中国研究生创新实践系列大赛

“华为杯”第二十届中国研究生数学建模竞赛

学　　校		江苏科技大学
参赛队号		**23102890028**
队员姓名	**1.**	丁凯星
	2.	刘文正
	3.	路航

中国研究生创新实践系列大赛
“华为杯”第二十届中国研究生数学建模竞赛

题　　目　同信道多 BSS 系统吞吐计算问题

摘　　要：

无线技术在局域网环境中的重要性日益凸显，它提供了低成本、高吞吐和便捷的无线通信服务。然而，随着人们生活水平的提高和联网设备数量的增加，局域网中信道冲突的情形也随之增多。特别是在教学区等场所，由于存在同频多 BSS 的情况，冲突更为频繁。虽然我们可以依靠 CSMA/CA 机制来避免冲突，但是在冲突情况下，特别是存在同频干扰和隐藏节点情况下，对信道吞吐量的估计研究仍然不够理想。本文针对同频多 BSS 场景下，对存在隐藏节点与非理想信道情况下的信道吞吐估计预测问题进行了建模研究。

问题一：针对理想信道下 2BSS 互听仅下行系统中存在同频干扰时吞吐性能进行评估。本题需计算该系统中 AP 发送消息的概率 τ 与消息发生碰撞的概率 p。针对该概率未知的情况，我们首先通过信道状态分析建立了 CSMA/CA 机制下的 Markov 状态转移模型，通过分析该 Markov 链的稳态解得到了该模型下任意状态出现的概率表达式；此后根据题意参照 Bianchi 模型解得 $\tau = p \approx 0.1046$。针对吞吐计算公式未知的情况，我们参照题意根据全概率公式推导得到了以 τ 与 p 表示的吞吐计算公式，并以此**估计此系统下吞吐 S 约为 67.174Mbps。**最后建立了仿真机模型，通过蒙特卡洛方法进行了独立仿真，得到仿真下的系统吞吐 $S = 65.263$ Mbps，仿真结果区间为 **[64.182,66.199](Mbps)。**仿真模拟结果与数值方法计算结果十分接近，验证了数值方法计算结果的准确性。

问题二：针对理想信道下 2BSS 互听仅下行系统中不存在同频干扰时的吞吐性能进行评估。本题同问题一一致，仍需计算该系统中 AP 发送消息的概率 τ 与消息发生碰撞的概率 p。由于此时信道中不存在同频干扰，我们针对该概率未知的情况，首先通过信道状态分析**建立了 CSMA/CA 机制下的改进后的 Markov 状态转移模型**，通过分析该 Markov 链的稳态解得到了该模型下任意状态出现的概率表达式；此后根据题意参照 Bianchi 模型解得 $\tau = p = \frac{2}{17}$。针对吞吐计算公式未知的情况，我们参照题意根据概率论知识推导得到了以 τ 与 p 表示的吞吐计算公式，并以此**估计此系统下吞吐 S 约为 70.558Mbps。**最后通过蒙特卡洛方法进行了独立仿真，得到仿真下的系统吞吐 $S = 68.95$ Mbps，仿真结果区间为 **[68.251,69.432](Mbps)。**仿真模拟结果与数值方法计算结果十分接近，验证了数值方法计算结果的准确性。

问题三：针对非理想信道下 2BSS 不互听仅下行系统中存在同频干扰时的吞吐性能进行评估。根据题目要求，此时信道并非理想信道且出现了隐藏节点问题。针对非理想信道问题，我们将**首先建立理想信道下的信道吞吐评价模型并以仿真机验证**，之后**再考虑模型在 10% 丢包率的非理想信道情景中的吞吐性能变化**，并依旧以仿真机进行验证。针对隐藏节点问题，我们参照参考文献将消息碰撞的概率分为互听节点发生碰撞的概率 p_1 与隐藏节点发生碰撞的概率 p_2。其后与上两问相同，仍需计算该系统中 AP 发送消息的概率 τ 与消息发生碰撞的概率 p_1 与 p_2。我们针对该概率未知的情况，首先通过信道状态分析**建立了 CSMA/CA 机制下的 Markov 状态转移模型**，通过分析该 Markov 链的稳态解得到了该模型下任意状态出现的概率表达式；此后根据题意参照参考文献中的改进后的 Bianchi 模型求解。针对吞吐计算公式未知的情况，我们参照题意根据概率论知识推导得到了以 τ 与 p 表示的吞吐计算公式，并以此**估计此系统在理想信道下吞吐 S 约为 60.191Mbps。之**后通过蒙特卡洛方法进行了多次独立仿真，得到仿真下的系统吞吐 $S = 63.077$ Mbps，仿真结果区间为 [60.718,65.163](Mbps)。仿真模拟结果与数值方法计算结果十分接近，验证了数值方法计算结果的准确性。

针对非理想信道下的模型信息传输成功率未知问题，通过考虑丢包率 P_e 改进了系统吞吐计算公式，得到存在 10% 丢包情况改进下的系统吞吐计算公式，计算得出**非理想信道下该系统吞吐 S 约为 52.37Mbps**，之后通过蒙特卡洛方法进行了独立仿真，得到**仿真下的系统吞吐** $S = 55.22$ Mbps，仿真结果区间为 **[53.38,57.58](Mbps)。**仿真模拟结果与数值方法计算结果十分接近。之后利用附件 6 中参数对模型估计与仿真验证结果进行对比，**结果贴合程度也十分理想**。

问题四：针对理想信道下 3BSS 仅下行系统的吞吐性能评估。根据题目要求，此时节点中同时存在互听与不互听，有同频干扰和无同频干扰的情况。在问题四的场景中，AP1、AP3 与 AP2 的互听节点数量以及隐藏节点数量是不一致的，故无法简单地使用问题三所得方程进行参数的求解。问题四首先根据各节点关系确定 AP1、AP2、AP3 发送的数据包在系统中占比分别为 $x_1 = \frac{1}{4}$, $x_2 = \frac{1}{2}$, $x_3 = \frac{1}{4}$，并据此确定系统整体的**等价互听节点数量** $n_c = \frac{2}{3}$ 与**等价隐藏节点数量** $n_H = \frac{7}{3}$。在吞吐计算公式推导时引入**等效传输乘数** k，并利**用附件 4 参数进行 1000 次蒙特卡洛模拟的结果，对** k **值进行简单的估计**，取 $k = 1.76855$ 时，代入该 k 值计算系统吞吐 S 的结果为 116.54，与蒙特卡洛模拟求解的结果基本吻合。使用该值代入系统吞吐 S 的表达式进行附件 4 与附件 6 参数的系统性能估计，结果显示，在 CW_{min} 为 16 的参数下，根据附件 4 参数估计的等效传输速率 k 进行系统性能估计的结果处在可接受的区间内，而在 CW_{min} 为 32 的参数下的系统性能估计则出现了较大的误差，说明初始窗口大小 CW_{min} 对于等效传输速率 k 的取值有较大的影响，在今后的工作中将对 CW_{min} 对于等效传输速率 k 的取值影响做进一步的研究，以取得更好的估计精度。

关键词：Bianchi 模型 Markov 链 蒙特卡洛模拟 隐藏节点 全概率公式

目录

1 问题重述与分析 6

1.1 问题背景 6

1.2 问题重述 7

1.2.1 问题一 7

1.2.2 问题二 7

1.2.3 问题三 7

1.2.4 问题四 8

1.3 问题分析 8

1.3.1 问题一 8

1.3.2 问题二 9

1.3.3 问题三 9

1.3.4 问题四 11

2 模型假设与符号说明 11

2.1 模型假设 11

2.2 符号说明 11

3 问题一模型建立与求解 12

3.1 模型建立 12

3.1.1 信道状态分析 12

3.1.2 Markov 状态转移模型 13

3.1.3 吞吐计算公式推导 15

3.2 问题求解 15

3.2.1 Bianchi 模型参数求解 15

3.2.2 数值方法计算系统吞吐 16

3.2.3 仿真算法设计与蒙特卡洛模拟求解 16

3.3 小结 18

4 问题二模型建立与求解 19

4.1 模型建立 19

4.1.1 信道状态分析 19

4.1.2 改进的 Bianchi 状态转移模型 19

4.1.3 吞吐计算公式推导 21
4.2 问题求解 21
4.2.1 数值方法计算系统吞吐 21
4.2.2 仿真算法设计与蒙特卡洛模拟求解 21
4.3 小结 23
5 问题三模型建立与求解 **23**
5.1 理想信道模型建立 24
5.1.1 信道状态分析 24
5.1.2 改进的 Bianchi 模型 25
5.1.3 吞吐量计算公式推导 27
5.2 理想信道下模型求解 27
5.2.1 数值方法计算系统吞吐 27
5.2.2 仿真算法设计与蒙特卡洛模拟求解 27
5.3 非理想信道下模型改进与求解 29
5.3.1 改进数值计算方法并求解 29
5.3.2 改进仿真算法并进行蒙特卡洛模拟求解 30
5.4 小结 31
6 问题四模型建立与求解 **32**
6.1 模型建立 33
6.1.1 信道状态分析 33
6.1.2 改进的 Bianchi 模型 34
6.1.3 吞吐计算公式推导 35
6.2 问题求解 35
6.2.1 仿真算法设计与蒙特卡洛模拟求解 35
6.2.2 数值方法计算系统吞吐 37
6.3 小结 37
参考文献 **39**
附录 A Python 源程序 **40**
1.1 工具函数（问题一至四共用，含关键数值求解过程） 40
1.2 问题一数值分析方法求解 46
1.3 问题一仿真算法 47

1.4 问题一蒙特卡洛求解 49
1.5 问题二数值分析方法求解 51
1.6 问题二仿真算法 52
1.7 问题二蒙特卡洛求解 54
1.8 问题三理想信道下数值分析方法求解 56
1.9 问题三理想信道下仿真算法 57
1.10 问题三理想信道下蒙特卡洛求解 60
1.11 问题三数值分析方法求解 63
1.12 问题三仿真算法 64
1.13 问题三蒙特卡洛求解 68
1.14 问题四数值分析方法求解 72
1.15 问题四仿真算法 73
1.16 问题四蒙特卡洛求解 79
附录 B 蒙特卡洛模拟结果 85
2.1 问题一蒙特卡洛模拟结果 85
2.2 问题二蒙特卡洛模拟结果 85
2.3 第三问理想信道下蒙特卡洛模拟结果 85
2.4 问题三非理想信道蒙特卡洛模拟结果（含 7 种参数） 86
2.5 问题四蒙特卡洛模拟结果（含 7 种参数） 87

1 问题重述与分析

1.1 问题背景

在 Wi-Fi 技术被广泛应用的当下，随着实际生活中设备数量、应用类型、网络流量的飞速增长，无线接入点部署日趋高密，由于可用信道数有限，不同的基本服务集（BSS, basic service set）复用同一个信道，而同频节点之间通信区域存在重叠时，存在相互干扰问题。即不同 BSS 下的节点在互听或者不互听时均可能会受到来自其他同信道 BSS 下节点所发送信息的干扰。如何避免 Wi-Fi 信号下数据传递的信道冲突成为了该技术下一大研究课题。分布式协调功能 （DCF, distributed coordination function）正是基于此需求被提出的一种分布式、基于竞争的信道接入功能。其主要分为信道可用评估（CCA，clear channel assessment）、随机回退、数据传输这三个阶段。对于单 BSS 下的 DCF 常用基于 Markov 链的 Bianchi 模型 [1] 进行系统性能分析。由于其精确度很高，Chatzimisios [1] 在其基础上进行了最大重传次数限制的媒体接入控制（MAC，medium access control） 层性能情况的研究；Huang 和 Ivan Marsic [2] 在其上进行了隐藏节点下网络模型和性能分析的研究；Chen [3] 在其上分析了多速率 MAC 协议的性能。而上述复数 BSS 下的干扰情景在本题中被简化为以下三种模式（2BSS 互听仅下行、2BSS 不互听与 3BSS）：

（1）**2BSS 互听仅下行：**两 AP 分别向各自关联的 STA 发送数据，如图 1-1 所示。以 AP1→STA1 方向的数据传输为例，其会受到相邻 BSS2 的干扰，即 AP2 → STA2 方向的数据传输在进行时 AP1 会认为信道忙而触发等待。

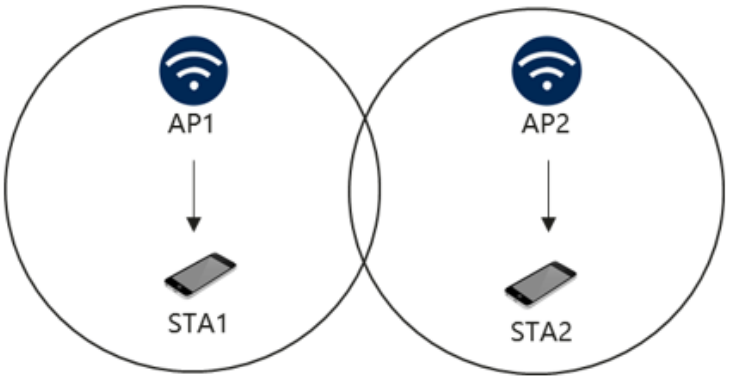

图 1-1　两同频 BSS 场景

（2）**2BSS 不互听：**AP 间 RSSI 低于 CCA 门限，也即隐藏节点问题。此时同 BSS 下数据包为信号包，否则为干扰包。信号前部分用于信号识别等功能的数据叫作前导 (Preamble)，如图 1-2 所示。当信号包先到时，接收机先解信号包的 Preamble 并锁定；当干扰包先到时，接收机先锁定到干扰包的 Preamble，由于小信号屏蔽算法的使用，在信号包到达时转为锁

定信号包。而在 SIR 比较小的情况下，以上两种情况均会传输失败。

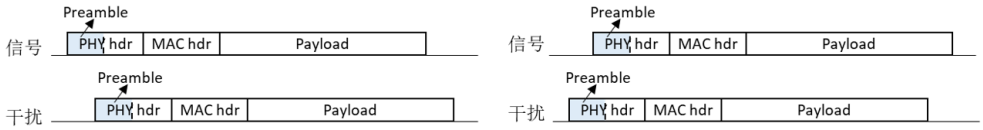

图 1-2　并发传输交叠示意图

（3）**3BSS：**考虑 3BSS 场景，该场景中，AP1 与 AP3 不互听，AP2 与两者都互听，即 AP2 的发送机会被 AP1 和 AP3 挤占。AP1 与 AP3 由于不互听可能同时或先后发送数据。假设 AP1 和 AP3 发包时间交叠时，SIR 较大，两者发送均成功。

1.2 问题重述

如上小节中提到的场景所描述的，在 AP 密集部署的情况下，会出现互听或者不互听的节点，存在同频干扰和不存在同频干扰的节点共用同一信道的情况。在这些情况并存的局域网中对信道吞吐做出评价与估计是比较困难的问题。本题正是基于这一系列场景，在不考虑环境噪声的前提下给出了以下四个待解决的网络模型下的性能分析问题。

1.2.1 问题一

假设 AP 发送包的载荷长度为 1500Bytes，MAC 头为 30Bytes，均采用物理层速率 455.8Mbps 发送，PHY 头时长固定为 13.6μs。AP 之间能互听，数据传输一定成功。当两个 AP 同时回退到 0 而同时发送数据时，存在同频干扰，即并发时因 SIR 较低，导致两个 AP 的数据传输都失败。请对该 2 BSS 系统进行建模，用数值分析方法求解，评估系统的吞吐。

1.2.2 问题二

假设两个 AP 采用物理层速率 275.3Mbps 发送数据，不存在同频干扰，即并发时因 SIR 较高，两个 AP 的数据传输都能成功。其他条件同问题 1。请对该 2 BSS 系统进行建模，用数值分析方法求解，评估系统的吞吐。

1.2.3 问题三

假设 AP 间不互听，除信道条件外其他条件同问题 1。在 Bianchi 模型中假设信道为理想信道，实际上经过实测，当仅有一个 AP 发送数据时，即便不存在邻 BSS 干扰，也会有 10% 以内不同程度的丢包。因此假设信道质量导致的丢包率 Pe = 10%。并发时存在同频干扰，即因 SIR 比较小，会导致两个 AP 的发包均失败。请对该 2 BSS 系统进行建模，尽量用数值分析方法求解，评估系统的吞吐。

1.2.4 问题四

考虑 3BSS 场景，其中 AP1 与 AP2 之间，AP2 与 AP3 之间 RSSI 均互听，AP1 与 AP3 之间不互听。可以预见的是，AP2 的发送机会被 AP1 和 AP3 挤占。AP1 与 AP3 由于不互听可能同时或先后发送数据。其他条件同问题 1。假设 AP1 和 AP3 间并发是不存在同频干扰，即因 SIR 较大，两者发送均成功。请对该 3BSS 系统进行建模，尽量用数值分析方法求解，评估系统的吞吐。

1.3 问题分析

1.3.1 问题一

问题一的难点在于针对 2BSS 互听仅下行系统中存在同频干扰时的吞吐性能进行评估。问题一中模拟的是理想情况下仅存在下行方向数据流动的信道，因此因信道质量引起的丢包问题可以暂时忽略，仅需关注双 AP 同信道互听时的双 BSS 系统吞吐。针对本双 BSS 系统，双 AP 同时发送数据时会因 SIR 较低而导致数据传输失败，必须计算出此碰撞发生的概率才能得到信道的利用率来对系统的吞吐做出评价。根据题意，此随机过程符合 Markov 过程。故本问将根据 Bianchi 模型计算出 Markov 链下的稳态解来对任一 AP 发送信息的概率与发送信息时产生碰撞的概率进行计算，据此建立本问设定下系统吞吐的评价模型，并建立仿真器对所建模型的精度进行验证。问题一流程如图 1-3 所示。

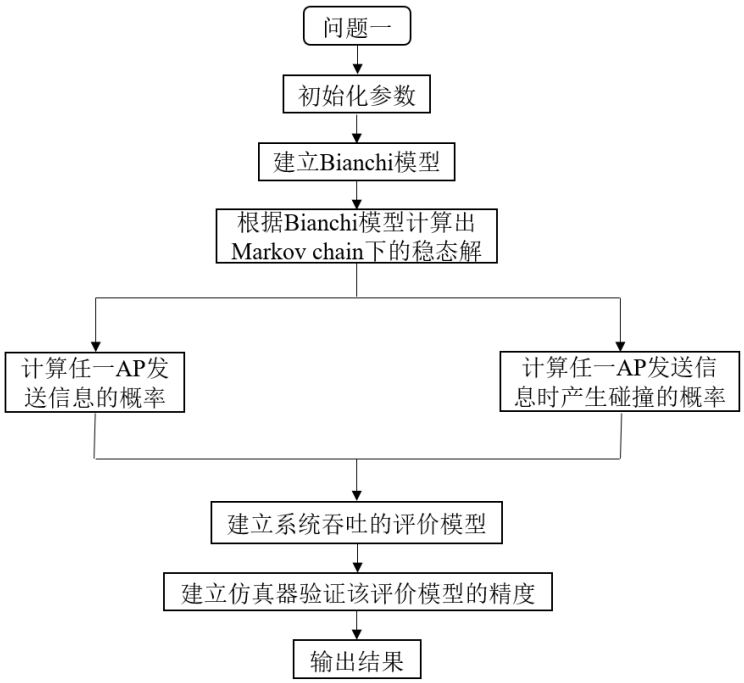

图 1-3　第一问流程图

1.3.2 问题二

问题二的难点在于针对 2BSS 互听仅下行系统中不存在同频干扰时的吞吐性能进行评估。与问题一相比，问题二同样是对理想情况下仅存在下行方向数据流动的信道中双 AP 同信道互听时的双 BSS 系统吞吐。但本问中双 AP 同时发送数据时因为 SIR 较高而使得两方信息均能发送成功（即不存在同频干扰），**所以本问将针对不存在碰撞的情况对第一问中所建立的 Bianchi 模型进行改进**，并在此改进后模型的基础上重新计算出 Markov 链下的稳态解来对任一 AP 发送信息的概率与发送信息时产生碰撞的概率进行计算。以此计算结果建立本问设定下系统吞吐的评价模型，并最终建立仿真器对所建模型的精度进行验证。问题二流程如图 1-4 所示。

