



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	中南大学
参赛队号	23105330383
队员姓名	1. 柳昭宇
	2. 姚锦
	3. 周浩堋

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：　WLAN 中多 BSS 场景下的信道竞争建模与性能分析

摘　要：

无线局域网 WLAN 是日常生活中广泛使用的无线通信服务，为了避免信道冲突，WLAN 使用分布式协调功能 DCF 提供基于竞争的信道接入功能。WLAN 组网中的多 BSS 建模问题是该领域的热门研究之一，由于同频 BSS 之间存在互听或不互听的情况，当节点发送数据碰撞时可能导致传输失败。针对复杂多 BSS 场景的建模和性能求解方法具有十分重要的应用价值。

针对**问题 1**，我们考虑 2BSS 互听的场景，其存在同频干扰，当节点并发时数据传输失败。我们基于 Bianchi 模型，建立二维马尔科夫模型表示单个 AP 节点的回退随机过程，并结合系统中数据传输和并发碰撞情况建立高次非线性方程，使用**松弛牛顿法**求解稳态概率，从而评估计算系统的吞吐，最后编写仿真器验证模型的精度。最终求解结果为**信道利用率 14.74%**以及**稳态吞吐量 67.174Mbps**，对应**仿真计算结果 13.69%和 62.389Mbps**, **信道利用率差距为$\Delta\eta = 1.05\%$**，表明模型有较好的准确度。

针对**问题 2**，同样考虑 2BSS 互听的场景，但是当节点并发时数据传输成功。因此只需将问题 1 中的模型简化为只存在一次回退的情况，此时可**直接求解稳态概率**，但是在计算吞吐时需要考虑并发时信道同时传输数据的情况。最终求解结果为**信道利用率 25.63%**以及**稳态吞吐量 70.562Mbps**，对应**仿真计算结果为 24.02%和 66.139Mbps**, **信道利用率差距为$\Delta\eta = 1.61\%$**，表明模型有较好的准确度。

针对**问题 3**，我们考虑 2BSS 不互听的场景，节点可能在信道上同时或先后传输数据，当两个 AP 发包时间交叠时数据传输失败，同时还需要考虑 10%的丢包率。我们建立二维马尔科夫模型，将失败概率表示为碰撞概率和丢包概率的组合，然后将发包时间换算为回退数，并结合隐藏节点在发包时间内回退的稳态概率对碰撞情况建模。通过**松弛牛顿-自适应扰动模拟退火混合算法**求解稳态概率，从而评估计算系统的吞吐。最终求解结果为**信道利用率 6.86%**以及**稳态吞吐量 31.269Mbps**，对应**仿真计算结果为 9.93%和 45.283Mbps**, **信道利用率差距为$\Delta\eta = 3.07\%$**。同时我们根据**附表 6** 改变了相关参数进行计算和对比分析。

针对**问题 4**，我们考虑 3BSS 场景，其中节点之间存在互听和不互听的情况。我们针对不同的节点分别建立二维马尔科夫模型，并考虑节点之间的关联，对各节点的碰撞和传输进行建模。通过**贝松自混算法**求解稳态概率，并在计算吞吐时考虑节点并发和发包时间覆盖情况。最终求解结果为**信道利用率 21.63%**以及**稳态吞吐量 98.567Mbps**，对应**仿真计算结果为 24.27%和 110.627Mbps**, **信道利用率差距为$\Delta\eta = 2.64\%$**。同时根据**附表 6** 改变了相关参数进行计算和对比分析。

关键词： DCF 多 BSS 建模 二维马尔科夫模型 数值分析 混合算法

目录

一、 问题重述

1.1 问题背景

无线局域网（WLAN），通常称为 Wi-Fi，已经成为我们日常生活和工作中不可或缺的一部分。它提供了一种低成本、高吞吐量和极其便利的无线通信服务，使我们能够轻松地连接到互联网，同时也促进了各种智能设备之间的互联互通。

WLAN 的基本构成单元是基本服务集（BSS）。在一个特定的覆盖区域内，我们可以找到一组站点（STA），它们与一个专门负责管理 BSS 的无线接入点（AP）相连接，形成了一个 BSS。这种连接过程被称为 STA 关联到 AP。AP 可以采用各种形式，如无线路由器、WiFi 热点等，而 STA 则包括了手机、笔记本电脑、物联设备等各种类型的终端设备。

在 WLAN 中，数据的传输通常分为两个方向：下行方向和上行方向。下行方向是指 AP 向 STA 发送数据，而上行方向则是 STA 向 AP 发送数据。需要注意的是，每个节点（无论是 AP 还是 STA）的发送和接收操作不能同时进行，这是由于无线信道的限制所决定的。

为了有效地管理共享的无线信道并避免数据冲突，WLAN 采用了一种称为载波侦听多址接入/退避（CSMA/CA）的机制，这一机制实现了分布式协调功能（DCF）。简言之，这意味着在发送数据之前，节点会先监听信道，以确保它是空闲的。如果信道被其他节点占用，节点会进行一定的等待，以避免碰撞发生。这个过程保证了 WLAN 中数据传输的可靠性和有效性。

1.2 问题描述

1.2.1 两BSS互听

考虑 2 个 BSS 互听的场景，仅下行，即两个 AP 分别向各自关联的 STA 发送数据。以 AP1->STA1 方向的数据传输为例，其会受到相邻 BSS2 的干扰，对于 STA1 来说，AP1->STA1 是信号，AP2->STA1 是干扰。对于 AP2->STA2 情况类似。假设 ACK 一定能发送成功。根据节点之间的 RSSI 估算两个 AP 并发时的 SIR，考虑不同的情景进行建模。

问题 1：

假设 AP 发送包的载荷长度为 1500Bytes（1Bytes = 8bits），PHY 头时长为 13.6μs，MAC 头为 30Bytes，MAC 头和有效载荷采用物理层速率 455.8Mbps 发送。AP 之间的 RSSI 为-70dBm。大部分时候只有一个 AP 能够接入信道，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。假设并发时的 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。（参数参考附录 4，可编写仿真器验证模型精确度）

问题 2：

假设两个 AP 采用物理层速率 275.3Mbps 发送数据，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。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。（参数参考附录 4，可编写仿真器验证模型精确度）

1.2.2 两BSS不互听

在 AP 密集部署时，同频 AP 之间的距离远，AP 间 RSSI 低于 CCA 门限，不互听。AP 认为信道空闲，因此总是在退避和发送数据。这是 Wi-Fi 里常见的隐藏节点问题，详见附录。可以预见的是，有很大概率出现二者同时或先后开始发送数据的情况。接收机解调信号时，PHY 头的前面部分码元用于 Wi-Fi 信号识别、频率纠错、定时等功能，叫作前导（Preamble）。当信号包先到时，接收机先解信号包的 Preamble 并锁定，干扰包被视为干扰，信号包是否接收成功由 SIR 决定；当干扰包先到时，接收机先锁定到干扰包的 Preamble，错过信号包的 Preamble，导致信号包无法解调。小信号屏蔽算法能有效解决这个问题，因为信号包 RSSI 一般大于邻小区的干扰包，接收机在信号包到达时转为锁定 RSSI 更大的信号包，此时信号包能否接收成功同样也由 SIR 决定。由此可以得知，在 SIR 比较小的情况下，如果信号包和干扰包在时间上有交叠时，一定会导致本次传输的失败。

问题 3：

假设 AP 间 RSSI 为-90dBm，AP 发送包的载荷长度为 1500Bytes，PHY 头时长为 13.6μs，MAC 头为 30Bytes，MAC 头和载荷采用物理层速率 455.8Mbps 发送。Bianchi 模型假设理想信道，实际上，无线传输环境是复杂多变的，当有遮挡物或者人走动时，无线信道都可能会快速发生比较大的变化。实测发现，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。假设因信道质量导致的丢包率$P_e = 10\%$。当两个 AP 发包在时间上有交叠时，假设 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。（参数参考附录 4 和 6，可编写仿真器验证模型精确度）

1.2.3 三BSS

问题 4：

考虑 3BSS 场景，其中 AP1 与 AP2 之间，AP2 与 AP3 之间 RSSI 均为-70dBm，AP1 与 AP3 之间 RSSI 为-96dBm。该场景中，AP1 与 AP3 不互听，AP2 与两者都互听，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。假设三个 AP 发送包的载荷长度为 1500Bytes，PHY 头时长为 13.6μs，MAC 头为 30Bytes，MAC 头和载荷采用物理层速率 455.8Mbps 发送。假设 AP1 和 AP3 发包时间交叠时，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。（参数参考附录 4 和 6，可编写仿真器验证模型精确度）

二、 问题分析

2.1 针对问题1的分析

问题 1 要求我们针对两个 BSS 互听，且两个 AP 同时发送数据会发生同频干扰而失败的情况建立 2BSS 系统模型，并使用数值分析方法求解、评估系统的吞吐并编写仿真器验证模型精确度。此问题需要结合 Bianchi 模型进行建模，需要着重考虑两个 AP 同时回退到 0 而同时发生数据会传输失败的情况如何在模型中表示。另外模型的求解可能涉及到高次非线性方程的求解，这时传统的精确解求解方法失效，需要使用数值分析方法求解得到τ和p的值，评估系统的吞吐时需要结合前面求得的τ和p值进行计算，最后编写仿真器验证模型精确度。

2.2 针对问题2的分析

问题 2 要求我们仍然是在两 BSS 互听且其他条件与问题 1 相同的前提下，考虑当两个 AP 并发时，数据传输都能成功的情况。对该 2BSS 系统进行建模，用数值分析方法求解并评估系统的吞吐，最后编写仿真器验证模型精确度。由于两个 AP 同时并发时也都能成功传输，不需要再考虑碰撞的问题，所以问题 2 的模型相对于问题 1 来说有所简化，可以基于 Bianchi 模型进行修改得到问题 2 的模型，通过模型化简求得结果。

2.3 针对问题3的分析

问题 3 的前提是两个 BSS 不互听，即 AP 一直认为信道空闲，总是回退或发送数据，这种情况下，两个 AP 之间有很大概率同时或先后开始发送数据，并且问题 3 假设 SIR 比较小，所以两个 AP 在发包时如果在时间上有交叠，则两个 AP 的发包均失。同时设定当仅有一个 AP 发送数据时，也会因信道质量导致有 10%的丢包率，如何在模型中表示两个 AP 发包时间交叠会导致失败以及一个 AP 发包有 10%的丢包率，是一个建模中的难点。而问题 3 的模型会比问题 1 更加复杂，所以求得的难度以及时间上可能会加大，可以考虑更为合适的数值求解方法，最后求出相关参数后评估系统的吞吐量并编写仿真器验证模型精确度。

2.4 针对问题4的分析

问题 4 考虑更为复杂的 3BSS 场景，其中 AP1 和 AP3 之间不互听，并且 SIR 比较大，2 个 AP 在发包时如果在时间上有交叠，都能发送成功；而 AP2 与另外 2 个 AP 之间互听，存在同频干扰，并发数据时会导致失败。此问题需要在 Bianchi 模型的基础上对各个 AP 进行建模，并要充分考虑各个 AP 之间的相互关联，从而推导碰撞和传输的情况。由于涉及到更多的 BSS 数量以及互听和不互听的情况，其模型的复杂度将会更高，可能需要优化数值分析方法。最后求出相关参数后评估系统的吞吐量并编写仿真器验证模型精确度。

三、 模型假设及符号说明

3.1 模型假设

1. 假设每个 BSS 中只存在一个 AP 和与其关联的一个 STA。
2. 假设每个节点的发送和接收不能同时发生。
3. 假设信道质量理想，发送数据不会丢包（除问题 3）。
4. 假设接收节点发送 ACK 一定成功。
5. 假设发送节点在传输数据成功或丢弃数据帧后马上开始下一个数据帧的回退。
6. 假设不同 BSS 之间只存在同频干扰问题，不存在异频干扰的情况。
7. 假设需要评估的是系统达到稳态时的信道利用率和吞吐。

3.2 主要符号说明

注：此为本文的主要缩略语说明，其他缩略语解释详见正文部分

简称	全称	意义
AP	access point	无线接入点
ACK	Acknowledgement	确认
ACKTimeout		确认超时
BSS	basic service set	基本服务集
CCA	clear channel assessment	信道可用评估
CSMA/CA	carrier sense multi-access and collision avoidance	载波监听多址接入/退避
CW	contention window	竞争窗口
DCF	distributed coordination function	分布式协调功能
DIFS	DCF inter-frame space	DCF 帧间距
MAC	medium access control	媒体控制
PHY	physical	物理层
RSSI	received signal strength indication	接收信号能量强度
SIFS	short inter-frame space	短帧间距
SIR	signal to interference ratio	信干比
STA	station	站点
WLAN	wireless local area network	无线局域网

注：此为本文的主要符号说明，其他符号解释详见正文部分。

符号	意义
T_e	空闲时隙
T_s	成功时隙

T_c	失败时隙
slotTime	回退时长
H	数据帧头传输时间
$E[P]$	数据帧的有效载荷传输时长
$E^*[P]$	冲突时较长数据帧的有效载荷传输时长
W_i	数据第i次发送时的回避窗口大小
τ	节点稳态传输概率
p	节点传输失败的概率
P_c	节点碰撞概率
P_e	丢包率
P_{tr}	虚拟时隙上至少一个节点传输的概率
P_s	有节点传输数据时传输成功的概率
η	信道利用率
S	系统的吞吐
V	易受干扰时段的换算回退数

四、 模型建立与求解

4.1 问题1模型的建立与求解

4.1.1 模型建立

无线局域网 WLAN 使用分布式协调功能 DCF 机制提供信道接入功能，每个节点接入信道进行传输的过程分为信道可用评估、随机回退和数据传输 3 个阶段。当一个节点打算发送数据时，首先进行一个固定时长的 DIFS 时段载波侦听，判断信道是否空闲。当信道空闲时，节点进行随机回退，即节点从一个[0, CW-1]中选取一个随机数作为回退数，CW 为竞争窗口，每次回退时长为 slotTime，在随机回退时段节点持续监听信道，如果期间信道变繁忙，则节点将回退暂停，直到信道在一个 DIFS 时长后变为空闲，再继续前面没有回退完的时间。如果节点回退到 0，则代表可以开始数据传输，发送节点发送一个数据帧，接收节点成功接收后等待 SIFS 时间后回复 ACK 确认帧，如果发送节点收到 ACK 则说明数据发送成功，如图 1(a)；如果由于冲突导致发送数据帧没被接收节点接收，或者 ACK 发送失败，或者 ACK 没有被发送节点收到，则说明数据传输失败，发送节点需要在等待超时时间 ACKTimeout 后重传数据，如图 1(b)。

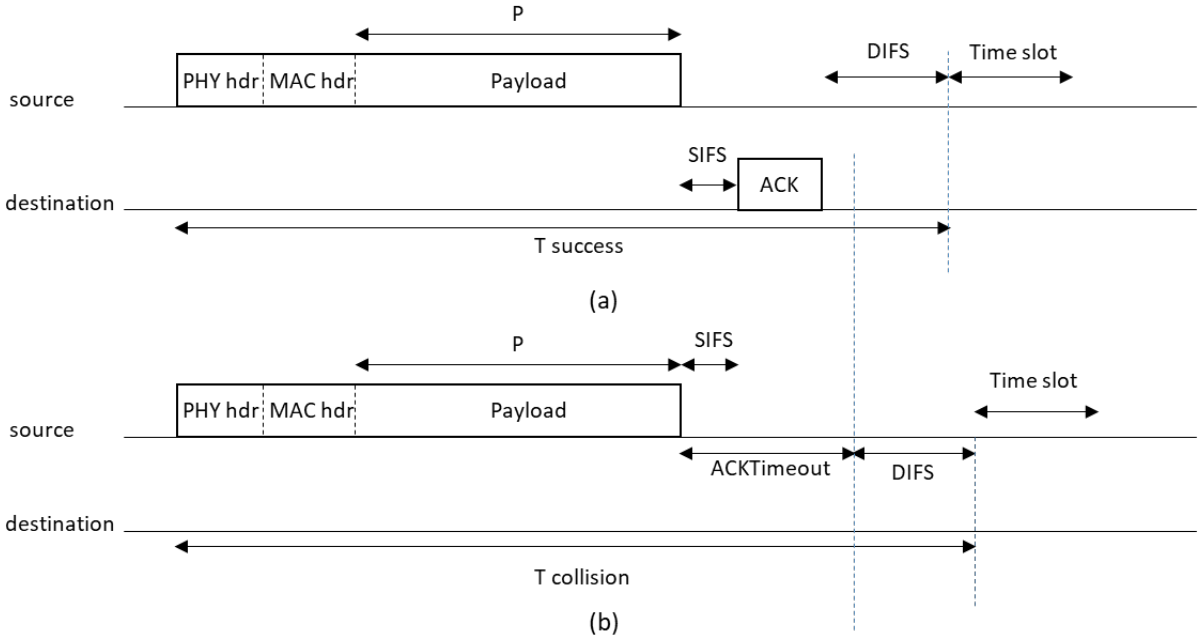

图 1 帧序列：(a) 发送成功 (b) 发送失败

那么信道可能处于三种状态：空闲、成功传输、失败传输，如图 2 所示，将每个状态看作一个虚拟时隙，那么信道在空闲时隙T_e、成功时隙T_s、失败时隙T_c三种虚拟时隙中转化。

