gc[j][k+1]
                        Then
                            Inc(t)
                        Else
                            Dec(t)
                Else
                    For k := 0 To codelength-l-1 Do
                      If gc[i][1+(k Mod codelength)] =
                        gc[j][1+((k+l) Mod codelength)]
                      Then
                          Inc(t)
                      Else
                          Dec(t);

                CCat[i,j]^.cc[l] := t;
            End;
        End;
End;{InitCCat}

{****************************************************************************}
{  DisposeCCat frees memory space taken up by the correlation catalog        }
{                                                                            }
{  Depends on: config                                                        }
{  Calls     : none                                                          }
{  Modifies  : none                                                          }
{****************************************************************************}
Procedure DisposeCCat(CCat:CArr);
Var
   i,                                                      { Loop variable }
   j      :Integer;                                        { Loop variable }
Begin
    For i := 1 To Nt Do                       { Loop over all catalog entries }
        For j := 1 To Nt Do
            Dispose(CCat[i,j]);               { Free memory of element i,j    }
End;{DisposeCCat}

{****************************************************************************}
{ InitTerm initializes all terminals as free                                 }
{                                                                            }
{  Depends on: config                                                        }
{  Calls     : none                                                          }
{  Modifies  : pfn                                                           }
{****************************************************************************}
Procedure InitTerm(Var Terminal:TermArr);
Var
   i :Integer; { Loop variable }
Begin
    For i:= 1 To Nt Do
        terminal[i] := IDLE;
End;{InitTerm}

{****************************************************************************}
{  Initcode reads Gold codes from a file and has the cross correlation       }
{  catalog setup                                                             }
{                                                                            }
{  Depends on: config                                                        }
{  Calls     : initCCat                                                      }
{  Modifies  : pfn                                                           }
{****************************************************************************}
Procedure InitCodes(Var CCat:CArr; groupSize, numCodes:Integer);
Var
```

```
    i,                         { Loop variable                          }
    j           :Integer;      { Loop variable                          }
    f           :text;         { File containing the Gold codes         }
    codeCount,                 { Length of transmission code            }
    cfile       :String;       { Codefile name                          }
    gc          :StringArr;    { Gold codes as read from file           }
Begin
    ClrScr;
    Str(codelength, codeCount);  { Assign appropriate codefile name     }
    cfile := 'codes\'+codeCount+'chip.dat';
    Assign(f, cfile);            { Open code file                       }
    Reset(f);
    For i := 1 To Nt Do          { Read the codes from file             }
        ReadLn(f,gc[i]);
        Close(f);
        If numCodes <> Nt        { Reassign codes according to reuse scheme }
        Then                     { e.g. 12345678 -> 11223344 for 2 term/grp }
            For i := numCodes DownTo 1 Do
                For j := groupSize DownTo 1 Do
                    gc[(i-1)*groupSize+j] := gc[i];
                    initCCat(CCat, gc);
                                 { Initialize catalog of cross correlations }
End;{InitCodes}

{*************************************************************************}
{  InitPathLoss sets up a table of transmitter to receiver path gains from  }
{  all transmitters to all receivers. All gains are between 0 and 1. When   }
{  groups are larger than one terminal duplicate entries will exist in the  }
{  table because there are less receivers than terminals then.              }
{                                                                           }
{  Depends on: config, lambda                                               }
{  Calls      : Power                                                       }
{  Modifies   : none                                                        }
{*************************************************************************}
Procedure InitPathGain(Var S:Matrix; groupSize, numberOfLines :Integer;
                              Rreceive,  Rterm, lossexp:Extended );
Var
   gamma,                      { Angle of receiver relative to x-axis     }
   theta    :Extended;         { Angle of transmitter relative to x-axis  }
   xt, yt,                     { Transmitter coordinates                  }
   xr, yr   :PosArr;           { Receiver coordinates                     }
   i,                          { Loop variable                            }
   j,                          { Loop variable                            }
   k        :Integer;          { Index variable                           }
Begin
    For i := 1 To numberOfLines Do{ Loop over all receivers               }
        For j := 1 To groupSize Do{ Loop over all group terminals          }
        Begin
          k := (i-1) * groupSize + j;         {Determine index current term. }
          gamma := 2*Pi * (i-1)/numberOfLines;  {Calc. receiver angle       }
          xr[k] := Rreceive * Cos(gamma);       {Calc. receiver x-coord.    }
          yr[k] := Rreceive * Sin(gamma);       {Calc. receiver y-coord.    }
          theta := 2*Pi*(j-1)/groupSize+gamma+Pi;{Calc. transmitter angle   }
          xt[k] := xr[k] + Rterm * Cos(theta);  {Calc. transmitter x-coord.}
          yt[k] := yr[k] + Rterm * Sin(theta);  {Calc. transmitter y-coord.}
        End;
        For i := 1 To Nt Do               {Double loop over all terminals}
            For j := 1 To Nt Do
            Begin
                { Calculate path gains according to far field model }
                S[i,j] := Power(lambda/(4*Pi*Sqrt(Sqr(xt[i]-xr[j])
                              +Sqr(yt[i]-yr[j]))),lossexp);
            End
End;{InitPathGain}