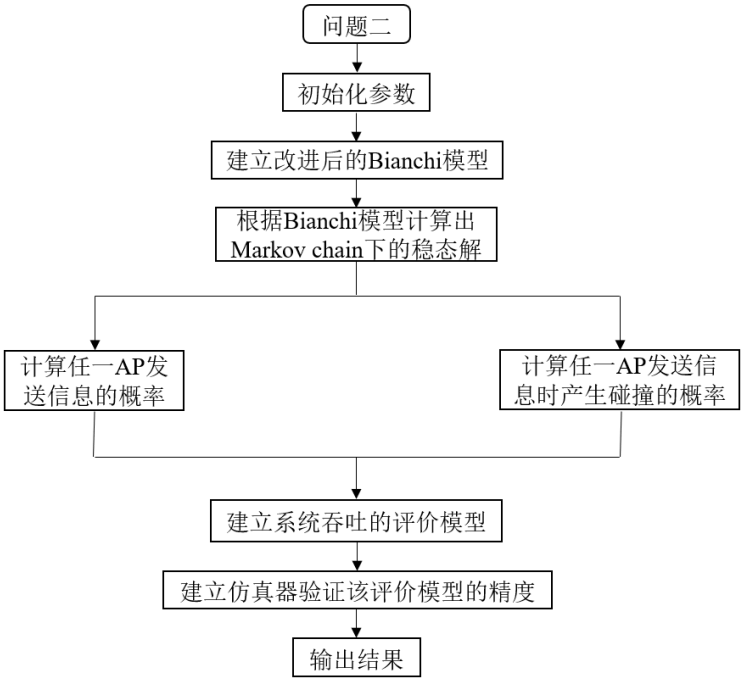

图 1-4　第二问流程图

1.3.3 问题三

问题三的难点在于针对 2BSS 不互听仅下行系统中存在同频干扰时的吞吐性能进行评估。相较于问题一和二的理想信道假设，问题三针对的是非理想信道（丢包率为 10%）。且此时双 AP 虽不互听，但发送数据包时间上存在交叠时因 SIR 较小，将导致双方发包均失败（即隐藏节点问题）。所以本问将**首先在假设理想信道状态下针对隐藏节点**送信碰撞概率问题在 Bianchi 模型上重新计算**覆盖站在随机选择的时隙中发生冲突的平稳概率**以及**隐藏站在易受攻击期间发生冲突的平稳概率**，并以此计算出理想信道下的此 2BSS 系统的发信概率与碰撞概率，最终建立仿真器模型来验证模型精度；之后再考虑丢包率对模型的影响，即**将非理想信道的因素纳入考量范围对模型进行修正，**最终得出非理想信道下的系统

吞吐评价模型，并建立仿真器来验证模型精度。问题三流程如图 1-5 所示。

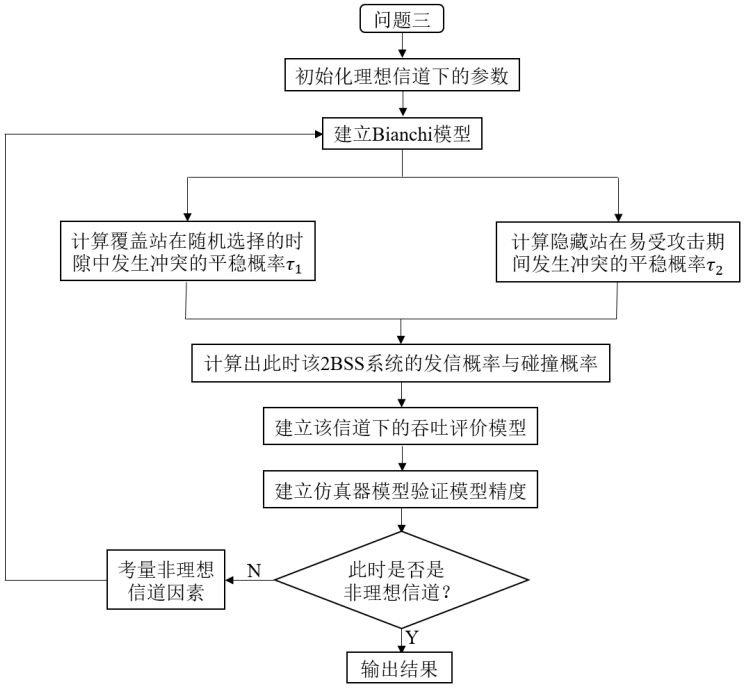

图 1-5　第三问流程图

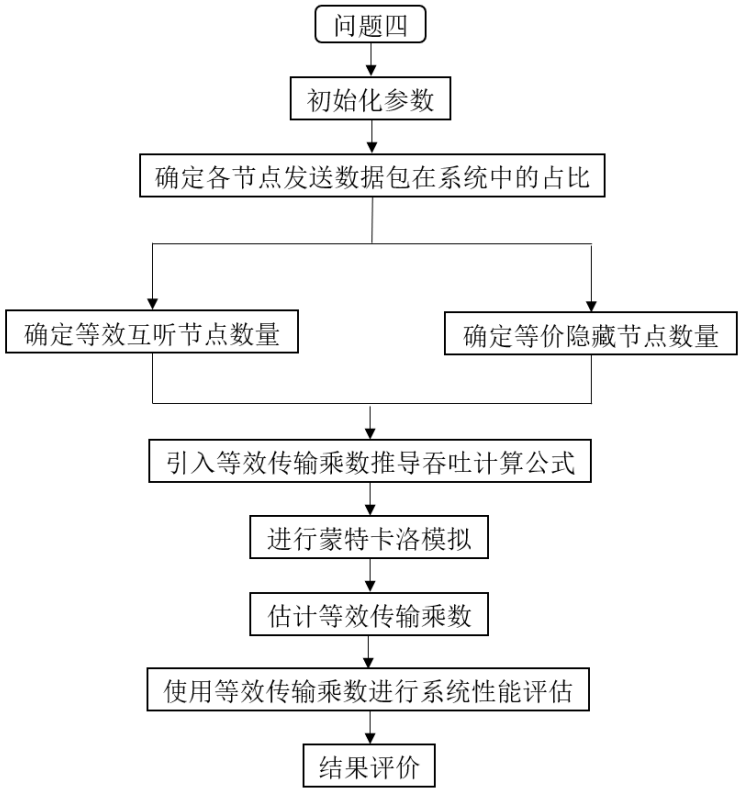

图 1-6　第四问流程图

1.3.4 问题四

问题四的难点在于针对 3BSS 存在互听与不互听仅下行系统中存在同频干扰时的吞吐性能进行评估，且本系统中还存在不会同频干扰的节点组。在问题四的场景中，AP1、AP3 与 AP2 的互听节点数量以及隐藏节点数量是不一致的，故无法简单地使用问题三所得方程进行参数的求解。问题四首先根据各节点关系对**各节点发送数据包在系统中的占比**，确定系统整体的**等价互听节点**与**等价隐藏节点**数量。在吞吐计算公式推导时引入**等效传输乘数**，并通过附件 4 的参数**进行 1000 次蒙特卡洛模拟，对乘数的值进行估计**，并使用估计得到的等效传输乘数在附件 6 的参数下进行系统性能的估计，对结果进行评价，并提出了今后工作的方向。问题四流程如图 1-6 所示。

2 模型假设与符号说明

2.1 模型假设

1、任意一个节点都是假设有数据包的，不存在节点竞争到信道之后，没有数据包待发送的情况出现（饱和信道）。

2、每次成功传输数据的事件独立，没有记忆性。

3、场景中的站点数目固定，每个信道所处位置的信号强度不发生变化。

4、如果题目中没有提及，则信道被视为理想信道。

2.2 符号说明

符号名	说明
T_e	空闲时隙
T_s	成功传输时隙
T_c	碰撞时隙
$DIFS$	DCF 帧间距
CW	竞争窗口
BO	退避计数器读数
$b_{i,j}$	节点处于 Markov 过程第 i 阶第 j 步的概率
$a_{i,j}$	节点处于 Markov 过程第 i 阶第 j 步的状态

符号名	说明
slotTime	空闲时隙时长
PHY 头	消息包物理层头
MAC 头	消息包媒体控制头
τ	节点在任一时刻发送消息的概率
p	节点在任一时刻发送消息时产生碰撞的概率
P_{tr}	只有一个节点在给定时隙内至少发生一次传输的概率
P_s	某一次传输成功的概率
p_1	互听节点与其他节点发送消息时产生碰撞的概率
p_2	隐藏节点与其他节点发送消息时产生碰撞的概率
VP	可能因碰撞产生发包失败的事件的长度
V	以空闲时隙为单位，脆弱周期的长度
n_C	互听节点的数量
n_H	隐藏节点的数量
$T_{c\_Covered}$	隐藏节点存在下，信道实际碰撞时隙时长
k	等效传输乘数

3 问题一模型建立与求解

由于本问中只考虑下行方向的数据传输的情况，故无需建立完整的 Bianchi 模型。因为在下行传输中，只有 AP 会发送数据，即不考虑上行传输的参数。故此处将建立基于 Bianchi 模型的简化 DCF 机制模型来对系统吞吐进行计算，并建立仿真器以确认模型精度。

3.1 模型建立

3.1.1 信道状态分析

根据题意，此时信道为理想信道（不因信道质量产生丢包）。丢包只因两 AP 同时发送数据，由于碰撞而产生。故信道占用状态可以分为三种：空闲时隙、成功传输时隙与碰撞

时隙，如图 3-7 所示。此时将其均视为一个虚拟时隙，则信道状态将在此三种时隙中变化。

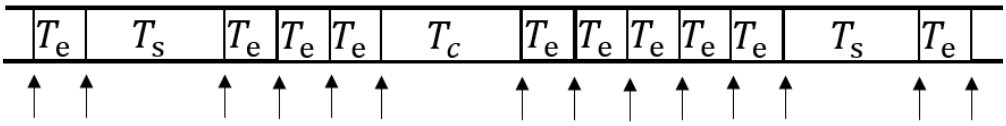

图 3-7　第一问信道状态示意

其中空闲时隙 T_e = slotTime，成功传输时隙 $T_s = H + E[P] + SIFS + ACK + DIFS$，碰撞时隙 $T_c = H + E^*[P] + DIFS + ACKTimeout$。

特别地，触发 T_c 时的随机回退机制如图 3-8 所示（图中不考虑触发碰撞上限的情况）：初始阶竞争窗口大小 CW_{min} 与每个节点的随机回退读数如均图所示。在初始的信道繁忙与 DIFS 之后，Station a 与 Station b 均开始随机回退，且都处于 CW_0，此时 Station a 取得回退数为 7，Station b 取得回退数为 12。在回退 7 个空闲时隙之后，Station a 进入信息传递阶段，Station b 检测到信道繁忙，将回退暂停，进入 hold 装填，信道进入成功传输时隙。Station a 信息传递完毕之后与 Station b 同时等待 DIFS 时间段后同时再次进入回退倒数阶段。假设此次 Station a 选择的回退数为 5，与此时的 Station b 剩余回退数相同。则此时 Station a 与 Station b 同时倒数到 0，同时发送信息，信道进入碰撞时隙。碰撞时隙过后 Station a 与 Station b 均将 CW 翻倍，之后重新随机选择回退数重复上述过程直到有一方信息成功传递之后重置自身的 CW 为 CW_0。

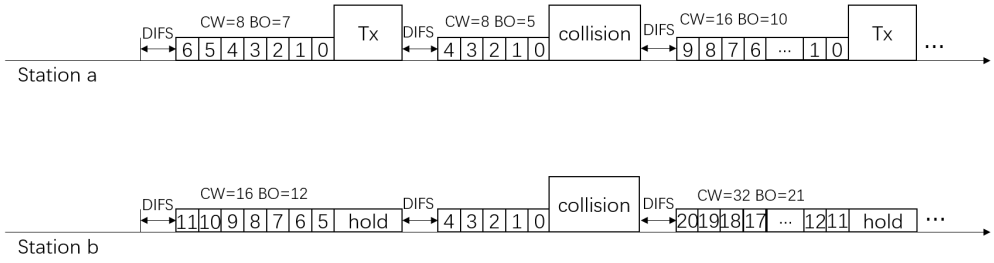

图 3-8　第一问二进制指数退及碰撞过程示意

3.1.2 Markov 状态转移模型

随机过程是一种数学概念，用于描述一系列相关联的随机事件的演变过程，如一支股票的价格波动等。在随机过程中，每个事件的结果都是随机的，而且可能受到前面事件的影响。这种过程在许多领域都有应用，包括物理学、化学、生物学、经济学和工程学。随机过程可以分为许多不同的类型，包括马尔可夫过程、高斯过程、泊松过程和维纳过程等。

Markov 过程是一种特殊的随机过程，其中每个事件的结果只依赖于前一个事件的结果，也即推导未来的某个状态只需要知道当前的状态。这很契合本问题中的随机回退机制，

并且 Bianchi 模型也是基于马尔可夫过程来建立的，故本题也使用 Markov 过程和 Markov 链来对该网络系统进行建模分析。

根据 Bianchi 模型有：τ 为某一个时隙发送数据帧的概率，p 为任意时刻发生碰撞的概率。则此时决定 CW 与 BO 的 Markov 状态转移模型如图 3-9 所示，根据 Bianchi 模型，用 $b(t)$ 与 $s(t)$ 分别代表 t 时刻一个节点的退避计数与阶数，设 $b_{i,k}=\lim_{t\to\infty}P\{s(t)=i,b(t)=k\}$, $a_{i,k}$ 表示此时 Markov 状态为 $\{s(t)=i,b(t)=k\}$ 。图中每个圆圈表示当前状态的 CW 与 BO（退避计数与阶数）每条箭头上的数字都表示在本一个状态下按箭头方向转移到下一个状态的概率（每一个点分出去的所有箭头的概率之和均为 1）。以第一层为例，A 点开始发送信息任务。则此时 A 点需要决定回退 k 步（$k\in[0,W_0-1]$），由于 k 取值为均匀离散分布，所以此时 A 将以 $\frac{1}{W_0}$ 的概率等可能的进入到 $a_{0,k}$ 中的任意一个状态，而当 $k\neq 0$ 时，状态将在经历一个 slotTime 后以 1 的概率进入到 $a_{0,k-1}$ 的状态。当 $k=0$ 时，状态 $a_{0,0}$ 在 slotTime 后将以 $(1-p)$ 的概率将消息发送出去，以 p 的概率进入下一层。

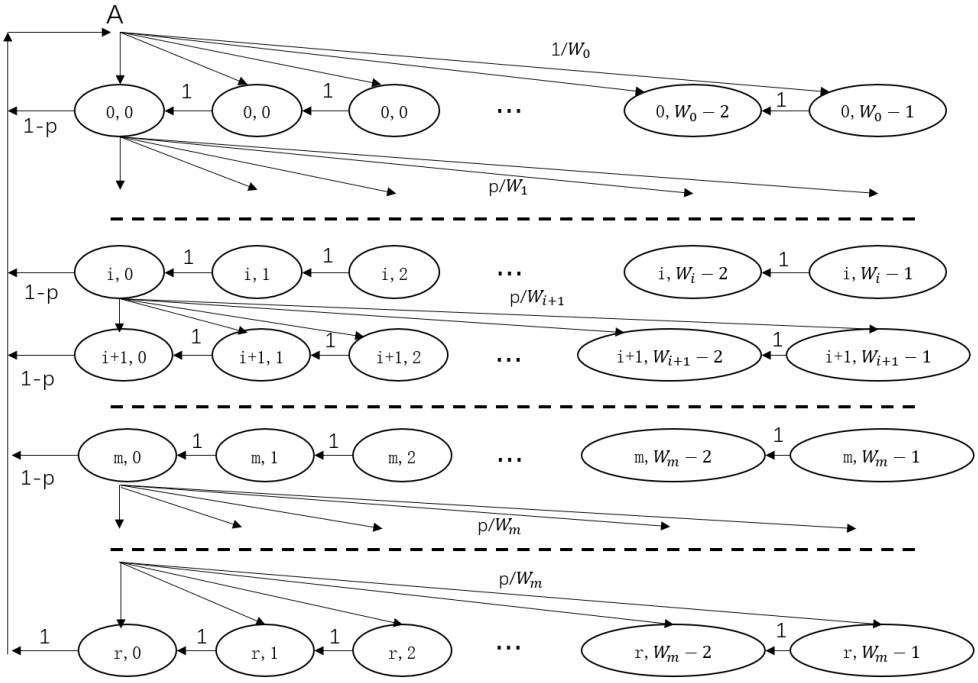

图 3-9　第一问马尔科夫状态转移示意图

综上，可得到进入任意状态的概率计算公式

$$b_{i,0}=p^i\cdot b_{0,0}\quad 0<i\le r$$

$$b_{i,k}=\begin{cases}b_{i-1,0}\cdot p\cdot\frac{w_{i-k}}{w_i} & 0<i<r\\ (1-p)\cdot\frac{w_{i-k}}{w_i}\cdot\sum_{j=0}^{r}b_{j,0} & i=0\end{cases}\tag{3-1}$$

3.1.3 吞吐计算公式推导

按照题意，可得信道吞吐 S（单位 bps）计算公式如下：

$$S = \frac{E[\text{一个时隙内传输的有效载荷发送时长}]}{E[\text{一个时隙长度}]} * v \tag{3-2}$$

其中 v 为信息传递的物理层速率。

因此，吞吐可以用一个时隙长度内传输的有效载荷信息所需的时间除以一个时隙时间的平均长度来表示，另根据 Bianchi 模型，将 P_{tr} 定义为 BSS 系统中只有一个 AP 在给定时隙内至少发生一次传输的概率。假设有 n 个 AP 竞争进行传输，且每个 AP 进行传输的概率均为 τ，则根据逆事件概率公式与独立事件概率公式可得

$$P_{tr} = 1-(1-\tau)^n \tag{3-3}$$

特别地，此处将某一次传输成功的概率定义为 P_s。由定义可知，若某一次传输成功，则情景必为：在给定时隙内，某一 AP 进行传输而其余 n-1 个站点均保持静默，则根据古典概型与独立事件概率公式可得

$$P_s = \frac{n \cdot \tau \cdot (1-\tau)^{n-1}}{1-(1-\tau)^n} \tag{3-4}$$

又根据定义与上式可得

$$E[\text{一个时隙内传输的有效载荷发送时长}] = \frac{P_{tr} \cdot P_s \cdot E[P]}{v} \tag{3-5}$$

综上，根据贝叶斯公式与文献 [4]，可得吞吐计算公式为

$$S = \frac{P_{tr} \cdot P_s \cdot E[P]}{(1-P_{tr}) \cdot T_e + P_{tr} \cdot P_s \cdot T_s + P_{tr} \cdot (1-P_s) \cdot T_c} \cdot v \tag{3-6}$$

3.2 问题求解

3.2.1 Bianchi 模型参数求解

用 i 表示一个数据的发送次数，在问题一中，CW 可用下式表示

$$W_i = \begin{cases} 2^{i+4} & 0 \le i \le 6 \\ 2^{10} & 6 < i \le 32 \end{cases} \tag{3-7}$$

根据 Bianchi 模型，有

$$b_{i,k} = \frac{w_{i-k}}{w_i} \cdot b_{i,0} \quad 0 \le i \le r,\ 0 \le k \le W_i - 1 \tag{3-8}$$

又由于所有稳态的概率之和为 1，因此有

$$\sum_{k=0}^{W_i-1} \sum_{i=0}^{r} b_{i,k} = \sum_{i=0}^{r} b_{i,0} \cdot \sum_{k=0}^{W_i-1} \frac{w_{i-k}}{w_i} = \sum_{i=0}^{r} b_{i,0} \frac{W_i+1}{2} = 1 \tag{3-9}$$

