NOTE TO USERS

This reproduction is the best copy available.

UMI®
Hong Lin
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)
GRADE / DEGREE

Department of Systems Science
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Performance of Back-off Algorithms
TITRE DE LA THÈSE / TITLE OF THESIS

Dr. David McDonald
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Dr. Vladimir Pestov

Dr. Tet Yeap

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies
PERFORMANCE OF BACK-OFF ALGORITHMS

Hong Lin

Thesis Submitted to the Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
for the degree of
Master of Science in Systems Science

Department of Systems Science
Faculty of Graduate and Postdoctoral Studies
University of Ottawa

© Hong Lin, Ottawa, Canada, 2008
Abstract

A well known defect of the standard exponential back-off algorithm used in IEEE 802.11 is its short-term unfairness between users. A user will have high throughput shortly after a successful transmitting but the user will have a low throughput temporarily after backing off again and again because of multiple collisions. This variation of short-term throughput induces unacceptable delay variation for the users that have real-time service demands such as Audio or Video. Recently algorithms like Idle Sense [1] has been proposed to solve this problem by reducing the short-term unfairness between users. It lets each user adapt its probability $p$ of accessing the channel to a predetermined optimal $p^*$ based on its observation of traffic on the channel. In this thesis, we discuss some improvements we made to Idle Sense by including a deterministic back-off of $[(1 - p)/p]$ time slots after a successful transmission. Our algorithm may be viewed as a combination of Idle Sense and Zero Collision [2]. Simulation results show the throughput our algorithm obtains is almost the same as that of Idle Sense but short-term unfairness is reduced. Packets of each user can be transmitted at much more regular intervals and in simulations for Voice over IP (VOIP), the number of VOIP users with a good Quality of Service (QoS) through a single access point can be substantially increased.
I would like to thank David McDonald, my supervisor, for his many suggestions and constant support during this research. He shared with me his abundant knowledge and provided many useful references and friendly encouragement. I had the pleasure of meeting A. Proutière and C. Bordenave. They are wonderful people and their support makes research like this possible. Of course, I am grateful to my wife, Ning for her patience and love. Without her this work would never have come into existence.
# Contents

Abstract iii

Acknowledgements v

List of Tables xi

List of Figures xiii

1 Introduction 1
   1.1 Brief history of contention-based random multi-access protocols 1
   1.2 The problems and research interests in different times 2
   1.3 Contribution and Structure of this thesis 3

2 Background and wireless packet network protocols 5
   2.1 Related contention-based random multi-access protocols 5
   2.2 IEEE wireless packet network protocols 6
   2.3 The binary exponential back-off algorithm in IEEE 802.11 10

3 Recent work 13
   3.1 Idle Sense 13
   3.2 Zero Collision 15
   3.3 Kalman Filter 16

4 Our Algorithm 19
   4.1 Throughput and Fairness 19
CONTENTS

4.2 Mathematical inference .................................................. 20
4.3 The idea of our algorithm ............................................... 22
4.4 Our algorithm ............................................................ 23

5 Simulation Study .................................................................. 25
5.1 Simulation Environment .................................................. 25
5.2 Simulation of a static experiment ...................................... 26
5.3 Simulation of a dynamic experiment ................................. 29
5.4 Simulation of VOIP application ........................................ 30
5.5 The study of the performance of our algorithm with varying parameters ......................................................... 32
5.6 Compatibility with the binary exponential back-off algorithm .................................................. 34
5.7 CSIM simulation Package and its application in our model .................................................. 36

6 Results of some experimental algorithms ................................. 39
6.1 Some results in the early time ............................................ 39
  6.1.1 MIMD ................................................................. 39
  6.1.2 Kalman Filter ......................................................... 40
6.2 Some recent results ........................................................ 42
  6.2.1 Some attempts that tried to reduce the fluctuation of user's access probability ........................................ 43
  6.2.2 Some attempts that tried to reduce collision .................. 44
  6.2.3 Some attempts that tried to increase the speed of convergence of users' access probability .................................. 45

7 Conclusion and future work .................................................. 47
7.1 Conclusion ......................................................................... 47
7.2 Future work ....................................................................... 48
CONTENTS

8 Appendix A: SOME DETAILED RESULTS OF EXPERIMENTS 49

9 Appendix B: SOURCE CODE OF SOME SIMULATION PROGRAMS 53
  9.1 Source code for idle sense and our algorithm for static experiment and dynamic experiment 53
  9.2 Source code for 802.11 with binary exponential back-off for static experiment and dynamic experiment 62
  9.3 Source code for compatibility of 802.11 with binary exponential back-off and our algorithm for static experiment 70

Bibliography 77
List of Tables

2.1 Comparison of wireless packet network protocols .................. 7
2.2 Comparison of variants of 802.11 ................................. 9

5.1 Results of dynamic experiment ................................. 30
5.2 Results of VOIP application experiment ......................... 31

8.1 The simulation result1 of Our Algorithm ......................... 49
8.2 The simulation result1 of Idle Sense .......................... 50
8.3 The simulation result1 of the basic 802.11b ..................... 50
8.4 The simulation result2 of Our Algorithm ......................... 50
8.5 The simulation result2 of 802.11 RTS/CTS Algorithm ............ 51
8.6 The simulation result2 of 802.11 basic Algorithm ............. 51
List of Figures

2.1 Classifications and Ranges of the Various IEEE Wireless Networking Standards .......................... 7
2.2 802.11 and 802.16 .................................................. 8
2.3 the brief description of operation of CSMA in 802.11 ................................................. 9
3.1 Principle of AIMD ...................................................... 14
3.2 Three transmission rounds and user a, b and n have allocated slots. .............................. 15
3.3 d is a successful new user; e and f are unsuccessful ...................................................... 16
4.1 Throughput as a function of average access probability of users .............................................. 21
4.2 The ideal picture of our algorithm .................................................. 22
5.1 The ideal picture of our algorithm .................................................. 26
5.2 The Scenario of the static experiment .................................................. 26
5.3 Throughput as a function of the number of users .................................................. 27
5.4 Relative SD: The ratio between SD and Mean between two consecutive successful transmissions for each individual user .................................................. 28
5.5 The Scenario of the dynamic experiment .................................................. 29
5.6 The Scenario of the VOIP application experiment .................................................. 31
5.7 The grouping of VOIP packets in a buffer into a 802.11b packet ........................................ 32

xiii

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Chapter 1

Introduction

1.1 Brief history of contention-based random multi-access protocols

The main purpose of random multi-access protocols is to share a common resource among competing users with a Distributed Coordination Function (DCF). The research on random multi-access protocols can be dated back to 1970. Norman Abramson and his colleagues at the University of Hawaii devised the ALOHA algorithm [3] to solve the problem of channel allocation. In ALOHA, the shared resource is the wireless communication channel provided by a satellite. Competing users are the stations scattered on the several islands. The idea is that, each user can try to transmit whenever it has packets. If the transmission is successful, it can continue. If a collision occurs, each user involved in the collision has to wait a random back-off time before they can retry the channel. From this original idea, lots of improvements have been made, and a large protocol family has been developed upon it. Among these improvements, slotted ALOA, Carrier Sense Multiple Access protocol (CSMA), Carrier Sense Multiple Access with Collision Detection (CSMA/CD) and Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)[4, 9] have been implemented. The technology of mobile computing and wireless portable equipment are developing
quickly. Consequently the research on random multi-access protocols has intensified to develop decentralized back-off random multi-access protocols for medium access control for Wireless Local Area Networks (WLAN) and Ad-Hoc networks.

1.2 The problems and research interests in different times

The research of contention-based random multi-access protocols mainly focuses on the study of the DCF. At first, the emphasis was on improving throughput [6]. After the RTS/CTS scheme was released, the maximal saturated throughput of system almost reaches its optimal value. Then research interest changed to another important area: fairness. As we all know, 802.11 can easily achieve long-term fairness, but not short-term fairness because of the bursty characteristics [7] of the DCF of 802.11 itself. This short-term unfairness can cause large service delay even when traffic is not very heavy. For the recent popular real-time services, this service delay becomes unacceptable.

Researchers have tried to conceive of new DCFs to improve short-term fairness. One important research direction is the study of adaptive back-off algorithms. These algorithms try to adjust the collision probability or the number of inactive time slots between transmissions to a predefined value by adapting the access probability of users. The adaptation is based on real-time traffic measurement. Idle Sense [1, 8] is the best known algorithm in this research direction.

Besides adaptive back-off algorithms such as Idle Sense, there are other kinds of algorithms aiming at improving short-term fairness and QoS (Quality of Service). An interesting proposal is Zero Collision [2]. This algorithm sets the number of slots of between successive transmissions to a predetermined global constant we could call a transmission round. Each user keeps track of the status of slots in every transmission round. Once a user succeeds in finding a slot to transmit in a transmission round, it will wait this predetermined number of slots until it transmits again in the next
transmission round. This slot changes its status to occupied and other users will not try to use it in next round. New users and users who suffered a collision compete only for the slots which are inactive in the last round or in the last several rounds.

In this research, while some improvements have been achieved, lots of problems arise. Some of these algorithms need to change the standard frame format of 802.11, some of them are too complicated to implement. Some like Zero Collision are not robust. There is still room for improvement.

Recently, as more and more real-time services have been provided, the QoS of the DCF measured by the standard deviation of packet delay becomes a critical index of performance, especially as a measurement for short-term fairness. The standard deviation of packet delay for the adaptive algorithms [1, 8] are smaller than for 802.11, but they are still large.

1.3 Contribution and Structure of this thesis

This thesis presents an adaptive algorithm which makes a considerable improvement in short-term fairness over the algorithm (Idle Sense) in [1]. To evaluate the performance of our algorithm, Idle Sense and 802.11b with binary exponential back-off, we use three kinds of measurement in our simulation: the overall saturated throughput; the overall mean of the period between two consecutive successful transmissions for each user and the standard deviation of the above period. The QoS of the DCF can also be estimated from the last two measurements. We also estimate the performance of three algorithms for Voice over IP (VOIP) applications by simulation. We measure the mean and standard deviation of voice signal delay of users in the system. From the simulation results, we found that we make a considerable improvement to Idle Sense in term of QoS for VOIP. This means substantially more users may sustain VOIP connections via a single access router.

The structure of this thesis is as follows: Chapter 2 introduces some terminology
and gives the description of some related wireless packet network protocols. Chapter 3 gives the description of some recent works in the back-off algorithm. Chapter 3 gives the description of our algorithm. In Chapter 5 we compare the performance between our algorithm, Idle Sense and 802.11 with binary exponential back-off using simulation. Chapter 5 is a brief description of some former results we got in the course when we developed our algorithm. Chapter 6 is some results of experimental algorithms. Chapter 7 is a summary of thesis and some work we want to do in the future. Appendix A lists some detailed results of simulations in Tables and Appendix B lists some important source code of the simulation experiments.
Chapter 2

Background and wireless packet network protocols

In this chapter, we will give a description of some related wireless packet network protocols.

2.1 Related contention-based random multi-access protocols

The principle of ALOHA [4] is quite simple. Each user that has data packets tries to transmit its data. An attempt may result in a successful transmission or a collision. If it is successful, the user will continue to transmit. Otherwise, if it is a collision, the user will wait a random back-off time before trying to retransmit this packet. ALOHA gave a good solution in the special scenario of transmissions via satellite from dispersed users without wire-line connections. However, when ALOHA was used in other scenarios, it was found that its efficiency is rather low, the maximum of its saturated throughput is only about 17 percent. Here the maximum of the saturated throughput is the theoretical throughput when each user accesses the channel with the optimal access probability, where the packet size is 1 and the overhead is 0.
To improve the saturated throughput, slotted ALOHA and CSMA [4, 9] were developed. Slotted ALOHA separates time into discrete time slots and the users can only transmit data at the beginning of a time slot. The maximum saturated throughput for slotted ALOHA can reach 36 percent. CSMA takes advantage of carrier sensing to avoid collisions between new transmission attempts and ongoing transmissions. Depending on the access probability with which the user accesses the channel when the channel is inactive, CSMA is also called p-persistence CSMA. The maximum saturated throughput can reach more than 85 percent when p is very small. CSMA/CD is mostly used in Ethernet local area networks. It can detect collisions and stop a transmission involved in a collision immediately after its occurrence. CSMA/CA is widely used in wireless situation in which collision detection is impossible and the RTS/CTS (Request To Send/Clear To Send) mechanism is used to avoid data packet collision. In 1999, IEEE released its 802.11 standard [5] for the WLAN.

2.2 IEEE wireless packet network protocols

In this section, we introduce some related IEEE wireless packet network protocols. In general, wireless packet network protocols are classified by the size of their coverage range. Figure 2.1 shows the classification of wireless packet network protocols with their coverage ranges.