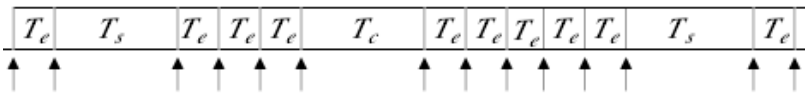

图 2 信道状态

每种虚拟时序的时长表示为

$$T_e = slotTime \tag{1}$$

$$T_s = H + E[P] + SIFS + ACK + DIFS \tag{2}$$

$$T_c = H + E^*[P] + DIFS + ACKTimeout \tag{3}$$

其中H为数据帧头传输时间，包括 MAC 层头和物理层头，$E[P]$为数据帧的有效载荷传输时长，$E^*[P]$为发生冲突时较长数据帧的有效载荷传输时长，PHY 头时长固定，MAC 头和有效载荷的发送时长由其字节长度除以物理层速率得到，即

$$H = t_{phy} + t_{mac\ header} = t_{phy} + L_{mac\ header}/rate \tag{4}$$

$$E[P] = t_{payload} = L_{payload} / rate \tag{5}$$

随机回退采用二进制指数退避算法确定回退时间，竞争窗口 CW 的初始值为 CW_{min}，每次数据传输失败后重传数据帧时，CW 翻倍。如果重传达到最大退避阶数 m，CW 达到了 CW_{max}，则保持此值。每次数据传输成功时 CW 重置，开始下一个数据帧的回退。若传输连续失败，重传次数达到 r 后，数据帧被丢弃，CW 重置传输下一个数据帧，CW 在第i次重传阶段可表示为

$$\begin{cases} \mathrm{W_i} = 2^{\mathrm{i}}\mathrm{W_0} & 0 \le i \le m \\ \mathrm{W_i} = 2^{\mathrm{m}}\mathrm{W_0} & m \le i \le r \end{cases} \tag{6}$$

问题 1 中考虑在一个 WLAN 组网中存在 2 个 BSS，每个 BSS 中分别有一个 AP 向其关联的 STA 下行发送数据，如图 3 所示。同时，2 个 BSS 之间互听，当一个 AP 接入信道后，另一个 AP 能监测到信号繁忙而停止回退，因此大部分时间数据传输成功，但是存在同频干扰的情况，当 2 个 AP 同时回退到 0 时，都检测到信道空闲，从而同时发送数据，在本问的场景下，假设并发时的 SIR 较低，两个 AP 的数据传输都会失败。要求针对该 2BSS 系统进行建模，并评估系统的吞吐。

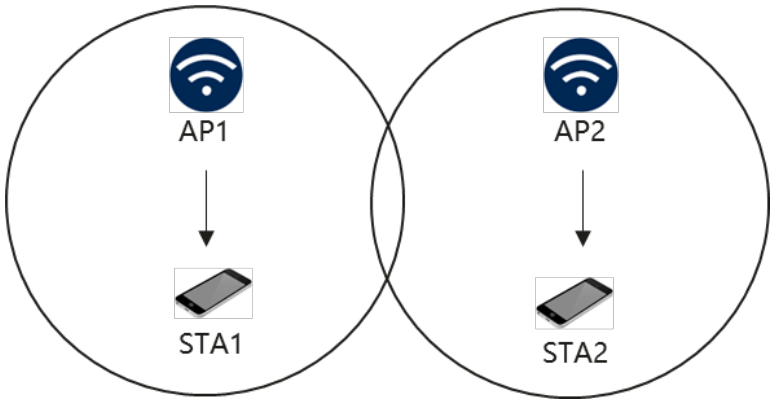

图 3 两同频 BSS 场景

参考 Bianchi 模型[1][2][4]，当整个系统处于稳态时，采用一个二维的马尔科夫模型$\{s(t), b(t)\}$表示单个 AP 节点在退避过程中所处的状态，如图 4 所示。$b(t)$和$s(t)$代表t时刻，即一个虚拟时隙的开始时刻，节点在其退避随机过程中的退避计数和退避阶数。

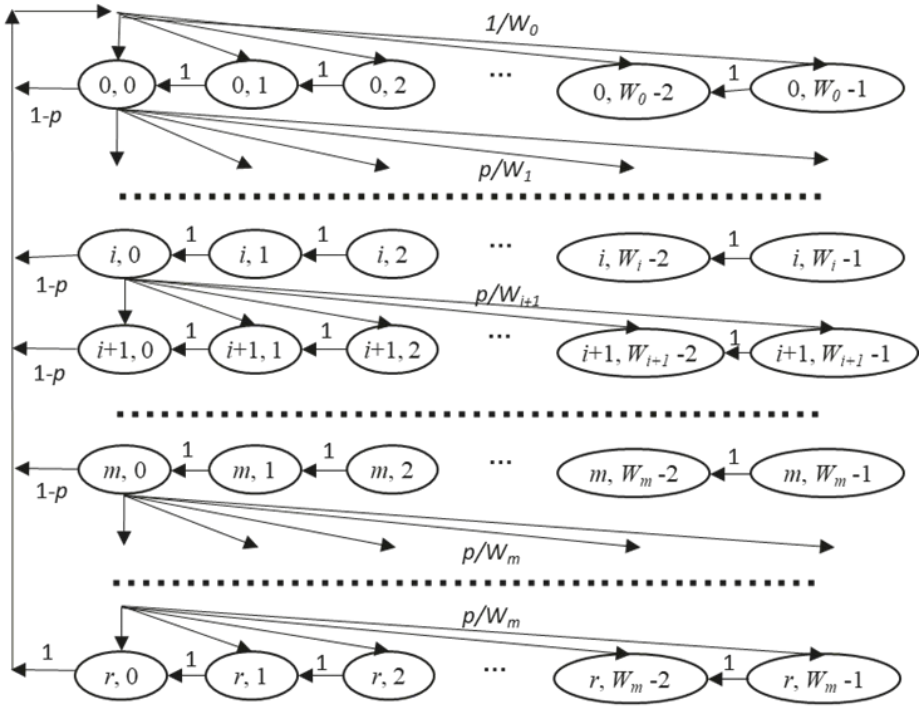

图 4 二维马尔科夫模型

在整个系统中，用p表示节点发送失败的概率，本问的场景下，p即多个节点同时回退到 0 从而导致同步发送，即节点发生碰撞的概率。马尔科夫链的状态转移过程可由式(7)到式(10)表示。

节点经过 1 个空闲间隙，回退数减 1，直到减为 0，每次回退的概率为 1。

$$P\{i,k|i,k+1\}=1\ ,k\in[0,W_i-2],i\in[0,r] \tag{7}$$

节点回退数达到 0，开始数据传输，如果不发生碰撞，即$1-p$的概率，发送成功，然后马上开始下一个数据帧的回退过程。

$$P\{0,k|i,0\}=(1-p)/W_0\ ,k\in[0,W_0-1],i\in[0,r] \tag{8}$$

如果在第$i-1$次回退到 0 发送数据然后发生碰撞，即p的概率，发送失败，则节点进入第i阶回退过程。

$$P\{i,k|i-1,0\}=p/W_i,k\in[0,W_i-1]\ \ i\in[1,r] \tag{9}$$

当节点到达最大的传输次数以后，无论成功还是失败，节点都会开始下一数据帧的回退。

$$P\{0,k|r,0\}=1/W_0,k\in[0,W_0-1] \tag{10}$$

该马尔可夫链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。从任何状态出发，都能到达另一状态，具有常返性。因此其具有稳态解，表示为

$$b_{i,k}=\lim_{t\to\infty}P\{s(t)=i,b(t)=k\},i\in[0,m],k\in[0,W_i-1] \tag{11}$$

对于一次发送失败的情况，状态$b_{i-1,0}$转移到状态$b_{i,0}$，表示为

$$b_{i,0}=b_{i-1,0}*\left(\mathrm{p}*\frac{1}{Wi}+\mathrm{p}*\frac{1}{Wi}*1+\cdots+\mathrm{p}*\frac{1}{Wi}*1^{Wi-1}\right)=p*b_{i-1,0}\ ,0<i\le r \tag{12}$$

对于任一状态$b_{i,k}$, 若$0<i<r$，则是从一次发送失败的状态，通过竞争窗口加倍之后转移过来的。若$i=0$，则是从任一阶发送成功，或达到重传次数限制后转移过来的，因此有

$$b_{i,k}=\begin{cases}b_{i-1,0}*p*\frac{W_i-k}{W_i} & 0<i\le r\\ \frac{W_i-k}{W_i}*\left(\sum_{j=0}^{r-1}(1-p)b_{j,0}+b_{r,0}\right) & i=0\end{cases} \tag{13}$$

将式(12)代入式(13)中，可得

$$b_{i,k} = \frac{W_i - k}{W_i} * b_{i,0}, 0 \le i \le r, 0 \le k \le W_i - 1 \tag{14}$$

根据马尔科夫链的性质，所有稳态的概率之和为 1，因此有

$$1 = \sum_{k=0}^{W_i-1} \sum_{i=0}^{r} b_{i,k} = \sum_{i=0}^{r} b_{i,0} \sum_{k=0}^{W_i-1} \frac{W_i - k}{W_i} = \sum_{i=0}^{r} b_{i,0} \frac{W_i + 1}{2} \tag{15}$$

联立求解可得

$$\mathrm{b}_{00} = \begin{cases} \dfrac{2(1-p)(1-2p)}{(1-2p)(1-p^{r+1}) + W_0(1-p)(1-(2p)^{r+1})}, r \le m \\ \dfrac{2(1-p)(1-2p)}{W_0(1-(2p)^{m+1})(1-p) + (1-2p)(1-p^{r+1}) + W_0 2^m p^{m+1}(1-p^{r-m})(1-2\mathrm{p})}, m < r \end{cases} \tag{16}$$

节点随机回退到 0 时，一定会发送数据，定义稳态传输概率τ，表示为

$$\tau = \sum_{0}^{r} b_{i,0} = b_{0,0} * \frac{1-p^{r+1}}{1-p} \tag{17}$$

在本问的场景中，n=2 个 AP 站点都以概率τ传输数据到同一个信道，并且两个 BSS 互听，因此当一个虚拟时隙最多只有一个站点发送数据时，不会发生碰撞，表示为

$$p = 1 - (1-\tau)^{n-1} = 1 - (1-\tau)^{2-1} = \tau \tag{18}$$

令P_{tr}为在一个虚拟时隙上最少有一个节点传输数据的概率，表示为

$$P_{tr} = 1 - (1-\tau)^n = 1 - (1-\tau)^2 \tag{19}$$

令P_s为在有节点传输数据的情况下，传输成功的概率，表示为

$$P_s = \frac{n\tau(1-\tau)^{n-1}}{P_{tr}} = \frac{2\tau(1-\tau)}{P_{tr}} \tag{20}$$

系统的吞吐是信道在单位时间内发送数据有效载荷的比特数，单位为 bps，可以由信道的利用率η与物理层速率 rate（单位 bps）的乘积表示，

$$\mathrm{S} = \frac{\mathrm{E}[\text{一个时隙内传输的有效载荷发送时长}]}{\mathrm{E}[\text{一个时隙长度}]} * \text{物理层速率} = \eta * rate \tag{21}$$

$$\eta = \frac{P_s P_{tr} E(p)}{(1-P_{tr})T_e + P_s P_{tr} T_s + P_{tr}(1-P_s)T_c} \tag{22}$$

根据问题 1 给定的参数，确定三种虚拟时隙T_e、T_s和T_c，并根据模型推导出处于每个虚拟时隙的概率，从而最终评估出系统的吞吐。

4.1.2 模型求解

1) 松弛牛顿法

牛顿法（Newton's Method）是一种用于求解非线性方程的数值迭代方法，它是一种迭代法，通过不断改进初始猜测值来逼近方程的根。是解决非线性方程和优化问题中的一种强大方法。

牛顿法的基本原理：对于一个一元非线性方程$f(x) = 0$,我们希望找到一个x值，使得$f(x)$足够接近 0，即找到$f(x)$的根。牛顿法基于以下观察来寻找这个x值：首先如果我们有一个近似的根x_0，则$f(x_0)$的值应该接近零。然后通过计算$f(x_0)$的导数$f'(x_0)$来估计x_0附近的切线，找到切线与x轴之间的交点x_1,则这个交点x_1就是更好的根估计，不断迭代这个过程，就可以逐步逼近真实的根。

松弛牛顿法（Relaxed Newton's Method）是标准牛顿法的一个变种。松弛牛顿法引入了一个松弛因子（relaxation factor）来控制迭代步骤的幅度，从而可以改善算法的收敛性和稳定性。

松弛牛顿法的步骤如下：

1. 选择一个初始猜测解x_0。
2. 计算当前猜测解x_k处的函数值：$f(x_k)$
3. 计算当前猜测解x_k处的导数值：$f'(x_k)$
4. 使用松弛牛顿法的迭代公式来计算新的猜测解x_{k+1}：

$$x_{k+1} = x_k - \alpha \frac{f(x_k)}{f'(x_k)} \tag{23}$$

式(23)中，x_k表示第 k 次迭代的估计解。$f(x_k)$表示函数在x_k处的值。$f'(x_k)$表示函数在x_k处的导数值。α表示松弛因子，决定迭代步骤的幅度，较小的因子会使迭代更加稳定，但也会使得收敛更慢，较大的因子可能导致快速收敛但不稳定的情况。

5. 检查是否满足收敛准则，若$|f(x_{k+1})| < tol$，（tol为设置的容差），则满足收敛准则，停止迭代；否则，继续迭代。
6. 重复步骤 2 到 5，直到满足收敛准则或达到最大迭代次数。

牛顿松弛法的伪代码如表 1 所示：

表 1 松弛牛顿法伪代码

Algorithm 1: Relaxation Newton Method

```
        Input: 初始猜测 x0，容差 tol，最大迭代次数 max  iter, 松弛因子 alpha
        Output: 近似解 x
1:      Initialize：x=x0
        for i in range(max_iter):
          //计算 x 处的函数值和导数值
2:        Calculate f(x)
3:        Calculate df(x)
          //使用迭代公式计算新的猜测解
4:        x_new = x - alpha * f(x) / df(x)
          //检查是否满足收敛准则
5:        if |f(x_new)| < tol:
            break
          else:
            x= x_new
6:      return x
```

牛顿松弛法的伪代码如图 5 所示：

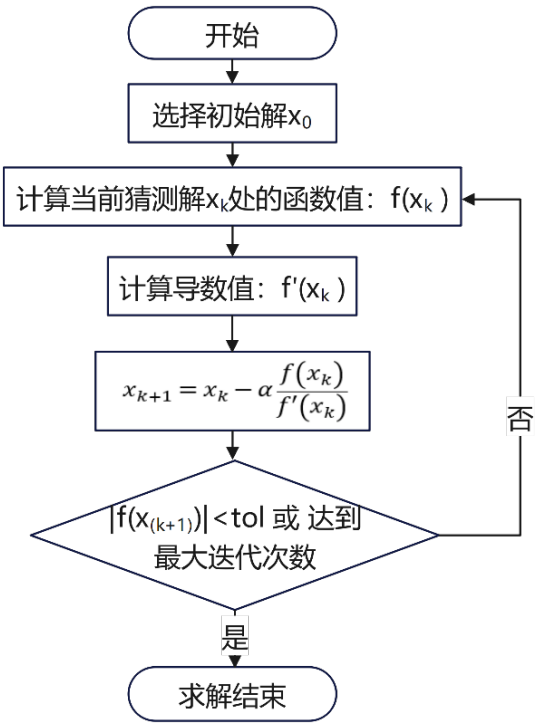

图 5 松弛牛顿法算法流程图

2) 松弛牛顿法求解模型

根据题意问题 1 所建立的模型，可以求得节点处于b_{00}状态的概率如式(16),节点在一个时隙发送数据帧的概率τ如式(17)，2 个节点的条件碰撞概率p，如式(18).