根据式 (3.7) 与 (3.9)，可以求得

$$b_{0,0}=\begin{cases}\dfrac{2(1-p)(1-2p)}{(1-2p)(1-p^{r+1})+W_0(1-p)(1-(2p)^{r+1})}, & r\le m\\ \dfrac{2(1-p)(1-2p)}{W_0(1-p)(1-(2p)^{m+1})+(1-2p)(1-p^{r+1})+W_0 2^m p^{m+1}(1-p^{r-m})(1-2p)}, & m<r\end{cases} \tag{3-10}$$

又由于节点回退到 0 时会发送消息，且传输数据时至少有一个节点在传输数据，假设此时共有 N 个节点，根据上式可得

$$\begin{cases}\tau=\sum_{i=0}^{r} b_{i,0}=b_{0,0}\cdot\frac{1-p^{r+1}}{1-p}\\ p=1-(1-\tau)^{N-1}\end{cases} \tag{3-11}$$

联立求解，解得 $\tau=p\approx 0.1046$。

3.2.2 数值方法计算系统吞吐

问题一 AP 发送包的载荷长度为 1500Bytes，PHY 头时长为 $13.6\mu s$，MAC 头为 30Bytes，MAC 头和有效载荷采用物理层速率 $v=455.8$Mbps 发送，于是数据帧头传输时长 $H=13.6+\frac{30\times 8}{455.8}\approx 14.1265\mu s$，数据帧的有效载荷传输时长 $E[P]=E^*[P]=\frac{1500\times 8}{455.8}\approx 26.3273\mu s$。因此成功传输和碰撞的传输时长分别为

$$T_s=H+E[P]+SIFS+ACK+DIFS\approx 131.45\mu s \tag{3-12}$$

$$T_c=H+E^*[P]+DIFS+ACKTimeout\approx 148.45\mu s \tag{3-13}$$

根据小节 3.2.3 得到的吞吐计算方法，首先根据 τ 和 p，计算得到 $P_{tr}\approx 0.198$，$P_s\approx 0.945$。于是系统吞吐的解析解为

$$S=\frac{P_{tr}\cdot P_s\cdot E[P]}{(1-P_{tr})\cdot T_e+P_{tr}\cdot P_s\cdot T_s+P_{tr}\cdot(1-P_s)\cdot T_c}\cdot v\approx 67.174\text{ Mbps} \tag{3-14}$$

3.2.3 仿真算法设计与蒙特卡洛模拟求解

为了验证上文数值计算模型的正确性，本文针对问题一场景设计了一个仿真算法。算法的关键思路是准确模拟 CSMA/CA 协议的退避和发送过程，从而统计系统的吞吐。本文实现了该仿真算法，算法伪代码如算法 1 所示。仿真过程初始化参数包括数据帧头传输时长、数据帧的有效载荷传输时长、时隙时间等，并设置竞争窗口的初始值和最大值，以及最大重传次数。然后进入循环体，模拟时间的推进，两个节点退避计数器的减小过程。当退避计数器减小到 0 时，根据信道状态，节点发送数据包或者发生碰撞。发送成功后更新统计变量，重新初始化相关参数。该仿真算法通过准确地模拟整个发送过程，能够有效验证所建立的数学模型和数值分析得到的结果。

算法 1 两 BSS 并发传输失败仿真算法

输入: $T_e, T_s, \mathrm{CW}_{\min}, \mathrm{CW}_{\max}, r$, DIFS, 迭代层数

输出: 模拟得到的系统吞吐

$t \leftarrow$ DIFS
$\mathrm{CW}_1, \mathrm{CW}_2 \leftarrow \mathrm{CW}_{\min}$
根据 $\mathrm{CW}_1, \mathrm{CW}_2$ 随机生成 BO_1, BO_2
总传输比特数, $c \leftarrow 0$
for i 从 1 到 迭代层数 **do**
    $BO \leftarrow \min(BO_1, BO_2)$
    $BO_1 \leftarrow BO_1 - BO$
    $BO_2 \leftarrow BO_2 - BO$
    $t \leftarrow t + BO * T_e$
    if $BO_1 = 0$ 且 $BO_2 = 0$ **then**
        $t \leftarrow t + T_c$
        $\mathrm{CW}_1, \mathrm{CW}_2 \leftarrow 2 * \mathrm{CW}_1, 2 * \mathrm{CW}_2$
        根据 $\mathrm{CW}_{\max}$ 限制 $\mathrm{CW}_1, \mathrm{CW}_2$ 的最大值
        根据 $\mathrm{CW}_1, \mathrm{CW}_2$ 随机生成新的 BO_1, BO_2
        $c \leftarrow c + 1$
        if $c = r$ **then**
            $\mathrm{CW}_1, \mathrm{CW}_2 \leftarrow \mathrm{CW}_{\min}$
            根据 $\mathrm{CW}_1, \mathrm{CW}_2$ 随机生成新的 BO_1, BO_2
            $c \leftarrow 0$
    else if $BO_1 = 0$ **then**
        $t \leftarrow t + t_s$
        总传输比特数 $\leftarrow$ 总传输比特数 $+ 1500 * 8$
        $\mathrm{CW}_1 \leftarrow \mathrm{CW}_{\min}$, 根据 CW_1 随机生成 BO_1
        $c \leftarrow 0$
    else if $BO_2 = 0$ **then**
        同上
吞吐 $\leftarrow$ 总传输比特数$/t$
返回结果

根据仿真算法，取迭代层数为 1000，得到吞吐随时间的变化的示意图如图 3-10 所示。从图中可以看出随着时间的增加，系统的吞吐逐渐平稳。在这一次模拟中，吞吐稳定在了 67 至 68 Mbps 之间。

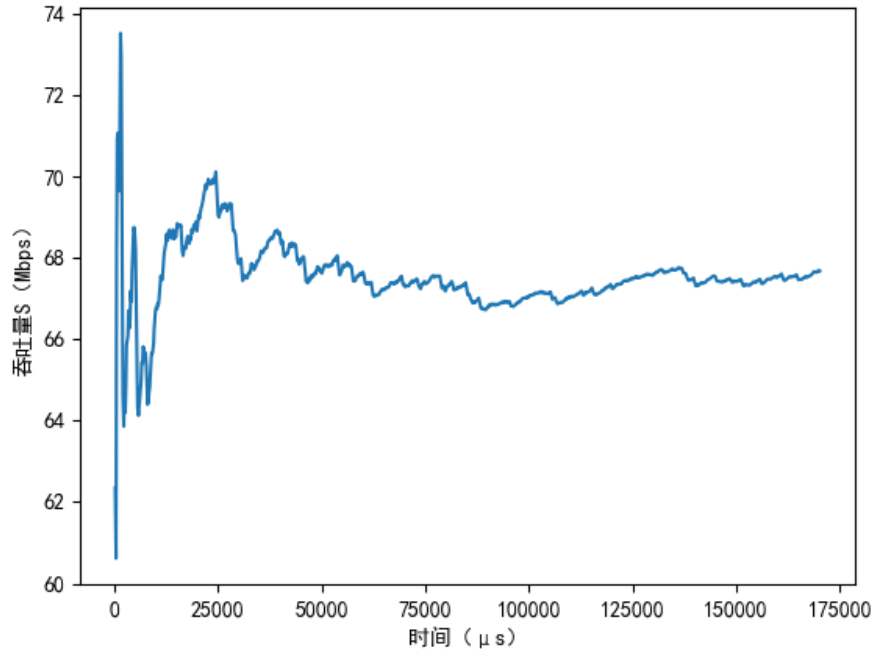

图 3-10　问题一仿真算法吞吐随时间的变化

由于仿真算法的随机性，一次仿真的结果无法完全反映系统的长期稳态性能。本文采用蒙特卡罗方法进行多次独立仿真，以提高结果的可靠性。具体来说，设置每次仿真的迭代层数设为 10000，并重复进行 1000 次仿真实验，作为最终的仿真结果。1000 次仿真实验分别模拟得到的系统吞吐保存在附件里 problem1_result.txt 文件中。经过蒙特卡罗仿真得到的系统吞吐结果区间为 [64.182,66.199]，结果均值为 65.249，与数值方法计算结果十分接近，验证了数值方法计算结果的准确性。

3.3 小结

本章通过构建 Bianchi 模型对理想信道下 2BSS 互听仅下行系统中存在同频干扰时进行了建模，并通过数值分析方法**估计该系统吞吐为 67.174Mbps**。此后通过蒙特卡罗方法对此系统进行了迭代层数为 10000，重复 1000 次的**仿真实验，得到系统吞吐结果区间为 [64.182,66.199]，结果均值为 65.249 Mbps**，与本章所建 Bianchi 模型求解结果相近，可以认为模型效果良好。

4 问题二模型建立与求解

本问仍然只考虑下行方向的数据传输的情况，且大部分情况与第一问相同，故不再对于吞吐量公式等进行完整推导，只针对第二问建立**改进后的 Markov 状态转移模型**（**本问中不会因碰撞导致信息发送失败，故 Markov 状态转移模型只有第一层**）与**改进后的 Bianchi 模型**来对吞吐进行估计，并建立仿真器以确认模型精度。

4.1 模型建立

4.1.1 信道状态分析

本问中信道情况与第一问中相似，但本问中不存在碰撞状态，只存在繁忙等待状态，故信道占用状态可分为两种：空闲时隙与成功传输时隙，如图 4-11 所示。

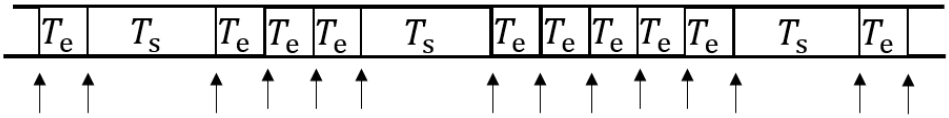

图 4-11　第二问信道状态示意

其中空闲时隙 T_e = slotTime，成功传输时隙 T_s = H+E[P]+SIFS+ACK+DIFS 。而碰撞不会影响信息传递，**所以不会产生碰撞时隙** T_c **。**

此时的随机回退过程如图 4-12 所示（不考虑触发碰撞上限的情况）：初始阶竞争窗口大小 CW_{min} 与每个节点的随机回退读数如均图所示。在初始的信道繁忙与 DIFS 之后，Station a 与 Station b 均开始随机回退，且都处于 CW_0，此时 Station a 取得回退数为 7，Station b 取得回退数为 12。在回退 7 个空闲时隙之后，Station a 进入信息传递阶段，Station b 检测到信道繁忙，将回退暂停，信道进入成功传输时隙。Station a 信息传递完毕之后与 Station b 同时等待 DIFS 时间段后同时再次进入回退倒数阶段。假设此次 Station a 选择的回退数为 5，与此时的 Station b 剩余回退数相同。则此时 Station a 与 Station b 同时倒数到 0，同时发送信息，由于此时 SIR 较高，两方同时发送成功，**故不会进入碰撞时隙**，也不会引起 Station a 与 Station b 的 CW 翻倍，之后重新随机选择回退数重复上述过程且每次传递信息后均重置自身的 CW 为 CW_0。

4.1.2 改进的 Bianchi 状态转移模型

问题二并发时两个终端接收到数据的 SIR 较高，两个 AP 的数据传输都能成功。令 $b(t)$ 和 $s(t)$ 代表 t 时刻一个节点退避随机过程的退避计数和退避阶数。在问题二的情况下，$s(t) \equiv 0$，随机过程 $\{b(t), s(t)\}$ 可以退化为用一维 Markkov 链表示，如图 4-13 所示。$b_{0,k} = \lim_{t\to\infty} P\{b(t) = k\}$ 代表 Markov 链的稳态解，$k \in [0, W_0 - 1]$。Markov 链中每一步状态

转移概率为

$$\begin{cases} P\{0,k \mid 0,k+1\}=1 \\ P\{0,k \mid 0,0\}=1/W_0 \end{cases} \tag{4-15}$$

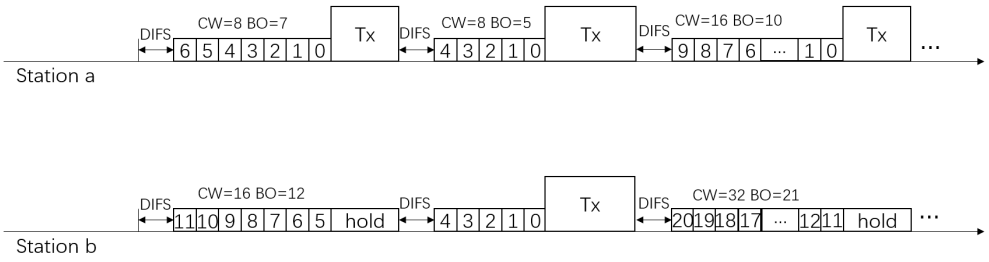

图 4-12　第二问二进制指数退避过程示意

该 Markov chain 的任意状态之间可达，是不可约的。任意状态到另一状态的步长不存在周期。以任何状态出发，都能到达零一状态，具有常返性。因此该二进制退避过程的非周期不可约 Markov chain 具有稳态解，且所有稳态的概率之和为 1。

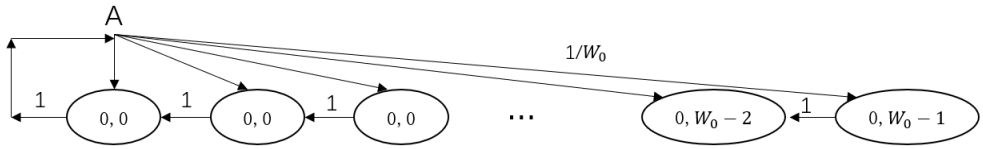

图 4-13　第二问马尔科夫状态转移示意图

对于状态 a_{0,W_0-1}，只能从状态 $a_{0,0}$ 转移而来，因此

$$b_{0,W_0-1}=b_{0,0}*\frac{1}{W_0} \tag{4-16}$$

对于状态 a_{0,W_0-2}，可以从状态 $a_{0,0}$ 转移而来，也可以从 a_{0,W_0-1} 转移而来，于是

$$b_{0,W_0-1}=b_{0,0}*\frac{2}{W_0} \tag{4-17}$$

推广可得，对于 $b_{0,k},\ 0\le k\le W_0-1$，有

$$b_{0,k}=b_{0,0}*\frac{W_0-k}{W_0} \tag{4-18}$$

根据 Markov 链的性质，所有稳态的概率之和为 1，因此有

$$1=\sum_{k=0}^{W_0-1}b_{0,k}=\sum_{k=0}^{W_0-1}b_{0,0}*\frac{W_0-k}{W_0}=\frac{1}{2}b_{0,0}(W_0+1) \tag{4-19}$$

节点随机回退到 0 时发送数据，于是 $\tau=b_{0,0}=\frac{2}{W_0+1}$。传输数据发生冲突时，至少有一个节点也传输数据，共有 2 个节点，因此碰撞概率 p 可表示为

$$p=1-(1-\tau)^{2-1}=\tau=\frac{2}{W_0+1} \tag{4-20}$$

4.1.3 吞吐计算公式推导

从上文信道状态中分析可得知，本问所使用吞吐计算公式中，一个时隙内传输的有效载荷发生变化：产生碰撞时会双方均会传输成功。所以根据定义可得

$$E[\text{一个时隙内传输的有效载荷发送时长}] = \frac{P_{tr} \cdot P_s \cdot E[P] + 2 \cdot P_{tr} \cdot (1 - P_s) \cdot E[P]}{v} \quad (4\text{-}21)$$

综上可得吞吐计算公式

$$S = \frac{P_{tr} \cdot P_s \cdot E[P] + 2 \cdot P_{tr} \cdot (1 - P_s) \cdot E[P]}{(1 - P_{tr}) \cdot T_e + P_{tr} \cdot P_s \cdot T_s + P_{tr} \cdot (1 - P_s) \cdot T_c} \cdot v \quad (4\text{-}22)$$

4.2 问题求解

4.2.1 数值方法计算系统吞吐

问题二 AP 发送包的载荷长度为 1500Bytes，PHY 头时长为 $13.6\mu s$，MAC 头为 30Bytes，MAC 头和有效载荷采用物理层速率 $v = 275.3$Mbps 发送，于是数据帧头传输时长 $H = 13.6 + \frac{30\times8}{275.3} \approx 14.47\mu s$，数据帧的有效载荷传输时长 $E[P] = E^*[P] = \frac{1500\times8}{275.3} \approx 43.59\mu s$。因此成功传输传输时长为

$$T_s = H + E[P] + SIFS + ACK + DIFS \approx 149.06\mu s \quad (4\text{-}23)$$

问题二所取 $W_0 = \text{CW}_{\min} = 16$，因此 $\tau = p = \frac{2}{W_0+1} = \frac{2}{17}$。根据 τ 和 p，计算得到 $P_{tr} \approx 0.22$，$P_s \approx 0.94$。于是得到系统吞吐的解析解为

$$S = \frac{P_{tr} \cdot P_s \cdot E[P] + P_{tr} \cdot (1 - P_s) \cdot 2 \cdot E^*[P]}{(1 - P_{tr}) \cdot T_e + P_{tr} \cdot T_s} \cdot v \approx 70.558 \text{ Mbps} \quad (4\text{-}24)$$

4.2.2 仿真算法设计与蒙特卡洛模拟求解

为了验证上文两 BSS 并发成功传输的数值计算模型的正确性，对于问题二本文同样设计了仿真算法并进行蒙特卡洛模拟。算法的关键思路是准确模拟 CSMA/CA 协议的退避和发送过程，区别在于模拟两个节点退避计数器同时减小到 0 时，认为都是成功发送。本文实现了该仿真算法，关键代码如算法 2 所示。仿真过程初始化参数包括数据帧头传输时长、数据帧有效载荷传输时长、时隙时间等，并设置竞争窗口初始值。然后进入循环，模拟时间推进，两个节点退避计数器减小过程。当两个退避计数器都为 0 时，统计为两次成功发送；当单个退避计数器为 0 时，模拟一次成功发送。发送后更新统计变量，重新初始化相关参数。该算法通过准确地模拟整个发送过程，能够有效验证所建立的数学模型和数值分析结果。

算法 2 两 BSS 并发传输成功仿真算法

输入: T_e, T_s, CWmin, CWmax, r, DIFS, 迭代层数

输出: 模拟得到的系统吞吐

$t \leftarrow$ DIFS
$\text{CW}_1, \text{CW2} \leftarrow$ CWmin
根据 CW_1, CW_2 随机生成 BO_1, BO_2
总传输比特数, $c \leftarrow 0$
for i 从 1 到迭代层数 **do**
    $BO \leftarrow \min(BO_1, BO_2)$
    $BO_1 \leftarrow BO_1 - BO$
    $BO_2 \leftarrow BO_2 - BO$
    $t \leftarrow t + BO * T_e$
    if $BO_1 = 0$ **and** $BO_2 = 0$ **then**
        $t \leftarrow t + T_s$
        根据 CW_1, CW_2 随机生成新的 BO_1, BO_2
        总传输比特数 ← 总传输比特数 + 1500 * 8 * 2
    else if $BO_1 = 0$ **then**
        $t \leftarrow t + T_s$
        总传输比特数 ← 总传输比特数 + 1500 * 8
        根据 CW_1 随机生成新的 BO_1
    else if $BO_2 = 0$ **then**
        同上
吞吐 ← 总传输比特数 / t
返回结果

根据仿真算法，取迭代层数为 1000，得到吞吐随时间的变化的示意图如图 4-14 所示。从图中可以看出随着时间的增加，系统的吞吐逐渐平稳。在这一次模拟中，吞吐稳定在了 70Mbps 左右。

由于仿真算法的随机性，一次仿真的结果无法完全反映系统的长期稳态性能。本文采用蒙特卡罗方法进行多次独立仿真，以提高结果的可靠性。具体来说，设置每次仿真的迭代层数设为 10000，并重复进行 1000 次仿真实验，作为最终的仿真结果。1000 次仿真实验分别模拟得到的系统吞吐保存在附件里 problem2_result.txt 文件中。经过蒙特卡罗仿真得到的系统吞吐结果区间为 [68.251,69.432]，结果均值为 68.95，与数值方法计算结果十分接近，验证了数值方法计算结果的准确性。