{*************************************************************************}
{*  InitStartConfig
{*************************************************************************}
PROCEDURE InitStartConfig (VAR Start:ConfigArr; X:Integer);
VAR
   i : Integer;
BEGIN
     FOR i := 1 TO X DO
         Start[i] := i+1;
END;{InitStartConfig}

{*************************************************************************}
{*  InitMaxConfig
{*************************************************************************}
PROCEDURE InitMaxConfig (VAR Max:ConfigArr; X:Integer);
VAR
   i : Integer;
BEGIN
     FOR i := X DOWNTO 1 DO
         Max[i] := Nt-(X-i);
END;{InitMaxConfig}

{*************************************************************************}
{*  WriteConfig                                                          *}
{*************************************************************************}
Procedure WriteConfig (Var Config:ConfigArr; X:Integer);
Var
   i : Integer;
Begin
     Write('Configuration = ');
     For i := 1 TO X DO
         Write(Config[i],' ');
     WriteLn;
End;{WriteConfig}

{*************************************************************************}
{  Rayleigh randomly draws a value according to the rayleigh distribution   }
{                                                                           }
{  Depends on: none                                                         }
{  Calls      : duni                                                        }
{  Modifies   : pfn                                                         }
{*************************************************************************}
Function Rayleigh(ave:Extended):Extended;
Var
   r,
```

```
   u,
   v,
   x,
   y      :Extended;
Begin
    Repeat
          u := 2*duni-1;
          v := 2*duni-1;
          r := sqr(u)+sqr(v);
    Until (r<1.0) and (r>0.0);
    r := Sqrt(-2*Ln(r)/r);
    x := u*r;
    y := v*r;
    Rayleigh:=Sqrt(Sqr(x)+Sqr(y))*ave*Sqrt(2/Pi);
End;

{****************************************************************************}
{  MarcumQ calculates the Marcum Q-function                                  }
{                                                                            }
{  Depends on: none                                                          }
{  Calls     : none                                                          }
{  Modifies  : pfn                                                           }
{****************************************************************************}
Function MarcumQ(a,b:Extended):Extended;
Var
   aa,
   ab,
   ac,
   bc,
   bb,
   ba,
   dd,
   d1     :Extended;
   n      :Integer;
Begin
    If a = 0
    Then
        If b < 30
        Then
            MarcumQ := Exp(-Sqr(b)/2)
        Else
            MarcumQ := 0
    Else
    Begin
        If a < b
        Then
        Begin
            ab := 1;
            dd  := a/b;
        End
        Else
        Begin
            ab := 0;
            dd  := b/a;
        End;
            aa := 0  ;
            bb := 0.5;
            ba := 0  ;
            d1 := dd;
            n  := 0;
            Repeat
                  n  := n + 1;
                  ac := d1+2*n*ab/(a*b)+aa;
                  bc := 1+2*n*bb/(a*b)+ba;
                  d1 := dd*d1;
                  aa := ab;
                  ab := ac;
                  ba := bb;
                  bb := bc;
            Until bc > 1e12;
            If a < b
            Then
               MarcumQ := (ac/(2*bc))*Exp(-Sqr(a-b)/2)
            Else
               MarcumQ := 1-(ac/(2*bc))*Exp(-Sqr(a-b)/2);
    End;
End;

{****************************************************************************}
{  NegExpI0 approximates the modified Bessel function of the first kind      }
{  using the recurrence relation I     (z) = I     (z) + 2n/z I   (z)        }
{                                 0,n-1      0,n+1            0,n            }
{                                                                            }
{  and then multiplies the result with Exp(-(a^2+b^2)/2).                    }
{                                                                            }
{  Depends on: none                                                          }
{  Calls     : none                                                          }
{  Modifies  : pfn                                                           }
{****************************************************************************}
Function NegExpI0(a,b:Extended):Extended;
Var
   n     :Integer;                                        { Loop variable      }
   x,
   Ja,                                                    { I0,n-1             }
   Jb,                                                    { I0,n               }
   Jc,                                                    { I0,n+1             }
   tmp,
   t     :Extended;                                       { Intermediate result }
Begin
    tmp := (Sqr(a)+Sqr(b))/2;
    x   := a*b;
    If x = 0
    Then
        If tmp < 150
        Then
            NegExpI0 := Exp(-tmp)/2
        Else
            negExpI0 := 0
    Else
    Begin
```