由题目可知，r是最大重传次数，为 32。W_0是初始竞争窗口大小，为 16。m 是最大退避阶数，为 6，故 m<r。将相关参数值带入后，可以得到：

$$b_{00}=\frac{2(1-p)(1-2p)}{16(1-(2p)^7)(1-p)+(1-2p)(1-p^{33})+1024p^7(1-p^{26})(1-2p)} \tag{24}$$

$$\tau=b_{0,0}*\frac{1-p^{33}}{1-p} \tag{25}$$

$$p=\tau \tag{26}$$

化简可得：

$$2050\tau^{35}-1029\tau^{34}+2\tau^{33}-1024\tau^8-18\tau^2+21\tau-2=0 \tag{27}$$

这是一个关于τ的一元非线性方程，τ最高次为 35 次，使用常规的精确解求解方法极难求解，故考虑使用数值分析方法松弛牛顿法求解。

首先为了使得松弛牛顿法能够稳定收敛且不会耗时太长，同时提高结果的精度，将松弛因子α设为 0.5，将最大迭代次数设为 10000 代，将结果容差tol设为1×10^{-6}。

考虑到非线性方程可能存在多解的情况，同时考虑到τ是概率，取值范围为[0,1]，故将初始猜测解 τ_0以 0.01 为步长，在[0,1]的范围内遍历调用松弛牛顿法，观察求得的估计近似解的分布情况：

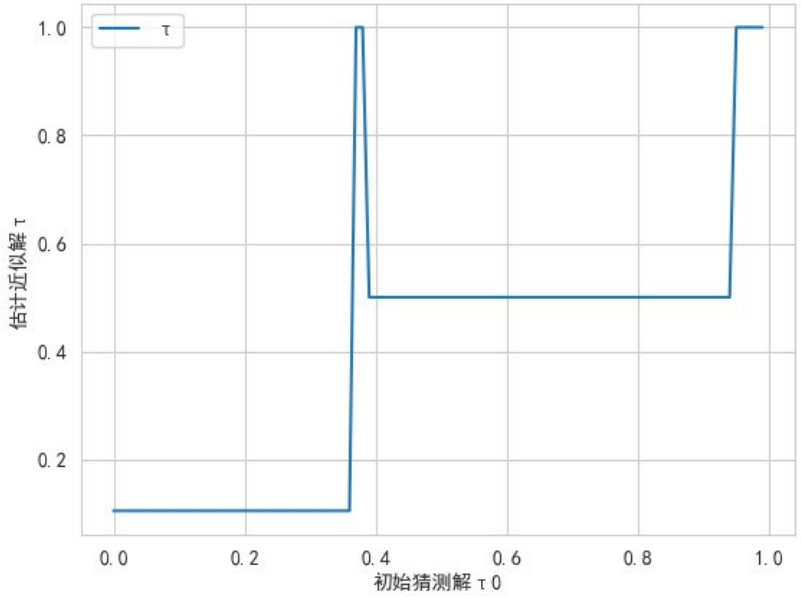

图 6 估计近似解分布图

由图 6 可以看出τ_0在[0,1]范围内变动时，估计近似解x的取值大抵有 3 类：0.10462 附近、0.5 附近、1 附近。

经过验证，$\tau = 0.5$、$\tau = 1$均可以使得式(27)等于 0，但是结合τ的现实意义来看，τ是节点在一个时隙发送数据帧的概率，且由式(26)知 p=τ。若$\tau = 0.5$，则 p=τ=0.5，意味着节点在某个时隙不是在发送数据就是在碰撞，没有了空闲状态，不符合题意和实际情况，故舍去$\tau = 0.5$。若$\tau = 1$，意味着节点在某个时隙总在发送数据，也不符合题意和实际情况，故舍去$\tau = 1$。当$\tau = 0.10462$时，符合题意和实际情况，为了提高τ的精度，进一步实验：

将结果容差tol设为1×10^{-10}，松弛因子和最大迭代次数均保持不变，将τ_0设为 0.10462，记录松弛牛顿法每次迭代得到的 τ，最终在迭代 36 次后，得到最终的估计近似解 τ= 0.10462063228。画出 τ 的迭代求解图。

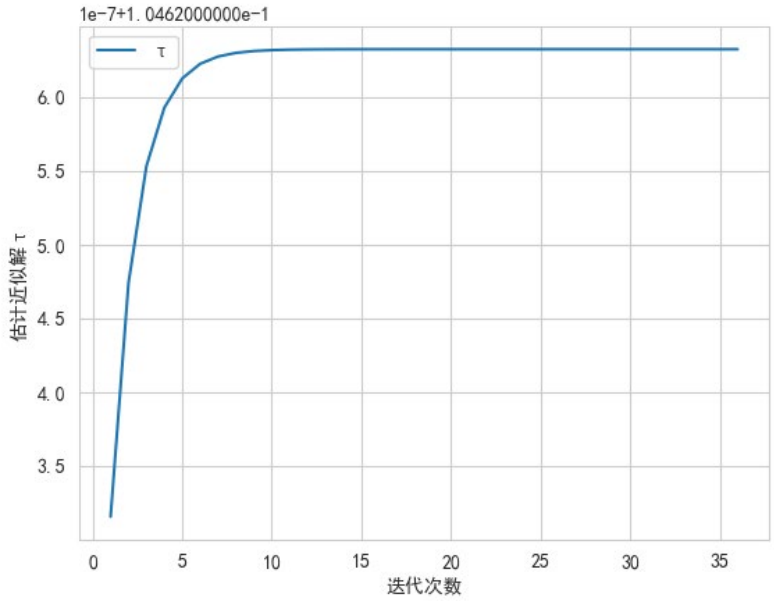

图 7 τ 的迭代求解图

由图 7 可看出 τ 的迭代收敛速度先快后慢，到达第 10 代后收敛速度变慢，τ 值趋

于稳定。

$p = \tau = 0.10462063228$，根据p和τ可以推导出$P_{tr} = 0.1983, P_s = 0.9448$。

表 2 问题 1 参数列表

参数名称	值
载荷长度	1500Bytes
PHY 头时长	13.6us
MAC 头长度	30Bytes
物理层速率 rate	455.8Mbps
ACK 时长	32μs
SIFS 时长	16μs
DIFS 时长	43μs
SLOT 时长	9μs
ACKTimeout 时长	65μs
CW min	16
CW max	1024
最大重传次数	32

根据表 2 给定参数，计算得到$E[p] = 26.327us, T_e = 9us, T_s = 131.454us, T_c = 148.454us$。代入信道利用率和吞吐量的计算公式中可得最终结果为信道利用率$\eta = 14.74\%$以及稳态吞吐量$S = 67.174Mbps$。

同时，我们用 C++语言建立程序进行仿真，程序模型严格参照问题 1 的模型假设：AP 信号塔之间互听，在两个 AP 信号塔的回退数同步回退到 0 的时候，AP 信号塔进入碰撞状态，信号作废；在其中一个 AP 信号塔(AP1)的回退数为 0，另外一个(AP2)不为 0 的时候，AP1 进入发射状态，并占据信道$Ts(us)$，此时 AP2 停止回退操作，冻结 AP2 状态，在$Ts(us)$之后，统计一次信号传输的有效荷载，AP1 的状态从发射状态进入初始回退状态，准备进行新一轮的数据传输，同时 AP2 解封，进如回退状态；在两个 AP 信号塔回退数都不为 0 的时候，此时信道为空，AP 信号塔正常进行回退操作，直至其中至少有一个回退数为 0。

此外，关于仿真部分系统的稳态吞吐量计算，我们在程序中设定如下：程序中设有模拟时间$nTim$，以 us 为单位，我们让程序先运行$bTim$，$bTim$的选定足够大，保证系统能够进入稳态，随后从$bTim$时刻开始逐一统计每次系统的有效荷载传输，得到总量T，随后，系统在每个时刻的吞吐量用$S = T/(nTim - bTim)$ 得到。

最终，我们将系统运行了 2000000 us，按照上述操作得到最后系统的信道利用率以及稳态吞吐量：$\eta = 13.69\%, S = 62.389Mbps$。与通过数据分析方式得到的结果对比，我们发现在信道利用率上差距为$\Delta\eta = 1.05\%$，较好地仿真了该模型。

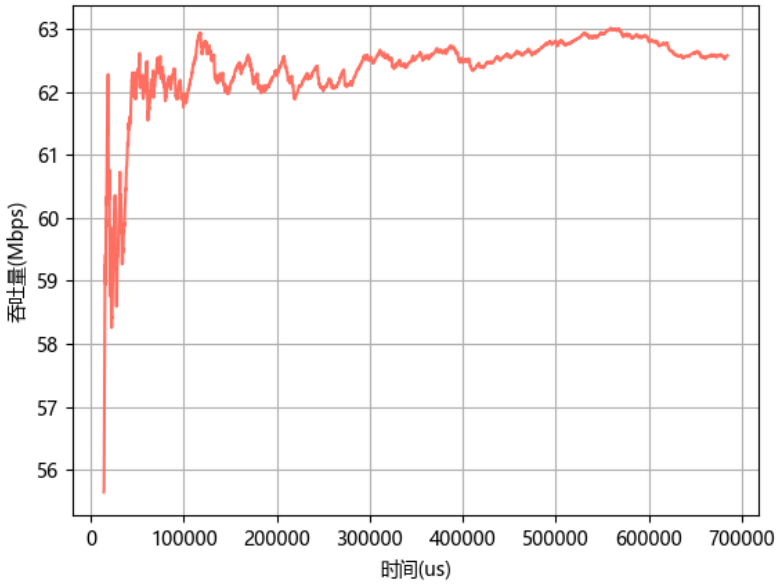

图 8 问题 1 仿真系统吞吐量变化曲线

4.2 问题2模型的建立与求解

4.2.1 模型建立

问题 2 与问题 1 都考虑在一个 WLAN 组网中存在 2 个 BSS 并且每个 BSS 中分别有一个 AP 向其关联的 STA 下行发送数据，并且 2 个 BSS 之间互听的场景。与问题 1 不同之处在于，当 2 个 AP 同时回退到 0 并发时，两个终端接收到数据的 SIR 较高，数据传输都能成功。要求针对该 2BSS 系统进行建模，并评估系统的吞吐。

由于 2 个 AP 并发时，数据仍然能传输成功，因此可以认为节点每次发送数据帧只有一次回退。在问题 1 的二维马尔科夫模型中，只存在$i = 0$的回退阶段，如图 9 所示。

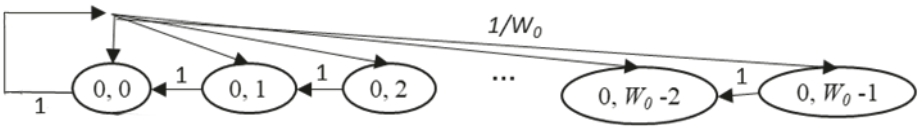

图 9 $i = 0$时的二维马尔科夫模型

马尔科夫链的状态转移过程可由式(28)到式(29)表示

$$P\{0,k|0,k+1\} = 1\,,k \in [0,W_0 - 2] \tag{28}$$

$$P\{0,k|0,0\} = 1/W_0\,,k \in [0,W_0 - 1] \tag{29}$$

对于任一状态$b_{0,k}$，可以表示为

$$b_{0,k} = \frac{W_0 - k}{W_0} * b_{0,0}, 0 \le k \le W_0 - 1 \tag{30}$$

所有稳态的概率之和为 1，因此有

$$1 = \sum_{k=0}^{W_0-1} b_{0,k} = \frac{w_0 + 1}{2} b_{0,0} \tag{31}$$

求解得

$$b_{0,0} = \frac{2}{w_0 + 1} \tag{32}$$

节点第一次随机回退到 0 时，一定会发送数据，稳态传输概率τ，表示为

$$\tau = b_{0,0} \tag{33}$$

在本问的场景中，由于同样是 2BSS 互听，概率p、P_{tr}、P_s计算的方式与问题 1 中的式(18)到式(20)相同。但是由于节点以p的概率碰撞时，数据传输都会成功，没有发送失败的情况，P_s意为有节点传输数据的情况下只发送一个 AP 站点成功的概率，$1-P_s$对应为 2 个 AP 站点并发且成功的概率，此时信道上传输的有效数据载荷是 2 个 AP 站点发送有效数据载荷之和（在本问中 2 个 AP 站点发送有效数据载荷相同）。因此信道利用率需修改为

$$\eta = \frac{P_{tr}P_sE(p) + 2P_{tr}(1-P_s)E(p)}{(1-P_{tr})T_e + P_{tr}T_s} \tag{34}$$

最终吞吐量的计算仍为式(21)。

根据问题 2 给定的参数，确定三种虚拟时隙T_e、T_s和T_c，并根据模型推导出处于每个虚拟时隙的概率，从而最终评估出系统的吞吐。

4.2.2 模型求解

由本问的模型建立部分可知，代入已知条件$w_0 = 16$，可直接计算得$\tau = b_{0,0} = \frac{2}{w_0+1} = 0.117647 = p$，根据$p$和τ可以推导出$P_{tr} = 0.2215, P_s = 0.9375$。

表 3 问题 2 参数列表

参数名称	值
载荷长度	1500Bytes
PHY 头时长	13.6us
MAC 头长度	30Bytes
物理层速率 rate	275.3Mbps
ACK 时长	32μs
SIFS 时长	16μs
DIFS 时长	43μs
SLOT 时长	9μs
ACKTimeout 时长	65μs
CW min	16
CW max	1024
最大重传次数	32

根据表 3 给定参数，计算得到$E[p] = 43.589us, T_e = 9us, T_s = 149.06us$。代入信道利用率和吞吐量的计算公式中可得最终结果为信道利用率$\eta = 25.63\%$，以及$S = 70.562Mbps$。

问题 2 的仿真过程较问题 1 而言相对简单，我们仍然建立 C++程序进行模型仿真，程序的设定严格遵照题意规定：两个 AP 信号塔互听，但是由于 SIR 较高，不存在干扰的情况，所以两个基塔在回退数同时为 0 的时候，一起发送数据不会发生碰撞，两个 AP 信号塔都会进入发送数据的状态，并且数据都会被成功发送，其他设定同问题 1。

最终，我们将系统运行了 2000000 us，按照上述操作得到最后系统的信道利用率以及稳态吞吐量：$\eta = 24.02\%, S = 66.139Mbps$。与通过数据分析方式得到的结果对比，我们发现在信道利用率上差距为$\Delta\eta = 1.61\%$，可以看到，程序结果和数据分析结果相

印证，问题 2 我们依然较好地仿真了该模型。

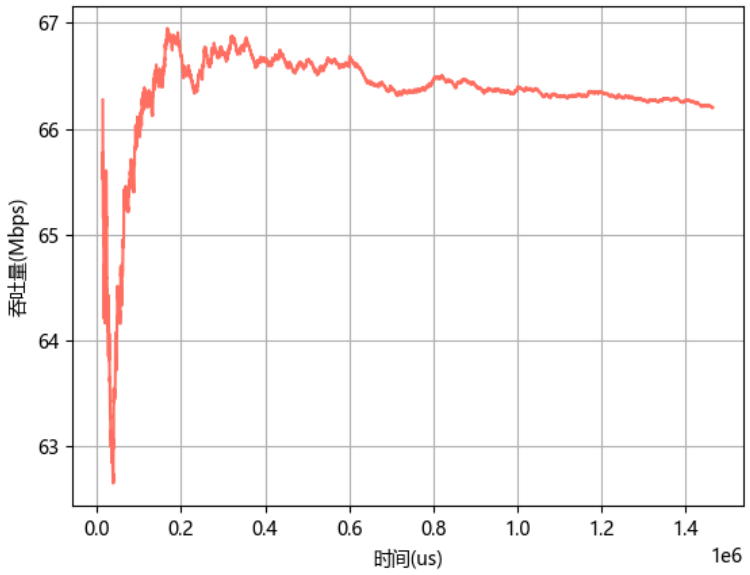

图 10 问题 2 仿真系统吞吐量变化曲线

4.3 问题3模型的建立与求解

4.3.1 模型建立

问题 3 也考虑在一个 WLAN 组网中存在 2 个 BSS 并且每个 BSS 中分别有一个 AP 向其关联的 STA 下行发送数据的场景。但是由于 AP 间 RSSI 低于 CCA 门限，2 个 BSS 之间不互听，当一个 AP 接入信道后，另一个 AP 不能监测到信道繁忙而认为信道空闲，仍然继续回退过程，从而导致有很大概率出现二者同时或先后开始发送数据的情况。当两个 AP 发包在时间上有交叠时（只考虑数据帧发送时段），假设 SIR 比较小，会导致两个 AP 的发包均失败。这意味着 2 个 AP 之间的并发并不局限于都回退为 0 同时发送的情况，还包含一个 AP 发送之后，另一个 AP 回退到 0 并发送造成冲突。同时，本问还考虑了实际情况中无线传输环境是复杂多变，无线信道受到干扰，从而导致当仅有一个 AP 发送数据，并且不存在另一个 AP 干扰时，会有$P_e = 10\%$的丢包率。要求针对该 2BSS 系统进行建模，并评估系统的吞吐。

我们仍然采用问题 1 中相同的二维的马尔科夫模型表示单个 AP 节点在退避过程中所处的状态，并且状态转移过程相同。但是与问题 1 不同的是，节点每次回退到 0 时发送失败的情况是由碰撞和丢包两种情况组成，概率p由碰撞概率P_c和丢包概率P_e表示，

$$p = P_c + (1 - P_c)P_e \tag{35}$$

每个节点回退到 0 时，一定会发送数据，稳态传输概率τ_1仍然可以表示为

$$\tau_1 = \sum_{0}^{r} b_{i,0} = b_{0,0} * \frac{1 - p^{r+1}}{1 - p} \tag{36}$$

由于 2 个 BSS 不互听，因此 1 个 AP 在传输数据时，另一个 AP 相当于一个隐藏节点，当其在 AP 的数据帧发送期间回退到 0 则会发送数据造成干扰。这个易受干扰的时期时长即为单个数据帧传输时长，可将其换算为V个空闲时隙。

$$V = \frac{H + E[p]}{T_e} \tag{37}$$

当一个 AP 开始发送数据时，如果隐藏节点会干扰到其数据发送，说明该隐藏节点的回退数已经回退到了 V 以内（包括 V）。令稳态概率τ_2表示节点当前回退数已经达到 V 以内的概率[3]，表示为

$$\tau_2 = \sum_{i=0}^{r}\sum_{k=0}^{V} b_{i,k}$$

$$= \begin{cases} \left[(V+1)\cdot\left(\frac{1-p^{r+1}}{1-p}\right)-\left(\frac{V(V+1)}{2W_0}\right)\cdot\left(\frac{1-\left(\frac{p}{2}\right)^{r+1}}{1-\left(\frac{p}{2}\right)}\right)\right]\cdot b_{0,0}, V < W_0 \\ \left[\begin{array}{c} \frac{1}{2}\cdot\left(\frac{1-p^{X}}{1-p}\right)+\frac{W_0}{2}\cdot\left(\frac{1-(2p)^{X}}{1-(2p)}\right)+ \\ (V+1)\cdot\left(\frac{p^{X}-p^{r+1}}{1-p}\right)-\left(\frac{V(V+1)}{2W_0}\right)\cdot\left(\frac{\left(\frac{p}{2}\right)^{X}-\left(\frac{p}{2}\right)^{r+1}}{1-\left(\frac{p}{2}\right)}\right) \end{array}\right]\cdot b_{0,0} \\ , W_{X-1} < V < W_X, 1 \le X \le r \\ 1, V > W_r \end{cases} \tag{38}$$