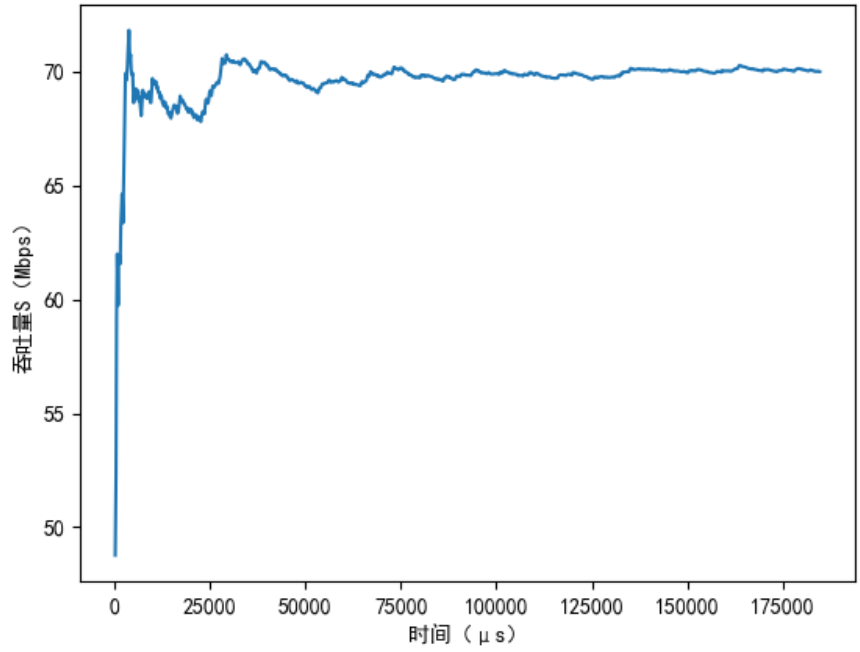

图 4-14　问题二仿真算法吞吐随时间的变化

4.3 小结

本章通过改进 Bianchi 模型对理想信道下 2BSS 互听仅下行系统中不存在同频干扰时进行了建模，并通过数值分析方法**估计该系统吞吐为 70.558Mbps**。此后通过蒙特卡罗方法对此系统进行了迭代层数为 10000，重复 1000 次的**仿真实验，得到系统吞吐结果区间为 [68.251,69.432]，结果均值为 68.95 Mbps**，与本章所建 Bianchi 模型求解结果相近，可以认为模型效果良好。

5 问题三模型建立与求解

本问仍然只考虑下行方向的数据传送，两个 AP 分别向目标 STA 发送信息但会受到对方 AP 的信息干扰，即若两 AP 发包存在时间上的重合部分就会导致双方均送信失败，此处将这种在一方发包过程中其他方有可能导致发包失败的时间称为脆弱周期（VP,vulnerable period）。除此之外，本问信道并非理想信道，即存在丢包问题（按照题设，此处丢包率为 10%）。

基于本问中信道非理想信道的考虑，此处将先对理想信道下的模型进行推导并以仿真器验证，确认模型正确之后在此模型基础上考虑非理想信道下的通信系统情况。

本问中信息传输的信道占用情况与前两问区别较大，但 Markov 过程与第一问区别相

同。以下将进行详细分析并进行相关参数推导计算。在每个模型的最后均建立仿真器以确认模型精度。

5.1 理想信道模型建立

5.1.1 信道状态分析

由于此时信道为理想信道，丢包只会因两方 AP 发包时间交叠导致，故信道占用状态可以分为三种：空闲时隙、成功传输时隙与碰撞时隙，如图如图 5-15 所示。

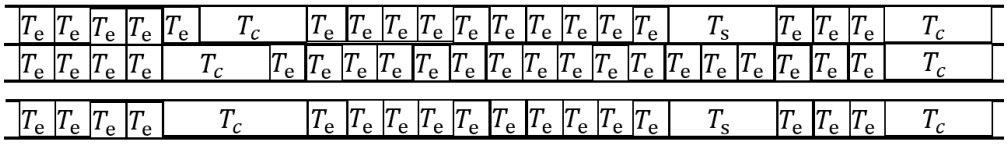

图 5-15　第三问信道状态示意

由于双方 AP 不互听，故此处用两条信道状态条来说明双方的信道占用情况。此时将其均视为一个虚拟时隙，则信道状态将在此三种时隙中变化。其中空闲时隙 T_e = slotTime，成功传输时隙 $T_s = H + E[P] + SIFS + ACK + DIFS$，碰撞时隙 $T_c = H + E^*[P] + DIFS + ACKTimeout$。特别地，从图中可以发现本问信道实际每次 T_c 时间长度并不固定，此处将其期望时长命名为 $T_{c\_Covered}$，将在小节 5.1.2 中对该时长的期望进行计算。

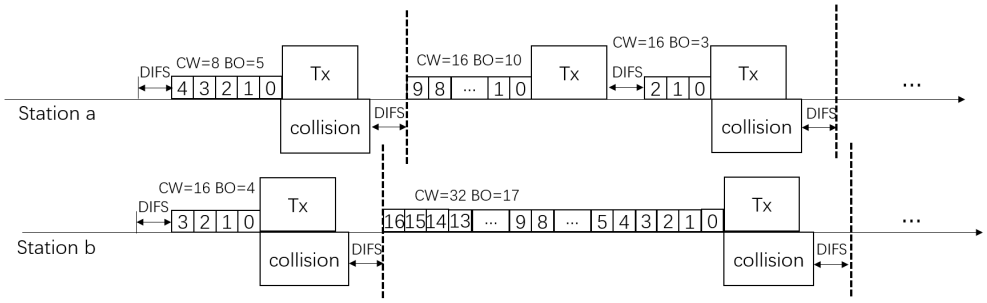

图 5-16　第三问二进制指数退及碰撞过程示意

此外，由于发包时间交叠时发包失败引起碰撞时隙后的随机回退机制如图 5-16 所示（不考虑触发碰撞上限的情况）：初始阶竞争窗口大小 CW_{min} 与每个节点的随机回退读数如均图所示。在初始的信道繁忙与 DIFS 之后，Station a 与 Station b 均开始随机回退，且都处于 CW_0，此时 Station a 取得回退数为 5，Station b 取得回退数为 4。在回退 4 个空闲时隙之后，Station b 进入信息传递阶段。由于 AP 不互听，Station a 无法检测到信道繁忙，将继续回退。在一个空闲时隙之后，Station a 进入信息传递状态。此时 Station a 与 Station b 之间发包时间交叠，双方将发包失败（实质上进入碰撞时隙，双方均不会接收到 ACK 信号）。在双方 AP 分别经历碰撞时隙之后 Station a 与 Station b 均将 CW 翻倍，之后重新随

机选择回退数重复上述过程。如图所假设，Station a 在 CW_1 状态下随机得到回退数为 10，Station b 在 CW_1 状态下随机得到回退数为 17，在此情况下 Station a 倒数 10 个空闲时隙后成功进行了信息传递并将自身 CW 重置为 CW_0，后续将重复以上过程。

5.1.2 改进的 Bianchi 模型

根据参考文献 [2]，此处设互听 AP 在随机选择的时隙中发包并与其他 AP 在发包帧发生碰撞的概率为 p_1，隐藏站在易受攻击期间发生冲突的平稳概率为 $p2$ ，本问中决定 CW 与 BO 的 Markov 状态转移模型如图 5-17 所示，与第一问的情形相同。

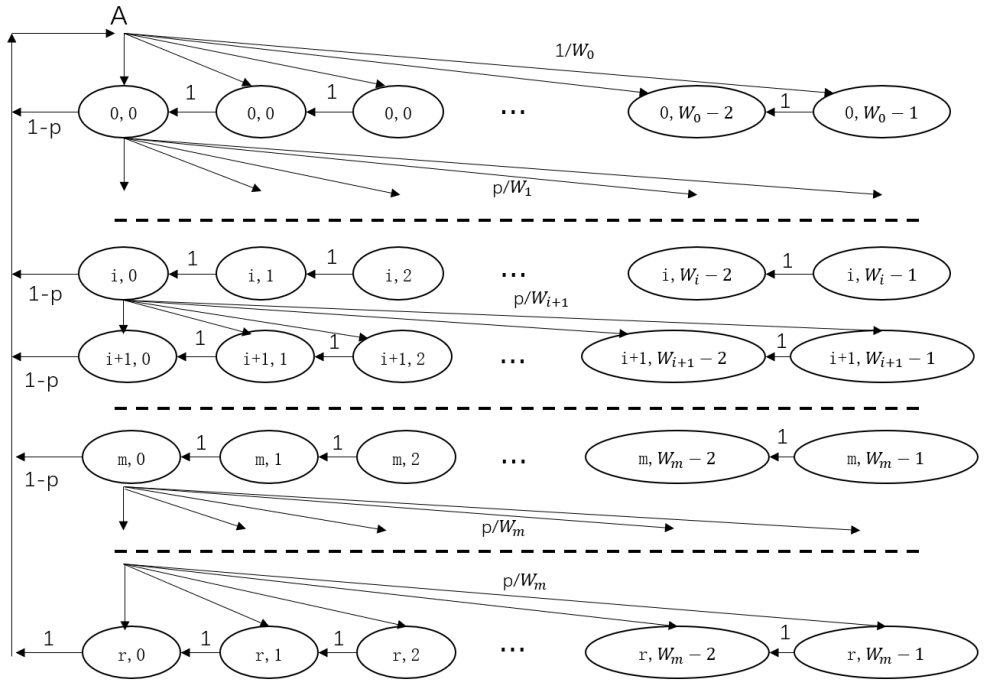

图 5-17　第三问马尔科夫状态转移示意图

同时，概率计算公式也与第一问基本相同，如下所示：

$$b_{i,0} = p^i \cdot b_{0,0} \quad 0 < i \le r b_{i,k} = \begin{cases} b_{i-1,0} \cdot p \cdot \frac{w_{i-k}}{w_i} & 0 < i < r \\ (1-p) \cdot \frac{w_{i-k}}{w_i} \cdot \sum_{j=0}^{r} b_{j,0} & i = 0 \end{cases} \tag{5-25}$$

此外，互听 AP 在随机选择的时隙中发包并与其他 AP 发包发生碰撞的概率 p_1 表达式与前两问相同

$$p_1 = \sum_{i=0}^{r} b_{i,0} = b_{0,0} \cdot \frac{1-p^{r+1}}{1-p} \tag{5-26}$$

针对隐藏站在发包期间发生冲突的概率为 p_2 ，在本题情况中，脆弱周期如图 5-18 所示。

显然，互听节点的脆弱周期为一个 slotTime，而隐藏节点的脆弱周期为信息发送时间（H+E[P]），产生冲突的概率分别在这两段时间上均匀分布。则可得产生冲突的期望分别为

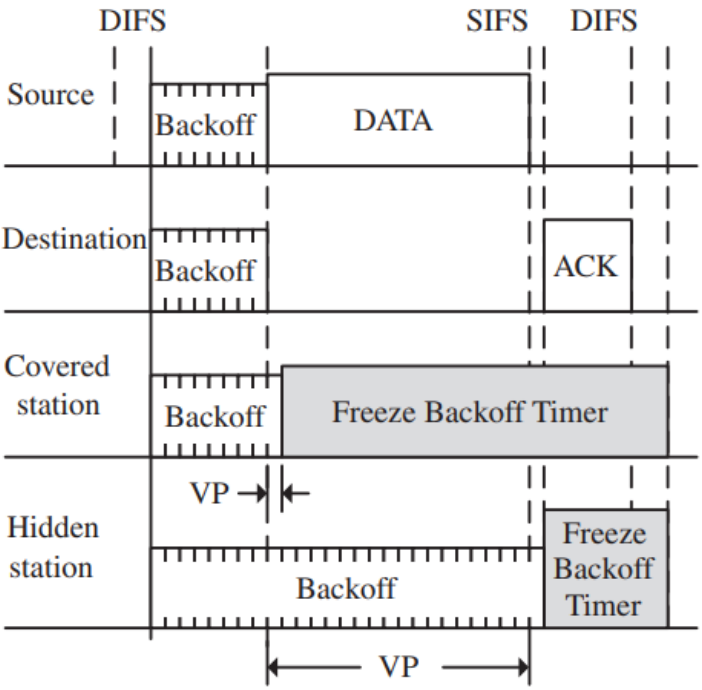

图 5-18　脆弱周期时长示意图

其时长的 $\frac{1}{2}$，也即

$$\begin{cases} VP_{Hidden} = \dfrac{H + E[P]}{2} \\ VP_{Covered} = \dfrac{T_e}{2} \end{cases} \tag{5-27}$$

考虑到在本题中存在一个互听节点（自身）与一个隐藏节点，则可得在本问中 $VP = \frac{1}{2}(VP_{Hidden} + VP_{Covered})$。

另外，根据参考文献 [2] 可知

$$T_{c\_Covered} = \frac{n_C}{n}(\frac{T_e}{2} + H + E[P] + ACK_{\text{Timeout}}) + \frac{n_H}{n}(\frac{H + E[P]}{2} + H + E[P] + ACK_{\text{Timeout}}) \tag{5-28}$$

其中 n_C 为互听节点的数量，n_H 为隐藏节点的数量。又有

$$p_2 = [(V+1) \cdot (\frac{1 - p^{m+1}}{1-p}) - (\frac{V(V+1)}{2W_0}) \cdot (\frac{1 - (\frac{p}{2})^{m+1}}{1 - (\frac{p}{2})})] \cdot b_{0,0},\ V < W_0 \tag{5-29}$$

其中 V 为 VP 与 T_e 的比值再行取整，此处为 1。在一般状态下，根据概率学知识，碰撞概率 p 的概率学意义为：至少有一个互听 AP 与目标 AP 在相同的时隙进行传输，或至少有一个隐藏 AP 在目标 AP 的脆弱周期内传输的概率。因此 p 可以表示为

$$p = 1 - (1 - \tau_1)^{n_C - 1}(1 - \tau_2)^{n_H} \tag{5-30}$$

5.1.3 吞吐量计算公式推导

据第一问和第二问推断，可知吞吐可以用一个时隙长度内传输的有效载荷信息所需的时间除以一个时隙时间的平均长度来表示，而在本问中有

$$E[\text{一个时隙内传输的有效载荷发送时长}] = P_{tr} \cdot P_s \cdot E[P] \tag{5-31}$$

其中 P_{tr} 为任一互听 AP 在给定时隙内至少发生一次传输的概率，P_s 为某一次传输成功的概率。假设有总共 n 个 AP 竞争进行传输，并已知每个 AP 进行传输的概率均为 τ_1，则根据概率论知识可得：

$$P_{tr} = 1 - (1 - \tau_1)^n \tag{5-32}$$

$$P_s = \frac{n\tau_1(1-\tau_1)^{n_C-1}(1-\tau_2)^{n_H}}{P_{tr}} \tag{5-33}$$

综上，根据贝叶斯公式与文献，可得吞吐计算公式为

$$S = \frac{P_{tr} \cdot P_s \cdot E[P]}{(1-P_{tr}) \cdot T_e + P_{tr} \cdot P_s \cdot T_s + P_{tr} \cdot (1-P_s) \cdot T_c} \cdot v \tag{5-34}$$

其中 v 为物理层速率。

5.2 理想信道下模型求解

5.2.1 数值方法计算系统吞吐

在问题三中初始题设 AP 发送包的载荷长度为 1500Bytes，PHY 头时长为 $13.6\mu s$，MAC 头为 30Bytes，MAC 头和有效载荷采用物理层速率 $v = 455.8$Mbps 发送，于是数据帧头传输时长 $H = 13.6 + \frac{30\times8}{455.8} \approx 14.1265\mu s$，数据帧的有效载荷传输时长 $E[P] = E^*[P] = \frac{1500\times8}{455.8} \approx 26.3273\mu s$。因此成功传输和碰撞的传输时长分别为

$$T_s = H + E[P] + SIFS + ACK + DIFS \approx 131.45\mu s \tag{5-35}$$

$$T_c = H + E^*[P] + DIFS + ACKTimeout \approx 148.45\mu s \tag{5-36}$$

根据小节 5.1.2 的方法建立方程组，解得 $p_1 \approx 0.093,\ p_2 \approx 0.181,\ p \approx 0.181$，根据这些参数计算得到 $P_{tr} \approx 0.178,\ P_s \approx 0.859$。于是数值方法计算得到的系统吞吐为

$$S = \frac{P_{tr} \cdot P_s \cdot E[P]}{(1-P_{tr}) \cdot T_e + P_{tr} \cdot P_s \cdot T_s + P_{tr} \cdot (1-P_s) \cdot T_c} \cdot v \approx 60.191 \text{ Mbps} \tag{5-37}$$

5.2.2 仿真算法设计与蒙特卡洛模拟求解

问题三场景下出现了 AP 双方不互听的情况，模拟算法的关键在于重新考虑对方站点为隐藏站情况下的时间隙推进，以及如何判断信号包的冲突。理想信道下问题三的仿真算法如算法 3 所示。

算法 3 问题三理想信道状况下仿真算法

输入:T_e, T_s, CWmin, CWmax, r, DIFS, 迭代层数

输出: 模拟得到的系统吞吐

$t_1, t_2 \leftarrow$ DIFS
$\text{CW}_1, \text{CW2} \leftarrow$ CWmin
根据 CW_1, CW_2 随机生成 BO_1, BO_2
$t_1 \leftarrow t_1 + BO_1 * T_e$
$t_2 \leftarrow t_2 + BO_2 * T_e$
for i 从 1 到迭代层数 **do**
    if $t_1 < t_2$ **then**
        if $t_1 + H + E[P] > t_2$ **then**
            记录节点 1 和 2 信号包冲突
            $\text{CW}_1, \text{CW}_2 \leftarrow 2 * \text{CW}_1, 2 * \text{CW}_2$
            限制 CW_1, CW_2 的最大值
            随机生成新的 BO_1, BO_2
            更新重传计数 c
            if $c == r$ **then**
                重置 $\text{CW}_1, \text{CW}_2, BO_1, BO_2, c$
            $t_1, t_2 \leftarrow t_1 + T_c + BO_1 * T_e,\ t_2 + T_c + BO_2 * T_e$
        else
            节点 1 成功发送, 更新统计变量
            重置 CW_1, BO_1
            $t_1 \leftarrow t_1 + BO_1 * T_e$
    else if $t_2 \leq t_1$ **then**
        同上 (交换节点 1 和 2)
返回结果

根据仿真算法，取迭代层数为 1000，得到吞吐随时间的变化的示意图如图 5-19 所示。从图中可以看出随着时间的增加，系统的吞吐逐渐平稳。在这一次模拟中，吞吐稳定在了 60Mbps 至 61 Mbps 之间。

由于仿真算法的随机性，一次仿真的结果无法完全反映系统的长期稳态性能。本文采用蒙特卡罗方法进行多次独立仿真，以提高结果的可靠性。具体来说，设置每次仿真的迭代层数设为 10000，并重复进行 1000 次仿真实验，作为最终的仿真结果。1000 次仿真实验分别模拟得到的系统吞吐保存在附件里 problem3_result_ideal.txt 文件中。经过蒙特卡罗仿