802.15, 802.11, 802.16 and 802.20 are the four best known wireless packet network protocols. 802.15 originated from Bluetooth technology, now also called WiMedia. It is used in a Wireless Personal Area Network (WPAN) such as a home, a small office or an operation room, etc. Its range of coverage is the smallest, less than 10 meters. 802.11 is the most widely used. One of its popular variants is 802.11b or Wi-Fi. It is used in a Wireless Local Area Network. Its range of coverage is larger than 802.15, about several ten meters. 802.16 is also called WiMax, which is used in Wireless Metropolitan Area Network (WMAN) as the backhaul to link Local Area
Network to core network. It covers a range about 50 kilometers. 802.20 called mobile WiMAX is also used in WMAN. It covers a range similar to 802.16. But it focuses on supporting mobile users moving with a velocity up to 250 kilometers per hour. The new 802.22, on the other hand, is used in Wireless Regional Area Network (WRAN) and aims to provide broadband access across entire "regions" for a coverage range up to 100 kilometers. Table 2.1 summarizes other features among these four protocols.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Spectrum</th>
<th>Max. Data rate</th>
<th>Channel Plan</th>
<th>Modulation</th>
<th>User type</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.15</td>
<td>2.4G/60G</td>
<td>11 M/2 G bps</td>
<td>TDD</td>
<td>OFDM</td>
<td>Portable</td>
</tr>
<tr>
<td>802.11</td>
<td>2.4G/5.5G</td>
<td>11 M/54 M bps</td>
<td>TDD</td>
<td>HRDSSS/OFDM</td>
<td>Portable</td>
</tr>
<tr>
<td>802.16</td>
<td>2G-11G</td>
<td>2 M-155 M bps</td>
<td>TDD</td>
<td>OFDM</td>
<td>Portable/Nomadic</td>
</tr>
<tr>
<td>802.20</td>
<td>&lt;3.5G</td>
<td>&gt;1 M bps</td>
<td>TDD/FDD</td>
<td>OFDM</td>
<td>Mobile</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of wireless packet network protocols

The focus of this thesis is 802.11. It is often used together with 802.16 as in Figure 2.2. 802.11 has several variants, 802.11a, 802.11b and 802.11g. Table 2.2 summarizes the features of physical layer of these variants.
For the MAC layer, all these variants of 802.11 has two options for their Medium Access Control. One option is central control with a Point Coordination Function (PCF). Another option is distributed control with a Distributed Coordination Function (DCF). This thesis focuses on the study of DCFs. All DCFs of 802.11 variants use Carrier Sensing Multiple Access (CSMA). Figure 2.3 is the brief description of operation of CSMA in 802.11.

First, the user monitors the state of the channel by carrier sensing; If the channel stays idle for a DIFS, the user reduces its back-off counter by one every time slots when channel keeps idle. If the channel is busy, the user freezes its back-off counter until the channel can keep idle for a DIFS. Second, when back-off counter reaches
2.2. IEEE WIRELESS PACKET NETWORK PROTOCOLS

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Spectrum</th>
<th>Max. Data rate</th>
<th>Modulation</th>
<th>Spread spectrum type</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11a</td>
<td>5.5G</td>
<td>54 Mbps</td>
<td>OFDM</td>
<td>FSS</td>
</tr>
<tr>
<td>802.11b</td>
<td>2.4G</td>
<td>11 Mbps</td>
<td>HR/DSSS</td>
<td>DSS</td>
</tr>
<tr>
<td>802.11g</td>
<td>2.4G</td>
<td>54 Mbps</td>
<td>OFDM</td>
<td>FSS</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of variants of 802.11

Figure 2.3: the brief description of operation of CSMA in 802.11.

0, the user transmits it data and reset its back-off counter according to the back-off algorithm. The most important factor to decide the MAC performance of these protocols is the back-off algorithm they use. This is the research interest of this thesis. The next section is an introduction of the back-off algorithm in IEEE 802.11.
2.3 The binary exponential back-off algorithm in IEEE 802.11

802.11 is a protocol based on CSMA/CA with a binary exponential back-off algorithm. Its core mechanism for medium access control is its DCF. The idea [6] is that all the users in the system share a common wireless channel. When a user has a data packet to transmit, it monitors the state of the channel by carrier sensing. If the channel stays inactive for a period of time equal to a distributed inter-frame space (DIFS), each user starts measuring fixed length (say 20 us) time slots. A user waits a random back-off time measured in slots before transmitting in order to minimize the probability of a collision with other users. Also, to avoid channel capture, a user must wait a random back-off time after successful transmissions, even if the channel is inactive. If the channel becomes active again, the user keeps monitoring the channel until it again stays inactive for a DIFS. For reasons of efficiency, the DCF of 802.11 uses discrete time slots. The user can transmit only at the beginning of each time slot. The size of time slot depends the propagation delay, the time to switch between receiving and transmitting, and the time that the MAC layer needs to be informed of the state of the channel from the Physical layer. 802.11 uses a binary exponential back-off DCF. At the beginning of transmission, each user sets the initial value for its back-off counter to a random value chosen uniformly between 0 and $CW_{min} - 1$. Here the back-off counter is the random time measured in time slots that a user waits before it begins transmitting. $CW_{min}$ is the minimal contention window. It is the interval in which each user uniformly chooses its back-off counter for the first transmission or any transmit attempt after a successful transmission. If the transmission is unsuccessful, the user doubles its contention window up to a maximum value $CW_{max} = 2^mCW_{min}$. Here $m$ is the maximal back-off stage. After the channel stays inactive for a DIFS, the back-off counter is reduced by one every time slot whenever the channel stays inactive but is kept "frozen" when the channel
2.3. THE BINARY EXPONENTIAL BACK-OFF ALGORITHM IN IEEE 802.11

is active (sensed active during a transmission or a collision), and the back-off counter starts decreasing again after the channel keeps inactive for a DIFS again. When the back-off counter reaches zero, the user transmits.

802.11 has bursty transmission characteristics because of its binary exponential back-off DCF. Under binary exponential back-off algorithm, if a user suffers several collisions, this user may have a very long back-off time before its next transmission. This is not a big problem for data transmission because it does not have a large impact on its overall throughput. But this bursty behavior causes short-term unfairness between users and results in increased variability in the time between packet transmissions and in unacceptable service delay, especially for services with a high real-time requirement.
Chapter 3

Recent work

In this chapter, we will give a description of some recent research on DCF for 802.11.

3.1 Idle Sense

Idle Sense, an adaptive back-off algorithm for the Ethernet or wireless LAN was presented in [1]. In this algorithm, as in 802.11, each user maintains a back-off counter and a contention window. If the channel is active, the back-off counters of users remain unchanged until the channel stays inactive for a DIFS. The back-off counters is reduced by one whenever the channel stays inactive for one more time slot. When a user's back-off counter reaches zero and the channel is inactive, this user begins its transmission.

The difference between this adaptive back-off algorithm and the back-off algorithm in 802.11 is its way of calculating the contention window. By observing the transmission event, each user $i$ adjusts its access probability $p_i$ towards an optimal value $p^*$ without ever knowing number of users $N$ in system. To do this user $i$ estimates the probability of an inactive slot with $\hat{q}_i$. In practice user $i$ examines the last $B$ slots. Slots which are inactive are counted 1. Slots which are active are counted 0. The estimate $\hat{q}_i$ is the number of ones divided by $B$. The target probability that an any time slot is an inactive slot is $\exp(-s^*)$. $s^*$ is a predefined global constant given
in [1]. The user can adjust its $p_i$ consistently to make the estimate $\hat{q}_i$ approach the target $\exp(-s^*)$. In particular Idle Sense essentially performs the following adjustment. User $i$ adjusts its transmission probability from the old value $p_i^{old}$ to the new value $p_i^{new}$:

Additive increase  if $\hat{q}_i \geq \exp(-s^*)$ then set $p_i^{new} = p_i^{old} + \epsilon$,

Multiplicative decrease  if $\hat{q}_i < \exp(-s^*)$ then set $p_i^{new} = p_i^{old}/\alpha$.

The effect of this algorithm is to stabilize each $p_i$ at $p^* = s^*/N$ where $\hat{q}_i$ will be close to $\exp(-s^*)$. The $p_i$ tend to the same value because its AIMD (Additive Increase and Multiplicative Decrease) ($\alpha > 1$) mechanism reduces the difference between the $p_i$'s. The associated contention window $CW_i$ for user $i$ is given by $p_i = 2/(CW_i + 1)$ and the back-off is chosen randomly from this window.

AIMD is a feedback control algorithm. We can intuitively understand how it works by looking at Figure 3.1.

![Figure 3.1: Principle of AIMD](image_url)
3.2 Zero Collision

Figure 3.1 shows two users who are synchronized and additively increase and multiplicatively decrease together. Assume user 1 and user 2 have access probabilities $p_1$ and $p_2$ respectively. The difference between the two access probabilities is $d_1 = p_1 - p_2$. After an additive increase, the difference between the two access probabilities is the same; i.e. $d_2 = d_1 = p_1 - p_2$. However, after a multiplicative decrease, the difference between the two access probabilities reduces to $d_3 = d_2 / \alpha = (p_1 - p_2) / \alpha$. In this 'saw tooth' manner, the access probabilities of the two different users approach the same value, but this common value continues the 'saw tooth' behavior. However, in Idle Sense and our algorithm, different users adapt their access probabilities asynchronously based on their asynchronous observation of traffic on the channel. So the access probabilities of different users can not in fact converge to the same value. Instead there is a constant fluctuation around a central value.

3.2 Zero Collision

Zero Collision was presented in [2]. There are two main ideas. One is that wasting some inactive slots only has small impact on the performance because the length of an inactive slot is much shorter than an active one. Another is that each specific network has a limitation on its maximum number of users because of its hardware and software restraints.

Figure 3.2 and 3.3 gives an illustration of operation of Zero Collision.

![Figure 3.2: Three transmission rounds and user a, b and n have allocated slots.](image)
3. RECENT WORK

Figure 3.3: d is a successful new user; e and f are unsuccessful.

In this algorithm, the number of slots in each transmission round is fixed. Each user keeps monitoring the status of slots in every transmission round. A user may keep transmitting in the fixed slots in the transmission round if it succeeded in last round. New users and users suffering a collision only compete for the slots which are inactive in last round or last several rounds.

In Figure 3.2, a,b...n are users who successfully transmit their data into fixed allocated slots. They will continue to transmit in the same slots in every transmission round until they have no data to transmit. In Figure 3.3, user d, user e and user f are new users. User d transmits successfully and will keep transmitting in the same slot. User e and user f compete for a slot but they collide. So they will wait for a random back-off time until they can again compete for a slot. [2] shows that this algorithm has very good performance when the capacity limit of network is not very large (maximum number of user up to 128). However, Zero Collision suffers from a lack of robustness. If a user fails to detect a transmission from another hidden user, then his internal description of a transmission round is false. This will lead to an incorrect calculation of his allocated time slot and that will result in a collision.

3.3 Kalman Filter

[12] presented an estimation method for the number of users in the system using the extended Kalman Filter. Simulation shows the extended Kalman Filter is more
accurate and effective than ARMA filter. We tried to use the extended Kalman Filter in our algorithm. Every user estimates the number of users $N$ based on the observation of the traffic using the extended Kalman Filter as in [12] and each user adapts its access probability as $p = s^*/N$. Here $s^*$ is the same constant as Idle Sense. But we found the simulation result is poor. The accuracy and the convergence speed of the estimate using the extended Kalman Filter are not very good, especially in the dynamic case when users arrive and depart.

The performance of 802.11 and Idle Sense will be compared with the performance of our algorithm in the next chapter.
Chapter 4

Our Algorithm

In this chapter, we will introduce our adaptive algorithm, which is an improvement for Idle Sense.

4.1 Throughput and Fairness

The two main factors in performance evaluation are throughput and fairness. The main goal is to improve these factors to ensure a good quality of service (QoS) for each user.

Throughput, Long-term fairness and short-term fairness

Throughput is the channel utilization; i.e. the long term proportion of time spent transmitting (payload) data. In this thesis, we use throughput as the primary performance measure. Long-term fairness is the equality of transmitting opportunities that each user gets over a long time period. Short-term fairness is the equality of transmitting opportunities that each user gets in a relative short time period.

In this thesis, we use the standard deviation of the period between two consecutive successful transmissions for each user divided by the mean time between these two successful transmissions to measure the relative variation of packet delay of the streams from each user as the measure of fairness. In the case of VOIP, we use the
mean and standard deviation of the voice signal delay for each user as the measure of service quality.

4.2 Mathematical inference

Suppose there are \( N \) users in the system with a common access channel which the users share in a decentralized manner [10]. Time is slotted and users can only try to transmit at the beginning of time slot. All users always have packets to transmit. Each user has a back-off counter \( k \) and keeps an access probability \( p \) as a state.

A hypothesis that the back-off’s of various users decouple when \( N \) gets large was developed in [6] and was justified theoretically in [11]. According to this hypothesis, the back-off’s of the various users are mutually independent as the number of users in the system becomes large. As a result, if the channel is inactive and the probability a user accesses the channel is \( p \), the chance that none of the \( N \) users access the channel is \((1 - p)^N\); that one user accesses the channel and successfully captures it is \( Np(1 - p)^{N-1} \) and that a collision occurs is \((1 - Np(1 - p)^{N-1} - (1 - p)^N)\).