只有当一个 AP 能互听到的节点，不同步并发，并且不互听的隐藏节点不在干扰时段发送数据时，不会发生碰撞。因此碰撞概率可以表示为

$$P_c = 1-(1-\tau_1)^{n_c-1}(1-\tau_2)^{n_H} \tag{39}$$

其中n_c=1 表示能互听的 AP 节点只有自身，n_H=1 表示存在一个不互听的隐藏节点，$n = n_c + n_H$表示所有 AP 节点数。联立以上式子可以求解出τ_1, τ_2, P_C等概率，P_{tr}的计算方式与问题 1 中的式(19)相同，但是当有节点传输数据的情况下，传输成功的概率P_s需修改为式(40)，表示不发生碰撞且不丢包时才能传输成功。

$$P_s = \frac{n\tau_1(1-\tau_1)^{n_c-1}(1-\tau_2)^{n_H}(1-P_e)}{P_{tr}} \tag{40}$$

信道利用率和吞吐量的计算仍与问题一中的式(21)和式(22)相同。

根据问题 3 给定的参数，确定三种虚拟时隙T_e、T_s和T_c，并根据模型推导出处于每个虚拟时隙的概率，从而最终评估出系统的吞吐。

4.3.2 模型求解

1) 自适应扰动模拟退火算法

自适应扰动模拟退火算法（Adaptive Perturbation Simulated Annealing，APSA）是一种强大的全局优化算法，特别适用于高维、复杂、多峰和非线性的优化问题。基于模拟退火原理，在模拟退火过程中引入自适应扰动策略，以提高搜索效率，用于在复杂问题中全局搜索多维解空间以找到最优解。模拟退火通过接受概率较低的差解，有助于避免陷入局部最优解，而自适应扰动策略根据搜索进展动态地调整扰动程度，以平衡全局探索和局部搜索的需求。

自适应扰动模拟退火算法的步骤如下：

1. 初始化：初始化初始温度T_0、初始解x、初始扰动参数d、给定温度阈值T_{th}。
2. 使用随机扰动策略生成一个扰动解x'，计算$\Delta f = f(x') - f(x)$。
3. 更新当前解：如果$\Delta f < 0$，则接受新解x'作为x，否则以$e^{-\frac{\Delta f}{T_k}}$的概率接受新解。
4. 递减温度T_k，更新扰动参数d_k。

5. 检查是否满足终止条件，如果温度降到T_{th}以下，则停止迭代，否则继续迭代。

6. 重复步骤 2 到 5，直到满足终止条件或达到最大迭代次数。

自适应扰动模拟退火算法伪代码如表 4 所示：

表 4 自适应扰动模拟退火算法伪代码

Algorithm 2: Adaptive Perturbation Simulated Annealing (APSA)

Input: 初始温度 T_0，扰动参数 d_0，迭代次数 **max iter**、温度阈值 T_{th}

Output: 最优解 x，对应的目标函数值 f(x)

Initialize：x = 随机生成初始解，$T = T_0$，$d = d_0$

for k in range(max_iter):

//生成扰动解 x’、计算 Δ f

generate x’

Δ f=f(x’)-f(x)

//判断是否接受 x’作为新解

if Δ f < 0 or random()<=exp(-Δf / T_k) :

x=x’

Update T_k

Update d_k

//检查是否满足终止条件

if $T_k < T_{th}$:

break

return x

自适应扰动模拟退火算法流程图如图 11 所示：

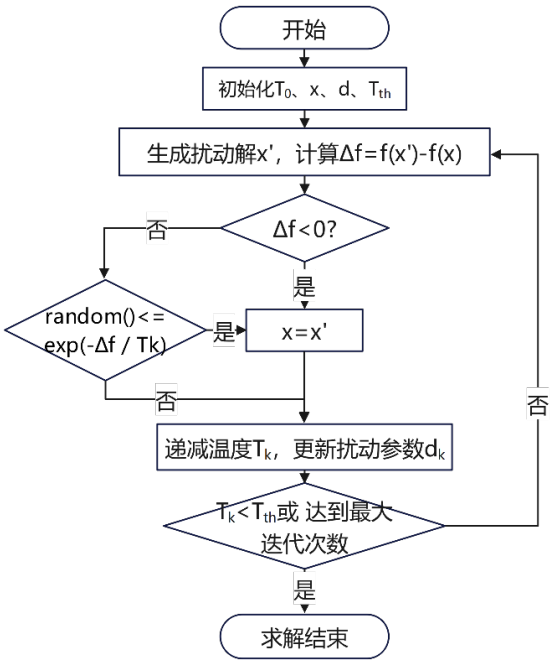

图 11 自适应扰动模拟退火算法流程图

2) 松弛牛顿-自适应扰动模拟退火混合算法

松弛牛顿-自适应扰动模拟退火混合算法（Relaxation Newton-Adaptive Perturbation Simulated Annealing Hybrid Algorithm，RNAH）是一种混合优化算法，结合了松弛牛顿

方法和自适应扰动模拟退火算法的特点，用于解决非线性优化问题。该算法的基本思想是通过松弛牛顿方法进行加速局部搜索，然后结合自适应扰动模拟退火以更全面地探索解空间。

松弛牛顿-自适应扰动模拟退火混合算法的步骤如下：

1. 初始化初始猜测解x_0、初始温度T_0、初始扰动参数d、给定温度阈值T_{th}、松弛因子α、容差tol、跳转容差tol_s。
2. 局部搜索（松弛牛顿方法）：
   a) 计算当前猜测解x_k处的函数值：$f(x_k)$和导数值：$f'(x_k)$。
   b) 使用松弛牛顿法的迭代公式来计算新的猜测解x_{k+1}：

$$x_{k+1} = x_k - \alpha \frac{f(x_k)}{f'(x_k)} \tag{41}$$

   c) 如果$|f(x_{k+1}) - f(x_k)| < tol_s$，则停止局部搜索，开始全局搜索，否则更新解$x_k = x_{k+1}$。
3. 全局搜索（自适应扰动模拟退火）：
   a) 使用随机扰动策略生成一个扰动解x_k'，计算$\Delta f = f(x_k') - f(x_k)$。
   b) 更新当前解：如果$\Delta f < 0$，则接受新解x_k'作为x_k，否则以$e^{-\frac{\Delta f}{T_k}}$的概率接受新解。
   c) 递减温度T_k，更新扰动参数d_k。
   d) 检查是否满足终止条件，如果温度降到T_{th}以下，或$|f(x_k)| < tol$，则停止迭代，否则继续迭代。
4. 重复步骤 2 到 3，直到满足终止条件或达到最大迭代次数。

松弛牛顿-自适应扰动模拟退火算法伪代码如表 5 所示：

表 5 松弛牛顿-自适应扰动模拟退火算法伪代码

Algorithm 3: Relaxation Newton-Adaptive Perturbation Simulated Annealing Hybrid (RNAH)

Input: 初始猜测解 x_0, 初始温度 T_0, 初始扰动参数 d, 温度阈值 T_{th}, 松弛因子 alpha, 容差 tol, 跳转容差 tol_s, 最大迭代次数 max_iter，全局搜索最大迭代次数 max_iter_A

Output: 最优解 x，对应的目标函数值 f(x)

```
Initialize：x = x0，T = T0
for i in range(max_iter):
    // 局部搜索（松弛牛顿方法）
    fk = Calculate f(x)
    f_prime_k = Calculate f'(x)
    x_new = x - alpha * fk / f_prime_k
    if |f(x_new) - fk| < tol_s:
        // 转到全局搜索
        while T > T_th:
            for m in range(max_iter_A):
            x_prime = Generate perturbed solution x' using a random perturbation strategy
            Δf = f(x_prime) - fk
            if Δf < 0 or with probability exp(-Δf / T) accept x':
                x = x_prime
```

```
                // 更新温度和扰动参数 T, d，根据性能和迭代次数动态调整
                Update current temperature T
                Update perturbation parameter d
                // 第一次检查终止条件
4:              if T <= T_th or |f(x_new)| < tol:
                    break
            // 第二次检查终止条件
            if T <= T_th or |f(x_new)| < tol:
                break
```

松弛牛顿-自适应扰动模拟退火算法流程图如图 12 所示：

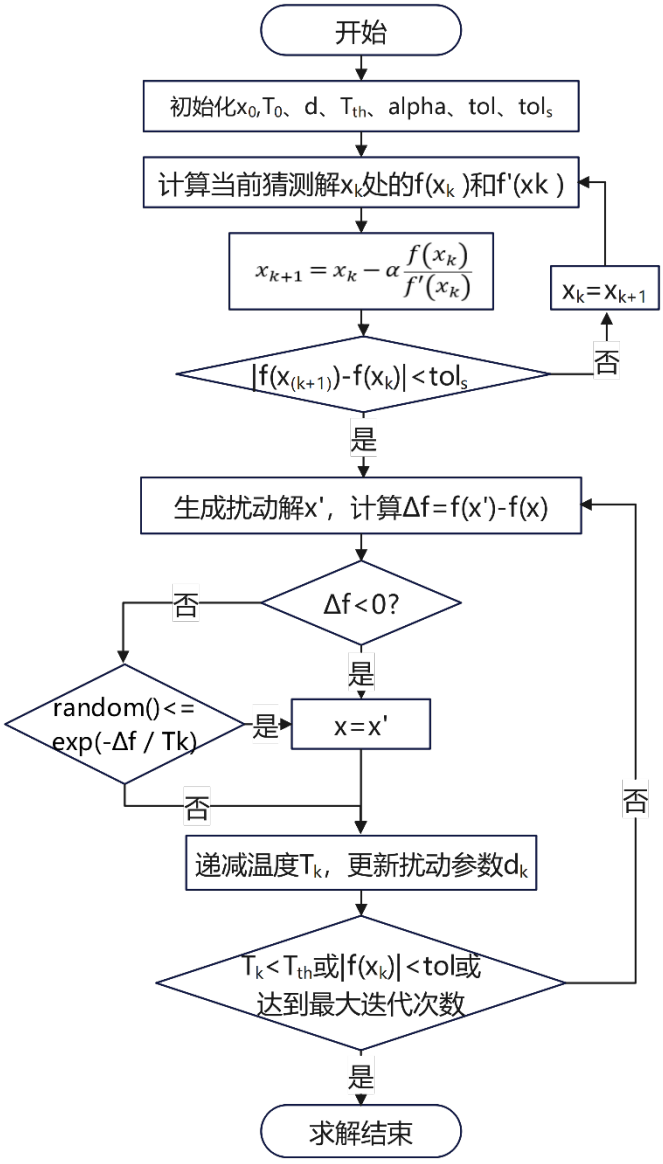

图 12 松弛牛顿-自适应扰动模拟退火算法流程图

3) 松弛牛顿-自适应扰动模拟退火混合算法求解问题 3 模型

因为问题 3 的能互听的节点只有自身，所以n_c=1，又因为问题 3 存在一个隐藏节点，所以n_H=1，代入式(39)得：

$$P_c = \tau_2 \tag{42}$$

由式(37)求得$V = 4.4949 \approx 5$，因为$V = 5 < 16 = W_0$，结合式(38)可得：

$$\tau_2 = \left[6 \cdot \left(\frac{1-p^{33}}{1-p}\right) - \left(\frac{15}{16}\right) \cdot \left(\frac{1-\left(\frac{p}{2}\right)^{33}}{1-\left(\frac{p}{2}\right)}\right)\right] \cdot b_{0,0} \tag{43}$$

结合式(35)和式(39)及式(43)，化简可得：

$$f(p) = 0 \tag{44}$$

$f(p)$是关于p的高次非线性方程，使用松弛牛顿-自适应扰动模拟退火混合算法求解这个方程：

首先设定初始温度 T_0 为 100 度，初始扰动参数 d 为 0.1，给定的温度阈值T_{th}为 0.001 度，松弛因子α为 0.5，容差为1×10^{-10}，跳转容差为1×10^{-9}。

考虑到非线性方程可能存在多解的情况，同时考虑到p是概率，取值范围为[0,1]，故将初始猜测解p_0以 0.01 为步长，在[0,1]的范围内遍历调用松弛牛顿-自适应扰动模拟退火算法，观察求得的估计近似解的分布情况：

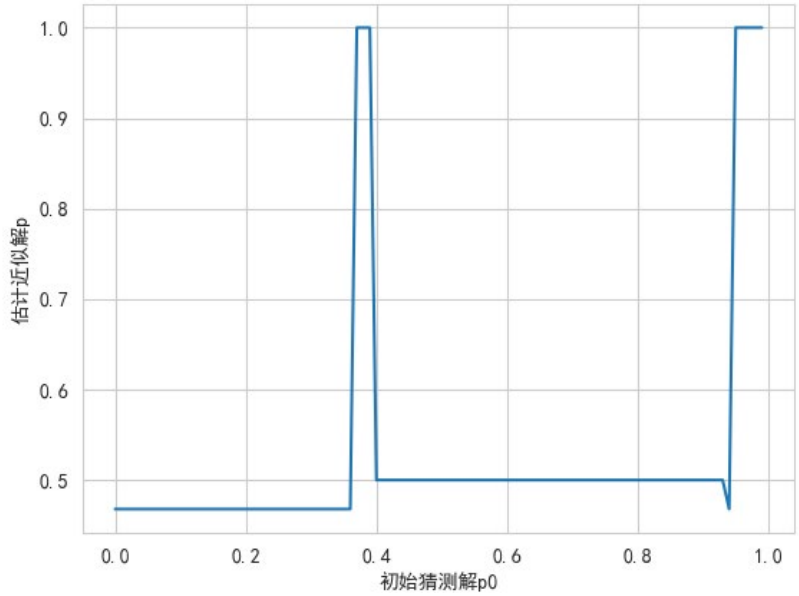

图 13 估计近似解分布图

由图 13 可以看出p_0在[0,1]范围内变动时，估计近似解p的取值大抵有 3 类：0.4647925 附近、0.5 附近、1 附近。

经过验证，当 p 取 0.5 或 1 时，求得的τ_1会无穷大，显然不符合题意，故舍去。当$p = 0.4647925$时，符合题意和实际情况，为了提高p的精度，进一步实验：

将结果容差tol设为1×10^{-10}，松弛因子和最大迭代次数均保持不变，将p_0设为 0.4647925，记录松弛牛顿-自适应扰动模拟退火混合算法每次迭代得到的p，最终在迭代 73 次后，得到最终的估计近似解$p = 0.46479313254$。画出 τ 的迭代求解图。

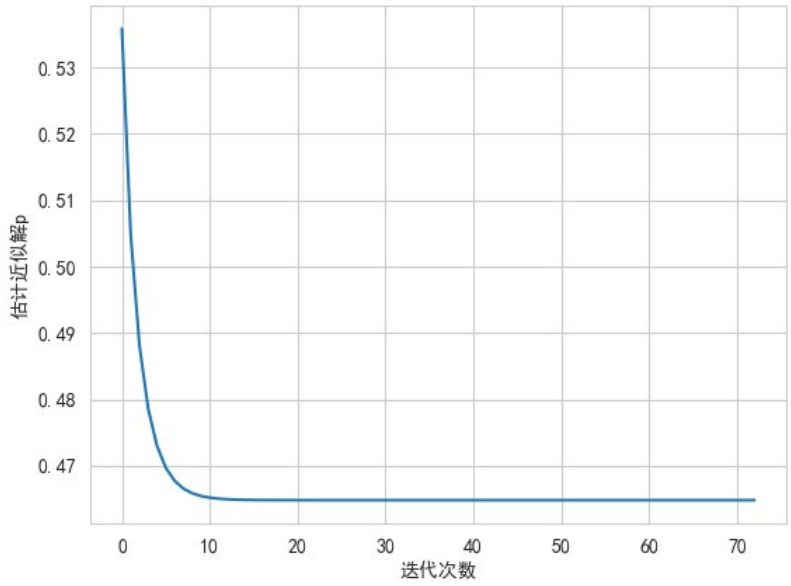

图 14 p的迭代求解图

由图 14 可知p的迭代收敛速度先快后慢，到达第 10 代后收敛速度变慢，p值趋于稳定。

求出p值后，可计算得到$\tau_2 = P_c$=0.405325，$\tau_1 = 0.057893$。然后推导出$P_{tr} = 0.0845, P_s = 0.6138$。