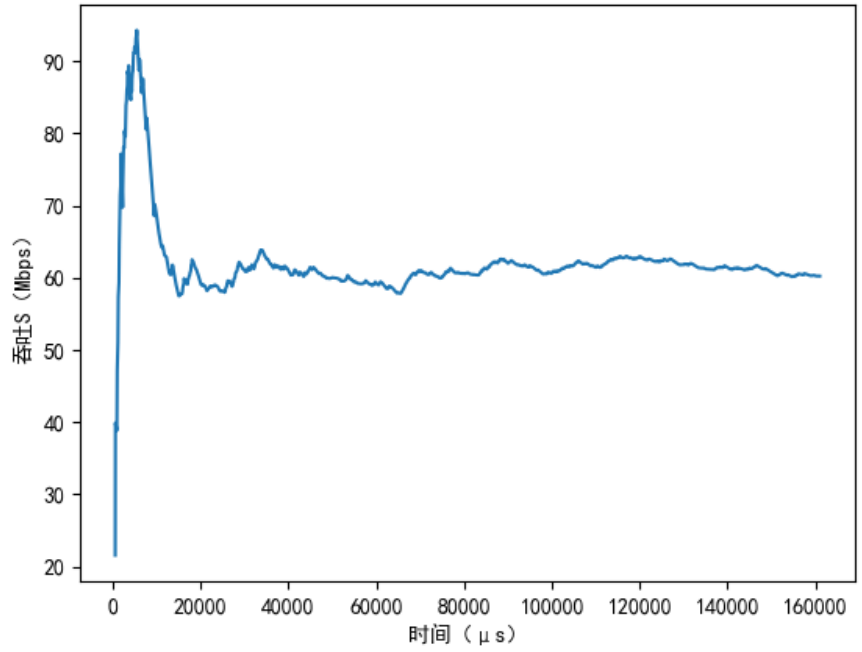

图 5-19　问题三理想信道下仿真算法吞吐随时间的变化

真得到的系统吞吐结果区间为 [60.718,65.163]，结果均值为 63.077，与数值方法计算结果十分接近，验证了数值方法计算结果的准确性。

5.3 非理想信道下模型改进与求解

5.3.1 改进数值计算方法并求解

问题三题设因信道质量导致的丢包率 $P_e = 10\%$，假如考虑丢包的问题，系统的吞吐会比理想信道状态下更低。本小节根据非理想信道条件下固定丢包率 P_e 改进系统吞吐 S 的计算公式，并进行附录 4 和 6 参数下系统吞吐的求解。

根据理想信道下的数值计算，已得 $P_{tr} \approx 0.178,\ P_s \approx 0.859$。在理想信道状态下，信息发送成功的概率为 $P_{tr} \cdot P_s$，考虑丢包率，成功发送的概率即为 $P_{tr} \cdot P_s \cdot (1 - P_e)$，发送失败的概率为 $P_{tr} \cdot P_s \cdot P_e$。于是考虑丢包情况下改进的系统吞吐计算公式如下：

$$S = \frac{P_{tr} \cdot P_s \cdot (1 - P_e) \cdot E[P]}{(1 - P_{tr}) \cdot T_e + P_{tr} \cdot P_s \cdot (1 - P_e) \cdot T_s + P_{tr} \cdot P_s \cdot P_e \cdot T_c + P_{tr} \cdot (1 - P_s) \cdot T_c} \cdot v \quad (5\text{-}38)$$

根据改进后的系统吞吐 S 的计算公式，计算得到附录 4 和 6 参数下系统吞吐的数值求解结果如表 5-1 所示。

表 5-1　问题三系统吞吐的数值求解结果

CW_min	16	16	32	16	16	32	16
CW_max	1024	1024	1024	1024	1024	1024	1024
最大重传次数	32	6	5	32	6	5	32
物理层速率 (Mbps)	455.8	286.8	286.8	286.8	158.4	158.4	158.4
计算结果	52.37	45.65	42.04	45.65	35.18	34.04	35.18

5.3.2 改进仿真算法并进行蒙特卡洛模拟求解

对于仿真算法，考虑丢包的情况，即在原本传输成功的情况下，考虑有丢包率 P_e 的概率会导致模型传输失败，其原本的成功传输需要取消并将成功传输的传输时长 T_s 更改为碰撞的传输时长 T_c。改进后的仿真算法如算法 4 所示。

根据改进后的仿真算法，对附录 4 和 6 参数下系统吞吐进行模拟。并且根据参考文献[3]，使用物理层速率不同的节点基于 CSMA/CA 规则进行组网效果不佳，故此处不再讨论物理层速率速不相同情况下的节点组成系统的吞吐，而是基于相同网速与回退窗口对系统吞吐进行评价。

模拟结果如表 5-2 所示。

表 5-2　问题三系统吞吐的蒙特卡洛模拟求解结果

CW_min	CW_max	最大重传次数	物理层速率 (Mbps)	蒙特卡洛模拟结果区间	蒙特卡洛模拟结果均值
16	1024	32	455.8	[53.38,57.58]	55.22
16	1024	6	268.8	[43.96,47.60]	46.04
32	1024	5	268.8	[38.41,41.26]	39.59
16	1024	32	268.8	[43.88,46.96]	45.53
16	1024	6	158.4	[29.83,32.55]	31.26
32	1024	5	158.4	[26.95,29.04]	28.06
16	1024	32	158.4	[34.86,37.05]	35.98

算法 4 问题三改进后的仿真算法

输入:$T_e, T_s, CW_{min}, CW_{max}, r, P_e, DIFS$, 迭代层数

输出: 模拟得到的系统吞吐

$t_1, t_2 \leftarrow DIFS$

$CW_1, CW_2 \leftarrow CW_{min}$

根据 CW_1, CW_2 随机生成 BO_1, BO_2

$t_1 \leftarrow t_1 + BO_1 * T_e$

$t_2 \leftarrow t_2 + BO_2 * T_e$

初始化统计变量

for i 从 1 到迭代层数 **do**

    if $t_1 < t_2$ **then**

        if $t_1 + H + E[P] > t_2$ **then**

            记录节点 1、2 信号包冲突

            $CW_1, CW_2 \leftarrow 2CW_1, 2CW_2$

            限制 CW_1, CW_2 的最大值

            随机生成新的 BO_1, BO_2

            更新重传计数 c_1, c_2

            if $c_1 == r$ 或 $c_2 == r$ **then**

                重置相关参数

            $t_1, t_2 \leftarrow t_1 + T_c + BO_1T_e, t_2 + T_c + BO_2T_e$

        else

            if 随机概率 $< P_e$ **then**

                更新节点 1 相关统计变量

            else

                节点 1 成功发送, 更新统计变量

                重置 CW_1, BO_1

                $t_1 \leftarrow t_1 + BO_1 * T_e$

    else if $t_2 < t_1$ **then**

        同上 (交换节点 1 和 2)

返回结果

5.4 小结

本章通过改进 Bianchi 模型对非理想信道下 2BSS 不互听仅下行系统中存在同频干扰时（隐藏节点情况）进行了建模，首先对理想状况下的信道吞吐进行了估计并通过仿真验

证，此后再考虑非理想信道情况对模型进行了改进。通过数值分析方法**估计该系统吞吐为 52.37Mbps**。此后通过蒙特卡罗方法对此系统进行了迭代层数为 10000，重复 1000 次的**仿真实验，得到系统吞吐结果区间为 [53.38,57.58]，结果均值为 55.22 Mbps**，与本章所建 Bianchi 模型求解结果相近，可以认为模型效果良好。

对于题干中提供的改变竞争窗口和最大重传次数各种网络系统，本章也进行了数值求解和仿真验证，结果均在可接受范围内，具体结果如表 5-3 所示。

表 5-3　问题三结果汇总

CW_min	CW_max	最大重传次数	物理层速率 (Mbps)	蒙特卡洛模拟结果区间	蒙特卡洛模拟结果均值	数值计算结果
16	1024	32	455.8	[53.38,57.58]	55.22	52.37
16	1024	6	268.8	[43.96,47.60]	46.04	45.65
32	1024	5	268.8	[38.41,41.26]	39.59	42.04
16	1024	32	268.8	[43.88,46.96]	45.53	45.65
16	1024	6	158.4	[29.83,32.55]	31.26	35.18
32	1024	5	158.4	[26.95,29.04]	28.06	34.04
16	1024	32	158.4	[34.86,37.05]	35.98	35.18

6 问题四模型建立与求解

本问仍然只考虑下行方向的理想信道中数据传输的情况，且 AP 的 Markov 状态转移模型与第一、三问相同，故不再对于 Markov 链单独求解稳定解。并且我们注意到，此时 AP1 与 AP3 之间并不存在发信冲突，故此问中并不存在附件 3 中所提及的隐藏节点问题。我们将利用第一问与第二问的结论对第三问中 3BSS 模型的吞吐进行估计，并建立仿真器以确认模型精度。

6.1 模型建立

6.1.1 信道状态分析

根据题意，此时信道为理想信道。丢包只会因为两 AP 中 AP1 与 AP2 或 AP3 与 AP2 同时发送数据，由于碰撞而产生。故信道占用状态可以分为三种：空闲时隙、成功传输时隙与碰撞时隙，如图 6-20 和图 6-21 所示。此时将其均视为一个虚拟时隙，则信道状态将在此三种时隙中变化。

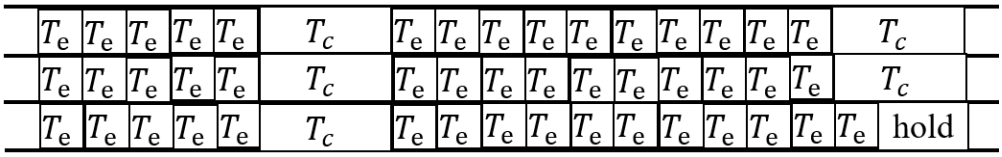

图 6-20　第四问信道状态示意图一

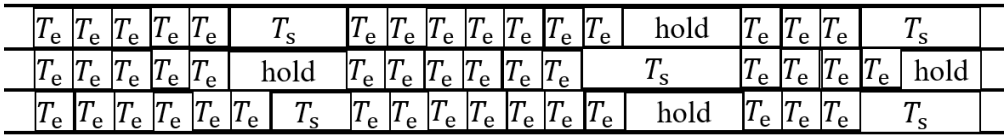

图 6-21　第四问信道状态示意图二

其中空闲时隙 T_e = slotTime，成功传输时隙 $T_s = H + E[P] + SIFS + ACK + DIFS$，碰撞时隙 $T_c = H + E^*[P] + DIFS + ACKTimeout$。特别地，触发 T_c 时的随机回退机制如图 6-22 所示（图中不考虑触发碰撞上限的情况）：初始阶竞争窗口大小 CW_{min} 与每个节点的随机回退读数如均图所示。在初始的信道繁忙与 DIFS 之后，Station a 、Station b 与 Station c 均开始随机回退，且都处于 CW_0 阶。此时三个 Station 取得回退数均为 5。在回退 5 个空闲时隙之后，三个 Station 同时开始信息传递，进入碰撞时隙。过后三个 Station 均将 CW 翻倍。此后 Station a 与 Station b 均取得回退数为 10，Station c 取得回退数 11，在经过 10 个 slotTime 后，信道进入碰撞时隙，而 Station c 将在读完第 11 个 slotTime 之后进入 hold 状态，之后 Station a 与 b 均将自身 CW 翻倍。

另外，随机回退机制另外还有如图 6-23 所示的可能性：初始阶竞争窗口大小 CW_{min} 与每个节点的随机回退读数如均图所示。在初始的信道繁忙与 DIFS 之后，Station a 、Station b 与 Station c 均开始随机回退，且都处于 CW_0 阶。此时 Station a 取得回退数 5，Station b 与 c 取得回退数 6，在 5 个 slotTime 之后，Station a 进入信息发送状态，Station b 侦测到信道繁忙进入 hold 状态，而 Station c 侦测不到信道繁忙，会继续回退。再经过一个 slotTime，Station c 同样进入信息发送状态，此时 Station b 需等待 Station c 的成功传输时隙之后才能继续回退。之后重新随机选择回退数重复上述过程直到有一方信息成功传递之后重置自身的 CW 为 CW_0。

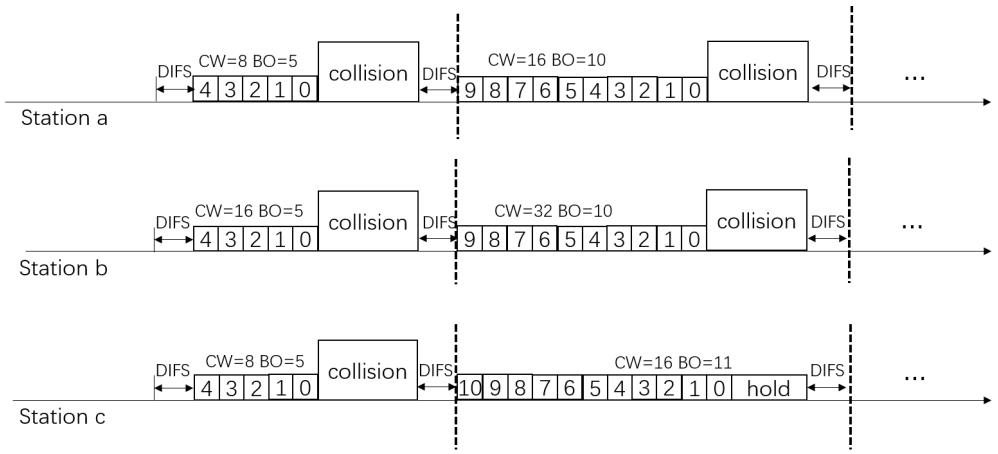

图 6-22　第四问二进制指数退及碰撞过程示意图一

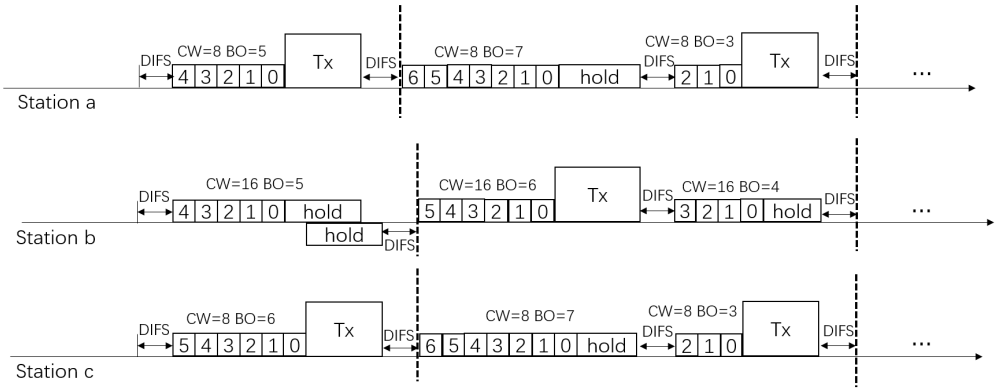

图 6-23　第四问二进制指数退及碰撞过程示意图二

6.1.2 改进的 Bianchi 模型

根据信道状态分析，本章中的 Markov 状态转移模型与第一、三问相同。此处沿用该两问中进入任意状态的概率计算公式如下：

$$
\begin{aligned}
&b_{i,0} = p^i \cdot b_{0,0} \quad 0 < i \leq r \\
&b_{i,k} = \begin{cases} b_{i-1,0} \cdot p \cdot \frac{w_{i-k}}{w_i} & 0 < i < r \\ (1-p) \cdot \frac{w_{i-k}}{w_i} \cdot \sum_{j=0}^{r} b_{j,0} & i = 0 \end{cases}
\end{aligned} \tag{6-39}
$$

在问题四的场景中，AP1、AP3 与 AP2 的互听节点数量以及隐藏节点数量是不一致的，故无法简单地使用问题三所得方程进行参数的求解。记 AP1、AP2、AP3 的互听节点分别为 n_{C1}, n_{C2}, n_{C3}，隐藏节点分别为 n_{H1}, n_{H2}, n_{H3}，则 $n_{C1} = n_{C3} = 2$, $n_{H1} = n_{H_3} = 1$, $n_{C2} = 3$, $n_{H2} = 0$。设 AP1、AP2、AP3 发送的数据包在系统中占比分别为 x_1, x_2, x_3。在本文中，将占比简单地根据如图 6-24 所示的关系取得 $x_1 = x_3 = \frac{1}{4}$, $x_2 = \frac{1}{2}$，在今后的工作中将对这里的占比进行进一步的研究，以使得参数更符合实际并取得更好的结果。于是系统整体的

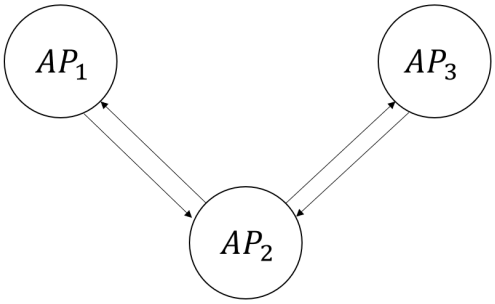

图 6-24　系统数据传输示意图

等价互听节点与**等价隐藏节点数量**分别为

$$\begin{cases} n_C = n_{C1} \cdot x_1 + n_{C2} \cdot x_2 + n_{C3} \cdot x_3 = \dfrac{2}{3} \\ n_H = n_{H1} \cdot x_1 + n_{H2} \cdot x_2 + n_{H3} \cdot x_3 = \dfrac{7}{3} \end{cases} \tag{6-40}$$

6.1.3 吞吐计算公式推导

问题四场景下，AP1 与 AP3 由于相互不互听，可以同时或先后发送数据，且当 AP1 和 AP3 发包时间交叠时，SIR 较大，两者均能发送成功。针对问题四中 3BSS 含隐藏站的复杂问题，引入等效传输乘数，记为 k。引入等效传输乘数 k 的系统吞吐的计算公式为

$$S = \frac{P_{tr} \cdot P_s \cdot k \cdot E[P]}{(1 - P_{tr}) \cdot T_e + P_{tr} \cdot P_s \cdot T_s + P_{tr} \cdot (1 - P_s) \cdot T_c} \cdot v \tag{6-41}$$

在问题四的复杂场景下，等效传输乘数 k 难以直接分析得到，其可能与窗口大小、最大重传次数、物理层速率等参数都有关。本文使用附件 4 参数进行 1000 次蒙特卡洛模拟的结果，对 k 值进行简单的估计，并使用代入该 k 值的系统吞吐 S 的表达式进行附件 6 参数的系统性能估计。在今后的工作中，将对等效传输速率 k 与一系列系统参数进行进一步研究，以得到一个等效传输速率 k 的表达式，能够更好地用于反映系统性能。

6.2 问题求解

6.2.1 仿真算法设计与蒙特卡洛模拟求解

问题四仿真算法比较复杂，算法 5 展现了仿真算法的主要逻辑，更新相关变量的逻辑与算法 1-4 基本相同。具体代码实现见附录与附件代码中的 problem4-simulation.py。

算法 5 问题四三 BSS 仿真算法

输入:$T_e, T_s, CW_{min}, CW_{max}, r, DIFS$, 迭代层数

输出: 模拟得到的系统吞吐

$t_1, t_2, t_3 \leftarrow DIFS$
$CW_1, CW_2, CW_3 \leftarrow CW_{min}$
根据 CW 随机生成 BO
更新 t
初始化统计变量
for i 从 1 到迭代层数 **do**
    if $t_1 == t_2$ **and** $t_2 == t_3$ **then**
        三节点冲突, 更新相关变量
    else if $t_1 == t_2$ **and** $t_2 < t_3$ **then**
        节点 1,2 冲突, 更新相关变量
        if $t_3 < t_2 + T_c$ **then**
            节点 3 与节点 2 对齐
    else if $t_2 == t_3$ **and** $t_2 < t_1$ **then**
        节点 2,3 冲突, 更新相关变量
        if $t_1 < t_2 + T_c$ **then**
            节点 1 与节点 2 对齐
    else if t_2 最小 **then**
        if 节点 1,3 需与节点 2 对齐 **then**
            对齐
        节点 2 发送成功, 更新统计变量
    else if t_1 最小 **then**
        节点 1 发送成功, 更新统计变量
    else if t_3 最小 **then**
        节点 3 发送成功, 更新统计变量
    else if $t_1 == t_3$ **and** $t_3 < t_2$ **then**
        if 节点 2 需与节点 1,3 对齐 **then**
            对齐
        节点 1,3 成功发送, 更新统计变量
返回结果

使用仿真算法，对附录 4 和 6 参数下系统吞吐进行模拟，模拟结果如表 6-4 所示。

表 6-4　问题四系统吞吐的蒙特卡洛模拟求解结果

CW_min	CW_max	最大重传次数	物理层速率 (Mbps)	蒙特卡洛模拟结果区间	蒙特卡洛模拟结果均值
16	1024	32	455.8	[115.46,117.50]	116.54
16	1024	6	268.8	[106.87,108.99]	107.86
32	1024	5	268.8	[86.77,88.27]	87.45
16	1024	32	268.8	[107.10,108.81]	107.87
16	1024	6	158.4	[92.42,93.73]	93.12
32	1024	5	158.4	[74.22,75.31]	74.75
16	1024	32	158.4	[92.40,93.83]	93.14