The state of the channel can be seen as a three-state semi-Markov chain: transmission with a length of \( L_{tr} \), collision with a length of \( L_c \) or inactive with a length of 1. \( L_{pd} \) is the payload of a transmission and the unit of length is an inactive time slot. Then the throughput is the proportion of time that the channel can be captured successfully by a user is:

\[
\frac{L_{pd}Np(1 - p)^{N-1}}{L_{tr}Np(1 - p)^{N-1} + L_c(1 - Np(1 - p)^{N-1} - (1 - p)^N) + (1 - p)^N}.
\]

Figure 4.1 gives the throughput as a function of access probability \( p \) when \( N \) is 3, 6, 15, 40, 100. Red line is for \( N = 3 \). Blue line is for \( N = 6 \). Green line is for \( N = 15 \). The two black ones are for \( N = 40 \) and \( N = 100 \). We can see that the larger the number of users \( N \) is, the smaller the optimal \( p \) is.

To optimize the throughput, we can maximize the above proportion. It is also
4.2. MATHEMATICAL INFERENCE

Figure 4.1: Throughput as a function of average access probability of users.

equivalent to minimize

\[
\frac{L_{tr}}{L_{pd}} + \frac{L_c}{L_{pd}Np(1-p)^{N-1}} - \frac{L_c}{L_{pd}} - \frac{L_c(1-p)}{L_{pd}Np} + \frac{1-p}{L_{pd}Np}.
\]

By setting the derivative respect to \(p\) equal to zero we find

\[
\frac{L_c - 1}{L_c}(1 - p)^{N} + Np - 1 = 0. \tag{4.2.1}
\]

The optimal point \(p^*\) at which the maximum reaches is independent of \(L_{tr}\) and \(L_{pd}\).

Substituting \(p\) by \(s/N\), (4.2.1) becomes

\[
0 = \frac{L_c - 1}{L_c}(1 - \frac{s}{N})^{N} + s - 1 \approx \frac{L_c - 1}{L_c}e^{-s} + s - 1. \tag{4.2.2}
\]

Solving (4.2.2), we get \(s = s^*\) to give the optimal throughput then the probability a slot is inactive is \((1 - p^*)^N = (1 - s^*/N)^N \approx \exp(-s^*)\).

The analysis above is the same mathematical basis for Idle Sense and our algorithm. Both Idle Sense and our algorithm let each user \(i\) in the system estimate the
probability of an inactive slot by observing the transmission events. Then each user can adjust its own access probability $p_i$ towards the optimal value $p^*$ respectively without ever knowing $N$. So the probability of a time slot being inactive approaches to the above approximate optimal value $\exp(-s^*)$ and gives the optimal throughput.

4.3 The idea of our algorithm

Actually, Idle Sense works rather well. But it does not take full advantage of the fact that, when several users succeed in their data transmissions in consecutive time slots, they all wait the same back-off time before their next transmissions, they will have a larger probability to succeed in their transmissions again. This is the basis of our improvement to Idle Sense.

The key idea of our algorithm is that every user adapts its access probability using Idle Sense. From the mathematical inference in previous section, we know that the overall throughput of the system will reach its optimum. Meanwhile, because AIMD reduces the difference between the access probabilities of different users, the access probabilities $p_i$ will approximately approach a common value. After a successful transmission, the user waits a deterministic back-off of $[(1 - p_i)/p_i]$ time slots before its next transmission.

![Figure 4.2: The ideal picture of our algorithm.](image)

Figure 4.2: The ideal picture of our algorithm. Assume user 1 and user 2 succeed in their transmissions. In our algorithm, user 1 will waits a deterministic back-
off of $[(1 - p_1)/p_1]$ time slots before its next transmission. User 2 will waits $[(1 - p_2)/p_2]$ time slots. In the ideal case, all users succeed in transmission and their transmission probabilities approximately approach to a common value ($P_1 = P_2 = \cdots$). In the next transmission round, all users will keep transmissions in the same sequence since they all reset their back-off counter as a same value. For the same reason, all users will keep this fixed optimal transmission sequence again and again in the future.

4.4 Our algorithm

In this section, we describe our algorithm in detail. We propose our algorithm, with two modifications to Idle Sense as follows. We estimate the probability a time slot is inactive in the same way as Idle Sense. User $i$ examines the last $B$ slots. Slots which are inactive are counted 1. Slots which are active are counted 0. $q_i$ is the number of ones divided by $B$.

User $i$ adjusts its access probability from the old value $p_i^{\text{old}}$ to the new value $p_i^{\text{new}}$:

Additive increase \quad if $q_i > \exp(-s^*)$ then set $p_i^{\text{new}} = p_i^{\text{old}} + \epsilon$,

Multiplicative decrease \quad if $q_i < \exp(-s^*)$ then set $p_i^{\text{new}} = p_i^{\text{old}}/\alpha$.

We take $\epsilon = 0.001$, $\alpha = 1.2$ and $B = 50$ as in Idle Sense and these are appropriate for $N \leq 100$.

Our first modification is that, instead of using above $p_i^{\text{new}}$ directly, we use the filtered estimate of the above $p_i^{\text{new}}$ as User $i$'s access probability. That is we set:

$p_i^{\text{new}} = \beta p_i^{\text{old}} + (1 - \beta)p_i^{\text{new}}$. Here $0 < \beta < 1$.

The next issue is how to pick the value of the back-off counter. Our second modification also is the most important one. After a successful transmission, instead of choosing randomly from $[0, \text{int}((1.0 - p_i^{\text{new}})/p_i^{\text{new}}) \times 2]$ as in Idle Sense, User $i$ will set its back-off counter to the closest multiple of 10 of the deterministic value $[(1 - p_i^{\text{new}})/p_i^{\text{new}}]$. After a collision the back-off counter is set to a random variable chosen from $[0, \text{int}((1.0 - p_i^{\text{new}})/p_i^{\text{new}})\times2]$ the same as Idle Sense. Most of the attempts
are successful so when the users have about the same access probability \( p = s^*/N \),
the users tend to follow in sequence. This dramatically reduces the jitter.

From the description for Figure 3.1, the access probabilities of different users can
not in fact converge to the same value.
Chapter 5

Simulation Study

In this chapter, we will compare it with 802.11 with binary exponential back-off and Idle Sense by simulation.

5.1 Simulation Environment

We use the object-oriented programming language C++ and CSIM simulation package to implement the simulation program under Visual Studio 2003. Our platform is described in section 5.7.

We use the parameters for 802.11 b in our simulation. Figure 5.1 gives the frame formats of $T_s$, a successful transmission and $T_c$, a collision. For the reason of convenience in simulation, we use the same length of $T_s$ and $T_c$.

We list these parameters in the table below. The length of one inactive time slot is 20 us, the payload is 1000 bytes, and transmission rate is 11 Mbps.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value (unit: time slot/ us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet payload</td>
<td>36.4 ts / 728 us</td>
</tr>
<tr>
<td>$T_s$</td>
<td>61.3 ts / 1226 us</td>
</tr>
<tr>
<td>$T_c$</td>
<td>61.3 ts / 1226 us</td>
</tr>
<tr>
<td>$t_s$</td>
<td>1 ts / 20 us</td>
</tr>
</tbody>
</table>
5. SIMULATION STUDY

5.2 Simulation of a static experiment

Figure 5.2 describes the scenario of the static experiment.

We assume the users arrive according to a Poisson process with a mean 8 seconds. Users stay in the system until simulation finishes. The arrival of users stops when the number of users reaches 5, 10, 20, 40, and 80 in 5 independent experiments respectively.
The throughput and relative Standard Deviation are collected as statistics after a warm-in period equal to half the simulation time. We run the simulation 40 minutes. For our algorithm and Idle Sense, we assume all the users try to access the channel with \( p_0 = \frac{1}{200} \) at the beginning of experiment. For the basic exponential back-off algorithm, all the users try to access the channel with \( CW = CW_{\text{min}} \) and back-off counter is 0 at the beginning of experiment. We set \( CW_{\text{max}} = 1024 \) and \( CW_{\text{min}} = 32 \).

We compare the overall throughput; i.e., the proportion of time spent in successful transmissions and the standard deviation of the period between two successful transmissions for each user divided by the mean time of that period for each of 802.11b, Idle Sense and our algorithm via simulation. This ratio of SD/mean mentioned above is the measure of the variation of the packet delay of each user.

Figure 5.3 gives the overall throughput for one channel for these three algorithms. We see our adaptive algorithm has essentially the same throughput (given by the green line) as Idle Sense (the blue line). The basic exponential back-off does poorly for a larger numbers of users (the red line).

![Figure 5.3: Throughput as a function of the number of users.](image-url)
Figure 5.4 gives the ratio of SD/mean for one channel for these three algorithms. We see our adaptive algorithm has about a 50 percent improvement (given by the green line) over Idle Sense (the blue line). This improvement is attributed to our two modifications: using the filtered estimate of the access probability and setting the back-off counter to the closest multiple of 10 of the deterministic value \([(1 - p)/p]\) after a successful transmission. The basic exponential back-off has a much larger SD/mean (the red line) than either Idle Sense or our algorithm because of its bursty characteristics.

Figure 5.4: Relative SD: The ratio between SD and Mean between two consecutive successful transmissions for each individual user.

The details of the experiment results are listed in Table 8.1, 8.2 and 8.3 in the Appendix.

In Table 8.4, 8.5 and 8.6 of Appendix A, we list the comparison results among the basic 802.11, the 802.11 with RTS/CTS and our algorithm via simulation from another experiment using the parameters in TABLE II in [6].
5.3 Simulation of a dynamic experiment

We also conducted a dynamic experiment with the same parameters as the above static experiment. Figure 5.5 describes the scenario of the dynamic experiment.

![Diagram](image)

Figure 5.5: The Scenario of the dynamic experiment.

We assume the users arrival according to a Poisson process with a mean 8 seconds. Users stay in the system an exponential time with a mean 240 seconds. So the mean number of users in the system is about 30. We run the simulation 80 minutes. In this experiment, for the 802.11 with binary exponential back-off, the average number of user in the system is 29.42 with a standard deviation of 4.75. The overall throughput is 42.35 percent. The mean of the period between two successful transmissions of each user is 50.89 ms with a standard deviation of 143.89 ms. For Idle Sense, the average number of user in the system is 29.52 with a standard deviation of 4.77. The overall throughput is 50.08 percent. The mean of the period between two successful transmissions of each user is 43.08 ms with a standard deviation of 32.72 ms. For our algorithm, the average number of user in the system in our algorithm
is 29.53 with a standard deviation of 4.78. The overall throughput is 52.24 percent. The mean of the period between two successful transmissions of each user is 41.30 ms with a standard deviation of 15.97 ms. Table 5.1 summarizes above result in table. Here the $prd^1$ here is the period between two successful transmissions of each user. 802.11$b^2$ here is the 802.11b with binary exponential back-off.

| Algorithm     | Aver. user Num | SD of user Num | Throughput | Mean of $prd^1$ | SD of $prd^1$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11$b^2$</td>
<td>29.42</td>
<td>4.75</td>
<td>0.4235</td>
<td>50.89 ms</td>
<td>143.89 ms</td>
</tr>
<tr>
<td>Idle Sense</td>
<td>29.52</td>
<td>4.77</td>
<td>0.5008</td>
<td>43.08 ms</td>
<td>32.72 ms</td>
</tr>
<tr>
<td>Our algorithm</td>
<td>29.53</td>
<td>4.78</td>
<td>0.5224</td>
<td>41.30 ms</td>
<td>15.97 ms</td>
</tr>
</tbody>
</table>

Table 5.1: Results of dynamic experiment

5.4 Simulation of VOIP application

We compared an VOIP application between our algorithm, Idle Sense and the 802.11 with binary exponential back-off under the same conditions as the above dynamic experiment. Figure 5.6 describes the scenario of the VOIP application experiment.

We assume each user generates VOIP packets every 10 ms. These packets are stored in a user’s buffer. When a user finds an available slot all waiting packets are grouped into one 802.11b packet and transmitted as one packet.

Figure 5.7 describes how VOIP packets in buffer are grouped in a 802.11b packet to be transmitted together in the VOIP application experiment.

We measure the delay between the generation of a VOIP packet and the successful transmission. We run the simulation 80 minutes. For 802.11b, the mean of delay of VOIP packet is 228.87 ms with a standard deviation of 367.83 ms. For Idle Sense, the mean of delay of VOIP packet is 33.96 ms with a standard deviation of 31.78 ms. For our algorithm, the mean of delay of VOIP packet is 23.73 ms with a
5.4. SIMULATION OF VOIP APPLICATION

Users arrive as a Poisson process with a mean inter-arrival $B$ seconds. While in the system, each user generates a VOIP packet every $10$ ms and store it in its buffer until it was sent. All packets in buffer are grouped in a 802.11b packet and are transmitted in next available slot.