表 6 问题 3 通用参数列表

参数名称	值
载荷长度	1500Bytes
PHY 头时长	13.6us
MAC 头长度	30Bytes
物理层速率 rate	455.8Mbps
ACK 时长	32μs
SIFS 时长	16μs
DIFS 时长	43μs
SLOT 时长	9μs
ACKTimeout 时长	65μs
CW min	16
CW max	1024
最大重传次数	32

根据表 6 给出的参数，计算得到$E[p] = 26.327us, T_e = 9us, T_s = 131.454us, T_c = 148.454us$。代入信道利用率和吞吐量的计算公式中可得最终结果为信道利用率$\eta = 6.86\%$，稳态吞吐量$S = 31.269Mbps$。

问题三的仿真过程是四个题中最为复杂的，我们除了要考虑两个 AP 信号塔在不互听和 SIR 较低的情况下出现碰撞的情形，还要考虑丢包的情况。首先我们确定，信号重叠指的是两个信号塔发送数据报 PHY+MAC+Payload 的时段内有交叠，如图：

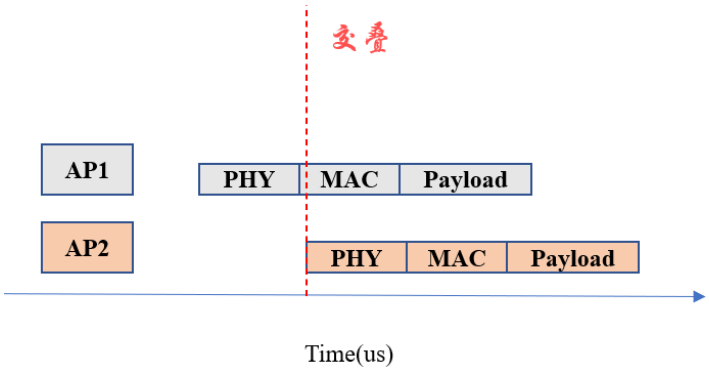

图 15 数据报交叠示意图

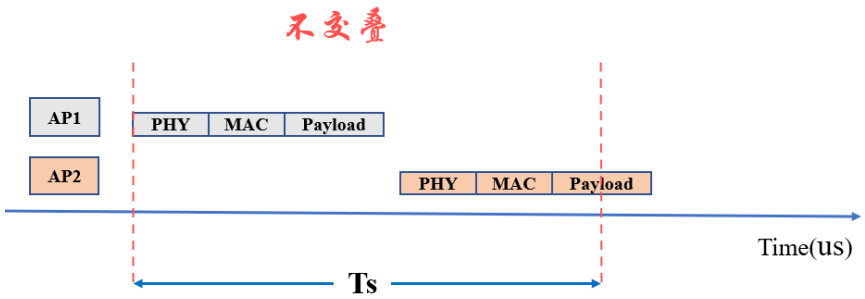

图 16 数据报不交叠示意图

由于 AP1 与 AP2 之间是不互听的，所以 AP1 与 AP2 发送数据的时候不会考虑此时对方是否正在发送数据，所以会出现碰撞。此外，在 AP 对象未发生碰撞，成功发送信号后，我们依然以 10%的概率让其进行丢包，进入碰撞状态。其他程序设定同问题 1、2 类似。我们将系统运行了 2000000 us，得到信道利用率以及稳态吞吐量：$\eta = 9.93\%, S = 45.283Mbps$。我们发现在信道利用率上差距为$\Delta\eta = 3.07\%$，我们发现程序的仿真结果和数据分析结果相差较大，并且程序仿真的系统吞吐量会比数据分析的吞吐量大。我们进行了分析，认为在建模的过程中，将易受干扰的数据帧传输时段近似换算为 V（整数）个空闲时隙，也就是说会出现时间上的偏差，而在我们的程序中，时间是连续变化的，也就是“严丝合缝”的，故程序仿真的系统吞吐量大于数值分析的系统吞吐量是合理的。

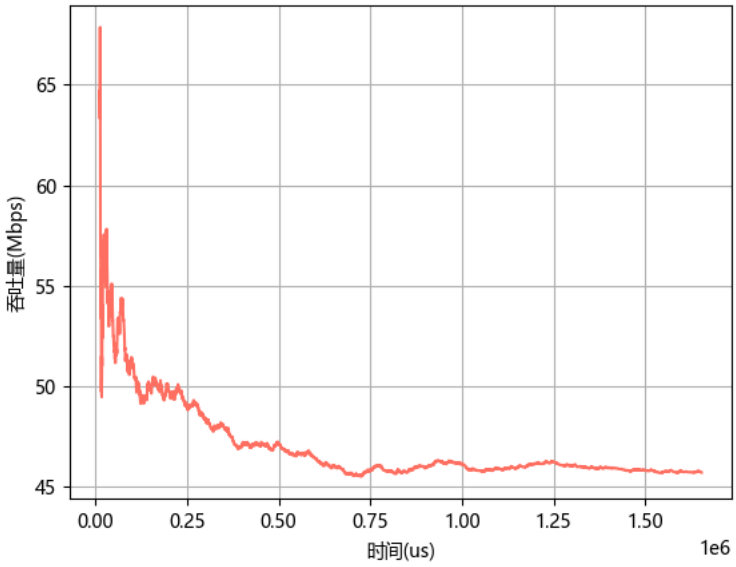

图 17 问题 3 仿真系统吞吐量变化曲线图

此外，我们尝试改变物理层速率、竞争窗口和最大重传次数等参数，重新对系统进行了数值分析和仿真计算，结果如表 7：

表 7 问题 3 不同参数下结果对比

参数名称	值						
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
数值分析							
S(Mbps)	31.269	29.655	25.554	30.631	25.265	18.707	26.484
η	6.86%	10.34%	8.91%	10.68%	15.95%	11.81%	16.72%
仿真结果							
S(Mbps)	45.283	37.236	30.722	37.104	27.758	22.715	29.267
η	9.93%	12.98%	10.71%	12.94%	17.52%	14.34%	18.48%

从上述表格中，我们总结出，对于问题三系统而言，物理层传输速率降低、CW_{min} 参数变大都会导致系统吞吐量的下降，此外对于最大重传次数，数值越小在一定程度上也会降低系统的吞吐量，但是我们数值分析和仿真过程中，其影响效果并不明显。

4.4 问题4模型的建立与求解

4.4.1 模型建立

问题 4 考虑在一个 WLAN 组网中存在 3 个 BSS 并且每个 BSS 中分别有一个 AP 向其关联的 STA 下行发送数据的场景，如图 18 所示。其中 AP1 与 AP3 之间的 RSSI 低于 CCA 门限，不互听，而 AP2 与 AP1、AP3 之间的 RSSI 都高于 CCA 门限，即 AP2 与两者互听。因此 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。并且假设 AP1 和 AP3 发包时间交叠时，SIR 较大，两者发送均成功。要求针对该 3BSS 系统进行建模，并评估系统的吞吐。

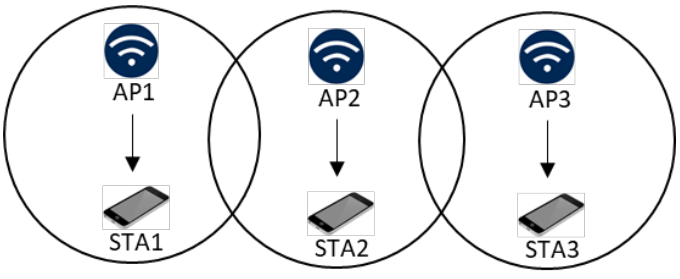

图 18 三同频 BSS 场景

我们仍然采用问题 1 中相同的二维的马尔科夫模型表示单个 AP 节点在退避过程中所处的状态，并且状态转移过程相同。但是与问题 1 不同的是，本问中 AP1、AP3 和 AP2 最终稳态时的状态并不相同，因此在本问中需要采用一个二维的马尔科夫模型 B^1 表示 AP1 和 AP3，其最终稳态为$b_{i,k}^1$，再采用一个二维的马尔科夫模型 B^2 表示 AP2，其最终稳态为$b_{i,k}^2$。两个模型的稳态传输概率分别表示为

$$\tau_1 = \sum_{0}^{r} b_{i,0}^{1} = b_{0,0}^{1} * \frac{1 - p_1^{r+1}}{1 - p_1} \tag{45}$$

$$\tau_2 = \sum_{0}^{r} b_{i,0}^{2} = b_{0,0}^{2} * \frac{1 - p_2^{r+1}}{1 - p_2} \tag{46}$$

对于 AP1 和 AP3，其传输失败只发生在与 AP2 并发时，因此其碰撞概率为

$$p_1 = 1 - (1 - \tau_2)^1 = \tau_2 \tag{47}$$

对于 AP2，其传输失败发生在与 AP1 或 AP2 并发时，因此其碰撞概率为

$$p_2 = 1 - (1 - \tau_1)^2 \tag{48}$$

联立以上式子，即可求解出τ_1, τ_2, p_1, p_2。

在本问中，分别对 B^1 和 B^2 求解对应的P_{tr}和P_s，以表示该 AP 站点能检测到在一个虚拟时隙上最少有一个节点传输数据的概率和在有节点传输数据的情况下，传输成功的概率。

$$P_{tr1} = 1 - (1 - \tau_1)(1 - \tau_2) \tag{49}$$

$$P_{tr2} = 1 - (1 - \tau_1)^2(1 - \tau_2) \tag{50}$$

$$P_{s1} = \frac{\tau_1(1 - \tau_2)}{P_{tr1}} \tag{51}$$

$$P_{s2} = \frac{\tau_2(1 - \tau_2)^2}{P_{tr2}} \tag{52}$$

然后计算 3 个 AP 对应的平均信道利用率和吞吐，最后得到系统总的吞吐。

$$\eta_1 = \eta_3 = \frac{P_{tr1}P_{s1}E(p)}{(1 - P_{tr1})T_e + P_{tr1}P_{s1}T_s + P_{tr1}P_{s1}T_s} \tag{53}$$

$$\eta_2 = \frac{P_{tr2}P_{s2}E(p)}{(1 - P_{tr2})T_e + P_{tr2}P_{s2}T_s + P_{tr2}P_{s2}T_s} \tag{54}$$

$$S_i = \eta_i * rate, i = 1,2,3 \tag{55}$$

$$S = S_1 + S_2 + S_3 \tag{56}$$

根据问题 4 给定的参数，确定三种虚拟时隙T_e、T_s和T_c，并根据模型推导出处于每个虚拟时隙的概率，从而最终评估出系统的吞吐。

4.4.2 模型求解

1) Bayesian 优化

Bayesian 优化是一种用于优化目标函数的迭代式方法，它可以在有限的迭代次数内找到尽可能好的解。该方法基于贝叶斯统计模型，通过不断地选择下一个采样点，并使用当前的信息来指导下一次采样，以有效地优化目标函数。

$$P(f(x)|D) = \frac{P\left(D\middle|f(x)\right) \cdot P\left(f(x)\right)}{P(D)} \tag{57}$$

式(57)是贝叶斯公式，根据贝叶斯公式，我们可以在给定先验概率和新的观测数据时更新目标函数的后验概率分布。其中：

- $P(f(x)|D)$是在给定观测数据D的情况下，目标函数$f(x)$的后验概率分布。

- $P(D|f(x))$是在给定目标函数值$f(x)$的情况下，观测数据D的似然。
- $P(f(x))$ 是目标函数$f(x)$的先验概率分布。
- $P(D)$ 是观测数据D的边缘概率。

Bayesian 优化的核心思想是选择下一个采样点，以最大化对目标函数$f(x)$的预测改进。这通常由一个采样策略（Acquisition Function）来定义，常见的策略包括：

a) 期望改进（Expected Improvement，EI）：用于最小化问题，它衡量下一个采样点是否有望在当前最优解的基础上提高。

$$EI(x) = E[\max(f(x^*) - f(x), 0)] \tag{58}$$

其中x^*是当前已知的最优解。

b) 高置信上限（Upper Confidence Bound，UCB）：它权衡探索和利用，鼓励在不确定性高的地方探索。

$$UCB(x) = \mu(x) + \kappa\sigma(x) \tag{59}$$

其中$\mu(x)$是目标函数的均值估计，$\sigma(x)$是方差估计，κ是探索参数。

Bayesian 优化是一个迭代过程。在每个迭代中，根据采样策略选择下一个采样点，然后在该点进行实际采样，获得观测数据。这些观测数据被用来更新贝叶斯模型的参数，然后进入下一轮迭代。迭代过程会不断提高对目标函数的建模准确性，最终收敛到一个局部最优解或全局最优解。

2) 贝松自混算法

贝叶斯-松弛牛顿-自适应扰动混合优化算法（贝松自混算法）（Bayesian Relaxation Newton-Adaptive Perturbation Simulated Annealing Hybrid Algorithm，BRNAH）在松弛牛顿-自适应扰动混合优化算法的基础上添加了贝叶斯优化过程，将贝叶斯优化部分用于全局搜索，有助于发现目标函数的多个潜在解。将松弛牛顿法和自适应扰动模拟退火用于局部优化，在已知的解附近精细调整，从而更加全面精确的找到非线性方程的解。

贝松自混算法的步骤如下：

1. 初始化初始猜测解x_0、初始温度T_0、初始扰动参数d、给定温度阈值T_{th}、松弛因子α、容差tol、跳转容差tol_s。
2. 初始化 Bayesian 优化模型，使用高斯过程（GP）建模目标函数$f(x)$的概率分布
3. 使用 Bayesian 优化算法来选择下一个采样点x_k，优化采样策略使用 UCB：

$$x_k = argmax_x Acquisition(x) \tag{60}$$

4. 局部搜索（松弛牛顿方法）：
   a) 计算当前解x_k处的函数值：$f(x_k)$和导数值：$f'(x_k)$。
   b) 使用松弛牛顿法的迭代公式来计算新的猜测解x_{k+1}：

$$x_{k+1} = x_k - \alpha\frac{f(x_k)}{f'(x_k)} \tag{61}$$

   c) 如果$|f(x_{k+1}) - f(x_k)| < tol_s$，则停止局部搜索，开始全局搜索，否则更新解$x_k = x_{k+1}$。
5. 全局搜索（自适应扰动模拟退火）：
   a) 使用随机扰动策略生成一个扰动解x_k'，计算$\Delta f = f(x_k') - f(x_k)$。
   b) 更新当前解：如果$\Delta f < 0$，则接受新解x_k'作为x_k，否则以$e^{-\frac{\Delta f}{T_k}}$的概率接受新解。

c) 递减温度T_k，更新扰动参数d_k。

d) 检查是否满足终止条件，如果温度降到T_{th}以下，或$|f(x_k)| < tol$，则停止迭代，否则继续迭代。

6. 重复步骤 2 到 5，直到满足终止条件或达到最大迭代次数。

贝叶斯-松弛牛顿-自适应扰动模拟退火算法伪代码如表 8 所示：

表 8 贝叶斯-松弛牛顿-自适应扰动模拟退火算法伪代码

Algorithm4: Relaxation Newton-Adaptive Perturbation Simulated Annealing Hybrid (RNAH)

Input: 初始猜测解 x_0, 初始温度 T_0, 初始扰动参数 d, 温度阈值 T_{th}, 松弛因子 alpha, 容差 tol, 跳转容差 tol_s, 最大迭代次数 max_iter，全局搜索最大迭代次数 max_iter_A

Output: 最优解 x，对应的目标函数值 f(x)

```
Initialize：x = x0，T = T0,
Build bayesian model
for i in range(max_iter):
  //使用 Bayesian 优化算法选择下一个采样点
  x=argmaxxAcquisition(x0)
  // 局部搜索（松弛牛顿方法）
  fk = Calculate f(x)
  f_prime_k = Calculate f'(x)
  x_new = x - alpha * fk / f_prime_k
  if |f(x_new) - fk| < tol_s:
    // 转到全局搜索
    while T > T_th:
      for m in range(max_iter_A):
       x_prime = Generate perturbed solution x' using a random perturbation strategy
       Δf = f(x_prime) - fk
       if Δf < 0 or with probability exp(-Δf / T) accept x':
           x = x_prime
       // 更新温度和扰动参数 T, d，根据性能和迭代次数动态调整
       Update current temperature T
       Update perturbation parameter d
       // 第一次检查终止条件
       if T <= T_th or |f(x_new)| < tol:
           break
    // 第二次检查终止条件
    if T <= T_th or |f(x_new)| < tol:
      break
```