6.2.2 数值方法计算系统吞吐

利用附件 4 参数进行 1000 次蒙特卡洛模拟的结果，对 k 值进行简单的估计，取 $k = 1.76855$ 时，代入该 k 值计算系统吞吐 S 的结果为 116.54，与蒙特卡洛模拟求解的结果基本吻合。使用该值代入系统吞吐 S 的表达式进行附件 4 与附件 6 参数的系统性能估计，计算结果如表 6-5 所示。

表 6-5　数值计算结果表

CW_min	16	16	32	16	16	32	16
CW_max	1024	1024	1024	1024	1024	1024	1024
最大重传次数	32	6	5	32	6	5	32
物理层速率 (Mbps)	455.8	286.8	286.8	286.8	158.4	158.4	158.4
计算结果	116.54	105.60	98.66	105.60	87.61	83.76	87.61

6.3 小结

针对附件 4 与附件 6 的参数，本章进行了蒙特卡洛模拟求解，并对等效传输速率 k 进行了估计，并代入系统吞吐 S 的表达式进行数值方法的求解，具体结果如表 6-6 所示。

表 6-6　问题四结果汇总

CW_min	CW_max	最大重传次数	物理层速率 (Mbps)	蒙特卡洛模拟结果区间	蒙特卡洛模拟结果均值	数值计算结果
16	1024	32	455.8	[115.46,117.50]	116.54	116.54
16	1024	6	268.8	[106.87,108.99]	107.86	105.60
32	1024	5	268.8	[86.77,88.27]	87.45	98.66
16	1024	32	268.8	[107.10,108.81]	107.87	105.60
16	1024	6	158.4	[92.42,93.73]	93.12	87.61
32	1024	5	158.4	[74.22,75.31]	74.75	83.76
16	1024	32	158.4	[92.40,93.83]	93.14	87.61

结果显示，在 CW_{min} 为 16 的参数下，根据附件 4 参数估计的等效传输速率 k 进行系统性能估计的结果处在可接受的区间内，而在 CW_{min} 为 32 的参数下的系统性能估计则出现了较大的误差，说明初始窗口大小 CW_{min} 对于等效传输速率 k 的取值有较大的影响，在今后的工作中将对 CW_{min} 对于等效传输速率 k 的取值影响做进一步的研究，以取得更好的估计精度。

本章工作不足及今后工作的研究方向如下：

（1）在本文中，将 AP1、AP2、AP3 发送的数据包在系统中占比简单地根据如图 6-24 所示的关系取得 $x_1 = x_3 = \frac{1}{4}$, $x_2 = \frac{1}{2}$，这样的取值有一些武断，在今后的工作中将对这里的占比进行进一步的研究，以使得参数更符合实际并取得更好的结果。

（2）在问题四的复杂场景下，等效传输乘数 k 难以直接分析得到，其可能与窗口大小、最大重传次数、物理层速率等参数都有关。本文使用附件 4 参数进行 1000 次蒙特卡洛模拟的结果，对 k 值进行简单的估计，并使用代入该 k 值的系统吞吐 S 的表达式进行附件 6 参数的系统性能估计。在今后的工作中，将对等效传输速率 k 与一系列系统参数进行进一步研究，以得到一个等效传输速率 k 的表达式，能够更好地用于反映系统性能。

参考文献

[1] Bianchi G, IEEE 802.11-saturation throughput analysis, IEEE communications letters, 2(12): 318-320, 1998.

[2] Hung F Y, Marsic I, Performance analysis of the IEEE 802.11 DCF in the presence of the hidden stations, Computer Networks, 54(15):2674-2687, 2010.

[3] Chen D R, Zhang Y J, Is dynamic backoff effective for multi-rate WLANs?, IEEE communications letters, 11(8):647-649, 2007.

[4] Chatzimisios P, Vitsas V, Boucouvalas A C, Throughput and delay analysis of IEEE 802.11 protocol, Proceedings 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, IEEE, 168-174, 2002.

附录 A　Python 源程序

1.1　工具函数（问题一至四共用，含关键数值求解过程）

utils.py

```
from scipy.optimize import fsolve
import numpy as np


def solve_tau_p(CW_min, m, r, nodeNum, tau_init=0.1,
    p_init=0.1):
"""
使用Bianchi模型求解发送概率τ和冲突概率p。

参数:
CW_min (int): CW窗口大小。
m (int): 最大退避阶数。
r (int): 最大重传次数。
nodeNum (int): 节点数目。
tau_init (float, 可选): τ的初始猜测值。默认为0.1。
p_init (float, 可选): p的初始猜测值。默认为0.1。

返回:
tau (float): 求解得到的τ值。
p (float): 求解得到的p值。
"""

W0 = CW_min
N = nodeNum

def equations(vars):
"""
定义Bianchi模型的两个方程。

参数:
vars (list): 包含两个元素的列表，其中vars[0]对应于τ，
    vars[1]对应于p。

返回:
```

```
        list: 包含两个元素的列表，其中第一个元素是第一个方程的结
            果，第二个元素是第二个方程的结果。
        """
        tau, p = vars

        if r <= m:
        b00 = (2 * (1 - p) * (1 - 2 * p)) / (
        (1 - 2 * p) * (1 - p ** (r + 1)) + (W0 * (1 - p) * (1 -
            (2 * p) ** (r + 1))))
        else:
        b00 = (2 * (1 - p) * (1 - 2 * p)) / (
        (W0 * (1 - (2 * p) ** (m + 1)) * (1 - p)) + ((1 - 2 * p)
            * (1 - p ** (r + 1))) + (
        W0 * (2 ** m) * (p ** (m + 1)) * (1 - p ** (r - m)) * (1
            - 2 * p)))

        eq1 = b00 * ((1 - (p ** (r + 1))) / (1 - p)) - tau
        eq2 = 1 - (1 - tau) ** (N - 1) - p
        return [eq1, eq2]

        # 使用fsolve函数求解方程
        tau, p = fsolve(equations, (tau_init, p_init))

        return tau, p


        def solve_tau_p_problem3(CW_min, m, r, H, EP, nodeNum,
            Ts, Tc, tau1_init=0.1, tau2_init=0.1, p_init=0.1):
        W0 = CW_min
        N = nodeNum
        V = np.around(((0.5 * 9 + 0.5 * (H + EP)) / 2) / 9)

        def equations(vars):
        tau1, tau2, p = vars

        if r <= m:
        b00 = (2 * (1 - p) * (1 - 2 * p)) / (
        (1 - 2 * p) * (1 - p ** (r + 1)) + (W0 * (1 - p) * (1 -
            (2 * p) ** (r + 1))))
```

```
        else:
        b00 = (2 * (1 - p) * (1 - 2 * p)) / (
        (W0 * (1 - (2 * p) ** (m + 1)) * (1 - p)) + ((1 - 2 * p)
            * (1 - p ** (r + 1))) + (
        W0 * (2 ** m) * (p ** (m + 1)) * (1 - p ** (r - m)) * (1
            - 2 * p)))

        eq1 = b00 * ((1 - (p ** (r + 1))) / (1 - p)) - tau1
        # eq2 = ((2 * tau * 13.6) / ((1 - tau) * 9.0 + tau * (1
           - p) * Ts + tau * p * Tc)) - p
        eq2 = b00 * (((V + 1) * (((1 - p ** (m + 1)) / (1 - p)))
           ) - (
        ((V * (V + 1)) / (2 * W0)) * ((1 - (p / 2) ** (m + 1)) /
            (1 - (p / 2))))) - tau2
        eq3 = tau2 - p
        return [eq1, eq2, eq3]

        # 使用fsolve函数求解方程
        tau1, tau2, p = fsolve(equations, (tau1_init, tau2_init,
            p_init))

        return tau1, tau2, p


        def solve_tau_p_problem4(CW_min, m, r, H, EP, nodeNum,
           Ts, Tc, tau1_init=0.1, tau2_init=0.1, p_init=0.1):

        W0 = CW_min
        N = nodeNum
        V = np.around(((0.5 * 9 + 0.5 * (H + EP)) / 2) / 9)

        def equations(vars):
        tau1, tau2, p = vars

        if r <= m:
        b00 = (2 * (1 - p) * (1 - 2 * p)) / (
        (1 - 2 * p) * (1 - p ** (r + 1)) + (W0 * (1 - p) * (1 -
           (2 * p) ** (r + 1))))
        else:
```

```
        b00 = (2 * (1 - p) * (1 - 2 * p)) / (
        (W0 * (1 - (2 * p) ** (m + 1)) * (1 - p)) + ((1 - 2 * p)
            * (1 - p ** (r + 1))) + (
        W0 * (2 ** m) * (p ** (m + 1)) * (1 - p ** (r - m)) * (1
            - 2 * p)))

        eq1 = b00 * ((1 - (p ** (r + 1))) / (1 - p)) - tau1
        # eq2 = ((2 * tau * 13.6) / ((1 - tau) * 9.0 + tau * (1
           - p) * Ts + tau * p * Tc)) - p
        eq2 = b00 * (((V + 1) * (((1 - p ** (m + 1)) / (1 - p)))
           ) - (((V * (V + 1)) / (2 * W0)) * ((1 - (p / 2) ** (m
            + 1)) / (1 - (p / 2))))) - tau2
        eq3 = 1 - ((1 - tau1) ** 1.8) * ((1 - tau2) ** 0.8) - p
        return [eq1, eq2, eq3]

        # 使用fsolve函数求解方程
        tau1, tau2, p = fsolve(equations, (tau1_init, tau2_init,
            p_init))

        return tau1, tau2, p


        def cal_m(CW_min, CW_max):
        """
        计算最大退避阶数m。

        参数:
        CW_min (int): CW窗口最小值。
        CW_max (int): CW窗口最大值。

        返回:
        m (int): 最大退避阶数。
        """
        m = int(np.log2(CW_max / CW_min))
        return m


        def cal_Ts(H, EP, SIFS=16, ACK=32, DIFS=43):
        return H + EP + SIFS + ACK + DIFS
```

```
def cal_Tc(H, EstarP, DIFS=43, ACKTimeout=65):
return H + EstarP + DIFS + ACKTimeout


def cal_Tc_problem3(H, EstarP, DIFS=43, ACKTimeout=65):
T_cover = 9.0 / 2 + H + EstarP + ACKTimeout
T_hidden = (H + EstarP) / 2 + H + EstarP + ACKTimeout
return T_hidden / 2 + T_cover / 2


def cal_S(tau, p, Tc, Ts, EP, v, Te=9):
Ptr = 1 - (1 - tau) ** 2
Ps = (2 * tau * (1 - tau)) / (1 - (1 - tau) ** 2)

print(Ptr, Ps)

return v * (((Ptr * Ps * EP)) / ((1 - Ptr) * Te + Ptr *
   Ps * Ts + Ptr * (1 - Ps) * Tc))


# def cal_S_problem2(tau, p, Ts, EP, v, Te=9):
#     return v * (((tau * (1 - p) * EP) + (tau * p * 2 *
    EP)) / ((1 - tau) * Te + tau * Ts))


def cal_S_problem2(tau, p, Ts, EP, v, Te=9):
Ptr = 1 - (1 - tau) ** 2
Ps = (2 * tau * (1 - tau)) / (1 - (1 - tau) ** 2)

print(Ptr, Ps)

return v * (((Ptr * Ps * EP) + (Ptr * (1 - Ps) * 2 * EP)
   ) / ((1 - Ptr) * Te + Ptr * Ts))


def cal_S_problem3_ideal(tau1, tau2, Tc, Ts, EP, v, Te
   =9):
```

```
    Ptr = 1 - (1 - tau1) ** 2
    Ps = (2 * tau1 * (1 - tau2)) / Ptr

    print(Ptr, Ps)

    return v * (((Ptr * Ps * EP)) / ((1 - Ptr) * Te + Ptr *
        Ps * Ts + Ptr * (1 - Ps) * Tc))


    def cal_S_problem3(tau1, tau2, Tc, Ts, EP, v, Te=9):
    Ptr = 1 - (1 - tau1) ** 2
    Ps = (2 * tau1 * (1 - tau2)) / Ptr

    print(Ptr, Ps)

    return v * (((Ptr * Ps * 1.8 * EP)) / (
    (1 - Ptr) * Te + Ptr * Ps * 0.9 * Ts + Ptr * Ps * 0.1 *
        Tc + Ptr * (1 - Ps) * Tc))


    def cal_S_problem4(tau1, tau2, Tc, Ts, EP, v, Te=9):

    Ptr = 1 - (1 - tau1) ** 3
    Ps = (3 * tau1 * ((1 - tau1) ** 1.5) * ((1 - tau2) **
        0.5)) / Ptr

    print(Ptr, Ps)

    return v * (((Ptr * Ps  * 1.76855 * EP)) / ((1 - Ptr) *
        Te + Ptr * Ps * Ts + Ptr * (1 - Ps) * Tc))


    def calculate_transfer_time(data_length_bytes,
        transfer_rate_mbps):
    # 1 Byte = 8 bits, 所以将数据长度从字节转换为比特
    data_length_bits = data_length_bytes * 8

    # 1 Mbps = 10^6 bits per second, 所以将传输速率从Mbps转
        换为每秒比特数
```

```
    transfer_rate_bps = transfer_rate_mbps * 10 ** 6

    # time = data / rate, 计算传输时间（以秒为单位）
    transfer_time_seconds = data_length_bits /
       transfer_rate_bps

    # 1 second = 10^6 microseconds, 将时间从秒转换为微秒
    transfer_time_microseconds = transfer_time_seconds * 10
       ** 6

    return transfer_time_microseconds
```

1.2 问题一数值分析方法求解

problem1-math.py

```
from utils import *

CW_min = 16
CW_max = 1024
m = cal_m(CW_min, CW_max)
r = 32
nodeNum = 2

tau, p = solve_tau_p(CW_min, m, r, nodeNum)
H = 13.6 + calculate_transfer_time(30, 455.8)
EP = calculate_transfer_time(1500, 455.8)
Ts = cal_Ts(H, EP)
Tc = cal_Tc(H, EP)
S = cal_S(tau, p, Tc, Ts, EP, 455.8)

print('tau:', tau)
print('p:', p)
print('H:', H)
print('EP:', EP)
print('Ts:', Ts)
print('Tc:', Tc)
print('S:', S)
```

1.3 问题一仿真算法

problem1-simulation.py

```
from utils import *
import numpy as np
from matplotlib import pyplot as plt

# 支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号


class Simulator:

def __init__(self):
H = 13.6 + calculate_transfer_time(30, 455.8)
self.EP = calculate_transfer_time(1500, 455.8)

self.Te = 9
self.Ts = cal_Ts(H, self.EP)
self.Tc = cal_Tc(H, self.EP)

self.CW_min = 16
self.CW_max = 1024
self.r = 32

self.DIFS = 43

def run(self):
t = self.DIFS
CW1 = self.CW_min
CW2 = self.CW_min

# 分别从[0,CW - 1]中随机选取一个数
BO1 = np.random.randint(0, CW1)
BO2 = np.random.randint(0, CW2)
```

```
        all_bits = 0
        c = 0

        ts = []
        Ss = []

        for i in range(1000):
        BO = min(BO1, BO2)
        BO1 -= BO
        BO2 -= BO
        t += BO * self.Te

        if BO1 == 0 and BO2 == 0:
        t += self.Tc
        CW1 *= 2
        CW2 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        print(f"当前时间点：{t}μs，两个节点冲突")
        c += 1
        if c == self.r:
        c = 0
        CW1 = self.CW_min
        CW2 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        else:
        if BO1 == 0:
        t += self.Ts
        all_bits += 1500 * 8
        S = all_bits / t
        ts.append(t)
        Ss.append(S)
        print(f"当前时间点：{t}μs，节点1发送数据包成功，当前计算
            得到的吞吐量为{S}")
```

```
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        c = 0
        if BO2 == 0:
        t += self.Ts
        all_bits += 1500 * 8
        S = all_bits / t
        ts.append(t)
        Ss.append(S)
        print(f"当前时间点：{t}μs，节点2发送数据包成功，当前计算
            得到的吞吐量为{S}")
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c = 0

        # 绘制吞吐量随时间的变化曲线
        plt.plot(ts, Ss)
        plt.xlabel('时间（μs）')
        plt.ylabel('吞吐量S（Mbps）')
        plt.show()


        simulator = Simulator()
        simulator.run()
```

1.4 问题一蒙特卡洛求解

problem1-MonteCarlo.py

```
        from utils import *
        import numpy as np


        class Simulator:

        def __init__(self):
        H = 13.6 + calculate_transfer_time(30, 455.8)
        self.EP = calculate_transfer_time(1500, 455.8)

        self.Te = 9
```

```
        self.Ts = cal_Ts(H, self.EP)
        self.Tc = cal_Tc(H, self.EP)

        self.CW_min = 16
        self.CW_max = 1024
        self.r = 32

        self.DIFS = 43

        def run(self):
        t = self.DIFS
        CW1 = self.CW_min
        CW2 = self.CW_min

        # 分别从[0,CW - 1]中随机选取一个数
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)

        all_bits = 0
        c = 0

        for i in range(10000):
        BO = min(BO1, BO2)
        BO1 -= BO
        BO2 -= BO
        t += BO * self.Te

        if BO1 == 0 and BO2 == 0:
        t += self.Tc
        CW1 *= 2
        CW2 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        c += 1
        if c == self.r:
```

```
        c = 0
        CW1 = self.CW_min
        CW2 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        else:
        if BO1 == 0:
        t += self.Ts
        all_bits += 1500 * 8
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        c = 0
        if BO2 == 0:
        t += self.Ts
        all_bits += 1500 * 8
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c = 0

        return all_bits / t


        simulator = Simulator()

        results = []

        for i in range(1000):
        results.append(simulator.run())

        print(f"[{np.min(results)},{np.max(results)}]")
        print(np.mean(results))

        # 将results内容一列一列地保存到problem3_{CW_min}_{CW_max
           }_r_v.txt文件中
        with open(f"./output/problem1_result.txt", "w") as f:
        for result in results:
        f.write(str(result) + "\n")
```

1.5 问题二数值分析方法求解

problem2-math.py

```
from utils import *

CW_min = 16
CW_max = 1024
m = cal_m(CW_min, CW_max)
r = 32
nodeNum = 2

tau, p = 2.0 / 17.0, 2.0 / 17.0
H = 13.6 + calculate_transfer_time(30, 275.3)
EP = calculate_transfer_time(1500, 275.3)
Ts = cal_Ts(H, EP)
S = cal_S_problem2(tau, p, Ts, EP, 275.3)

print('tau:', tau)
print('p:', p)
print('H:', H)
print('EP:', EP)
print('Ts:', Ts)
print('S:', S)
```

1.6 问题二仿真算法

problem2-simulation.py

```
from utils import *
import numpy as np
from matplotlib import pyplot as plt

# 支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号


class Simulator:

def __init__(self):
```

```
        H = 13.6 + calculate_transfer_time(30, 275.3)
        self.EP = calculate_transfer_time(1500, 275.3)

        self.Te = 9
        self.Ts = cal_Ts(H, self.EP)

        self.CW_min = 16
        self.CW_max = 1024
        self.r = 32

        self.DIFS = 43

        def run(self):
        t = self.DIFS
        CW1 = self.CW_min
        CW2 = self.CW_min

        # 分别从[0,CW - 1]中随机选取一个数
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)

        all_bits = 0
        c = 0

        ts = []
        Ss = []

        for i in range(1000):
        BO = min(BO1, BO2)
        BO1 -= BO
        BO2 -= BO
        t += BO * self.Te

        if BO1 == 0 and BO2 == 0:
        t += self.Ts
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        all_bits += 1500 * 8 * 2
        S = all_bits / t
```

```
            print(f"当前时间点：{t}μs，两个节点冲突，但是可以一起
                走，当前计算得到的吞吐量为{S}")
            else:
            if BO1 == 0:
            t += self.Ts
            all_bits += 1500 * 8
            S = all_bits / t
            ts.append(t)
            Ss.append(S)
            print(f"当前时间点：{t}μs，节点1发送数据包成功，当前计算
                得到的吞吐量为{S}")
            BO1 = np.random.randint(0, CW1)
            if BO2 == 0:
            t += self.Ts
            all_bits += 1500 * 8
            S = all_bits / t
            ts.append(t)
            Ss.append(S)
            print(f"当前时间点：{t}μs，节点2发送数据包成功，当前计算
                得到的吞吐量为{S}")
            BO2 = np.random.randint(0, CW2)

            # 绘制吞吐量随时间的变化曲线
            plt.plot(ts, Ss)
            plt.xlabel('时间（μs）')
            plt.ylabel('吞吐量S（Mbps）')
            plt.show()


            simulator = Simulator()
            simulator.run()
```