Users stay in the system an exponential time with a mean 240 seconds and standard deviation of 20.20 ms. Here, we see a huge improvement in the delay of our algorithm than ones 802.11b and about 30.12 percent improvement over Idle Sense in terms of the mean of the delay. Though the mean of period between two successful transmissions of each user is similar between these three algorithms, 802.11b has a much much larger waiting time because its period has a higher variance. Table 5.2 summarizes above result in table. Here $802.11b^2$ here is the 802.11b with binary exponential back-off.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Aver. user Num</th>
<th>Mean of VOIP pac. delay</th>
<th>SD of VOIP pac. delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>$802.11b^2$</td>
<td>29.42</td>
<td>228.87 ms</td>
<td>367.83 ms</td>
</tr>
<tr>
<td>Idle Sense</td>
<td>29.52</td>
<td>33.96 ms</td>
<td>31.78 ms</td>
</tr>
<tr>
<td>Our algorithm</td>
<td>29.53</td>
<td>23.73 ms</td>
<td>20.20 ms</td>
</tr>
</tbody>
</table>

Table 5.2: Results of VOIP application experiment

Figure 5.6: The Scenario of the VOIP application experiment.
5.5 The study of the performance of our algorithm with varying parameters

In this section, we discuss how the parameters influence the performance of our algorithm. In our algorithm, there are four explicit parameters. We will discuss them one by one.

1. The increase and decrease parameters in Idle Sense, $\epsilon$ and $\alpha$

$\epsilon$ is the increment which a user add to its access probability when the probability of an arbitrary slot is inactive is too high. Alternatively, $\alpha$ is the factor with which a user decreases its access probability when the traffic is too heavy. Using smaller $\epsilon$ and $\alpha$ can reduce the fluctuation of the access probabilities of users and, to some extent, can reduce the SD of the period between two successful transmissions for the same users. However, the effect is not obvious. For an example, in a static experiment with 40 users, when we reduce the $\epsilon$ from 0.001 to 0.0001 and $\alpha$ from 1.2 to 1.02, the SD of above period reduces from 20.70 ms to 11.02 ms. But reducing these two parameters will decrease the convergence speed of the access probabilities of users and that is important especially in the dynamic case.

2. The average filter parameter $\beta$

We use the average filter to calculate the moving average of access probability. Using a larger $\beta$ leads to a longer average filter. It helps get a more stable average
value; different users have more chance to stabilize at a common access probability. Also, to some extent, it can reduce the SD of the period between two successful transmissions for the same users. Using a smaller $\beta$ will cause more fluctuations for the access probabilities of users but a faster convergence speed. However, as above, the improvement is not obvious. In a static experiment with 80 users, when we change $\beta$ from 0.99 to 0.999, the SD of the period between two successful transmissions for the same users only reduces from 51.70 ms to 35.51 ms with the cost of decreasing the convergence speed.

3. The rounding parameter

We use the rounding parameter to round the back-off counter to the multiple of an integer when a user resets its back-off counter after transmission. From simulation, we found that, if the rounding parameter is larger, it is easier for users to have some common values for resetting their back-off counter. However, these common values may fall further away from the optimal value which we force users to approach. This caused a fluctuation of these common values themselves. On the other hand, if the rounding parameter is smaller, different users tend to reset to different back-off counters because of their different asynchronous observation of traffic on channel.

4. The access probability updating period, $B$

Every $B$ slots, each user updates its access probability based on their observation of the traffic on channel. In stationary case, for the same user, the larger $B$ is, the more accurate the estimate of the probability of a slot inactive is. But in the dynamic case, a larger $B$ will lead to a less frequent estimation and update of access probabilities of users which results in a slow convergence speed. Also, because different users do their estimation asynchronously, using a larger $B$ helps to get a longer length of overlapping observation time which may lead to a close estimation of the traffic on the channel.
5.6 Compatibility with the binary exponential back-off algorithm.

If we change the configuration parameters of our algorithm and 802.11b slightly, the users using these two different algorithms can work compatibly in the same system. The principle that balances the users using different algorithms is to make the mean of the users’ access probability roughly equal. For 802.11b, by [11], when all users go to equilibrium, the mean of a user's access probability $\rho/N$ where $\rho$ satisfies the equation $\rho/(2 - \exp(\rho)) = p_0$. Here $p_0/N$ is the access probability when back-off window is $CW_{\text{min}}$; i.e. $p_0/N = 1/(CW_{\text{min}}/2)$. We conclude that $\rho/(2 - \exp(\rho)) = N/(CW_{\text{min}}/2)^{-1}$. We calculate $\rho$ in the various cases.

<table>
<thead>
<tr>
<th>$N = 10$</th>
<th>$CW_{\text{min}}$</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>0.231</td>
<td>0.134</td>
<td>0.072</td>
<td></td>
</tr>
<tr>
<td>$N = 20$</td>
<td>$CW_{\text{min}}$</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.357</td>
<td>0.231</td>
<td>0.134</td>
<td></td>
</tr>
<tr>
<td>$N = 30$</td>
<td>$CW_{\text{min}}$</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.430</td>
<td>0.303</td>
<td>0.183</td>
<td></td>
</tr>
</tbody>
</table>

We would like to balance the access probability $\rho/N$ with $s^*/N$. Clearly we have to compromise since $\rho$ depends on $N$ while $s^*$ only varies with the payload size. If we pick a payload of 1000 bytes then $s^* \approx 0.17$ so in the case $N = 30$ with $CW_{\text{min}} = 128$, we get $\rho = 0.303 \approx 1.78 \times s^*$. We therefore alter our algorithm by using $1.78 \times s^*$ instead of $s^*$ in our algorithm. This will drive the access probability for our algorithm toward $1.78 \times s^*/N$. For $N < 30$ this gives the users of our algorithm a advantage over binary exponentials back-off, so we set $0.134/10=0.0134$ as the upper bound for access probability for users with our algorithm. In this way, for $10 < N < 30$, The two different algorithms can always reach an approximate balance.

We notice that the throughput does not decrease much by making this alteration which is progressively bigger than $\rho/N$ for $N \leq 30$. The new throughput is
5.6. COMPATIBILITY WITH THE BINARY EXPONENTIAL BACK-OFF ALGORITHM.

compared with the optimal throughput at $s^*$ in the following.

We compare the mean time between successful transmissions MTS in Milliseconds and the standard deviation SDTS for a mixture of a number of users using our algorithm and a number of users using EXP (binary exponential back-off). We also give THRU as the overall throughput, as well as the mean throughput THRDCF for users using our algorithm and THREXP for users using binary exponential back-off. The throughput of an individual user is within the mean throughput plus or minus 0.0007.

For $N = 10$:

<table>
<thead>
<tr>
<th>Mixture</th>
<th>1 DCF: 9 EXP</th>
<th>5 DCF: 5 EXP</th>
<th>9 DCF: 1 EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRU</td>
<td>0.502</td>
<td>0.510</td>
<td>0.531</td>
</tr>
<tr>
<td>SDTS</td>
<td>13.497</td>
<td>10.755</td>
<td>5.388</td>
</tr>
<tr>
<td>THRDCF</td>
<td>0.0517</td>
<td>0.0533</td>
<td>0.0539</td>
</tr>
<tr>
<td>THREXT</td>
<td>0.0497</td>
<td>0.0485</td>
<td>0.0454</td>
</tr>
</tbody>
</table>

For $N = 20$:

<table>
<thead>
<tr>
<th>Mixture</th>
<th>1 DCF: 19 EXP</th>
<th>10 DCF: 10 EXP</th>
<th>19 DCF: 1 EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRU</td>
<td>0.499</td>
<td>0.509</td>
<td>0.557</td>
</tr>
<tr>
<td>MTS</td>
<td>29.188</td>
<td>28.587</td>
<td>26.158</td>
</tr>
<tr>
<td>SDTS</td>
<td>43.533</td>
<td>33.163</td>
<td>13.104</td>
</tr>
<tr>
<td>THRDCF</td>
<td>0.0271</td>
<td>0.0277</td>
<td>0.0282</td>
</tr>
<tr>
<td>THREXT</td>
<td>0.0249</td>
<td>0.0232</td>
<td>0.0194</td>
</tr>
</tbody>
</table>

For $N = 30$:

<table>
<thead>
<tr>
<th>Mixture</th>
<th>1 DCF: 29 EXP</th>
<th>15 DCF: 15 EXP</th>
<th>29 DCF: 1 EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRU</td>
<td>0.488</td>
<td>0.500</td>
<td>0.556</td>
</tr>
<tr>
<td>MTS</td>
<td>44.762</td>
<td>43.662</td>
<td>39.314</td>
</tr>
<tr>
<td>SDTS</td>
<td>84.377</td>
<td>59.364</td>
<td>15.709</td>
</tr>
<tr>
<td>THRDCF</td>
<td>0.0114</td>
<td>0.0160</td>
<td>0.0185</td>
</tr>
<tr>
<td>THREXT</td>
<td>0.0165</td>
<td>0.0173</td>
<td>0.0181</td>
</tr>
</tbody>
</table>

As expected, a user using our algorithm takes a larger share of the bandwidth.
in most cases but overall there is a reasonable share for every user. More users using our algorithm have a reduced standard deviation of the time between successful transmissions (SDTS).

Targeting $1.78^*s^* / N$ does not hurt our results for VOIP much. In the dynamic simulation with a mean of $N=30$ users using our algorithm with the 1.78 factor, we get throughput is 0.521354 and the mean time between two successful transmissions of 41.38 ms with a standard deviation of 22.96 ms. The mean delay of VOIP packets is 27.06 ms with a standard deviation of 27.50. The mean numbers of users throughout this simulation is 29.562069 with a standard deviation of 4.786050. We see that using the factor $1.78^*s^*$ has very little effect when compared with the results of dynamic experiment using the optimal factor $s^*$ in 5.3. We see for the case of $N = 10, 20$, the mean delay of VOIP packets is shorter.

5.7 CSIM simulation Package and its application in our model

CSIM is a process-oriented, discrete-event simulation toolkit originally developed in 1984 by Herb Schwetman at Microelectronics and Computer Technology Corporation (MCC). It is a C/C++ library which consists of a set of classes, functions, procedures that enables researchers to construct simulation models for large complex systems in order to better understand, analyze and predict the dynamic behaviors of systems and further help make the best decision in configuring system resources to improving system performance.

The version of CSIM we use runs on a Windows platform. We use Visual Studio 2003 as the development environment. cpp.h of CSIM is the header file which include the definitions of classes, procedures and functions in CSIM. The application development interface of CSIM provides researchers with a means to create their own simulation model. Generally, in CSIM models, processes are used to simulate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
5.7. CSIM SIMULATION PACKAGE AND ITS APPLICATION IN OUR MODEL

Dynamic entities (such as customers who arrive or depart in a dynamic way) and facilities are used to simulate stable resource entities (such as servers who provide services to customers). Customers use functions (reserve() and release()) of the class facility to take the control of the server and the function hold() is used to advance the simulation time. Processes of customers will terminate once customers get serviced and leave the system.

However, in our model, each user needs to keep monitoring the status of the channel and transmits only at the beginning of time slots after a DIFS. If we use processes to simulate users and a facility to simulate the channel, the synchronization between users will make model-building complicated. Also, users need to retry the channel repetitively after transmitting but not leave the system. In the collision case, several users may use the channel simultaneously but none take control of channel as is generally the case in CSIM models. So our model is not a typical scenario in CSIM models.

In this thesis, we just use the function hold() together with CSIM built-in random number generator to control the flow of user arrivals, departures and channel access attempts. The process birthp() is created to control the user arrival stream and the process deathp() is created to control the user departure stream, and the process chnln() is created to control the users' channel access. These three processes run simultaneously and interact with each other to build a dynamic system.

Also we use class table and histogram in CSIM to collect data and generate the statistics.

Lastly, the version of CSIM we used does not support generic programming. We create a class station to simulate users and create a class slist as a user list to manage the life-cycle of users in the system.
Chapter 6

Results of some experimental algorithms

In this chapter, we describe some attempts that were unsuccessful. For some of them, the results are not very clear or unsatisfactory; for others, the results are similar to our algorithm and do not make obvious improvements. So, we just give a chronological brief description of them.