```
        Jc := 0;
        Jb := 1;
        t  := 0;
        For n := 20 DownTo 1 Do
        Begin
            Ja := (2*n/x)*Jb+Jc;
            Jc := Jb;
            Jb := Ja;
            t  := t + Ja;
        End;
        NegExpI0 := Exp(x-(Sqr(a)+Sqr(b))/2)*Ja/(2*t-Ja);
    End;
End;

{****************************************************************************}
{  CalcMu1 calculates the Mu1 parameter for the bit error formula in         }
{  Rayleigh fading environments                                              }
{                                                                            }
{  Depends on: L                                                             }
{  Calls     : R, Rhat                                                       }
{  Modifies  : none                                                          }
{****************************************************************************}
Function CalcMu1(u, h:Integer; tau:RArr; mu2:Extended):Extended;
Var
   tmp :Extended;                                     { Intermediate result }
   i   :Integer;                                      { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To L Do
        If i <> h
        Then
            tmp := tmp + R(u,u,tau[u,i])*Rhat(u,u,tau[u,i])*s[u,u];

    CalcMu1 := 4*2*Pt*tmp + mu2
End;

{****************************************************************************}
{  CalcMu2 calculates the Mu2 parameter for the bit error formula in         }
{  Rayleigh fading environments                                              }
{                                                                            }
{  Depends on: config, L, Pt, terminal                                       }
{  Calls     : R, Rhat                                                       }
{  Modifies  : none                                                          }
{****************************************************************************}
Function CalcMu2(u, h:Integer; tau:RArr; sigma_n2:Extended):Extended;
Var
   tmp :Extended;                                     { Intermediate result }
   i,                                                 { Loop variable       }
   j   :Integer;                                      { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To Nt Do
      If (terminal[i] = INTERFERE) And (i <> u)
      Then
          For j := 1 To L Do
                tmp := tmp + (Sqr(R(u,i,tau[i,j]))
                           +  Sqr(Rhat(u,i,tau[i,j])))*s[i,u];

    For j := 1 To L Do
       If j <> h
       Then
           tmp := tmp + (Sqr(R(u,u,tau[u,j]))
                      +  Sqr(Rhat(u,u,tau[u,j])))*s[u,u];

    CalcMu2 := 2*(2*Pt*tmp+sigma_n2);
End;

{****************************************************************************}
{  CalcMu12 calculates the Mu12 parameter for the bit error formula in       }
{  Rayleigh fading environments                                              }
{                                                                            }
{  Depends on: config, L, Pt, terminal                                       }
{  Calls     : R, Rhat                                                       }
{  Modifies  : none                                                          }
{****************************************************************************}
Function CalcMu12(u,h:Integer;tau:RArr):Extended;
Var
   tmp :Extended;                                     { Intermediate result }
   i,                                                 { Loop variable       }
   j   :Integer;                                      { Loop variable       }
Begin
    tmp := 0;
    For i := 1 To Nt Do
       If (terminal[i] = INTERFERE) And (i<>u)
       Then
           For j := 1 To L Do
              tmp := tmp + R(u,i,tau[i,j])*Rhat(u,i,tau[i,j])*s[i,u];

    For j := 1 To L Do
       If j <> h
       Then
           tmp := tmp + (Sqr(Rhat(u,u,tau[u,j]))
                      + R(u,u,tau[u,j])*Rhat(u,u,tau[u,j]))*S[u,u];

    CalcMu12 := 2*2*Pt*tmp;
End;

{****************************************************************************}
{  CalculatePbe computes the bit error probability for a terminal, given     }
{  which other terminals transmit                                            }
{                                                                            }
{  Depends on: config, L, Pt, S, Tb, terminal                                }
{  Calls     : duni, I0, MarcumQ, Rayleigh                                   }
{  Modifies  : none                                                          }
{****************************************************************************}
Function CalculatePbe(u:Integer):Extended;
Var
   mu1,
   mu2,
   mu12,
```