1.7 问题二蒙特卡洛求解

problem2-MonteCarlo.py

```
            from utils import *
            import numpy as np
```

```
class Simulator:

def __init__(self):
H = 13.6 + calculate_transfer_time(30, 275.3)
self.EP = calculate_transfer_time(1500, 275.3)

self.Te = 9
self.Ts = cal_Ts(H, self.EP)

self.CW_min = 16
self.CW_max = 1024
self.r = 32

self.DIFS = 43

def run(self):
t = self.DIFS
CW1 = self.CW_min
CW2 = self.CW_min

# 分别从[0,CW - 1]中随机选取一个数
BO1 = np.random.randint(0, CW1)
BO2 = np.random.randint(0, CW2)

all_bits = 0

for i in range(10000):
BO = min(BO1, BO2)
BO1 -= BO
BO2 -= BO
t += BO * self.Te

if BO1 == 0 and BO2 == 0:
t += self.Ts
BO1 = np.random.randint(0, CW1)
BO2 = np.random.randint(0, CW2)
all_bits += 1500 * 8 * 2
else:
if BO1 == 0:
```

```
        t += self.Ts
        all_bits += 1500 * 8
        BO1 = np.random.randint(0, CW1)
        if BO2 == 0:
        t += self.Ts
        all_bits += 1500 * 8
        BO2 = np.random.randint(0, CW2)

        return all_bits / t


        simulator = Simulator()

        results = []

        for i in range(1000):
        results.append(simulator.run())

        print(f"[{np.min(results)},{np.max(results)}]")
        print(np.mean(results))

        # 将results内容一列一列地保存到problem3_{CW_min}_{CW_max
           }_r_v.txt文件中
        with open(f"./output/problem2_result.txt", "w") as f:
        for result in results:
        f.write(str(result) + "\n")
```

1.8 问题三理想信道下数值分析方法求解

problem3-math-ideal.py

```
        from utils import *

        CW_min = 16
        CW_max = 1024
        m = cal_m(CW_min, CW_max)
        r = 32
        nodeNum = 2

        H = 13.6 + calculate_transfer_time(30, 455.8)
```

```
EP = calculate_transfer_time(1500, 455.8)
Ts = cal_Ts(H, EP)
Tc = cal_Tc(H, EP)
tau1, tau2, p = solve_tau_p_problem3(CW_min, m, r,
    nodeNum, Ts, Tc)
S = cal_S(tau1, tau2, Tc, Ts, EP, 455.8)

print('tau1:', tau1)
print('tau2:', tau2)
print('p:', p)
print('H:', H)
print('EP:', EP)
print('Ts:', Ts)
print('Tc:', Tc)
print('S:', S)
```

1.9 问题三理想信道下仿真算法

problem3-simulation-ideal.py

```
from utils import *
import numpy as np
from matplotlib import pyplot as plt

# 支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常
    显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显
    示负号


class Simulator:

def __init__(self, CW_min, CW_max, r, v):
self.H = 13.6 + calculate_transfer_time(30, v)
self.EP = calculate_transfer_time(1500, v)

self.Te = 9
self.Ts = cal_Ts(self.H, self.EP)
self.Tc = cal_Tc(self.H, self.EP)
```

```
self.CW_min = CW_min
self.CW_max = CW_max
self.r = r

self.DIFS = 43

def run(self):
t1 = self.DIFS
t2 = self.DIFS
CW1 = self.CW_min
CW2 = self.CW_min

# 分别从[0,CW - 1]中随机选取一个数
BO1 = np.random.randint(0, CW1)
BO2 = np.random.randint(0, CW2)

t1 += BO1 * self.Te
t2 += BO2 * self.Te

all_bits = 0
c = 0

ts = []
Ss = []

for i in range(1000):

if t1 < t2:
if t1 + self.H + self.EP > t2:
print(f"t={t1}μs和t={t2}μs时间点，节点1与节点2分别发出了
    会导致交叠的信号包")
CW1 *= 2
CW2 *= 2
if CW1 > self.CW_max:
CW1 = self.CW_max
if CW2 > self.CW_max:
CW2 = self.CW_max
BO1 = np.random.randint(0, CW1)
```

```
            BO2 = np.random.randint(0, CW2)
            c += 1
            if c == self.r:
            c = 0
            CW1 = self.CW_min
            CW2 = self.CW_min
            BO1 = np.random.randint(0, CW1)
            BO2 = np.random.randint(0, CW2)
            t1 += self.Tc + BO1 * self.Te
            t2 += self.Tc + BO2 * self.Te
            else:
            t1 += self.Ts
            all_bits += 1500 * 8
            S = all_bits / t1
            print(f"当前时间点：{t1}μs，节点1发送数据包成功，当前计
                算得到的吞吐量为{S}")
            ts.append(t1)
            Ss.append(S)
            CW1 = self.CW_min
            BO1 = np.random.randint(0, CW1)
            c = 0
            t1 += BO1 * self.Te
            else:
            if t2 + self.H + self.EP > t1:
            print(f"t={t2}μs和t={t1}μs时间点，节点2与节点1分别发出了
                会导致交叠的信号包")
            CW1 *= 2
            CW2 *= 2
            if CW1 > self.CW_max:
            CW1 = self.CW_max
            if CW2 > self.CW_max:
            CW2 = self.CW_max
            BO1 = np.random.randint(0, CW1)
            BO2 = np.random.randint(0, CW2)
            c += 1
            if c == self.r:
            c = 0
            CW1 = self.CW_min
            CW2 = self.CW_min
```

```
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        else:
        t2 += self.Ts
        all_bits += 1500 * 8
        S = all_bits / t2
        print(f"当前时间点：{t2}μs，节点2发送数据包成功，当前计
            算得到的吞吐量为{S}")
        ts.append(t2)
        Ss.append(S)
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c = 0
        t2 += BO2 * self.Te

        # 绘制吞吐量随时间的变化曲线
        plt.plot(ts, Ss)
        plt.xlabel('时间（μs）')
        plt.ylabel('吞吐S（Mbps）')
        plt.show()


        simulator = Simulator(16, 1024, 32, 455.8)
        simulator.run()
```

1.10 问题三理想信道下蒙特卡洛求解

problem3-MonteCarlo-ideal.py

```
        from utils import *
        import numpy as np


        class Simulator:

        def __init__(self, CW_min, CW_max, r, v):
        H = 13.6 + calculate_transfer_time(30, v)
        self.EP = calculate_transfer_time(1500, v)
```

```
        self.Te = 9
        self.Ts = cal_Ts(H, self.EP)
        self.Tc = cal_Tc(H, self.EP)

        self.CW_min = CW_min
        self.CW_max = CW_max
        self.r = r

        self.DIFS = 43

    def run(self):
        t1 = self.DIFS
        t2 = self.DIFS
        CW1 = self.CW_min
        CW2 = self.CW_min

        # 分别从[0,CW - 1]中随机选取一个数
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)

        t1 += BO1 * self.Te
        t2 += BO2 * self.Te

        all_bits = 0
        c = 0

        ts = []
        Ss = []

        for i in range(10000):

            if t1 < t2:
                if t1 + 14.126546731022378 + 26.32733655111891 > t2:
                    CW1 *= 2
                    CW2 *= 2
                    if CW1 > self.CW_max:
                        CW1 = self.CW_max
                    if CW2 > self.CW_max:
```

```
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        c += 1
        if c == self.r:
        c = 0
        CW1 = self.CW_min
        CW2 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        else:
        t1 += self.Ts
        all_bits += 1500 * 8
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        c = 0
        t1 += BO1 * self.Te
        else:
        if t2 + 14.126546731022378 + 26.32733655111891 > t1:
        CW1 *= 2
        CW2 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        c += 1
        if c == self.r:
        c = 0
        CW1 = self.CW_min
        CW2 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        else:
```

```
        t2 += self.Ts
        all_bits += 1500 * 8
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c = 0
        t2 += BO2 * self.Te

        return all_bits / max(t1, t2)


        simulator = Simulator(16, 1024, 32, 455.8)
        results = []

        for i in range(1000):
        results.append(simulator.run())

        print(f"[{np.min(results)},{np.max(results)}]")
        print(np.mean(results))

        # 将results内容一列一列地保存到problem3_{CW_min}_{CW_max
           }_r_v.txt文件中
        with open(f"./output/problem3_result_ideal.txt", "w") as
            f:
        for result in results:
        f.write(str(result) + "\n")
```

1.11 问题三数值分析方法求解

problem3-math.py

```
        from utils import *

        CW_min = 16
        CW_max = 1024
        r = 32
        v = 455.8

        m = cal_m(CW_min, CW_max)
        H = 13.6 + calculate_transfer_time(30, v)
        EP = calculate_transfer_time(1500, v)
```

```
Ts = cal_Ts(H, EP)
Tc = cal_Tc_problem3(H, EP)
tau1, tau2, p = solve_tau_p_problem3(CW_min, m, r, H, EP
    , 2, Ts, Tc)
S = cal_S_problem3(tau1, tau2, Tc, Ts, EP, v)

print('tau1:', tau1)
print('tau2:', tau2)
print('p:', p)
print('H:', H)
print('EP:', EP)
print('Ts:', Ts)
print('Tc:', Tc)
print('S:', S)
```

1.12 问题三仿真算法

problem3-simulation.py

```
from utils import *
import numpy as np
from matplotlib import pyplot as plt

# 支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常
    显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显
    示负号


class Simulator:

def __init__(self, CW_min, CW_max, r, v):
self.H = 13.6 + calculate_transfer_time(30, v)
self.EP = calculate_transfer_time(1500, v)

self.Te = 9
self.Ts = cal_Ts(self.H, self.EP)
self.Tc = cal_Tc(self.H, self.EP)
```

```
        self.CW_min = CW_min
        self.CW_max = CW_max
        self.r = r

        self.Pe = 0.1

        self.DIFS = 43

        def run(self):
        t1 = self.DIFS
        t2 = self.DIFS
        CW1 = self.CW_min
        CW2 = self.CW_min

        # 分别从[0,CW - 1]中随机选取一个数
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)

        t1 += BO1 * self.Te
        t2 += BO2 * self.Te

        all_bits = 0
        c1 = 0
        c2 = 0

        ts = []
        Ss = []

        for i in range(1000):

        if t1 < t2:
        if t1 + self.H + self.EP > t2:
        print(f"t={t1}μs和t={t2}μs时间点，节点1与节点2分别发出了
            会导致交叠的信号包")
        CW1 *= 2
        CW2 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
```

```
            CW2 = self.CW_max
            BO1 = np.random.randint(0, CW1)
            BO2 = np.random.randint(0, CW2)
            c1 += 1
            c2 += 1
            if c1 == self.r:
            c1 = 0
            CW1 = self.CW_min
            BO1 = np.random.randint(0, CW1)
            if c2 == self.r:
            c2 = 0
            CW2 = self.CW_min
            BO2 = np.random.randint(0, CW2)
            t1 += self.Tc + BO1 * self.Te
            t2 += self.Tc + BO2 * self.Te
            else:
            # 存在Pe的概率会丢包
            if np.random.random() < self.Pe:
            print(f"当前时间点：{t1}μs，节点1发送的数据包发生丢包")
            CW1 *= 2
            if CW1 > self.CW_max:
            CW1 = self.CW_max
            BO1 = np.random.randint(0, CW1)
            c1 += 1
            if c1 == self.r:
            c1 = 0
            CW1 = self.CW_min
            BO1 = np.random.randint(0, CW1)
            t1 += self.Tc + BO1 * self.Te
            else:
            t1 += self.Ts
            all_bits += 1500 * 8
            S = all_bits / t1
            print(f"当前时间点：{t1}μs，节点1发送数据包成功，当前计
                算得到的吞吐量为{S}")
            ts.append(t1)
            Ss.append(S)
            CW1 = self.CW_min
            BO1 = np.random.randint(0, CW1)
```

```
        c = 0
        t1 += BO1 * self.Te
        else:
        if t2 + self.H + self.EP > t1:
        print(f"t={t2}μs和t={t1}μs时间点，节点2与节点1分别发出了
            会导致交叠的信号包")
        CW1 *= 2
        CW2 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        c1 += 1
        c2 += 1
        if c1 == self.r:
        c1 = 0
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        if c2 == self.r:
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        else:
        # 存在Pe的概率会丢包
        if np.random.random() < self.Pe:
        print(f"当前时间点：{t2}μs，节点2发送的数据包发生丢包")
        CW2 *= 2
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO2 = np.random.randint(0, CW2)
        c2 += 1
        if c2 == self.r:
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
```

```
        t2 += self.Tc + BO2 * self.Te
        else:
        t2 += self.Ts
        all_bits += 1500 * 8
        S = all_bits / t2
        print(f"当前时间点：{t2}μs，节点2发送数据包成功，当前计
            算得到的吞吐量为{S}")
        ts.append(t2)
        Ss.append(S)
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c = 0
        t2 += BO2 * self.Te

        # 绘制吞吐量随时间的变化曲线
        plt.plot(ts, Ss)
        plt.xlabel('时间（μs）')
        plt.ylabel('吞吐S（Mbps）')
        plt.show()


        simulator = Simulator(16, 1024, 32, 455.8)
        simulator.run()
```

1.13 问题三蒙特卡洛求解

problem3-MonteCarlo.py

```
        from utils import *
        import numpy as np


        class Simulator:

        def __init__(self, CW_min, CW_max, r, v):
        self.H = 13.6 + calculate_transfer_time(30, v)
        self.EP = calculate_transfer_time(1500, v)

        self.Te = 9
        self.Ts = cal_Ts(self.H, self.EP)
```

```
        self.Tc = cal_Tc(self.H, self.EP)

        self.CW_min = CW_min
        self.CW_max = CW_max
        self.r = r

        self.Pe = 0.1

        self.DIFS = 43

        def run(self):
        t1 = self.DIFS
        t2 = self.DIFS
        CW1 = self.CW_min
        CW2 = self.CW_min

        # 分别从[0,CW - 1]中随机选取一个数
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)

        t1 += BO1 * self.Te
        t2 += BO2 * self.Te

        all_bits = 0
        c1 = 0
        c2 = 0

        ts = []
        Ss = []

        for i in range(10000):

        if t1 < t2:
        if t1 + self.H + self.EP > t2:
        CW1 *= 2
        CW2 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
```

```
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        c1 += 1
        c2 += 1
        if c1 == self.r:
        c1 = 0
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        if c2 == self.r:
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        else:
        # 存在Pe的概率会丢包
        if np.random.random() < self.Pe:
        CW1 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        c1 += 1
        if c1 == self.r:
        c1 = 0
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        t1 += self.Tc + BO1 * self.Te
        else:
        t1 += self.Ts
        all_bits += 1500 * 8
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        c = 0
        t1 += BO1 * self.Te
        else:
        if t2 + self.H + self.EP > t1:
        CW1 *= 2
        CW2 *= 2
```

```
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        c1 += 1
        c2 += 1
        if c1 == self.r:
        c1 = 0
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        if c2 == self.r:
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        else:
        # 存在Pe的概率会丢包
        if np.random.random() < self.Pe:
        CW2 *= 2
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO2 = np.random.randint(0, CW2)
        c2 += 1
        if c2 == self.r:
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        t2 += self.Tc + BO2 * self.Te
        else:
        t2 += self.Ts
        all_bits += 1500 * 8
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c = 0
        t2 += BO2 * self.Te
```

```
        return all_bits / max(t1, t2)

        # CW_min = 16
        # CW_max = 1024
        # r = 32
        # v = 455.8

        CW_min = 16
        CW_max = 1024
        r = 32
        v = 158.4

        simulator = Simulator(CW_min, CW_max, r, v)
        results = []

        for i in range(1000):
        results.append(simulator.run())

        print(f"[{np.min(results)},{np.max(results)}]")
        print(np.mean(results))

        # 将results内容一列一列地保存到problem3_{CW_min}_{CW_max
           }_r_v.txt文件中
        with open(f"./output/problem3_{CW_min}_{CW_max}_{r}_{v}.
           txt", "w") as f:
        for result in results:
        f.write(str(result) + "\n")
```

1.14 问题四数值分析方法求解

problem4-math.py

```
        from utils import *

        CW_min = 16
        CW_max = 1024
        r = 6
        v = 268.8

        m = cal_m(CW_min, CW_max)
```

```
H = 13.6 + calculate_transfer_time(30, v)
EP = calculate_transfer_time(1500, v)
Ts = cal_Ts(H, EP)
Tc = cal_Tc_problem3(H, EP)
tau1, tau2, p = solve_tau_p_problem4(CW_min, m, r, H, EP
    , 3, Ts, Tc)
# tau, p = solve_tau_p_problem4(CW_min, m, r)
S = cal_S_problem4(tau1, tau2, Tc, Ts, EP, v)

print('tau1:', tau1)
print('tau2:', tau2)
# print('tau:', tau)
print('p:', p)
print('H:', H)
print('EP:', EP)
print('Ts:', Ts)
print('Tc:', Tc)
print('S:', S)
```