6.1 Some results in the early time

6.1.1 MIMD

At the start of this thesis research, we use a MIMD (Multiplicatively Increase and Multiplicatively decrease) algorithm as follows: $\hat{q}_i$ is the proportion of inactive time slots observed by user $i$ from the traffic on the channel that does not involve the transmission activities of itself. For user $i$, if it does not transmit in this slot, then if the slot is idle one, count 1, otherwise, count 0. If user $i$ transmits in this slot, then if the transmission is a collision, count 0, otherwise count 1. Then $\hat{q}_i$ is the number of ones divide by the total number of both ones and zeros. Each user adapts its access probability periodically as:
\[ p_i^\text{new} = (1 - p_i^\text{old}) \cdot 1 / \exp(-s^*) \cdot p_i^\text{old}. \]

Here \( 1 - p_i^\text{old} \) is the probability that user \( i \) itself does not transmit. Then \( (1 - p_i^\text{old}) \cdot 1 \) will be the probability that an arbitrary slot is inactive. The same as in our algorithm or Idle Sense, if this probability is larger than \( \exp(-s^*) \cdot p_i^\text{old} \), as the formula shows, \( p_i^\text{old} \) will be multiplied by a factor larger than 1. Otherwise, \( p_i^\text{old} \) will be multiplied by a factor smaller than 1. On the other hand, if \( p_i^\text{old} \) is larger, the factor \( 1 - p_i^\text{old} \) is smaller, otherwise, the factor \( 1 - p_i^\text{old} \) is larger. In this way, the access probabilities of different users will gradually approach to a common value.

The problem we found is that: when the system is approximately in the optimal equilibrium, if a new user comes into the system with a small initial access probability, it will take a relatively long time until its access probability can catch up with those of users already in the system.

### 6.1.2 Kalman Filter

[12] presents an approach to estimate the number of active users in system based on extended Kalman filter with a change detection. We tried to use this method in our algorithm as follows. The estimation in [12] builds on the relationship between \( n_k \) the number of active users in system and \( p_k \) the packet collision probability on the channel condition on the considered user transmits. The relationship between this conditional collision probability and the number of users in system is:

\[ p_k = h(n_k) + v_k; \]
\[ h(n_k) = 1 - (1 - s^*/n_k)^{n_k-1}; \]
\[ h_k = \frac{\partial h(n)}{\partial n} |_{n=n_k-1} \]
\[ R_k = \text{Var}[V_k] = \frac{h(n_k) \cdot (1 - h(n_k))}{B} \]

\( R_k \) is the estimated variance of measurement \( p_k \), the collision probability condition on the considered user transmits. \( h_k \) is the changing rate of the measurement, linearized around the state estimate \( n_k-1 \). It is the partial derivative of \( h(n) \) in term of \( n \). We applied this estimation in our algorithm as follows:
1. We observe the channel status for B time slots. As in [12], if the considered user transmits in this slot, a successful transmission count 0 and a collision count 1 (the considered user can tell the difference from if it receives ACK when transmitting); if the considered user does not transmit in this slot, a transmission count 1 (no matter it is a successful one or a collision and a collision) and an idle slot count 1. The conditional probability $p$ of an user experiences a collision when it transmits its data is the number of ones divided by $B$ (the sum of number of both ones and zero).

2. The extended Kalman filter (EKF) consists of two sets of equations. One set of equations are time update equations which use to predict the system state at the next moment based on the estimated system state at the previous moment. Another set of equations are measurement update equations used to get the new estimated system state by correcting the predicted system state with the discrepancies between the actual measurement and the predicted measurement. In our algorithm, they are as follows:

EKF time update equations:
\[
\hat{n}_{k+1}^- = \hat{n}_k + w_k;
\]
\[
P_{k+1}^- = P_k + Q_k.
\]
Here $\hat{n}_{k+1}^-$ is the predicted number of users in system at moment $k + 1$ given by $\hat{n}_k$, the estimated number of users in system at moment $k$ plus a random variable $w_k$. $w_k$ is a state noise which accounts for the arrival and departure of users of system between the two moments. $P_{k+1}^-$ is the error variance. $Q_k$ is the variance of the state noise $w_k$.

EKF measurement update equations:
\[
K_k = (P_{k-1}^- + Q_{k-1}) \ast h_k / ((P_{k-1}^- + Q_{k-1}) \ast h_k^2 + R_k);
\]
\[
\hat{n}_k = \hat{n}_k^- + K_k \ast (p_k - h(\hat{n}_{k-1}));
\]
\[
P_k = P_k^- \ast (1 - K_k \ast h_k);
\]
\[
z_k = p_k - h(\hat{n}_{k-1}^-)
\]
$z_k$ is the innovation given by the discrepancies between the actual measurement $p_k$ and the predicted measurement $h(\hat{n}_{k-1})$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
3. In [12], a change detection mechanism is implemented with a change detection filter based on CUSUM (Cumulative SUMmary) test. It uses two filtered versions $g^+_k$ and $g^-_k$ of the innovation process $z_k$. A related process $s_k$ is used to represent the innovation process normalized with respect to its standard deviation.

$$s_k = \frac{z_k}{\sqrt{(P_{k-1}+Q_k)\cdot h_k^2+Q_k^2}}$$

The $g^+_k$ and $g^-_k$ are derived from $s_k$ as follows:

$$g^+_k = \max(0, g^+_k + s_k - v);$$
$$g^-_k = \min(0, g^-_k - s_k - v).$$

Here $v$ is a constant parameter. In CUSUM, there is another parameter $h$, called "alarm threshold". When $g^+_k > h$ or $g^-_k < -h$, a change alarm is sent. After an alarm, both $g^+_k$ and $g^-_k$ are reset to 0. Alarms from CUSUM are used to adaptively set the variance $Q_k$ of the noise $w_k$. Upon an alarm coming at moment $k$, $Q_k$ is set to a sufficiently large constant which represents a noise impulse in the time update equation so that Kalman filter can move away from its former estimate and converge to a new estimate rapidly. Otherwise, no alarm comes, $Q_k$ will stay 0 to stabilize the estimate.

From simulation, in the static experiment, EKF works well, but in the dynamic case with lots of users arriving and departing, EKF does poorly. It cannot give accurate estimation in an acceptable time.

### 6.2 Some recent results

Our algorithm gives an improvement to Idle Sense in terms of two measurements of short term fairness. However, the standard deviation of the period between two successful transmissions and the mean of voice signal delay are not decreased substantially. We consider the following factors may cause the problem.

1. In the AIMD mechanism of Idle Sense, the access probability is increased by a fixed increment (0.001 in Idle Sense). When $N$ is large, the access probability has a relatively large fluctuation and this cause a large jitter for back off counter.
2. The proportion of collisions is small but still exists. When collisions occur, the back off waiting time is a random variable with a large standard deviation.

3. The speed of the convergence of the access probability of each user to the optimal value is slow.

Here we use the simulation results of a static experiment with 40 users using Idle Sense plus our modification (wait a deterministic back-off of \([1/p]\) time slots but without rounding to the multiples of 10 before retransmit after a successful transmission) as a reference. The following comparisons in this chapter are all based on static experiment of 40 users. For Idle Sense with our modification, the overall throughput is 50.01 percent. The mean and standard deviation of the period between two successful transmissions are 58.23 ms, 33.61 ms respectively. For the original Idle Sense, the overall throughput is 50.04 percent. The mean and standard deviation of the period between two successful transmissions are 58.20 ms, 44.18 ms respectively.

### 6.2.1 Some attempts that tried to reduce the fluctuation of a user’s access probability

When we increase the access probability, we set \(p=p+f(N)\) instead of a fixed number 0.001. Here \(f(N)\) is function of \(N\) and it decreases when \(N\) increases. Because \(f(N)\) has arbitrary number of choices, we only tried \(f(N)=1/(C+k*N)\), \(C,k\) is a constant and \(C=1000\). We applied this method to Idle Sense with our modification. We found that when \(k\) is small, the overall throughput almost remains the same. The standard deviation between two successful transmissions had a little improvement. For an example for \(C=1000,k=1\), the overall throughput is 49.9879 percent. The standard deviation of the period between two successful transmissions is 33.48584 ms. But we found when \(N\) gets large, the speed the access probability of each user approaches the optimal value decreases. When \(N\) gets larger, the access probability of some users even diverges from the optimal value.
In [8], the algorithm, TES (Time-fair Efficient and Scalable MAC protocol) was presented. A MIMD mechanism in TES is used as an improvement for Idle Sense's AIMD. Its MIMD algorithm is as following:

The back off counter for user $i$ is chosen uniformly from $[0, CW_i^k]$, $CW_i^k$ is the size of adaptive contention window. $CW_i^k = \frac{2}{p_i^k} - 1$, $p_i^k$ is the access probability. $\hat{q}_i$ is the proportion of inactive time slots observed by user $i$. Here the increase and decrease are for contention window $CW$. In [8], $k_{base} = 1.01$, $k_{dec} = 0.0075$, $k_{inc} = 0.6$, $\alpha = 0.5$, $CW_i^{k+1} = \alpha \times CW_i^k + (1 - \alpha) \times CW_i^{k+1}$.

Multiplicative decrease if $\hat{q}_i > \exp(-s^*)$ then set $CW_i^{k+1} = \frac{CW_i^k}{k_{base} + k_{dec} \times \sqrt{CW_i^k}}$

Multiplicative increase if $\hat{q}_i < \exp(-s^*)$ then set $CW_i^{k+1} = CW_i^k \times (k_{base} + k_{inc} / \sqrt{CW_i^k})$

This algorithm has some improvement than Idle Sense. Also our modification (wait a deterministic back-off of $[1/p]$ time slots before retransmit after a successful transmission) to Idle Sense can apply to this algorithm and makes a similar improvement as it does to Idle Sense. The simulation results are as below: For the original TES we implemented on our platform, the overall throughput is 49.9414 percent. The mean and standard deviation of the period between two successful transmissions are 58.31092 ms, 39.58232 ms. For TES with our modification, the overall throughput is 50.3324 percent. The mean and standard deviation of the period between two successful transmissions are 57.85382 ms, and 33.40198 ms respectively.

6.2.2 Some attempts that tried to reduce collisions

We use a smaller number $\alpha \times s^*$ take the place of $s^*$. This method decreases the standard deviation of the period between two successful transmissions since the probability of a collision is smaller. But at the same time it decreases the overall throughput and increases the mean of the period between two successful transmissions.

We choose the back-off counter from a small window $[-3,3]$ instead of choosing from $[0, CW]$, $CW = \frac{2}{p} - 1$ or use a geometry random variable with mean $\text{int}(1/p)$ as
6.2. SOME RECENT RESULTS

back off counter. We apply this method to TES with our modification. For TES with our modification, the overall throughput is 50.2888 percent. The mean and standard deviation of the period between two successful transmissions are 57.9059 ms, 25.60254 ms respectively.

6.2.3 Some attempts that tried to increase the speed of convergence of users’ access probabilities

When we increase the access probability, we set $p = p + f(N)$ instead of a fixed number 0.001 the same as for the first factor. Here $f(N)$ is function of $N$. Instead, we set $f(N) = C \times N / (k + N)$. $C, k$ is a constant. When a user has a larger $p$ and a smaller $N$, we want its $p$ has a smaller increment. Conversely, its $p$ can have a larger increment when a user with a smaller $p$ and a larger $N$. In this way, different users with different access probabilities can approach to the same value faster than without this method. We apply this method to Idle Sense with our modification. It makes improvements in many cases. For a example for $C = 0.0015, k = 25$, the overall throughput is 49.9879 percent. The mean and standard deviation of the period between two successful transmissions are 58.1185 ms, 32.69748 ms respectively. The problem is that the fluctuation of access probability depends on the constant $C$, especially when $N$ get large.
Chapter 7

Conclusion and future work

7.1 Conclusion

This thesis presents an adaptive back-off algorithm with two improvements over the adaptive algorithm Idle Sense. The first modification is to use the filtered estimate of the users' adaptive access probability as the user's access probability. The second modification is the key idea of our improvement. After a successful transmission, a user will set its back-off counter to the closest multiple of 10 of the deterministic value \( \left\lfloor \frac{1 - p}{p} \right\rfloor \) where \( p \) is the filtered estimate of the users' adaptive access probability. Combining with these two modifications, to some extent, we implement both the ideas from Idle Sense and Zero Collision together.

In this thesis, a C++ program was developed as the simulation platform to compare the performance of our algorithm, Idle Sense and 802.11 with binary exponential back-off. The program can be used by other researchers with only slight modification as a simulation platform to evaluate the performance of other new algorithms.

We use simulations to compare our algorithm with Idle Sense, we conclude our algorithm considerably reduces the short term unfairness while having almost the same throughput as that of Idle Sense. The waiting time before packets of each user can be transmitted are much more regular and in simulations of VOIP, more VOIP
users can communicate through a single access point with an acceptable QoS range.