贝叶斯-松弛牛顿-自适应扰动模拟退火算法流程图如图 19 所示：

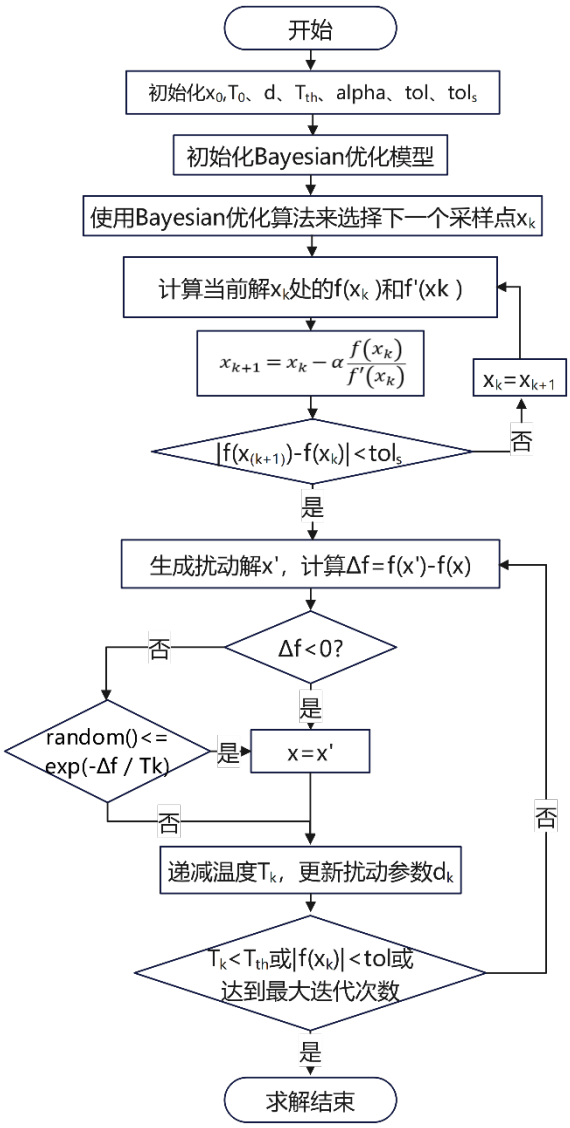

图 19 贝叶斯-松弛牛顿-自适应扰动模拟退火算法流程图

3)　贝叶斯-松弛牛顿-自适应扰动模拟退火算法求解模型

在问题 4 的模型建立部分提到，联系式(45)到式(48)，可以得到：

$$g(p_2) = 0 \tag{62}$$

我们使用贝叶斯-松弛牛顿-自适应扰动模拟退火算法求解p_2：

首先设定初始温度 T_0 为 100 度，初始扰动参数 d 为 0.1，给定的温度阈值T_{th}为 0.001 度，松弛因子α为 0.5，容差为1×10^{-10}，跳转容差为1×10^{-9}，并初始化 Bayesian 优化模型。

然后使用 Bayesian 优化算法来选择下一个采样点x_k。

考虑到非线性方程可能存在多解的情况，同时考虑到p_2是概率，取值范围为[0,1]，故将初始猜测解p_0以 0.01 为步长，在[0,1]的范围内遍历调用松弛牛顿-自适应扰动模拟退火算法，观察求得的估计近似解的分布情况：

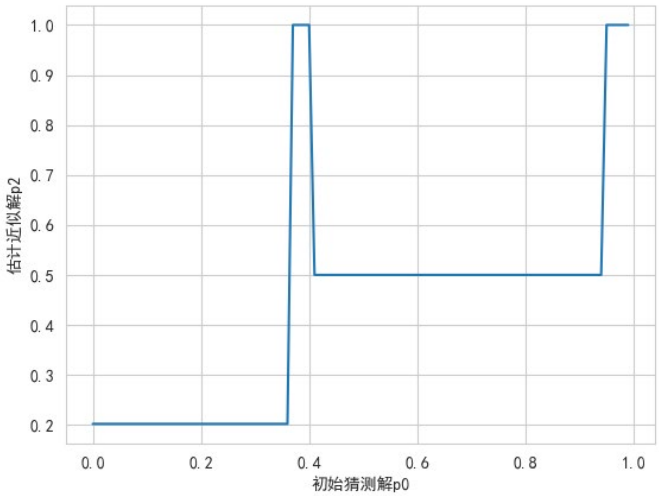

图 20 估计近似解分布图

由图可以看出p_0在[0,1]范围内变动时，估计近似解p_2的取值大抵有 3 类：0.2020674 附近、0.5 附近、1 附近。

经过验证，当p_2取 0.5 或 1 时，求得的τ_2会无穷大，显然不符合题意，故舍去。当$p_2 = 0.2020674$时，符合题意和实际情况，为了提高p的精度，进一步实验：

将结果容差tol设为1×10^{-10}，松弛因子和最大迭代次数均保持不变，将p_0设为0.2020674，记录贝叶斯-松弛牛顿-自适应扰动模拟退火混合算法每次迭代得到的p，最终在迭代 240 次后，得到最终的估计近似解$p_2 = 0.20206063243$。画出 τ 的迭代求解图。

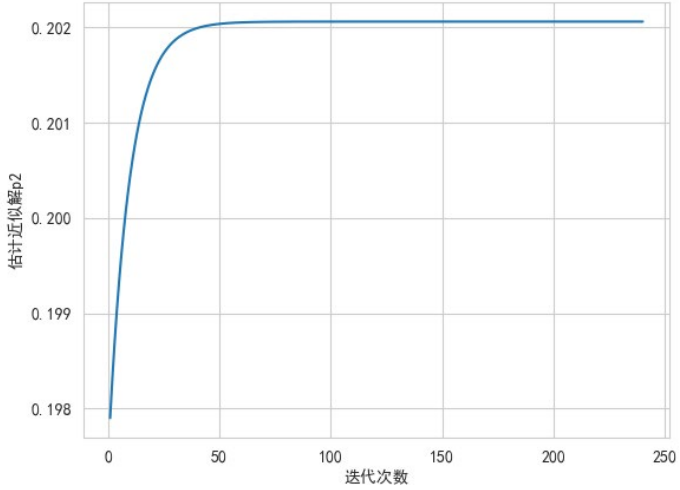

图 21 p_2的迭代求解图

解出$p_2 = 0.20206$后，可以求得$\tau_1 = 0.10673$、$p_1 = \tau_2 = 0.08928$。然后推导出$P_{tr1} = 0.1865, P_{tr2} = 0.2733, P_{s1} = 0.5212, P_{s2} = 0.2607$。

表 9 问题 4 通用参数列表

参数名称	值
载荷长度	1500Bytes
PHY 头时长	13.6us
MAC 头长度	30Bytes
物理层速率 rate	455.8Mbps

ACK 时长	32μs
SIFS 时长	16μs
DIFS 时长	43μs
SLOT 时长	9μs
ACKTimeout 时长	65μs
CW min	16
CW max	1024
最大重传次数	32

根据表 9 给出的参数，计算得到$E[p]=26.327us, T_e=9us, T_s=131.454us, T_c=148.454us$。代入信道利用率和吞吐量的计算公式中可得结果为$\eta_1=8.77\%, S_1=39.972Mbps, \eta_2=4.09\%, S_2=18.624Mbps, \eta_3=8.77\%, S_3=39.972Mbps$，总的信道利用率和吞吐量为$\eta=21.63\%, S=98.567Mbps$。

问题四的程序仿真遵照题意，AP1 与 AP3 不互听，并且 SIR 高，不出现干扰，AP2 同 AP1、AP2 都互听，并且 SIR 较低，出现干扰现象。程序的机制设定类似问题一、二中的模型，在此不再赘述，我们在将仿真模型运行 20000000us 后，得到结果$\eta_1=10.23\%$，$S_1=46.623\ Mbps$，$\eta_2=3.82\%$，$S_2=17.402\ Mbps$，$\eta_3=10.22\%$，$S_3=46.602\ Mbps$，总的信道利用率和吞吐量为$\eta=24.27\%, S=110.627\ Mbps$。对比仿真系统和数值分析系统吞吐量有：$\Delta\eta_1=1.46\%$，$\Delta\eta_2=0.27\%$，$\Delta\eta_3=1.45\%$，$\Delta\eta=2.64\%$。仿真系统和数值分析系统在三个 AP 信号塔上的吞吐都很好地吻合，但是在系统总的吞吐上还是略有差距。

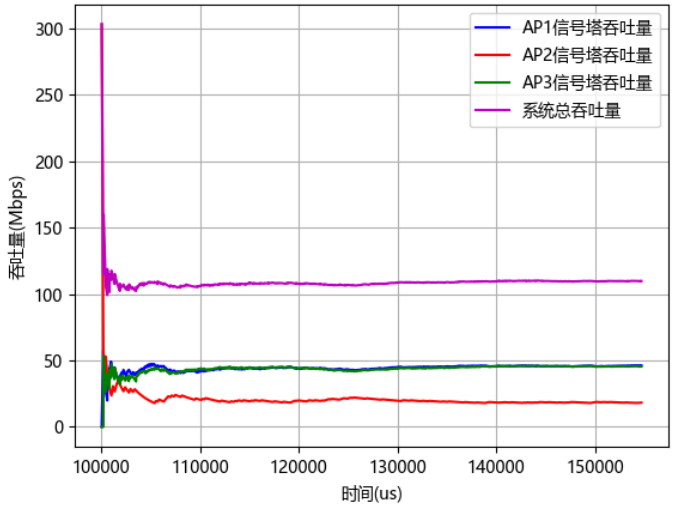

图 22 问题 4 仿真系统吞吐量变化曲线图

另外无论是仿真系统还是数值分析系统，都很好的反映了一个现象：AP1 和 AP3 信号塔会挤兑 AP2 的信道，导致 AP2 发送信号的频率相对较低，这种现象会随着竞争窗口、最大重传次数和物理层速率的改变而发生变化，特别是在物理层速率降低的情况下，该挤兑现象变得更加严重：

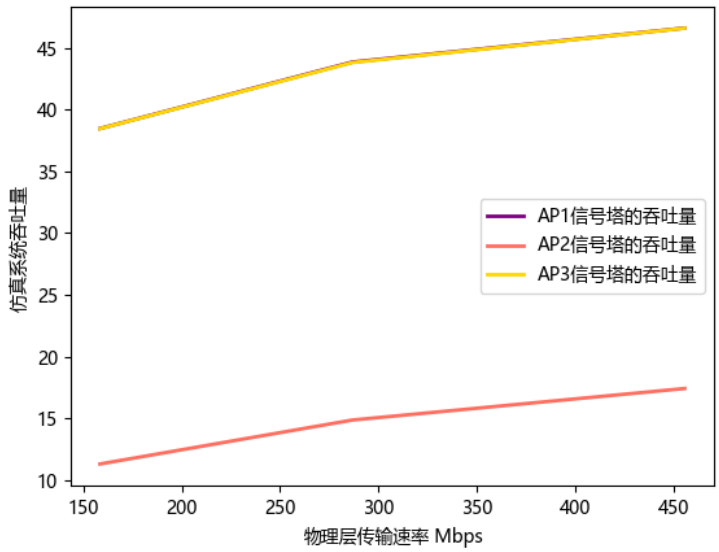

图 23 AP 信号塔吞吐量随物理层传输速率的变化曲线

上图中，由于 AP1 和 AP3 的吞吐量几乎一样，所以两条曲线重叠了，我们只能看到两条曲线。另外，可以看到，AP1 和 AP3 保持着一个较高的吞吐量，同时 AP2 的吞吐量仅占 AP1 和 AP3 的 1/4 左右，很好地契合了题意，另外，随着物理层传输的降低，所有 AP 信号塔的吞吐都有一定程度上的降低，但是 AP1 和 AP3 的吞吐量仍然保持在一个比较高的水平，但是 AP2 的吞吐量已经降到了 10Mbps 左右了。

此外，我们尝试改变物理层速率、竞争窗口和最大重传次数等参数，重新对系统进行了数值分析和仿真计算，结果如表 10：

表 10 问题 4 不同参数下结果对比

参数名称	值						
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
数值分析							
S(Mbps)	98.567	95.88	75.14	96.19	86.25	68.33	86.74
η	21.63%	33.43%	26.20%	33.54%	54.45%	43.14%	54.76%
仿真结果							
S(Mbps)	110.627	102.562	81.297	102.58	88.3517	71.3827	88.2808
η	24.27%	35.76%	28.35%	35.77%	55.78%	45.06%	55.73%

五、 模型的评价

5.1 模型的优点

1、综合性方法： 模型的求解采用了综合性的方法，结合了 Bayesian 优化、松弛牛顿法和自适应扰动模拟退火，使其具备了全局和局部搜索的能力，从而能够应对非线性优化问题的复杂性。

2、贝叶斯优化： 引入了贝叶斯优化作为全局搜索的一部分，这有助于有效地探索目标函数的多个潜在解，提高了找到全局最优解的机会。

3、适应性参数调整： 模型中的参数，如温度、扰动参数等，都是根据性能和迭代次数动态调整的，这增强了算法的适应性，有助于更快地收敛到最优解。

4、仿真器验证模型精确度：编写了仿真器对信道接入机制进行仿真，使用仿真结果对模型求解的结果进行精确度验证，提高了结果可信度。

5.2 模型的缺点

1、复杂性： 该模型是一个综合性算法，涉及多个部分和参数的调整，因此相对较复杂。这可能需要更多的时间和计算资源来实施和调整。

2、计算开销： 由于模型涉及多次迭代和对目标函数的不断采样，可能需要大量的计算资源，特别是在高维问题上。

3、收敛速度： 在一些情况下，贝叶斯优化可能需要较长的时间来收敛到最优解，这可能会增加算法的运行时间。

六、 模型的改进

在本文所用求解模型的基础上还可以作进一步的改进，比如：

1、模型改进：通过对问题的数学建模进行改进，可以使得模型更加紧凑或更加适合数值分析方法求解，从而提高求解精度或者效率。例如，可以考虑模型的简化版本，减少不必要的复杂性。简化后的模型可能更容易理解和调整。

2、算法改进：通过对求解算法的改进，可以进一步优化模型的性能，比如考虑设计新的采样策略，以提高模型的效率和性能。可以探索不同的采样策略，例如基于梯度信息的策略或基于进化算法的策略，以更快地收敛到最优解。

3、求解过程优化：对求解过程进行优化，可以在相同时间内获得更好的解或者更加稳定的解。例如，可以通过采用并行化的方法来加速求解过程，或者开发自适应参数调整策略，使算法能够根据问题的性质和当前状态来自动调整温度、扰动参数等关键参数。

七、 参考文献

[1] Bianchi, Giuseppe. "IEEE 802.11-saturation throughput analysis." IEEE communi*cations letters* 2.12 (1998): 318-320.

[2] Xiao, Yang, and Jon Rosdahl. "Throughput and delay limits of IEEE 802.11." *IEEE Communications letters* 6.8 (2002): 355-357.

[3] Bianchi, Giuseppe. "Performance analysis of the IEEE 802.11 distributed coordination function." IEEE Journal on selected areas in communi*cations* 18.3 (2000): 535-547.

[4] Chen, Da Rui, and Ying Jun Zhang. "Is dynamic backoff effective for multi-rate WLANs?." IEEE communications letters 11.8 (2007): 647-649.

[5] 段志荣. IEEE 802.11 DCF 协议退避算法的研究. MS thesis. 燕山大学, 2018.

八、 附录

8.1 使用的软件

PyCharm 2023.1 (Professional Edition)
Microsoft 365
Visual Studio 2022

8.2 相关的源程序代码

数值分析部分代码：

```
'''
松弛牛顿法
'''
def f(x):
    return 2050*x**35 - 1029*x**34 + 2*x**33 - 1024*x**8 - 18*x**2 + 21*x -2

def df(x):
    return 71750*x**34 - 34986*x**33 + 66*x**32 - 8192*x**7 - 36*x + 21

def relaxed_newton(f, df, x0, tol=1e-20, max_iter=1000, alpha=0.5):
    x = x0
    iteration = 0
    while abs(f(x)) > tol and iteration < max_iter:
        x_new = x - alpha * f(x) / df(x)
        x = x_new
        iteration += 1
    if abs(f(x)) <= tol:
        print("方程的根为:", x)
        x0_list.append(x0)
        x_list.append(x)
        iteration_list.append(iteration)
        x010462_list.append(x)
        return x
    else:
        print("超过最大迭代次数，未能找到足够接近的根")
        return None
iteration_list=[]
x010462_list=[]
# 设置初始猜测值和其他参数
x0 = 0.10462
alpha = 0.5
x0_list = []
x_list = []
relaxed_newton(f, df, x0, alpha=alpha)
```

```
print(iteration_list)
print(x010462_list)

'''
松弛牛顿法和自适应扰动模拟退火算法混合优化算法
'''
import numpy as np
import scipy.optimize as opt
import random
def nonlinear_equation(x):
    return 2050*x**35 - 1029*x**34 + 2*x**33 - 1024*x**8 - 18*x**2 + 21*x -2

def objective_function(x):
    return nonlinear_equation(x)**2

# 松弛牛顿法求解一元非线性方程
def relaxed_newton_solver(x0, tol=1e-9):
    x = x0
    while abs(nonlinear_equation(x)) > tol:
        x -= nonlinear_equation(x) / (71750*x**34 - 34986*x**33 + 66*x**32 - 8192*x**7 - 36*x + 21)
    return x

# 自适应扰动模拟退火算法求解一元非线性方程
def adaptive_perturbation_sa_solver(x0, max_iter=10000, initial_temperature=100, min_temperature=0.
001):
    x = x0
    current_temperature = initial_temperature
    for _ in range(max_iter):
        for _ in range(100):  # 每个温度下进行 100 次扰动
            perturbation = random.uniform(-0.1, 0.1)  # 随机扰动
            new_x = x + perturbation
            delta_e = objective_function(new_x) - objective_function(x)
            if delta_e < 0 or random.random() < np.exp(-delta_e / current_temperature):
                x = new_x
        current_temperature *= 0.7 # 降低温度
        if current_temperature < min_temperature:
            break
    return x
# 使用混合优化算法求解一元非线性方程
def hybrid_optimizer():
    # 使用松弛牛顿法找到一个初始解
    initial_guess = 0.5
    initial_solution = relaxed_newton_solver(initial_guess)
    # 使用自适应扰动模拟退火算法对初始解进行优化
    result = adaptive_perturbation_sa_solver(initial_solution)
    return result
x0_list = []
x_list = []
```

```
# 求解非线性方程
root = hybrid_optimizer()
print("方程的根为:", root)
print("方程的值为:", nonlinear_equation(root))
for i in range(100):
    x0 = i/100
    x0_list.append(x0)
    # 调用混合方法求解方程的根
    root = hybrid_optimizer()
    x_list.append(root)
    print("方程的根为:", root)
    print("方程的值为:", nonlinear_equation(root))
print(x0_list)
print(x_list)

'''
贝叶斯-松弛牛顿-自适应扰动混合优化算法
'''
import numpy as np

def f(x):
    return 2050*x**35 - 1029*x**34 + 2*x**33 - 1024*x**8 - 18*x**2 + 21*x -2

def df(x):
    return 71750*x**34 - 34986*x**33 + 66*x**32 - 8192*x**7 - 36*x + 21

def bayesian_relaxation_newton(f, df, x0, tol=1e-6, max_iter=100):
    x = x0
    for iteration in range(max_iter):
        # 贝叶斯估计更新
        uncertainty = 1.0 / (df(x)**2 + 1e-6)
        update = uncertainty * df(x) * (f(x) / df(x) - x)
        # 松弛牛顿法更新
        alpha = 0.5
        x_new = x + alpha * update
        # 自适应扰动
        if abs(x_new - x) < tol:
            return x_new, iteration + 1
        else:
            x = x_new + np.random.normal(0, 0.1)  # 添加随机扰动
    raise Exception("算法未能收敛")
x0 = 0.0  # 初始猜测值
root, iterations = bayesian_relaxation_newton(f, df, x0)
print("方程的根:", root)
print("迭代次数:", iterations)
```