1.15 问题四仿真算法

problem4-simulation.py

```
from utils import *
import numpy as np
from matplotlib import pyplot as plt

# 支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常
    显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显
    示负号


class Simulator:

def __init__(self, CW_min, CW_max, r, v):
self.H = 13.6 + calculate_transfer_time(30, v)
self.EP = calculate_transfer_time(1500, v)
```

```
self.Te = 9
self.Ts = cal_Ts(self.H, self.EP)
self.Tc = cal_Tc(self.H, self.EP)

self.CW_min = CW_min
self.CW_max = CW_max
self.r = r

self.DIFS = 43

def run(self):
t1 = self.DIFS
t2 = self.DIFS
t3 = self.DIFS
CW1 = self.CW_min
CW2 = self.CW_min
CW3 = self.CW_min

# 分别从[0,CW - 1]中随机选取一个数
BO1 = np.random.randint(0, CW1)
BO2 = np.random.randint(0, CW2)
BO3 = np.random.randint(0, CW3)

t1 += BO1 * self.Te
t2 += BO2 * self.Te
t3 += BO3 * self.Te

all_bits = 0
c1 = 0
c2 = 0
c3 = 0

ts = []
Ss = []

for i in range(1000):
if t1 == t2 and t2 == t3:
print(f"t={t1}μs时间点，三个节点同时准备发包，发生冲突")
```

```
            CW1 *= 2
            CW2 *= 2
            CW3 *= 2
            if CW1 > self.CW_max:
            CW1 = self.CW_max
            if CW2 > self.CW_max:
            CW2 = self.CW_max
            if CW3 > self.CW_max:
            CW3 = self.CW_max
            BO1 = np.random.randint(0, CW1)
            BO2 = np.random.randint(0, CW2)
            BO3 = np.random.randint(0, CW3)
            c1 += 1
            if c1 == self.r:
            c1 = 0
            CW1 = self.CW_min
            BO1 = np.random.randint(0, CW1)
            c2 += 1
            if c2 == self.r:
            c2 = 0
            CW2 = self.CW_min
            BO2 = np.random.randint(0, CW2)
            c3 += 1
            if c3 == self.r:
            c3 = 0
            CW3 = self.CW_min
            BO3 = np.random.randint(0, CW3)
            t1 += self.Tc + BO1 * self.Te
            t2 += self.Tc + BO2 * self.Te
            t3 += self.Tc + BO3 * self.Te
            elif t1 == t2 and t2 < t3:
            print(f"t={t2}μs时间点，节点1和节点2同时准备发包，发生冲
                突")
            CW1 *= 2
            CW2 *= 2
            if CW1 > self.CW_max:
            CW1 = self.CW_max
            if CW2 > self.CW_max:
            CW2 = self.CW_max
```

```
            BO1 = np.random.randint(0, CW1)
            BO2 = np.random.randint(0, CW2)
            c1 += 1
            if c1 == self.r:
            c1 = 0
            CW1 = self.CW_min
            BO1 = np.random.randint(0, CW1)
            c2 += 1
            if c2 == self.r:
            c2 = 0
            CW2 = self.CW_min
            BO2 = np.random.randint(0, CW2)

            if t3 < t2 + self.Tc:
            # 节点2与节点3互听，节点3需要与节点2对齐
            t3 = t2 + self.Tc + (t3 - t2)

            t1 += self.Tc + BO1 * self.Te
            t2 += self.Tc + BO2 * self.Te
            elif t3 == t2 and t2 < t1:
            print(f"t={t2}μs时间点，节点2和节点3同时准备发包，发生冲
                突")
            CW2 *= 2
            CW3 *= 2
            if CW2 > self.CW_max:
            CW2 = self.CW_max
            if CW3 > self.CW_max:
            CW3 = self.CW_max
            BO2 = np.random.randint(0, CW2)
            BO3 = np.random.randint(0, CW3)
            c2 += 1
            if c2 == self.r:
            c2 = 0
            CW2 = self.CW_min
            BO2 = np.random.randint(0, CW2)
            c3 += 1
            if c3 == self.r:
            c3 = 0
            CW3 = self.CW_min
```

```
        BO3 = np.random.randint(0, CW3)

        if t1 < t2 + self.Tc:
        # 节点2与节点1互听，节点1需要与节点2对齐
        t1 = t2 + self.Tc + (t1 - t2)

        t2 += self.Tc + BO2 * self.Te
        t3 += self.Tc + BO3 * self.Te
        elif t2 < t1 and t2 < t3:

        if t1 < t2 + self.Ts:
        # 节点2与节点1互听，节点1需要与节点2对齐
        t1 = t2 + self.Ts + (t1 - t2)
        if t3 < t2 + self.Ts:
        # 节点2与节点3互听，节点3需要与节点2对齐
        t3 = t2 + self.Ts + (t3 - t2)

        t2 += self.Ts
        all_bits += 1500 * 8
        S = all_bits / t2
        print(f"当前时间点：{t2}μs，节点2发送数据包成功，当前计
            算得到的吞吐量为{S}")
        ts.append(t2)
        Ss.append(S)
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c2 = 0
        t2 += BO2 * self.Te
        elif t1 < t2 and t1 < t3:

        if t2 < t1 + self.Ts:
        # 节点1与节点2互听，节点2需要与节点1对齐
        t2 = t1 + self.Ts + (t2 - t1)

        t1 += self.Ts
        all_bits += 1500 * 8
        S = all_bits / t1
        print(f"当前时间点：{t1}μs，节点1发送数据包成功，当前计
            算得到的吞吐量为{S}")
```

```
            ts.append(t1)
            Ss.append(S)
            CW1 = self.CW_min
            BO1 = np.random.randint(0, CW1)
            c1 = 0
            t1 += BO1 * self.Te
            elif t3 < t2 and t3 < t1:

            if t2 < t3 + self.Ts:
            # 节点3与节点2互听，节点2需要与节点3对齐
            t2 = t3 + self.Ts + (t2 - t3)

            t3 += self.Ts
            all_bits += 1500 * 8
            S = all_bits / t3
            print(f"当前时间点：{t3}μs，节点3发送数据包成功，当前计
                算得到的吞吐量为{S}")
            ts.append(t3)
            Ss.append(S)
            CW3 = self.CW_min
            BO3 = np.random.randint(0, CW3)
            c3 = 0
            t3 += BO3 * self.Te
            elif t1 == t3 and t3 < t2:
            # 节点1和节点3两者发送均成功

            if t2 < t1 + self.Ts:
            # 节点2与节点1和节点3都互听，节点2需要与节点1和节点3对齐
            t2 = t1 + self.Ts + (t2 - t1)

            t1 += self.Ts
            t3 += self.Ts
            all_bits += 1500 * 8 * 2
            S = all_bits / t1
            print(f"当前时间点：{t1}μs，节点1和节点3发送数据包成功，
                当前计算得到的吞吐量为{S}")
            ts.append(t1)
            Ss.append(S)
            CW1 = self.CW_min
```

```
        CW3 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        BO3 = np.random.randint(0, CW3)
        c1 = 0
        c3 = 0
        t1 += BO1 * self.Te
        t3 += BO3 * self.Te
        else:
        print(f"{t1}:{t2}:{t3}时间点，发生了未知错误")

        # 绘制吞吐量随时间的变化曲线
        plt.plot(ts, Ss)
        plt.xlabel('时间（μs）')
        plt.ylabel('吞吐S（Mbps）')
        plt.show()


        # simulator = Simulator(16, 1024, 32, 455.8)
        simulator = Simulator(16, 1024, 6, 286.8)
        simulator.run()
```

1.16 问题四蒙特卡洛求解

problem4-MonteCarlo.py

```
        from utils import *
        import numpy as np


        class Simulator:

        def __init__(self, CW_min, CW_max, r, v):
        self.H = 13.6 + calculate_transfer_time(30, v)
        self.EP = calculate_transfer_time(1500, v)

        self.Te = 9
        self.Ts = cal_Ts(self.H, self.EP)
        self.Tc = cal_Tc(self.H, self.EP)

        self.CW_min = CW_min
```

```
        self.CW_max = CW_max
        self.r = r

        self.DIFS = 43

        def run(self):
        t1 = self.DIFS
        t2 = self.DIFS
        t3 = self.DIFS
        CW1 = self.CW_min
        CW2 = self.CW_min
        CW3 = self.CW_min

        # 分别从[0,CW - 1]中随机选取一个数
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        BO3 = np.random.randint(0, CW3)

        t1 += BO1 * self.Te
        t2 += BO2 * self.Te
        t3 += BO3 * self.Te

        all_bits = 0
        c1 = 0
        c2 = 0
        c3 = 0

        ts = []
        Ss = []

        for i in range(10000):
        if t1 == t2 and t2 == t3:
        CW1 *= 2
        CW2 *= 2
        CW3 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
        CW2 = self.CW_max
```

```
        if CW3 > self.CW_max:
        CW3 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        BO3 = np.random.randint(0, CW3)
        c1 += 1
        if c1 == self.r:
        c1 = 0
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        c2 += 1
        if c2 == self.r:
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c3 += 1
        if c3 == self.r:
        c3 = 0
        CW3 = self.CW_min
        BO3 = np.random.randint(0, CW3)
        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        t3 += self.Tc + BO3 * self.Te
        elif t1 == t2 and t2 < t3:
        CW1 *= 2
        CW2 *= 2
        if CW1 > self.CW_max:
        CW1 = self.CW_max
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        BO1 = np.random.randint(0, CW1)
        BO2 = np.random.randint(0, CW2)
        c1 += 1
        if c1 == self.r:
        c1 = 0
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        c2 += 1
        if c2 == self.r:
```

```
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)

        if t3 < t2 + self.Tc:
        # 节点2与节点3互听，节点3需要与节点2对齐
        t3 = t2 + self.Tc + (t3 - t2)

        t1 += self.Tc + BO1 * self.Te
        t2 += self.Tc + BO2 * self.Te
        elif t3 == t2 and t2 < t1:
        CW2 *= 2
        CW3 *= 2
        if CW2 > self.CW_max:
        CW2 = self.CW_max
        if CW3 > self.CW_max:
        CW3 = self.CW_max
        BO2 = np.random.randint(0, CW2)
        BO3 = np.random.randint(0, CW3)
        c2 += 1
        if c2 == self.r:
        c2 = 0
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c3 += 1
        if c3 == self.r:
        c3 = 0
        CW3 = self.CW_min
        BO3 = np.random.randint(0, CW3)

        if t1 < t2 + self.Tc:
        # 节点2与节点1互听，节点1需要与节点2对齐
        t1 = t2 + self.Tc + (t1 - t2)

        t2 += self.Tc + BO2 * self.Te
        t3 += self.Tc + BO3 * self.Te
        elif t2 < t1 and t2 < t3:

        if t1 < t2 + self.Ts:
```

```
        # 节点2与节点1互听，节点1需要与节点2对齐
        t1 = t2 + self.Ts + (t1 - t2)
        if t3 < t2 + self.Ts:
        # 节点2与节点3互听，节点3需要与节点2对齐
        t3 = t2 + self.Ts + (t3 - t2)

        t2 += self.Ts
        all_bits += 1500 * 8
        CW2 = self.CW_min
        BO2 = np.random.randint(0, CW2)
        c2 = 0
        t2 += BO2 * self.Te
        elif t1 < t2 and t1 < t3:

        if t2 < t1 + self.Ts:
        # 节点1与节点2互听，节点2需要与节点1对齐
        t2 = t1 + self.Ts + (t2 - t1)

        t1 += self.Ts
        all_bits += 1500 * 8
        CW1 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        c1 = 0
        t1 += BO1 * self.Te
        elif t3 < t2 and t3 < t1:

        if t2 < t3 + self.Ts:
        # 节点3与节点2互听，节点2需要与节点3对齐
        t2 = t3 + self.Ts + (t2 - t3)

        t3 += self.Ts
        all_bits += 1500 * 8
        CW3 = self.CW_min
        BO3 = np.random.randint(0, CW3)
        c3 = 0
        t3 += BO3 * self.Te
        elif t1 == t3 and t3 < t2:
        # 节点1和节点3两者发送均成功
```

```
        if t2 < t1 + self.Ts:
        # 节点2与节点1和节点3都互听，节点2需要与节点1和节点3对齐
        t2 = t1 + self.Ts + (t2 - t1)

        t1 += self.Ts
        t3 += self.Ts
        all_bits += 1500 * 8 * 2
        CW1 = self.CW_min
        CW3 = self.CW_min
        BO1 = np.random.randint(0, CW1)
        BO3 = np.random.randint(0, CW3)
        c1 = 0
        c3 = 0
        t1 += BO1 * self.Te
        t3 += BO3 * self.Te
        else:
        print(f"{t1}:{t2}:{t3}时间点，发生了未知错误")

        return all_bits / max(t1, t2)

        # CW_min = 16
        # CW_max = 1024
        # r = 32
        # v = 455.8

        CW_min = 16
        CW_max = 1024
        r = 32
        v = 286.8

        # CW_min = 16
        # CW_max = 1024
        # r = 32
        # v = 158.4

        simulator = Simulator(CW_min, CW_max, r, v)
        results = []

        for i in range(1000):
```

```
        if i % 100 == 0:
        print(i)
        results.append(simulator.run())

        print(f"[{np.min(results)},{np.max(results)}]")
        print(np.mean(results))

        # 将results内容一列一列地保存到problem4_{CW_min}_{CW_max
           }_r_v.txt文件中
        with open(f"./output/problem4_{CW_min}_{CW_max}_{r}_{v}.
           txt", "w") as f:
        for result in results:
        f.write(str(result) + "\n")
```

附录 B　蒙特卡洛模拟结果

2.1　问题一蒙特卡洛模拟结果

完整结果见 problem1_result.txt

64.90784567908578,65.39910956415656,65.49853000798251,65.52770339649499,
65.56831071462855,64.68582870317485,65.35599308546618,64.59284503988293,
65.004494534769,……,65.17653327180301,65.57085448497061,64.74095265091641,
65.29143185173045,65.26747405877595,65.29777461596579,65.40455501248151,
64.93874258572735,65.27304310708645,65.12920807290816,64.77485785459399,……

2.2　问题二蒙特卡洛模拟结果

完整结果见 problem2_result.txt

69.03148837252989,68.57584914548134,68.9627267626008,68.84991983427712,
69.26646748355498,69.03023298738631,68.92206852641812,68.80786434496933,
69.09916287168787,68.69624913997608,……,69.05396185119346,69.2695011971095,
69.06140971560674,68.83195819107273,68.9641775024397,69.00078177199423,
68.97157557097604,68.99089422744382,68.93492276875016,68.77513559657947,
69.07162165668424,……

2.3　第三问理想信道下蒙特卡洛模拟结果

完整结果见 problem3_result_ideal.txt

64.254771248045,63.52888648660662,63.5284584606886,62.982354655653914,

62.49209597012088,63.08677533453148,61.883918933723045,63.892405163461916,
63.751828224054115,63.92798535219872,……,63.276346188088674,63.69613773283893,
62.613602335614964,64.18131912103982,62.763600656402254,64.08181755487253,
64.42250549495809,63.722833228680976,62.661032714747975,61.11432512176272,……

2.4 问题三非理想信道蒙特卡洛模拟结果（含 7 种参数）

此为 CWmin=16，CWmax=1024，最大回退次数 32 次，物理层速度 158.4Mbps 下的模拟结果，完整结果见 problem3_16_1024_32_158.4.txt

36.83053622461003,35.50570760784499,35.99516487686191,35.59380229948058,
35.95813918584998,35.84419612369009,35.36547090800394,35.260379103634286,
35.899135237956315,35.599429614302956,……,35.991049107054785,35.45041039655092,
35.33336438960069,36.12461046970174,35.426309977028005,36.3819879016139,
35.85369147464684,35.73781266378518,36.23420822669498,36.18146744070258,……

此为 CWmin=16，CWmax=1024，最大回退次数 32 次，物理层速度 286.8Mbps 下的模拟结果，完整结果见 problem3_16_1024_32_286.8.txt

46.36769432999539,45.68529473389357,46.02556540714549,44.68083009964789,
45.976142430707554,45.37533362655497,45.47740721507663,45.75063365856003,
45.34861443672611,45.85178770958056,……,45.216078307626915,45.90223373435163,
46.40790908424792,44.928088446348305,45.468532734549726,45.89383355543242,
45.475560858532994,45.81450043733051,45.56598561328725,45.64933306106393,
43.884452450940266,45.42047771544308,45.580755781680146,……

此为 CWmin=16，CWmax=1024，最大回退次数 32 次，物理层速度 455.8Mbps 下的模拟结果，完整结果见 problem3_16_1024_32_455.8.txt

57.358052187808305,55.053603697675825,55.1601870514309,55.76974823983048,
54.94893217675527,54.80393711434892,55.488142511340584,54.917821126636994,
54.790407970960146,55.45207508617789,……,55.797925972099165,54.49046477360876,
56.37024812199584,54.421584922387225,56.28342505233387,56.325756262818956,
54.82310476938647,55.064978788966926,54.80911705081833,55.17283605382332,……

此为 CWmin=16，CWmax=1024，最大回退次数 6 次，物理层速度 158.4Mbps 下的模拟结果，完整结果见 problem3_16_1024_6_158.4.txt

31.422022153938485,30.923220190957355,31.09910635829773,31.266955428938605,
30.82360229217345,31.589147511481073,31.640938315805695,30.871273557545226,
30.633450896295706,31.140323186422133,……,30.6778934691781,31.52426336467853,
31.054265374765137,31.404539171146936,31.94808288904497,31.448044454710676,
30.952952159847985,30.895258765366908,30.94412964385138,31.178657289455476,……

此为 CWmin=16，CWmax=1024，最大回退次数 6 次，物理层速度 286.8Mbps 下的模拟结果，完整结果见 problem3_16_1024_6_286.8.txt

46.201089976819866,46.074510597712404,46.30660720714648,45.55590956313405,
46.236843434753304,45.32656346295624,47.02973052030008,45.536532879365254,
46.697593297184376,45.67276363534936,……,45.689189735367535,45.995914317414936,
45.75835554640782,46.7260802599277,46.77091166336935,46.06213740279884,
46.80923285147499,46.054768450108455,46.61973492185942,46.88703805460673,……

此为 CWmin=32，CWmax=1024，最大回退次数 5 次，物理层速度 158.4Mbps 下的模拟结果，完整结果见 problem3_32_1024_5_158.4.txt

28.125858100702025,27.257284899124272,28.46973068460599,27.565539390198353,
27.847974303170783,28.455403912640993,28.073371008290717,27.69671780700179,
27.47371737347943,28.411366000485312,……,27.69358008371573,27.629512823723484,
27.96918075777387,27.879862430953633,28.07395733733616,28.555301990453618,
28.051153431240568,27.810918801372374,28.39385433206989,28.030370580355687,……

此为 CWmin=32，CWmax=1024，最大回退次数 5 次，物理层速度 286.8Mbps 下的模拟结果，完整结果见 problem3_32_1024_5_286.8.txt

39.44333692257815,39.514073705669695,39.09418527600656,39.68103325044487,
39.81260284387604,40.204475764068285,39.31563658893856,40.09419739339485,
38.911789832888765,39.46094229363377,……,39.609020106291766,38.991837465800174,
39.19475546131576,39.50802464120062,39.60634957319153,38.869117731460825,
39.53053563200248,39.62013269778412,39.458632788415905,40.20429049058356,……

2.5 问题四蒙特卡洛模拟结果（含 7 种参数）

此为 CWmin=16，CWmax=1024，最大回退次数 32 次，物理层速度 158.4Mbps 下的模拟结果，完整结果见 problem4_16_1024_32_158.4.txt

93.26269842645922,93.13449870461933,93.19929805677792,93.22693202328614,
93.22895440324385,93.07300362955124,93.1120373672594,93.1945051010986,
93.2462742935899,93.25303749872246,92.95296529254594,……,93.52714219051879,
93.27656832235586,93.54436450621844,93.13716043426419,93.070616741417,
93.3334643903945,92.85859311943449,93.00798409290323,93.18131202431482,
93.27580244904635,93.27452456288229,……

此为 CWmin=16，CWmax=1024，最大回退次数 32 次，物理层速度 286.8Mbps 下的模拟结果，完整结果见 problem4_16_1024_32_286.8.txt

107.7648753330895,107.7837107253724,107.90562319603339,107.82414534180343,
107.90306024284077,107.81191682851889,107.8755246708462,107.67969226031222,

108.07470683555478,107.92303885882517,……,107.99465704000882,107.84204407391101,
107.75252243199813,107.85791360767678,108.23750897654834,107.72809725242351,
107.42093992874882,107.57343965947808,108.38077977629679,108.11908142460193,……

此为 CWmin=16，CWmax=1024，最大回退次数 32 次，物理层速度 455.8Mbps 下的模拟结果，完整结果见 problem4_16_1024_32_455.8.txt

116.61467750520552,117.22741027751168,116.59542482281921,117.09800379468753,
116.26206190673203,116.21851688757644,116.60070571586729,116.24221858118987,
116.01889535060991,116.5283478555086,……,115.9886806965169,116.45849120157666,
116.55889801213849,116.55677785349023,117.13832202616689,115.88221593985703,
116.70602182767188,116.82173498468772,116.63109807923338,116.63228510074896,……

此为 CWmin=16，CWmax=1024，最大回退次数 6 次，物理层速度 158.4Mbps 下的模拟结果，完整结果见 problem4_16_1024_6_158.4.txt

93.41880909022795,93.2915952940243,93.16022037699449,93.15211918658656,
93.0026605873634,93.18192480847992,92.93265324508872,93.02288604542134,
93.40700258711962,92.86236171752336,92.94224527547684,……,92.90133966406245,
93.30341764305173,93.15577597780461,93.3346370013744,93.52715265969132,
93.05684210615381,93.36746069082973,93.00902469328716,93.53623551186378,
93.34381386361025,……

此为 CWmin=16，CWmax=1024，最大回退次数 6 次，物理层速度 286.8Mbps 下的模拟结果，完整结果见 problem4_16_1024_6_286.8txt

107.76110019861937,108.01780702927178,107.72793049724655,108.01398871757645,
108.10962499398961,107.98423621224687,108.08277229712247,107.56346583932708,
107.85932260202148,107.95282894392751,……,108.25046687751563,107.98151213019761,
107.83688807082613,107.70400272238338,107.27560204396548,