7.2 Future work

In our algorithm, the access probabilities of users are adapted using AIMD asynchronously based on the estimate of the probability that an arbitrary time slot is idle. But maybe AIMD is not the optimal procedure, even though it is used in TCP/IP. If we can improve the rate of convergence and reduce the standard deviation of our estimate, then we can improve the performance of our algorithm.
Chapter 8

Appendix A: SOME DETAILED RESULTS OF EXPERIMENTS

<table>
<thead>
<tr>
<th>Number of users</th>
<th>Overall throughput</th>
<th>Mean of period</th>
<th>SD of period</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.514422</td>
<td>7.08</td>
<td>2.22</td>
</tr>
<tr>
<td>10</td>
<td>0.540060</td>
<td>13.48</td>
<td>0.00</td>
</tr>
<tr>
<td>14</td>
<td>0.530088</td>
<td>19.23</td>
<td>4.04</td>
</tr>
<tr>
<td>20</td>
<td>0.544503</td>
<td>26.74</td>
<td>0.00</td>
</tr>
<tr>
<td>24</td>
<td>0.522969</td>
<td>33.41</td>
<td>10.70</td>
</tr>
<tr>
<td>34</td>
<td>0.521690</td>
<td>47.45</td>
<td>16.46</td>
</tr>
<tr>
<td>36</td>
<td>0.522287</td>
<td>50.18</td>
<td>17.50</td>
</tr>
<tr>
<td>40</td>
<td>0.519626</td>
<td>56.04</td>
<td>20.70</td>
</tr>
<tr>
<td>66</td>
<td>0.509548</td>
<td>94.30</td>
<td>41.16</td>
</tr>
<tr>
<td>80</td>
<td>0.507670</td>
<td>114.72</td>
<td>51.70</td>
</tr>
</tbody>
</table>

Table 8.1: The simulation result1 of Our Algorithm

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
### Table 8.2: The simulation result1 of Idle Sense

<table>
<thead>
<tr>
<th>Number of users</th>
<th>Overall throughput</th>
<th>Mean of $period^1$</th>
<th>SD of $period^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.503013</td>
<td>7.24</td>
<td>4.47</td>
</tr>
<tr>
<td>10</td>
<td>0.501909</td>
<td>14.51</td>
<td>9.55</td>
</tr>
<tr>
<td>20</td>
<td>0.501469</td>
<td>29.04</td>
<td>20.10</td>
</tr>
<tr>
<td>40</td>
<td>0.500351</td>
<td>58.20</td>
<td>44.18</td>
</tr>
<tr>
<td>80</td>
<td>0.498588</td>
<td>116.81</td>
<td>104.85</td>
</tr>
</tbody>
</table>

### The simulation result of the basic 802.11b

<table>
<thead>
<tr>
<th>Number of users</th>
<th>Overall throughput</th>
<th>Mean of $period^1$</th>
<th>SD of $period^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.506555</td>
<td>7.19</td>
<td>8.75</td>
</tr>
<tr>
<td>10</td>
<td>0.480109</td>
<td>15.16</td>
<td>31.50</td>
</tr>
<tr>
<td>20</td>
<td>0.444995</td>
<td>32.72</td>
<td>86.73</td>
</tr>
<tr>
<td>40</td>
<td>0.405222</td>
<td>71.86</td>
<td>199.64</td>
</tr>
<tr>
<td>80</td>
<td>0.360377</td>
<td>161.61</td>
<td>418.19</td>
</tr>
</tbody>
</table>

### Table 8.3: The simulation result1 of the basic 802.11b

<table>
<thead>
<tr>
<th>Number of users</th>
<th>Overall throughput</th>
<th>Mean of $period^1$</th>
<th>SD of $period^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.857035</td>
<td>47.75</td>
<td>2.10</td>
</tr>
<tr>
<td>10</td>
<td>0.858133</td>
<td>95.37</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>0.843129</td>
<td>194.14</td>
<td>50.94</td>
</tr>
<tr>
<td>40</td>
<td>0.834202</td>
<td>392.42</td>
<td>131.86</td>
</tr>
<tr>
<td>80</td>
<td>0.826529</td>
<td>792.08</td>
<td>305.48</td>
</tr>
</tbody>
</table>

### Table 8.4: The simulation result2 of Our Algorithm

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Number of users</th>
<th>Overall throughput</th>
<th>Mean of \textit{period}¹</th>
<th>SD of \textit{period}¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.834071</td>
<td>49.06085</td>
<td>60.0876</td>
</tr>
<tr>
<td>10</td>
<td>0.837005</td>
<td>97.7752</td>
<td>202.8425</td>
</tr>
<tr>
<td>20</td>
<td>0.836168</td>
<td>195.80635</td>
<td>521.8316</td>
</tr>
<tr>
<td>40</td>
<td>0.833378</td>
<td>393.2468</td>
<td>1097.93095</td>
</tr>
<tr>
<td>80</td>
<td>0.828024</td>
<td>789.62</td>
<td>2038.2342</td>
</tr>
</tbody>
</table>

Table 8.5: The simulation result² of 802.11 RTS/CTS Algorithm

<table>
<thead>
<tr>
<th>Number of users</th>
<th>Overall throughput</th>
<th>Mean of \textit{period}¹</th>
<th>SD of \textit{period}¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.809350</td>
<td>50.55345</td>
<td>62.34885</td>
</tr>
<tr>
<td>10</td>
<td>0.758204</td>
<td>107.85045</td>
<td>229.2177</td>
</tr>
<tr>
<td>20</td>
<td>0.697290</td>
<td>234.5899</td>
<td>609.79525</td>
</tr>
<tr>
<td>40</td>
<td>0.633335</td>
<td>516.63705</td>
<td>1449.8989</td>
</tr>
<tr>
<td>80</td>
<td>0.563936</td>
<td>1160.7523</td>
<td>3043.27305</td>
</tr>
</tbody>
</table>

Table 8.6: The simulation result² of 802.11 basic Algorithm

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Chapter 9

Appendix B: SOURCE CODE OF SOME SIMULATION PROGRAMS

In appendix, we list the source code of some simulation program.

9.1 Source code for idle sense and our algorithm for static experiment and dynamic experiment

List for source code in cpp file

```cpp
// C++/CSIIM Model

#include "cpp.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include "ex.h" /* class definitions */

/// global variables////
int n=0; /* counter of users*/
double thrpt=0.0;
facility *channel; /* pointer to facility channel*/
```
station *sta;
FILE *fp;
table *tabl[50],*tabp[50],*tbthru,*tblov,*tabv,*tbvov,*tbn; // *tbcol
double sstar, ustar;
slist *slst;
stream *bs,*ds,*bfs;

/////processes declaration/////
void birthpO;
void deathpO;
void chnnlO;

/////CSIM entry/////
extern "C" void sim(int argc, char *argv[]) {
  initO;
  create("sim");
  birthpO;
  deathpO;
  chnnlO;
  hold(TMAX);
  printf("num of user: %d\n",m);
  printf("simtime V,g\n", simtimeO);
  tbthru->record(2.0*thrpt/simtime());
  report(); // model report
  mdlstatO; // model statistics
}

/////initialization/////  
void initO() {
  fp = fopen("csim.out", "w");
  set_output_file(fp);
  set_model_name("802.11 std");
  sstar«calcsstar(lcc);
  ustar«double(exp(-sstar)/(l-exp(-sstar)));
  printf("Xg \n",sstar);
  printf("'/tg \n",ustar);
  printf("/»g \n",exp(-sstar));
  channel= new facility("channel"); // instantiate facility f
  slst= new slist();
  tbn= new table("numberofusers");
  tblov= new table("prdoverall");
  tbvov= new table("voicewaitingtimeoverall");
  tbvov->add_histogram(60,0.0,0.20);
}

/////main process/////  
void chnnlO() {
  int s,j,jn, rsold[60];
9.1. SOURCE CODE FOR IDLE SENSE AND OUR ALGORITHM FOR STATIC EXPERIMENT AND DYNAMIC EXPERIMENT

double test, tl, vt;
slistitem *tmp;
create ("ch")
bfsnew stream();
while (sintime()<TMAX)
{
    in=0; i=0;
tmp=NULL;
tmp=slist->gethd();
if (tmp!=NULL)
{
    for (j=0; j<N; j++) rsold[j]=0;
do
    for (i=0; i<N; i++) rsold[i]=0;
    tmp=slist->gethd();
in=in+rsold[i]->gen();
    tmp=tmp->getnext();
i++;} while (tmp!=NULL);
i=0;
tmp=NULL;
tmp=slist->gethd();
    j=int(clock/40.0);
do
    (tmp->getval())->obser(in); /*for experiment for Idle Sense and our algorithm*/
if (j<50) (tmp->gettbp(j))->record((tmp->getval())->getp());
if (rsold[i]<l) /* collision */
{
    if (in>l)
        (tmp->getval())->transmit(2);
    else /* transmit */
        tl=(tmp->getval())->gettl();
        vt=(tmp->getval())->getvt();
        if ((vt<0.0) && (sintime() > 0.5*TMAX))
        {
            test=vt;
            while (clock < test)
            {
                (tmp->gettbv())->record(clock-test);tobs->record(clock-test);
                test=test+40.0;
            }
            (tmp->getval())->setvt(test);
        }
    if ((tt<0.0) && (sintime() > 0.5*TMAX) && (j<50)) (tmp->gettbp(j))->record(clock-tl);
    if ((tt<0.0) && (sintime() > 0.5*TMAX)) (tobs->record(clock-tl))
    (tmp->getval())->transmit(tl);
}
if (rsold[i]==0)
9. APPENDIX B: SOURCE CODE OF SOME SIMULATION PROGRAMS

```c
(tmp->getval())->idle(in);
while (tmp!=NULL):
    if (simtime()>0.00000001) //in idle, 1 transmit, 2 collision
        if (tmp->getnext()==0) //channel
            if (in>0) (channel).reserve();
            if (in>1) {hold(i);} else hold(ltr);
        else (*channel).release();
    else hold(0.00000001);
} //endofwhile

/////Arrival of users/////
void birthp0
{
    create("birthp");
    int i,j,rn,dec,sgn;
    double lftm,rd,vttmp;
    char *sl;
    char sE["stn00000000"];
    slistitem *slstitm;
    bstream();ds->new stream();
    while ((simtimeO<TMAX)&&(n<M))//(simtimeO<300)
    {
        n++; //rd=ds->exponential(LIFETM);lftm=simtimeO+rd; /*for dynamic*/
        lftm-TMAX; /*for static*/
        vttmp=max(clock,0.6*TMAX);
        slstitm=new slistitem(sta,tabl,tabp,tabv);
        slist->insslst(slstitm);
    }
}
```
9.1. SOURCE CODE FOR IDLE SENSE AND OUR ALGORITHM FOR STATIC EXPERIMENT AND
DYNAMIC EXPERIMENT

```c
///Departure of users/////
void deathp()
{
    create("deathp");
    slistitem *tmp,*tmpl;
    while (simtime()<TMAX)
    {
        tmp=NULL;
        tmp=slist->gethd();
        if (tmp!=NULL)
        {
            do
            {
                if (((tmp->getval())->die))
                {
                    tmp1-trap;tmp->getnext();
                    slist->delslist(tmp1);
                }
                else tmp=tmp->getnext();
            }while (tmp!=NULL);
            hold(0.00001);
        }
    }

//List for source code in header file
extern double thrpt,sstar;
extern stream *bfs;
extern double mmin(double a,double b);
extern double mmax(double a,double b);

//simulation control parameter
#define N 200  // number of sources table/
#define N 1000// number of sources for dynamic experiments/
#define N 5    // number of sources is 5 for static experiments/
#define N 10   // number of sources is 10 for static experiments/
#define N 20   // number of sources is 20 for static experiments/
#define N 40   // number of sources is 40 for static experiments/
#define N 80   // number of sources is 80 for static experiments/
#define TMAX 4800.0 // simulation time for dynamic experiments/
#define TMAX 2400.0 // simulation time for static experiments/
#define IARVTM 8.0 /* mean of interarrival*/
#define LIFETM 240.000 /* mean of lifetime*/
#define VOCSAMFRq 0.01 /* interval of voice signal generation*/

//algorithm and protocol parameter
#define maxbs 5
#define cumin 32
#define cwmax 1024
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
#define lpd 0.000728
#define ts 0.00002
#define lcc 61.3
#define lc 0.001226
#define ltr 0.001226  
/*length of collision for basic model*/
/*length of successful transmission for basic model*/
/*for basic models*/

//////////////////////////////////// user class definition

class station : public csim.facility {
protected:
    double birthtime;
    double lifetime;
    double p;
    int bftyp; // 2 geometry rv, 1 fix 1/p
    int bftyp;  
    int rs;
    double tlastsuc;
    double voicetmstp;
    int sumblank;
    int gemnum;
    int owngem;
    double thru;
    unsigned long trhis;
    int r, l, butn;

public:
    station(const char* s, double bt=0.0, double lt=0.0, double pp=1.0/32.0, int btyp=0, int bcnt=0, int r=0, int sblank=0, int gnum=0, double tl=0.0, double th=0.0,  
        int ogm=0, double vtmstp=0.0, unsigned long trh=131071, int rr=0, int ll=0, int butnw=0) : facility(s) {
        p = pp; birthtime = bt; lifetime = lt; bftyp = btyp; bfcnt = bcnt; rs = r; sumblank = sblank; gemnum = gnum; tlastsuc = tl; thru = th; owngem = gnum; voicetmstp = vtmstp; trhis = trh; rr = rr;  
        ll = ll; butn = butnw;  
    }
    station();  
    void setval(double bt=0.0, double lt=0.0, double pp=1.0/32.0, int btyp=0, int bcnt=0, int r=0, int sblank=0, int gnum=0, double tl=0.0, double th=0.0,  
        int ogm=0, double vtmstp=0.0, unsigned long trh=131071, int rr=0, int ll=0, int butnw=0) {
        birthtime = bt; lifetime = lt; bftyp = btyp; bfcnt = bcnt; rs = r; sumblank = sblank; gemnum = gnum; tlastsuc = tl; thru = th; owngem = gnum; voicetmstp = vtmstp; trhis = trh; rr = rr;  
        ll = ll; butn = butnw;  
    }
    void dspO;  
    void dspstatO;  
    int getrsO {return rs;}  
    double gettlO {return tlastsuc;}  
    double getvtO {return voicetmstp;}  
    double setpO(double pp){}
    void obser(int);  
    void transmit(int in);  
    int gem();  
    void select(int in);  
    int die();  
};

//////////////////////////////////////// user list definition

class slistitem{
private:
  station *val;
  table *tbl[50];
  table *tbp[50];
  table *tbvtm;
  slistitem *next;

public:
  slistitem(station *st, table *tp[], table *tb[])
  {val=st;tbvtm=tbv;next=NULL;int i;for(i=0;i<50;i++)
    {tbl[i]=tp[i];tbp[i]=tb[i];}}
  slistitem() {next=NULL;delete val; /*printf("\nSli
");*/}
  station* getval() {return val;}
  table* gettbl(int i) {return tbl[i];}
  table* gettbp(int i) {return tbp[i];}
  table* getbv() {return tbvtm;}
  slistitem* getnext() {return next;}
  void setnext(slistitem *slstitm) {next=slstitm;}
  void disp() {val->disp();}
};

class slist {
private:
  slistitem *hd, *tl; // *iter

public:
  slist() {hd=NULL;tl=hd;}//iter=hd;
  slistitem* gethd() {return hd;}
  slistitem* gettl() {return tl;}
  void inslst(slistitem *slstitm)
  {if (hd==NULL)
    {hd=slstitm;tl=hd;}
  else
    
    }/*return next*/
  slistitem* fndpreslst(slistitem *slstitm)
  {slistitem *tmp,*tmpl;
    tmp=hd;
    do
    {tmpl=tmp->getnext();
      if (tmpl==slstitm)
        return tmp;
    else
      tmp=tmpl;
    }while (tmpl!=tl);
    return NULL;
  }
  void delslist(slistitem *slstitm)
  {slistitem *tmp;
    //tmp=fndpreslst(slstitm);
    if (slstitm==hd)
      {hd=hd->getnext();delete slstitm;}
    else
      
    }
    return NULL;
  }
  void fndprlst(slistitem *slstitm)
  {slistitem *tmp;
    //tmp=fndpreslst(slstitm);
    if (slstitm==hd)
      {hd=hd->getnext();delete slstitm;}
    else
      
  }

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX B: SOURCE CODE OF SOME SIMULATION PROGRAMS