仿真代码（C++）

第一问

```
#pragma once
#include<iostream>
#include <random>
#include <ctime>
#include<vector>
using namespace std;
static const int cwMax=1024,cwMin=16;

class AP {
public:
    int r_max, m, backNum, wi, r_now;
    static std::mt19937 gen;
    AP() {};
    AP(int r1, int m1);
    void selectBackNum();
    void collision();
    void reBuild();
};

#pragma once
#include<iostream>
#include<vector>
#include <fstream>
#include"AP.h"
using namespace std;

static const int beginTime = 10000, endTime = 2000000;
static const double Ts = 13.6 + 0.5265 + 26.327 + 32 + 43 + 16, Tc = 13.6 + 26.327 + 0.5265 + 43 +
65;

class BSS_System
{
public:
    AP AP1, AP2;
    long long throughput;
    long double TimeNow;
    void function();
    void addResult(int cargo);
    vector<long double>TimeNow_Arr;
    vector<long long>throughput_Arr;
    vector<long double>average_Throughput_Arr;
    BSS_System() :AP1(32, 6), AP2(32, 6), throughput(0), TimeNow(0.0)
    {
        TimeNow_Arr.reserve(20000);
        throughput_Arr.reserve(20000);
        average_Throughput_Arr.reserve(20000);
```

```
        cout << "系统建立成功!" << endl;
    }
    void writeFileMy();
};

#include"AP.h"
std::mt19937 AP::gen(static_cast<unsigned int>(std::time(nullptr))); // 初始化静态成员
AP::AP(int r1, int m1) :r_max(r1), m(m1), wi(16), backNum(), r_now(0)
{
    selectBackNum();
    cout << "AP 基塔搭建成功!" << endl;
}
void AP::selectBackNum() {
    // 定义均匀分布
    std::uniform_int_distribution<int> dist(0, wi - 1);
    // 生成随机整数
    this->backNum = dist(gen); // 使用静态生成器
}
void AP::collision() {
    if (r_now < r_max)
    {
        if (wi < 1024)
        {
            wi *= 2;
        }
        this->selectBackNum();
        r_now++;
    }
    else
    {
        this->reBuild();
    }
}
void AP::reBuild() {  //传输成功的时候 直接 rebuild 就行!
    this->r_now = 0;
    this->wi = 16;
    this->selectBackNum();
}

#include"System1.h"
#include"AP.h"
void BSS_System::addResult(int cargo) {
    this->throughput += cargo;
    long double Temp = this->throughput / ((TimeNow - beginTime) / 1000);
    cout << "NowTime(us) : " << TimeNow
        << "\t  Throughout ( 需要 *100)(单位是 Bit): " << this->throughput
        << "\t  average Thoughout    () " << Temp << endl;
    this->TimeNow_Arr.push_back(TimeNow);
```

```
    this->throughput_Arr.push_back(throughput);
    this->average_Throughput_Arr.push_back(Temp);
}

void BSS_System::function() {
    int timeGap = 0;
    for (int i = 0; i <= 100000 && TimeNow <= endTime; i++)
    {
        if (AP1.backNum == 0 && AP2.backNum == 0)
        {
            TimeNow += Tc;
            AP1.collision();
            AP2.collision();
        }
        else if (AP1.backNum != 0 && AP2.backNum != 0)
        {
            timeGap = min(AP1.backNum, AP2.backNum);
            AP1.backNum -= timeGap;
            AP2.backNum -= timeGap;
            timeGap += 1;
            timeGap *= 9;
            TimeNow += timeGap;
        }
        else
        {
            AP* AP_Point1 = AP1.backNum == 0 ? (&AP1) : (&AP2);
            TimeNow += Ts;
            AP_Point1->reBuild();
            if (TimeNow >= beginTime)
            {
                this->addResult(12);
            }
        }
    }
    cout << " 系统结束运行!  " << endl;
    writeFileMy();
}

void BSS_System::writeFileMy() {
    std::ofstream outputFile("data.txt");
    if (!outputFile) {
        std::cerr << "无法打开文件以写入数据。" << std::endl;
        return;
    }
    int len_Arr = TimeNow_Arr.size();
    // 将数据逐行写入文件
    for (size_t i = 0; i < len_Arr; ++i) {
```

```
        outputFile << TimeNow_Arr[i] << " " << throughput_Arr[i]*100 << " " << average_Throughput_A
rr[i] << "\n";

    // 关闭文件
    outputFile.close();
    std::cout << "数据已成功写入文件! " << std::endl;
}

#include"System1.h"
int main() {
    BSS_System system1;
    //system1.function();
    cout<<Ts<<endl;
    cout<<Tc<<endl;
    return 0;
}
```

第二问

```
#pragma once
#include<iostream>
#include <random>
#include <ctime>
#include<vector>
using namespace std;
static const int cwMax=1024,cwMin=16;
class AP {
public:
    int r_max, m, backNum, wi, r_now;
    static std::mt19937 gen;
    AP() {};
    AP(int r1, int m1);
    void selectBackNum();
    void collision();
    void reBuild();
};

#pragma once
#include<iostream>
#include<vector>
#include <fstream>
#include"AP.h"
using namespace std;
static const int beginTime = 10000, endTime = 2000000;
static const double Ts = 13.6 + (30+1500)*8/275.3+ 32 + 43 + 16, Tc = 13.6 + (30 + 1500) * 8 / 275.
3 + 43 + 65;
class BSS_System
{
public:
    AP AP1, AP2;
```

```
    long long throughput;
    long double TimeNow;
    void function();
    void addResult(int cargo);
    vector<long double>TimeNow_Arr;
    vector<long long>throughput_Arr;
    vector<long double>average_Throughput_Arr;
    BSS_System() :AP1(32, 6), AP2(32, 6), throughput(0), TimeNow(0.0)
    {
        TimeNow_Arr.reserve(20000);
        throughput_Arr.reserve(20000);
        average_Throughput_Arr.reserve(20000);
        cout << "系统建立成功!" << endl;
    }
    void writeFileMy();
};

#include"AP.h"
std::mt19937 AP::gen(static_cast<unsigned int>(std::time(nullptr))); // 初始化静态成员
AP::AP(int r1, int m1) :r_max(r1), m(m1), wi(16), backNum(), r_now(0)
{
    selectBackNum();
    cout << "AP 基塔搭建成功!" << endl;
}
void AP::selectBackNum() {
    // 定义均匀分布
    std::uniform_int_distribution<int> dist(0, wi - 1);
    // 生成随机整数
    this->backNum = dist(gen); // 使用静态生成器
}
void AP::collision() {
    if (r_now < r_max)
    {
        if (wi < 1024)
        {
            wi *= 2;
        }
        this->selectBackNum();
        r_now++;
    }
    else
    {
        this->reBuild();
    }
}
void AP::reBuild() {  //传输成功的时候 直接 rebuild 就行!
    this->r_now = 0;
    this->wi = 16;
```

```
    this->selectBackNum();
}

#include"System1.h"
#include"AP.h"
void BSS_System::addResult(int cargo) {
    this->throughput += cargo;
    long double Temp = this->throughput / ((TimeNow - beginTime) / 1000);
    cout << "NowTime(us) : " << TimeNow
        << "\t  Throughout( 需要 *100)(单位是 Bit): " << this->throughput
        << "\t  average Thoughout: " << Temp << endl;
    this->TimeNow_Arr.push_back(TimeNow);
    this->throughput_Arr.push_back(throughput);
    this->average_Throughput_Arr.push_back(Temp);
}
void BSS_System::function() {
    int timeGap = 0;
    for (int i = 0; i <= 1000000 && TimeNow <= endTime; i++)
    {
        if (AP1.backNum == 0 && AP2.backNum == 0)
        {
            TimeNow += Ts;
            AP1.reBuild();
            AP2.reBuild();
            if (TimeNow >= beginTime)
            {
                this->addResult(24);
            }
        }
        else if (AP1.backNum != 0 && AP2.backNum != 0)
        {
            timeGap = min(AP1.backNum, AP2.backNum);
            AP1.backNum -= timeGap;
            AP2.backNum -= timeGap;
            timeGap += 1;
            timeGap *= 9;
            TimeNow += timeGap;
        }
        else
        {
            AP* AP_Point1 = AP1.backNum == 0 ? (&AP1) : (&AP2);
            TimeNow += Ts;
            AP_Point1->reBuild();
            if (TimeNow >= beginTime)
            {
                this->addResult(12);
            }
        }
```

```
    }
    cout << " 系统结束运行！ " << endl;
    writeFileMy();
}

void BSS_System::writeFileMy() {
    std::ofstream outputFile("data.txt");
    if (!outputFile) {
        std::cerr << "无法打开文件以写入数据。" << std::endl;
        return;
    }
    int len_Arr = TimeNow_Arr.size();
    // 将数据逐行写入文件
    for (size_t i = 0; i < len_Arr; ++i) {
        outputFile << TimeNow_Arr[i] << " " << throughput_Arr[i]*100 << " " << average_Throughput_A
rr[i] << "\n";
    }
    // 关闭文件
    outputFile.close();
    std::cout << "数据已成功写入文件！" << std::endl;
}

#include"System1.h"
int main() {
    BSS_System system1;
    system1.function();
    return 0;
}
```