```
   a,
   b,
   bmax,                            { Maximum path gain                    }
   tmp,
   m,
   mq,
   mb,
   sigma_n2:Extended;               { Noise power                          }
   beta,                            { Rayleigh distributed path gains      }
   tau      :RArr;                  { Uniformly [0,Tb] distributed path delays }
   h,                               { Number of path with largest gain     }
   i,                               { Loop variable                        }
   j        :Integer;               { Loop variable                        }
Begin
     { Assign random delays and path gains to transmitting terminals }
     For i := 1 To Nt Do
        If terminal[i] = INTERFERE
        Then
            For j := 1 To L Do
            Begin
                 tau[i,j]  := Tb*duni;
                 beta[i,j] := Rayleigh(Sqrt(S[i,u]));
            End
        Else
            For j := 1 To L Do
            Begin
                 tau[i,j]  := 0;
                 beta[i,j] := 0;
            End;

     { Determine what the largest gain is }
     bmax := 0;
     h    := 1;
     For i := 1 To L Do
        If beta[u,i]>bmax
        Then
        Begin
             bmax := beta[u,i];
             h    := i;
        End;

     { Determine noise power relative to largest incoming signal }
     sigma_n2 := power(10, -snr/10)*Sqr(bmax)*Pt*Tb*Tb;

     { Calculate bit error probability parameters }
     mu2  := CalcMu2(u, h, tau, sigma_n2);
     mu1  := CalcMu1(u, h, tau, mu2);
     mu12 := CalcMu12(u, h ,tau);
     m    := Sqrt(2*Pt)*bmax*Tb;
     a    := m*Abs(1/Sqrt(mu1)-1/Sqrt(mu2))/Sqrt(2);
     b    := m*(1/Sqrt(mu1)+1/Sqrt(mu2))/Sqrt(2);
     mq   := MarcumQ(a,b);
     mb   := NegExpI0(a,b);

     { Calculate the bit error probability }
     tmp := ( (1 + mu12/Sqrt(mu1*mu2))*mb )/2;
     CalculatePbe := mq-tmp
End;

{*************************************************************************}
{*  CalcPpsXZero                                                         *}
{*************************************************************************}
FUNCTION CalcPpsXZero : Extended;
VAR
   i           : Integer;
   PpsTot      : Extended;
BEGIN
     InitTerm(Terminal);
     Terminal[1] := INTERFERE;
     PpsTot      := 0;
     If L=1
     Then
         CalcPpsXZero := Power(1-CalculatePbe(1), PacketLength)
     Else
     Begin
         For i := 1 To MonteCarlo Do
         BEGIN
            PpsTot := PpsTot + Power(1-CalculatePbe(1), PacketLength);
         END;
         CalcPpsXZero := PpsTot/MonteCarlo;
     End
End;{CalcPpsXZero}

{*************************************************************************}
{*  CalcPpsAveX                                                          *}
{*************************************************************************}
FUNCTION CalcPpsAveX (X:Integer):Extended;
VAR
   i,ii,j      : Integer;
   Pbe,
   PpsTot      : Extended;
   TempConfig  : ConfigArr;
BEGIN
     InitTerm(Terminal);
     Terminal[1] := INTERFERE;
     TempConfig := StartConfig;
     PpsTot := 0;
     CountConfig := 0;
     FOR ii := 1 TO X DO
         Terminal[TempConfig[ii]] := INTERFERE;
     FOR ii := 1 TO MonteCarlo DO
     Begin
          Pbe := CalculatePbe(1);
          PpsTot := PpsTot + Power(1-Pbe, PacketLength);
     End;
     CountConfig := CountConfig + 1;
     {WriteConfig(TempConfig, X);}
     i := X;
     WHILE (i<>1) OR (TempConfig[1]<>MaxConfig[1]) DO
     BEGIN
```

```
         i := X;
         IF TempConfig[i] = Nt
         THEN
         BEGIN
             WHILE TempConfig[i] = MaxConfig[i] DO
                 i := i-1;
             TempConfig[i] := TempConfig[i] + 1;
             FOR j := (i+1) TO X DO
                 TempConfig[j] := TempConfig[j-1] + 1;
         END
         ELSE
             TempConfig[i] := TempConfig[i] + 1;
         InitTerm(Terminal);
         Terminal[1] := INTERFERE;
         FOR ii := 1 TO X DO
             Terminal[TempConfig[ii]] := INTERFERE;
         ShowState;
         FOR ii := 1 TO MonteCarlo DO
         Begin
              Pbe := CalculatePbe(1);
              PpsTot := PpsTot + Power(1-Pbe, PacketLength);
         End;
         CountConfig := CountConfig + 1;
         {WriteConfig(TempConfig, X);}
    END;
    CalcPpsAveX := PpsTot/(MonteCarlo*CountConfig);
    {WriteLn(X, ' ', CountConfig, ' ', MonteCarlo*CountConfig);}
END;{CalcPpsAveX}