```c
//void dspO();
void dsp0()
    slistitem *tmp=bd;
    do
    { (tmp->getval())->dsp(); tmp=tmp->getnext();
    }while (tmp!=NULL); 

};
//Method or function of users////////
void station::transmit(int in)
{
    int i;
    if (in==0)
    { 
        rs=0;
        bftype=1;
        if (simtime()>0.5*TMAX) (thru+thru;thrt=thrt;lpd=lpd;)
        tlastsuc=clock;
        bfcnt=int((1.0-p)/p); /*for our algorithm*/
        bfcnt=bfcnt->uniform_int(0,int((1.0-p)/p)*2); /*for idle sense*/
        bfcnt=int(bfcnt<int); /*for our algorithm*/
    }else
    { 
        rs=0;
        bftype=bfcnt->uniform_int(0,int((1.0-p)/p)*2); 
    } } 
void station::dsp()
{
    printf("last");
    printf("%s \t\t\n" ,name());
}

void station::dspstat()
{
    printf("Xs ",name());printf("%g \t\n",p);
}

void station::idle(int in)
{//double rd;
    int tfld=0.1;
    if (bftype==1)
    { 
        bfcnt==
    } 

int station::die()
{
    if (((lifetime<simtime())&&((rs==0))||(simtime()=TMAX))) return 1; else return 0;
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
9.1. SOURCE CODE FOR IDLE SENSE AND OUR ALGORITHM FOR STATIC EXPERIMENT AND DYNAMIC EXPERIMENT

```c
int station::gen()
{
    double rd;
    int bfcntold=0,rtmp=0;
    bfcntold=bfcnt;
    if (rs==0)
    {
        if (bftyp==1)
        {
            bfcntold=bfcnt;
            if (bfcntold>0) rs=1;
        }
    }
    return rs;
}

void station::obser(int in)
{
    int i=0;
    double qhat,ptmp;
    if (in==0)
    {
      sumblank++;
    }
    else
    {
        if (i>1) {gennum++;}
        else {if (rs==0) {gennum++;} else {gennum++;}}
    }

    if (((sumblank+gennum)>500) && (gennum<0))
    {
        qhat=double(sumblank)/(gennum+sumblank);
        ptmp=p;
        if ((qhat/exp(-cstar))>1.0)
            p=min(0.999, max(0.00001,p*0.001));
        else
            p=min(0.999, max(0.00001,p/1.2));
        p=p*ptmp*0.01;p; //for our algorithm
        gennum=0;sumblank=0;
    }

    double min(double a,double b)
    {
        if(a<b) return a;
        else return b;
    }

    //////////////////////////////////////////////////////////////////////
    double mmin(double a,double b)
    {
        if(a<b) return a;
        else return b;
    }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
9.2 Source code for 802.11 with binary exponential back-off for static experiment and dynamic experiment

List for source code in cpp file

```c
// C++/CSIM Model of M/M/1 queue
#include "cpp.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include "ex.h" /* class definitions */
#include "ex.h" /* class definitions */

int m=0; /* counter of sources*/
double thrpt=0.0; /* counter of sources*/
double colpt=0.0;

facility *channel; /* pointer to facility channel*/
station *sta; /* pointer to facility channel*/
FILE *fp;
```
9.2. SOURCE CODE FOR 802.11 WITH BINARY EXPONENTIAL BACK-OFF FOR STATIC EXPERIMENT AND DYNAMIC EXPERIMENT

```c
// table *tabl[50],*tabp[50],*tbthru,*tblov,*tabv,*tbvov,*tbn;//*tbcol
double estar,estar;
stream *bs,*ds,*bfs;

void initO;
void birthpO;
void deathpO;
void chnnlO;

extern "C" void sim(int argc, char *argv[])
{
    initO();
    create("sta");
    birthpO();
    deathpO();
    chnnlO();
    hold(TMAX);
    printf("num of user %d\n",m);
    printf("simtime %g\n",simtimeO);
    tbthru->record(2.0*thrpt/simtime());
    report(); // model report
    mdlstat(); // model statistics
}

void initO()
{
    fp = fopen("csim.out", "w");
    set_output_file(fp);
    set_model_name("802.11 std");
    channel* new facilityO'channel"; // instantiate facility f
    slst-neu slist();
    tbn->new table("numberofusers");
    tblov->new table("prdoverall");
    tbvov->add_histogram(50,0.0,0.20);
    tbthru->new table("throughput");
}
////////////////////generate users////////////////
void chnnlO
{
    int i,j,in, rsold[N];
    double test,tl,vt;
    slistitem *tmp;
    create("chn");
    bfs->new streamO;
    while (simtime()<TMAX)
    {
        in=0;i=0;
        tmp=NULL;
        tmp=slst->gethd();
        if (tmp1-NULL)
        {
            for (j=0;j<N;j++)rsold[j]=0;
```
do
{
  rsold[i]=(tmp->getval())->gen();
  in=in+rsold[i];
  tmp=tmp->getnext();
  i++;
}while (tmp=NULL);
  i=0;
  tmp=NULL;
  tmp=slst->gethd();
  j=int(clock/3.0);
  do
  {
    // (tmp->getval())->observe(in); /*for experiment for Idle Sense and our algorithm*/
    if (rsold[i]==1) /* collision */
      {
        if (in>1)
          (tmp->getval())->transmit(2);
        else /* transmit */
          {
            t=(tmp->getval())->gett();
            vt=(tmp->getval())->getvt();
            if ((t!=0.0)&&(simtime()>0.5*TMAX))
              {
                test=vt;
                while (clock>test)
                  {
                    (tmp->getval())->record(clock-test);tvow->record(clock-test);
                    test=VDriverFR
                  }
                (tmp->getval())->setvt(test);
              }
            if ((t!=0.0)&&(simtime()>0.5*TMAX)&&((j<50))){(tmp->getval())->record(clock-t1);}
            if ((t!=0.0)&&(simtime()>0.5*TMAX)){tbvow->record(clock-t1);}
            (tmp->getval())->transmit(1);
          }
        if (rsold[i]==0)
          {
            (tmp->getval())->idle(in);
          }
        tmp=tmp->getnext();i++;
      }
    while (tmp=NULL);
  }while (tmp!=NULL);
  if (simtime()>0.5*TMAX) tbo->record(i);
  //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\}
9.2. SOURCE CODE FOR 802.11 WITH BINARY EXPONENTIAL BACK-OFF FOR STATIC EXPERIMENT AND DYNAMIC EXPERIMENT

```c
else hold(0.000001);
}

void birthp()
{
create("birthp");
int i,j,rn,dec,sgn;
double lftm,rd,vttmp;
char *sl;
char s["stnOOOOOOOO"];
slistitem *slstitm;
bs*new streamO ;ds-new streamO;
while ((simtime()<TMAX)&6(m<M))
{
    m++;
    rd=ds->exponential(IARVTM);
    hold(rd);
    lftm=0.0;
    ifm=0.0; //for dynamic experiment
    ifm=simtime()+rd;//for dynamic experiment
    lftm=TMAX;//for static experiment
    vttmp=max(clock,0.5*TMAX); //for static experiment
    s[0]='s'[i];s[1]='t';s[2]='n';for(j=3;j<len(sl);j++) s[j]=0;
    sta*new station(s,clock,lftm,0,0,0,0,0,0,vttmp);
    tabv=new table(s);
    s[0]='v';s[1]='0';s[2]='0';
    for(i=0;i<50;i++)
    {
        s[0]='p';
        if (i<10) s[1]=char(64+i/10);s[2]=char(64+i%10); else s[1]=char(64);s[2]=char(64+i);};
        tabp[i]=new table(s);
        s[0]='p';
        tabl[i]=new table(s);
    }
slistitem *new slistitem(slistitem,tabl,tabp,tabv);
    slist->inssl8t(slstitm);
}

void deathp()
{
create("deathp");
slistitem *tmp,*tmpl;
while ((simtime()<TMAX)
{
    tmp=NULL;
    tmp=slist->gethd();
    if (tmp=NULL)
    {
        do
        {
            if ((tmp->getval())->die0)
```
9. APPENDIX B: SOURCE CODE OF SOME SIMULATION PROGRAMS

```c
    { tmp=tmp;tmp=tmp->getnext();
        slst->deletelst(tmp);
    }
    else tmp=tmp->getnext();
    while (tmp!=NULL);
}  //endof if tmp!=null

hold(0.00001);
} //endwhile

List for source code in header file

extern double thrpt;
extern stream *bfs;

///////////////////////////////////////////////////////////////////simulation control parameter
#define N 200  /* number of sources table*/
#define M 1000 /* number of sources for dynamic experiment*/
// #define M 5  /*number of sources is 5 for static experiment*/
// #define M 10 /*number of sources is 10 for static experiment*/
// #define M 20 /*number of sources is 20 for static experiment*/
// #define M 40 /*number of sources is 40 for static experiment*/
// #define M 80 /*number of sources is 80 for static experiment*/
#define TMAX 4800.0 /*simulation time for dynamic experiment*/
// #define TMAX 2400.0 /*simulation time for static experiment*/
#define IAVTM 8.0
#define LIFETM 240.000
#define V0CSAMFRQ 0.01

// #define M 5
// #define M 10
// #define M 20
// #define M 40
// #define M 80
#define TMAX 2400.0 /*simulation time for static experiment*/
#define IAVTM 8.0
#define LIFETM 240.000
#define V0CSAMFRQ 0.01

#define maxbs 5
#define cwmin 32
#define cwmax 1024
#define lpd 0.000728
#define ts 0.00002
#define lc 0.001226 /*length of collision for basic model*/
#define ltr 0.001226 /*length of successful transmission for basic model*/
#define Ice 61.3 /*for basic model*/