第三问

```
#pragma once
#include<iostream>
#include <random>
#include <ctime>
#include<vector>
using namespace std;
static const int cwMax=1024,cwMin=16;
class AP {
public:
    int r_max, m, backNum, wi, r_now,AP_state;  //r_max 是 最大重传次数
    long double AP_NextTime;
    static std::mt19937 gen;
    AP() {};
    AP(int r1, int m1);
    void selectBackNum();
    void collision();
    void reBuild();
};
```

```
#pragma once
#include<iostream>
#include<vector>
#include <fstream>
#include"AP.h"
using namespace std;
static const double rat = 158.4;   //物理层的传输速率!
static const int beginTime = 10000, endTime = 20000000, payload_Capacity=10;
static const double TData = 13.6 + (1530 * 8) / rat,
                                Ts = TData + 32 + 43 + 16,
                                Tc = TData + 43 + 65,
                                Tc_s= Tc-Ts,
                                Ts_d= 32 + 43 + 16,
                                Tc_d= 43 + 65;
class BSS_System
{
public:
    AP AP1, AP2;
    long long throughput;
    long double TimeNow;
    void function();
    void addResult();
    vector<long double>TimeNow_Arr;
    vector<long long>throughput_Arr;
    vector<long double>average_Throughput_Arr;
    static std::mt19937 gen1;
    BSS_System() :AP1(5, 6), AP2(5, 6), throughput(0), TimeNow(0.0)
    {
        TimeNow_Arr.reserve(20000);
        throughput_Arr.reserve(20000);
        average_Throughput_Arr.reserve(20000);
        cout << "系统建立成功!" << endl;
    }
    void writeFileMy();
    void stateTransition0(AP& aP_point1, AP& aP_point2);
    void stateTransition1(AP& aP_point1, AP& aP_point2);
    void stateTransition2(AP& aP_point1, AP& aP_point2);
    void stateTransition3(AP& aP_point1, AP& aP_point2);
    void stateTransition4(AP& aP_point1, AP& aP_point2);
};

#include"AP.h"
std::mt19937 AP::gen(static_cast<unsigned int>(std::time(nullptr))); // 初始化静态成员
AP::AP(int r1, int m1) :r_max(r1), m(m1), wi(32), backNum(), r_now(0), AP_state(0), AP_NextTime(0)
{
    selectBackNum();
    cout << "AP 基塔搭建成功!" << endl;
}
```

```
void AP::selectBackNum() {
    // 定义均匀分布
    std::uniform_int_distribution<int> dist(0, wi - 1);
    // 生成随机整数
    this->backNum = dist(gen); // 使用静态生成器
    AP_state = 0;
    AP_NextTime = (this->backNum + 1) * 9;
}
void AP::collision() {
    if (r_now < r_max)
    {
        if (wi < 1024)
        {
            wi *= 2;
        }
        this->selectBackNum();
        r_now++;
    }
    else
    {
        this->reBuild();
    }
}
void AP::reBuild() {  //传输成功的时候 直接 rebuild 就行!
    this->r_now = 0;
    this->wi = 32;
    this->selectBackNum();
}

#include"System1.h"
#include"AP.h"
std::mt19937 BSS_System::gen1(static_cast<unsigned int>(std::time(nullptr))); // 初始化静态成员
std::bernoulli_distribution dist(0.9); // 0.9 的概率生成 1，0.1 的概率生成 0
void BSS_System::addResult() {
    if (this->TimeNow<beginTime)
    {
        return;
    }
    this->throughput += payload_Capacity;
    long double TimeGap= ((TimeNow - beginTime) / 1000);
    long double Temp = this->throughput / TimeGap;
    cout << "NowTime(us) : " << TimeNow
        << "\t 系统总吞吐量(从 10000 us 开始计算 )(单位是 Bit): " << this->throughput * 100
        << "\t 平均吞吐量: " << Temp << endl;
    this->TimeNow_Arr.push_back(TimeNow);
    this->throughput_Arr.push_back(throughput);
    this->average_Throughput_Arr.push_back(Temp);
}
```

```
void BSS_System::function() {
    for (int i = 0; i <= 10000000 && TimeNow <= endTime; i++)
    {
        AP& aP_point1 = AP1.AP_NextTime <= AP2.AP_NextTime? AP1:AP2;
        AP &aP_point2 = AP1.AP_NextTime > AP2.AP_NextTime ? AP1 : AP2;
        TimeNow += aP_point1.AP_NextTime;
        aP_point2.AP_NextTime -= aP_point1.AP_NextTime;
        aP_point1.AP_NextTime -= aP_point1.AP_NextTime;
        switch (aP_point1.AP_state)
        {
        case 0:
            stateTransition0(aP_point1,aP_point2);
            break;
        case 1:
            stateTransition1(aP_point1, aP_point2);
            break;
        case 2:
            stateTransition2(aP_point1, aP_point2);
            break;
        case 3:
            stateTransition3(aP_point1, aP_point2);
            break;
        default:
            stateTransition4(aP_point1, aP_point2);
            break;
        }
    }
    cout << " 系统结束运行!  " << endl;
    writeFileMy();
}
void BSS_System::writeFileMy() {
    std::ofstream outputFile("data.txt");
    if (!outputFile) {
        std::cerr << "无法打开文件以写入数据。" << std::endl;
        return;
    }
    int len_Arr = TimeNow_Arr.size();
    // 将数据逐行写入文件
    for (size_t i = 0; i < len_Arr; ++i) {
        outputFile << TimeNow_Arr[i] << " " << throughput_Arr[i]*100 << " " << average_Throughput_A
rr[i] << "\n";
    }
    // 关闭文件
    outputFile.close();
    std::cout << "数据已成功写入文件! " << std::endl;
}
void BSS_System::stateTransition0(AP& aP_point1, AP& aP_point2) {
    int randomeNum = dist(gen1);
```

```
int indexNext = randomeNum==1?2:4;
long double timeTmep = randomeNum==1? Ts_d : Tc_d;
if (aP_point2.AP_state==0)
{
    if (aP_point2.AP_NextTime==0)  //[0,0] [0,0]
    {
        this->TimeNow += Tc;
        aP_point1.collision();
        aP_point2.collision();
        return;
    }
    else {                                      //[0,d1]
        aP_point1.AP_state = 1;
        aP_point1.AP_NextTime = TData;
        return;
    }
}
else if (aP_point2.AP_state == 1) {
    if (aP_point2.AP_NextTime == 0)  //[1,0]
    {
        aP_point1.AP_state = 1;
        aP_point1.AP_NextTime = TData;
        aP_point2.AP_state = indexNext;
        aP_point2.AP_NextTime = timeTmep;
        return;
    }
    else {                                      //[1,d1]
        aP_point1.AP_state = 3;
        aP_point1.AP_NextTime = TData;
        aP_point2.AP_state = 3;
        return;
    }
}
else if (aP_point2.AP_state == 2) {
    if (aP_point2.AP_NextTime == 0)  //[2,0]
    {
        aP_point1.AP_state = 1;
        aP_point1.AP_NextTime = TData;
        this->addResult();
        aP_point2.reBuild();
        return;
    }
    else {                                      //[2,d1]
        aP_point1.AP_state = 1;
        aP_point1.AP_NextTime = TData;
        return;
    }
}
```

```
    else if (aP_point2.AP_state == 3) {
        if (aP_point2.AP_NextTime == 0)  //[3,0]
        {
            aP_point1.AP_state = 1;
            aP_point1.AP_NextTime = TData;
            aP_point2.AP_state = 4;
            aP_point2.AP_NextTime=Tc_d;
            return;
        }
        else {                                      //[3,d1]
            aP_point1.AP_state = 3;
            aP_point1.AP_NextTime = TData;
            return;
        }
    }
    else {
        if (aP_point2.AP_NextTime == 0)  //[4,0]
        {
            aP_point1.AP_state = 1;
            aP_point1.AP_NextTime = TData;
            aP_point2.collision();
            return;
        }
        else {                                      //[4,d1]
            aP_point1.AP_state = 1;
            aP_point1.AP_NextTime = TData;
            return;
        }
    }
}
void BSS_System::stateTransition1(AP& aP_point1, AP& aP_point2) {
    int randomeNum = dist(gen1);
    int indexNext = randomeNum == 1 ? 2 : 4;
    long double timeTmep = randomeNum == 1 ? Ts_d : Tc_d;
    if (aP_point2.AP_state == 0)
    {
        if (aP_point2.AP_NextTime == 0)  //[1,0] [0,0]
        {
            aP_point1.AP_state = indexNext;
            aP_point1.AP_NextTime = timeTmep;
            aP_point2.AP_state = 1;
            aP_point2.AP_NextTime = TData;
            return;
        }
        else {                                      //[0,d1]
            aP_point1.AP_state = indexNext;
            aP_point1.AP_NextTime = timeTmep;
            return;
```

```
        }
    }
    else if (aP_point2.AP_state == 2) {
        if (aP_point2.AP_NextTime == 0)  //[2,0]
        {
            aP_point1.AP_state = indexNext;
            aP_point1.AP_NextTime = timeTmep;
            this->addResult();
            aP_point2.reBuild();
            return;
        }
        else {                                      //[2,d1]
            aP_point1.AP_state = indexNext;
            aP_point1.AP_NextTime = timeTmep;
            return;
        }
    }
    else if (aP_point2.AP_state == 3) {
        if (aP_point2.AP_NextTime == 0)  //[3,0]
        {
            aP_point1.AP_state = 1;
            aP_point1.AP_NextTime = TData;
            aP_point2.AP_state = 4;
            aP_point2.AP_NextTime = Tc_d;
            return;
        }
        else {                                      //[3,d1]
            aP_point1.AP_state = 3;
            aP_point1.AP_NextTime = TData;
            return;
        }
    }
    else {
        if (aP_point2.AP_NextTime == 0)  //[4,0]
        {
            aP_point1.AP_state = indexNext;
            aP_point1.AP_NextTime = timeTmep;
            aP_point2.collision();
            return;
        }
        else {                                      //[4,d1]
            aP_point1.AP_state = indexNext;
            aP_point1.AP_NextTime = timeTmep;
            return;
        }
    }
}
void BSS_System::stateTransition2(AP& aP_point1, AP& aP_point2) {
```

```
long double temp1 = aP_point1.AP_state, temp2 = aP_point1.AP_NextTime;
aP_point1.reBuild();
this->addResult();
int randomeNum = dist(gen1);
int indexNext = randomeNum == 1 ? 2 : 4;
long double timeTmep = randomeNum == 1 ? Ts_d : Tc_d;
if (aP_point2.AP_state == 0)
{
    if (aP_point2.AP_NextTime == 0)  //[1,0] [0,0]
    {
        aP_point2.AP_state = 1;
        aP_point2.AP_NextTime = TData;
        return;
    }
    else {                                      //[0,d1]
        return;
    }
}
else if (aP_point2.AP_state == 1) {
    if (aP_point2.AP_NextTime == 0)  //[1,0]
    {
        aP_point2.AP_state = indexNext;
        aP_point2.AP_NextTime = timeTmep;
        return;
    }
    else {                                      //[1,d1]
        return;
    }
}
else if (aP_point2.AP_state == 2) {
    return;
}
else if (aP_point2.AP_state == 3) {
    if (aP_point2.AP_NextTime == 0)  //[3,0]
    {
        aP_point2.AP_state = 4;
        aP_point2.AP_NextTime = Tc_d;
        return;
    }
    else {                                      //[3,d1]
        return;
    }
}
else {
    if (aP_point2.AP_NextTime == 0)  //[4,0]
    {
        aP_point2.collision();
        return;
```

```
        }
        else {                                      //[4,d1]
            return;
        }
    }
}
void BSS_System::stateTransition3(AP& aP_point1, AP& aP_point2) {
    int randomeNum = dist(gen1);
    int indexNext = randomeNum == 1 ? 2 : 4;
    long double timeTmep = randomeNum == 1 ? Ts_d : Tc_d;
    aP_point1.AP_state = 4;
    aP_point1.AP_NextTime = Tc_d;
    if (aP_point2.AP_state == 0)
    {
        if (aP_point2.AP_NextTime == 0)  //[1,0] [0,0]
        {
            aP_point2.AP_state = 1;
            aP_point2.AP_NextTime = TData;
            return;
        }
        else {                                      //[0,d1]
            return;
        }
    }
    else if (aP_point2.AP_state == 2) {
        return;
}
    }
    else if (aP_point2.AP_state == 3) {
        if (aP_point2.AP_NextTime == 0)  //[3,0]
        {
            aP_point2.AP_state = 4;
            aP_point2.AP_NextTime = Tc_d;
            return;
        }
        else {                                      //[3,d1]
            return;
        }
    }
    else {
        if (aP_point2.AP_NextTime == 0)  //[4,0]
        {
            aP_point2.collision();
            return;
        }
        else {                                      //[4,d1]
            return;
        }
```

```
    }
}
void BSS_System::stateTransition4(AP& aP_point1, AP& aP_point2) {
    int randomeNum = dist(gen1);
    int indexNext = randomeNum == 1 ? 2 : 4;
    long double timeTmep = randomeNum == 1 ? Ts_d : Tc_d;
    aP_point1.collision();
    if (aP_point2.AP_state == 0)
    {
        if (aP_point2.AP_NextTime == 0)  //[1,0] [0,0]
        {
            aP_point2.AP_state = 1;
            aP_point2.AP_NextTime = TData;
            return;
        }
        else {                                      //[0,d1]
            /*aP_point1.AP_state = 2;
            aP_point1.AP_NextTime = Ts_d;*/
            return;
        }
    }
    else if (aP_point2.AP_state == 1) {
        if (aP_point2.AP_NextTime == 0)  //[1,0]
        {
            aP_point2.AP_state = indexNext;
            aP_point2.AP_NextTime = timeTmep;
            return;
        }
        else {                                      //[1,d1]
            /*aP_point1.AP_state = 3;
            aP_point1.AP_NextTime = TData;
            aP_point2.AP_state = 3;*/
            //aP_point1.AP_NextTime = TData;
            return;
        }
    }
    else if (aP_point2.AP_state == 2) {
        return;
        if (aP_point2.AP_NextTime == 0)  //[2,0]
        {
            aP_point2.reBuild();
            this->addResult();
            return;
        }
        else {                                      //[2,d1]
            /*aP_point1.AP_state = 2;
            aP_point1.AP_NextTime = Ts_d;*/
            /*aP_point2.AP_state = 3;*/
```

```
            //aP_point1.AP_NextTime = TData;
            return;
        }
    }
    else if (aP_point2.AP_state == 3) {
        if (aP_point2.AP_NextTime == 0)  //[3,0]
        {
            aP_point2.AP_state = 4;
            aP_point2.AP_NextTime = Tc_d;
            return;
        }
        else {                                      //[3,d1]
            /*aP_point1.AP_state = 3;
            aP_point1.AP_NextTime = TData;*/
            /*aP_point2.AP_state = 3;*/
            return;
        }
    }
    else {
        if (aP_point2.AP_NextTime == 0)  //[4,0]
        {
            /*aP_point1.AP_state = 2;
            aP_point1.AP_NextTime = Ts_d;*/
            aP_point2.collision();
            return;
        }
        else {                                      //[4,d1]
            return;
        }
    }
}

#include"System1.h"
int main() {
    BSS_System system1;
    system1.function();
    return 0;
}
```

第四问

```
#pragma once
#include<iostream>
#include <random>
#include <ctime>
#include<vector>
using namespace std;
static const int cwMax=1024,cwMin=16;
class AP {
public:
```

```
    int r_max, m, wi, r_now,state_ap;
    double backNum1;
    static std::mt19937 gen;
    AP() {};
    AP(int r1, int m1);
    void selectBackNum();
    void collision();
    void reBuild();
};

#pragma once
#include<iostream>
#include<vector>
#include <fstream>
#include"AP.h"
using namespace std;
static const double rat = 286.8;   //物理层的传输速率!
static const int beginTime = 100000, endTime = 20000000, payload_Capacity = 12;
static const double TData = 13.6 + (30 + 1500) * 8 / rat,
                                Ts = TData + 32 + 43 + 16,
                                Tc = TData + 43 + 65;;
                                /*Tc_s = Tc - Ts,
                                Ts_d = 32 + 43 + 16,
                                Tc_d = 43 + 65;*/
class BSS_System
{
public:
    AP AP1, AP2, AP3;
    long long throughput_all, throughput_ap1, throughput_ap2, throughput_ap3;
    long double TimeNow;
    void function();
    void addResult(int type_index);
    vector<long double>TimeNow_Arr;
    vector<long long>throughput_Arr_all;
    vector<long double>average_Throughput_Arr_all;
    vector<long double>average_Throughput_Arr_AP1;
    vector<long double>average_Throughput_Arr_AP2;
    vector<long double>average_Throughput_Arr_AP3;
    BSS_System() :AP1(5, 6), AP2(5, 6), AP3(5, 6), throughput_all(0), TimeNow(0.0), throughput_ap1
(0), throughput_ap2(0), throughput_ap3(0)
    {
        TimeNow_Arr.reserve(20000);
        throughput_Arr_all.reserve(20000);
        average_Throughput_Arr_all.reserve(20000);
        average_Throughput_Arr_AP1.reserve(20000);
        average_Throughput_Arr_AP2.reserve(20000);
        average_Throughput_Arr_AP3.reserve(20000);
        cout << "系统建立成功!" << endl;
```

```
    }
    void writeFileMy();
};

#include"AP.h"
std::mt19937 AP::gen(static_cast<unsigned int>(std::time(nullptr))); // 初始化静态成员
const int cwMin1 = 32;
AP::AP(int r1, int m1) :r_max(r1), m(m1), wi(cwMin1), backNum1(), r_now(0), state_ap(0)
{
    selectBackNum();
    cout << "AP 基塔搭建成功!" << endl;
}
void AP::selectBackNum() {
    // 定义均匀分布
    std::uniform_int_distribution<int> dist(0, wi - 1);
    // 生成随机整数
    this->backNum1 = (dist(gen)+1)*9; // 使用静态生成器
    this->state_ap = 0;
}
void AP::collision() {
    if (r_now < r_max)
    {
        if (wi < 1024)
        {
            wi *= 2;
        }
        this->selectBackNum();
        r_now++;
    }
    else
    {
        this->reBuild();
    }
}
void AP::reBuild() {  //传输成功的时候 直接 rebuild 就行!
    this->r_now = 0;
    this->wi = cwMin1;
    this->selectBackNum();
}

#include"System1.h"
#include"AP.h"
void BSS_System::addResult(int type_index) {
    if (TimeNow<beginTime)
    {
        return;
    }
    this->throughput_all += payload_Capacity;
```

```
    if (type_index==1){ throughput_ap1 += payload_Capacity;}
    else if (type_index == 2) { throughput_ap2 += payload_Capacity; }
    else{throughput_ap3 += payload_Capacity;}
    long double timeGap = ((TimeNow - beginTime) / 1000);
    long double Temp = this->throughput_all / timeGap;   // 平均总吞吐量
    long double Temp1 = this->   throughput_ap1 / timeGap;
    long double Temp2 = this->   throughput_ap2 / timeGap;
    long double Temp3 = this->   throughput_ap3 / timeGap;
    cout << "NowTime(us) : " << TimeNow
        << "\t AP1 平均吞吐量: " << Temp1
        << "\t AP2 平均吞吐量: " << Temp2
        << "\t AP3 平均吞吐量: " << Temp3
        << "\t  平均总吞吐量: " << Temp
        << "\t  总吞吐量(单位 Bit): " << this->throughput_all * 100
        << "\r" << endl;
    this->TimeNow_Arr.push_back(TimeNow);
    this->throughput_Arr_all.push_back(throughput_all);
    this->average_Throughput_Arr_all.push_back(Temp);
    this->average_Throughput_Arr_AP1.push_back(Temp1);
    this->average_Throughput_Arr_AP2.push_back(Temp2);
    this->average_Throughput_Arr_AP3.push_back(Temp3);
}
void BSS_System::function() {
    int timeGap = 0;
    for (int i = 0; i <= 100000 && TimeNow <= endTime; i++)
    {
        if (AP2.backNum1==0)
        {
            if (AP1.backNum1!=0&&AP3.backNum1!=0)
            {
                TimeNow += Ts;
                AP2.reBuild();
                this->addResult(2);
            }
            else {
                TimeNow += Tc;
                AP2.collision();
                if (AP1.backNum1==0)
                {
                    AP1.collision();
                }
                if (AP3.backNum1 == 0)
                {
                    AP3.collision();
                }
            }
        }
        else if(AP2.backNum1 != 0 && AP1.backNum1!=0 && AP3.backNum1!=0){
```

```
            long double backNumTemp = min(AP1.backNum1,min(AP2.backNum1, AP3.backNum1));
            AP1.backNum1 -= backNumTemp;
            AP2.backNum1 -= backNumTemp;
            AP3.backNum1 -= backNumTemp;
            TimeNow += backNumTemp;
            continue;
        }
        else
        {
            while (!((AP1.backNum1 > 0&&AP1.state_ap==0)&&(AP3.backNum1 > 0&&AP3.state_ap==0)))
            {
                AP* ap_point1 = AP1.backNum1 <= AP3.backNum1 ?&AP1 : &AP3;  //指向剩余时间较少的ap
                AP* ap_point2 = AP1.backNum1 > AP3.backNum1 ? &AP1 : &AP3;   //指向较多的ap
                (ap_point2)->backNum1 -= (ap_point1)->backNum1;
                TimeNow+= (ap_point1)->backNum1;
                (ap_point1)->backNum1 -= (ap_point1)->backNum1;
                if (AP1.backNum1 == 0&&AP1.state_ap==0)
                {
                    AP1.state_ap = 1;
                    AP1.backNum1 = Ts;
                }
                else if(AP1.backNum1 == 0)
                {
                    this->addResult(1);
                    AP1.reBuild();
                }
                if (AP3.backNum1 == 0&&AP3.state_ap==0)
                {
                    AP3.state_ap = 1;
                    AP3.backNum1 = Ts;
                }
                else if(AP3.backNum1 == 0)
                {
                    this->addResult(3);
                    AP3.reBuild();
                }
            }
        }
    }
    cout << " 系统结束运行!  " << endl;
    writeFileMy();
}
void BSS_System::writeFileMy() {
    std::ofstream outputFile("data.txt");
    if (!outputFile) {
        std::cerr << "无法打开文件以写入数据。" << std::endl;
        return;
    }
```

```
    int len_Arr = TimeNow_Arr.size();
    // 将数据逐行写入文件
    outputFile << " 时间节点(us) :  " << " " << "AP1 平均吞吐量 : " << " " << " AP2 平均吞吐量 : " <
< " " << "AP3 平均吞吐量:" << " " << " 平均总吞吐量: " << " " <<"总吞吐量(单位 Bit): " << " " << "\n
";
    for (size_t i = 0; i < len_Arr; ++i) {
        outputFile << TimeNow_Arr[i] << " " << average_Throughput_Arr_AP1[i] << " " << average_Thro
ughput_Arr_AP2[i] << " " << average_Throughput_Arr_AP3[i] <<" " << average_Throughput_Arr_all[i] <<
 " " << throughput_Arr_all[i] * 100 << " " << "\n";
    }
    // 关闭文件
    outputFile.close();
    std::cout << "数据已成功写入文件! " << std::endl;
}


#include"System1.h"
int main() {
    BSS_System system1;
    system1.function();
    return 0;
}
```