{****************************************************************************}
PROCEDURE OpenResultFile(VAR OutFile:Text);
VAR
   GS,
   NL,
   FileName : String;
BEGIN
    Str(GroupSize, GS);
    Str(NumberOfLines, NL);
    FileName := '2SDK7L2.dat';
    Assign(OutFile, FileName);

    ReWrite(OutFile);
{    WriteLn(OutFile,'H.L.A. Le');
    WriteLn(OutFile,'RESULTS FROM CALCULATIONS (v.2):');
    WriteLn(OutFile);
    WriteLn(OutFile,'Receiver Radius: ', Rreceive);
    WriteLn(OutFile,'Terminal Radius: ', Rterm);
    WriteLn(OutFile);
    WriteLn(OutFile,'Number of users : ', Nt);
    WriteLn(OutFile,'Size of group   : ', GroupSize);
    WriteLn(OutFile,'Number of groups: ', NumberOfLines);
    WriteLn(OutFile);
    WriteLn(OutFile,'Code length     : ', codelength);
    WriteLn(OutFile,'Packet length   : ', Packetlength);
    WriteLn(OutFile,'Inhibit delay fraction: ', Fraction);
    WriteLn(OutFile,'SNR             : ', SNR);
    WriteLn(OutFile);
    WriteLn(OutFile);}
END;{OpenResultFile}

{****************************************************************************}
{  WarnBeep emits a half second warning tone and then waits 4.5 seconds     }
{                                                                            }
{  Depends on: none                                                          }
{  Calls      : none                                                         }
{  Modifies   : pfn                                                          }
{****************************************************************************}
Procedure WarnBeep;
VAR
   i : Integer;
BEGIN
     FOR i := 1 TO 5 DO
     BEGIN
          Sound(500+i*20);
          Delay(300);
     END;
     NoSound;
     Delay(1000);
End;{WarnBeep}

{****************************************************************************}
{                            M A I N   L I N E                               }
{****************************************************************************}
Begin
     ClrScr;
     WriteLn('Output Testprogramma:');WriteLn;

     {Initialize Variables}
     Rreceive      := 30;
     Rterm         :=  5;
     SNR           :=  6;

     Nt            := 32;
     NumberOfLines :=  2;
     GroupSize     := 16;

     { Initialize Functions and Procedures }
     InitCodes(CCat, GroupSize, NumberOfLines);
     InitPathGain(S, GroupSize, NumberOfLines, Rreceive, Rterm, LossExp);
     InitFact(Faculteit);
     WriteLn('NumberOfLines: ',NumberOfLines);
     WriteLn;

     { Calculate PpsAveX }
     OpenResultFile(ResultFile);
     FOR X := 1 TO XMax Do
     BEGIN
          InitStartConfig(StartConfig, X);
          InitMaxConfig(MaxConfig, X);
          PpsAveX := CalcPpsAveX(X);
```

```
            PpsAveXArr[X] := PpsAveX;
            WriteLn(ResultFile, PpsAveX);
            Flush(ResultFile);
        END;
        WriteLn(ResultFile);
        WriteLn(ResultFile, ' G:                  ':24,
                            'S(4):                ':23,
                            'D(4):                ':23);

        { Throughput and Delay Calculation }
        PpsXZero := CalcPpsXZero;
        For i := 0 To NumSim Do
        Begin
            G          := Power(10, -(LRangeMin+i*(LRangeMax-LRangeMin)/NumSim));
            ExpGd      := Exp(-G*fraction);
            TCycle     := (1+(2*fraction))+(ExpGd/G);
            PNoConfl   := ExpGd*PpsXZero;
            Pidle      := 1/(G*(1+(2*fraction))+ExpGd);
            PConflTemp := 0;
            Write(ResultFile, G:23);
            For j := 1 To XMax Do
            Begin
                Poisson    := Power(G*fraction,j) * ExpGd/Faculteit[j];
                PConflTemp := PConflTemp + Poisson*PpsAveXArr[j];
            End;
            PConfl     := PConflTemp;
            Psuccess   := PNoConfl+PConfl;
            Throughput := Psuccess/TCycle;
            D          := (((G*Pidle)/Throughput)-1)*(1+fraction+(1/Alpha))
                          +(G*(1-Pidle)/Throughput)*(1/Alpha)
                          +(1+fraction);
            Write(ResultFile, Throughput:23);
            WriteLn(ResultFile, D:23);
        End;
        Close(ResultFile);


        {WarnBeep;}
        WriteLn;
        WriteLn;
        Write('I AM READY MISTER LE !!!!!!');
        GoToXY(35,17);
        Write('Press ENTER to return to program.');
        {ReadLn;}
END.
```