////////////////////////////////////////////////////////////////////////////

class station : public csim_facility {
protected:
    double birthtime;
    double lifetime;
    int bfrnt;
    int http://2 geometry rv, 1 back-off counter
    int bfrnt;
    int rs;
    double sletrem;
    double voicetmstp;
    int sumblank;
    int gemnum;
    int owngem;
```
9.2. SOURCE CODE FOR 802.11 WITH BINARY EXPONENTIAL BACK-OFF FOR STATIC EXPERIMENT AND DYNAMIC EXPERIMENT

```cpp
double thru;
public:
    station(const char* s, double bt=0.0, double lt=0.0, int b=0, int bttyp=0, int bcnt=0, int gem=0, double tl=0.0, double th=0.0, int ogm=0, double vtmstp=0.0): facility(s)
    {birthime=bt; lifetime=lt; birthime=bt; lifetime=lt; bftyp=btyp; bfcnt=bcnt; br=b; sumblank=sblank; gemnum=gnum; tlastsuc=tl; thro=th; ogm=ogm; voicetmstp=vtmstp; }
station();
    "stationO {/*printf ("st\n") ;*/ >
    void setval(double bt=0.0, double lt=0.0, int b=0, int bttyp=0, int bcnt=0, int r=0, int sblank=0, int gnum=0, double tl=0.0, double th=0.0, int ogm=0, double vtmstp=0.0)
    {birthime=bt; lifetime=lt; birthime=bt; lifetime=lt; bftyp=btyp; bfcnt=bcnt; br=b; sumblank=sblank; gemnum=gnum; tlastsuc=tl; thro=th; ogm=ogm; voicetmstp=vtmstp; }
    void dep();
    void display();
    int getrs() {return rs;}
    double gettl() {return tlastsuc;}
    double getvt() {return voicetmstp;}
    void setvt(double vt){voicetmstp=vt;}
    int getbs() {return bs;}
    void transmit(int in);
    int genO;
    void idle(int in);
    int dieO;
};
class slistitem{
private:
    station *val;
    table *tab[50];
    table *tbp[50];
    table *tbvtm;
    slistitem *snext;
public:
    slistitem(station *st, table *tl[], table *tbp[], table *tbvtm){val=st; tvtm=tbvtm; snext=NULL; int i; for(i=0;i<50;i++) {tbp[i]=tp[i]; tbl[i]=tl[i];}}
    slistitem() {snext=NULL; delete val;}/> printf("sli	\n"); */ >
    station* getval() {return val;}
    table* gettbl(int i){return tbl[i];}
    table* gettbp(int i){return tbp[i];}
    slistitem * getnextO {return snext;}
    void setnext(slistitem * slstitm){snext=slstitm;}
    void sdisp(){val->dsp();}
};
class slist{
private:
    slistitem *hd, *tl;
public:
    slist() {hd=NULL; tl=hd;}
    slist* gethd(){return hd;}
    slist* gettl() {return tl;}
    void inslettem(slistitem * slstitm)
    {if (hd==NULL)
        (hd=slstitm;tl=slstitm;)
    else
        (tl->next=slstitm;tl=slstitm;)
    }
    slist* findnext(slistitem * slstitm)
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
{slistitem *tmp,*tmpl;
tmp=hd;
do{
tmpl=tmp->getnext();
if (tmpl==slstitm)
    return tmp;
else
    tmp=tmpl;
}while (tmp!=tl);
return NULL;
}

void delslst(slistitem *slstitm)
{
    slistitem *tmp;
    if (slstitm==hd)
        {hd=hd->getnext();delete slstitm;}
    else
        {tmp=fndpreslst(slstitm);
         if (slstitm->getnext()) delete slstitm;
          tmp->setnext(slstitm->getnext());
          delete slstitm;}
}

void dsp()
{
    slistitem *tmp=hd;
do{
        if (tmp->getval()) tmp->dsp();tmp=tmp->getnext();
    }while (tmp!=NULL);
}

/////Method or function of users////////
void station::transmit(int in)
{
    extern int imin(int a,int b);
    int i;
    if (in==1)
    {
        rs=0;
        bftyp=1;//
        bi=0;
        if (simtime()>0.5*TMAX) {thru=thru+lpd;thrpt=thrpt+lpd;}
        tlastsuc=clock;
        bfcnt=bfs->uniform_int(0,(cwmin-1));
    }
    else
    {
        rs=0;bftyp=1;bi=imin(max(bi,bs+1));
        bfcnt=bfs->uniform_int(0,(cwmin-int(pow(2.0,double(bs))))-1));
    }
}

void station::dsp()
{
    printf("last");
    printf("\r\n",name());
}
9.2. SOURCE CODE FOR 802.11 WITH BINARY EXPONENTIAL BACK-OFF FOR STATIC EXPERIMENT AND DYNAMIC EXPERIMENT

```c
void station::dspstat()
{
  //printf("V.s ",nameO); sprintf(" Xg Yt ",p);
}

void station::idle(int in)
{
  int tflg=0,i;
  if (bftyp==1)
  {
    bflas--; 
  }
}

int station::die()
{
  if (((lifetime<simtime())&&(rs==0))||((simtime()<=TMAX)) return 1; else return 0;
}

int station::gen()
{
  double rd;
  int bftcnt0=0,rtmp0;
  bftcnt0=bftcnt;
  if (rs==0)
  {
    if (bftyp==1)
    {
      bftcnt0=bftcnt;
      if (bftcnt0<0) rs=1; 
    }
  }
  return rs;
}

int imin(int a,int b)
{
  if(a<b) return a;
  else return b;
}

int imax(int a,int b)
{
  if(a>b) return a;
  else return b;
}

double numax(double a,double b)
{
  if(a>b) return b;
  else return a;
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
9.3 Source code for compatibility of 802.11 with binary exponential back-off and our algorithm for static experiment

```c
#include "cpp.h" // class definitions
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include "ex.h"

#define N 10 /* number of sources */
#define TMAX 60000.0

facility *channel; // pointer to facility channel
station *sta[N];
FILE *fp;
table *tbl[N][60], *tbCN[N][50], *tbthru, *tblov, *tbhp;
double ustar, sstar;
stream *bfa, *bfe;
void initO;

extern "C" void sim(int argc, char *argv[])
(int i, j, in, resld[N])
double test, xt;
init();
create("sim");
while (simtime()<TMAX)
(j=int(clock/20.0);
in=0;
for(;in<N;i++)
{resld[i]=sta[i]->gen();
```
for(i=0;i<N;i++)
{ sta[i]->observe(in);
  if ((simtime()>0.5*TMAX)&&(j<50)) tblp[i][j]->record(sta[i]->getp());
  if ((simtime()>0.04*TMAX)&&(j<10)) tblp->record(sta[i]->getp());
}
if (in>0) //channel.reserve();
  if (in=0) (hold(tl)); else hold(tl); //O idle, 1 transmit, 2 collision
 else (hold(ts));
for(i=0;i<N;i++)
{ if (resold[i]==1) /* collision */
  {
    if (in>0) sta[i]->transmit(2);
    else /* transmit */
    { tl=sta[i]->gettl();
      if (tl>0.04&&(simtime()>0.04*TMAX)) {tbl[i][j]->record(clock-tl);}
    }
  }
if(resold[i]==0) (sta[i]->idle());
} //end while
for(i=0;i<N;i++) sta[i]->dsp();
report(); // model report
mdlstat(); // model statistics

void init()
{ int i,j;
  double ustar;
  char s[]="stationprot#";
  fp = fopen("cmin.out", "w");
  set_output_file(fp);
  set_model_name("802.11 std");
  ustar=calcstar(100);
  ustar=double(exp(-ustar)/(1-exp(-ustar)));
  printf("\%lg \n",ustar);
  printf("\%lg \n",ustar);
  bfa=new stream();bfe=new stream();
  channel+ new facility("channel"); // instantiate facility f
  for(i=0;i<N;i++)
  {
if (i>9) {s[10] = char(48 + i/10); s[11] = char(48 + i);}
else {s[10] = char(48); s[11] = char(48 + i);}
for(j=0; j<50; j++) {
  if (j>9) {s[8] = char(48 + j/10); s[9] = char(48 + j);}
  else {s[8] = char(48); s[9] = char(48 + j);}
  s[7] = 'r';
tbl[i][j] = new table(s);
  s[7] = 'p';
tblp[i][j] = new table(s);
}

if (i<5) {sta[i]->setval(1.0/32.0, 1, 0, 0, 0, 0, 0, 0.0, 0.0, 0, 1, 0, 131071);

for(j=0; j<50; j++) {
  if (j>9) {s[8] = char(48 + j/10); s[9] = char(48 + j);}
  else {s[8] = char(48); s[9] = char(48 + j);}
  s[7] = 'r';
tbl[i][j] = new table(s);
  s[7] = 'p';
tblp[i][j] = new table(s);
}

if (usrtyp == 1) {
  double rd;
  If (in == 1) {
    rs = 0;
    bftyp = 1;
    if (simtime() > 0.5 * TMAX) thru = thru + lpd;
    lstact = clock;
    bfcnt = int(1.0/p); bfcnt = int((bfcnt + 5)/10) * 10;
  }
  else {
    rs = 0;
    bftyp = 1;
    rs = 0;
    if (simtime() > 0.5 * TMAX) thru = thru + lpd;
    lstact = clock;
    bfcnt = bfe->uniform_int(0, 1 - 1.0 * p);
  }
}

if (in == 1) {
  bll = 0;
  rs = 0;
  if (simtime() > 0.5 * TMAX) thru = thru + lpd;
  lstact = clock;
  bfcnt = bfe->uniform_int(0, 1 - 1.0 * p);
source code for compatibility of 802.11 with binary exponential back-off and our algorithm for static experiment

```c
9.3. SOURCE CODE FOR COMPATIBILITY OF 802.11 WITH BINARY EXPONENTIAL BACK-OFF AND OUR ALGORITHM FOR STATIC EXPERIMENT

```
74
9. APPENDIX B: SOURCE CODE OF SOME SIMULATION PROGRAMS

(p=min(0.0134,max(0.00001,p+0.001));)
else
(p=min(0.0134,max(0.00001,p/1.2));)
p=0.999*p;pmp=0.01*p;
gemnum=0;sumblank=0;
}
}
///////////
List for source code in header file

#define maxbs 5
#define cumin 128
#define cvmax 1024
#define lpd 0.0364
#define ts 0.001
#define lc 0.0613 /*length of collision for basic model*/
#define ltr 0.0613 /*length of successful transmission for basic model*/
#define lcc 61.3 /*for basic model*/

// class station : public csim_facility {
protected:
int usrtyp; // 0 binary exp, 1 adp
int bs;
double p;
int bftyp; // 2 geometry rv, 1 fix 1/p
int bfcnt;
int rs;
double tlastsuc;
int sumblank;
int gemnum;
int owngem;
double thru;
unsigned long trhis;
public:
station(const char* s, double pp=1.0/32.0, int btyp=0, int bcnt=0,
int r=0, int sblank=0, int gnum=0, double tl=0.0, double th=0.0,
int ogm=0, int urty=0,
int bs=0, unsigned long trh=131071): facility(s)
{
p=pp;bftyp=btyp;bfcnt=bcnt;rs=r;sblank=sblank;gemnum=gnum;
tlastsuc=tl;thru=th;owngem=gm;usrtyp=urty;bs=bs;trhis=trh;
}
station();
stationO();
void setval(double pp=1.0/32.0, int btyp=0, int bcnt=0,
int r=0, int sblank=0, int gnum=0, double tl=0.0, double th=0.0,
int ogm=0, int urty=0,
int bs=0, unsigned long trh=131071)
{
p=pp;bftyp=btyp;bfcnt=bcnt;rs=r;sblank=sblank;gemnum=gnum;
tlastsuc=tl;thru=th;owngem=gm;usrtyp=urty;bs=bs;trhis=trh;
}
void dsp();
int getrs(){return rs;}
double gettl(){return tlastsuc;}
double getp(){if (usrtyp=0) return p; else return (2.0/(cumin*int(pow(2.0,double(bs))))); /* return bs;*/
void observer(int);
void transmit(int in);
int gem();

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
9.3. SOURCE CODE FOR COMPATIBILITY OF 802.11 WITH BINARY EXPONENTIAL BACK-OFF AND OUR ALGORITHM FOR STATIC EXPERIMENT

```c
void idle();

/*******************************************************************************/
int lmin(int a, int b)
{
    if (a < b) return a;
    else return b;
}

int lmax(int a, int b)
{
    if (a < b) return b;
    else return a;
}

double mmin(double a, double b)
{
    if (a < b) return a;
    else return b;
}

double mmax(double a, double b)
{
    if (a < b) return b;
    else return a;
}

int len(char* ptr)
{
    int i = 0;
    do
    { i++;
    } while (*(++ptr) != '\0');
    return i;
}

double calcsstar(double s)
{
    double ss, ssstar;
    ss = 0.5; ssstar = 0.0;
    while (abs(ssstar - ss) > 0.0001)
    { ss = ssstar;
      ssstar = 1.0 + exp(-ss) * (1.0/usc - 1.0);
    } return (1.0 + ssstar);
}
```
Bibliography


