LTE-V2X System Simulation 项目

系统级仿真平台建设

无线资源管理设计文档

北京邮电大学移动通信实验室

**目录**

[1. 概论 1](#_Toc468872895)

[2. 场景介绍以及仿真假设 2](#_Toc468872896)

[2.1. 场景介绍 2](#_Toc468872897)

[2.2. 仿真假设 2](#_Toc468872898)

[3. RRM\_TDM\_DRA 4](#_Toc468872899)

[3.1. 概述 4](#_Toc468872900)

[3.2. 分簇 4](#_Toc468872901)

[3.3. 频域资源划分 6](#_Toc468872902)

[3.4. 资源预留机制 7](#_Toc468872903)

[3.5. 冲突避让机制 8](#_Toc468872904)

[3.6. 详细流程 9](#_Toc468872905)

[4. RRM\_RR 11](#_Toc468872906)

[4.1. 频域资源划分 11](#_Toc468872907)

[4.2. 优先级维护 11](#_Toc468872908)

[4.3. 详细流程 12](#_Toc468872909)

[5. RRM\_ICC\_DRA 14](#_Toc468872910)

[5.1. 概述 14](#_Toc468872911)

[5.2. 分簇 14](#_Toc468872912)

[5.3. 冲突避让机制以及优先级维护 15](#_Toc468872913)

[5.4. 详细流程 15](#_Toc468872914)

[6. 数据结构 17](#_Toc468872915)

[6.1. 概述 17](#_Toc468872916)

[6.2. 详细参数 17](#_Toc468872917)

[参考文献 23](#_Toc468872918)

[附录A 修改历史 24](#_Toc468872919)

# 概论

本文档详细介绍了无线资源管理单元的设计框架，共有三种实现方式，第一种是集中式资源管理方案(RRM\_RR)，第二种是基于时分复用的分布式资源管理方案(RRM\_TDM\_DRA)，第三种是并发式的分布式资源管理方案(RRM\_ICC\_DRA)。第2章介绍了3GPP-36.885协议定义的两种仿真通信场景，并给出了仿真假设。第3章详细介绍了分布式资源管理的设计方案。第4章介绍了集中式资源管理的设计方案。第5章给出了系统级仿真平台中该模块的各个数据结构的定义以及详细含义

# 场景介绍以及仿真假设

## 场景介绍

根据3GPP-36.885协议[1]，通信方式分为3-A和3-B，见图 2‑1。其中对于模式3-A，车辆A，通过uplink将信息发送给eNodeB，然后eNodeB在downlink通以广播的方式将信息发送给覆盖范围内的RSU，最后每个RSU将信息通过sidelink转发给RSU覆盖范围内的UE群。对于模式3-B，车辆A，通过sidelink将信息发送给路边单元RSU，随后RSU通过uplink将接收到的来自车辆A的信息发送给eNodeB，随后eNodeB在downlink以广播方式将信息发送给覆盖范围内的UE群。

图 2‑1 通信场景示意图

## 仿真假设

在LTE-V2X仿真平台中，根据3GPP-36.885协议[1]，作出了以下假设：

* 场景3-A与3-B中的RSU均为UE Type。
* RSU在仿真持续时间内均为静止状态。
* 对于高速环境， RSU均匀分布在道路中心，两个RSU的间隔为100m，见图 2‑2。
* 对于城区环境，RSU的分布于所处的道路位置有关，见图 2‑3。
* 每个十字路口的中心放置一个RSU
* 对于纵向道路，在两个十字路口RSU之间插入2个RSU，相邻RSU的间隔为144.3m
* 对于横向道路，在两个十字路口RSU之间插入1个RSU，相邻RUS的间隔为125m



图 2‑2高速环境RSU分布图



图 2‑3 城区环境RSU分布图

# RRM\_TDM\_DRA

## 概述

LTEV2X仿真平台采用的RRM\_TDM\_DRA分布式方案包含了三种策略

* Collision avoidance based on sensing (P1)：冲突避免机制，车辆在自由选择资源块进行数据包传输前，会先检测可用的资源块，通过测量对应频段上的参考信号功率来判断该资源块是否被占用。当车辆成功占用资源块后，在该车辆剩余的传输时间段内，会在该频段上持续发送参考信号，以表明持续占有该资源块
* Enhanced random resource selection (P2)：根据不同的业务将频域资源划分为多个独立的资源块，每种业务将在对应的块中进行资源的选择，另外，独立划分出一个共有的区域用于具有最高优先级的紧急事件的传输
* Location-based resource allocation (P3)：在RSU的配合下，以地理位置以及道路(有向道路)为单位进行地理位置的划分，将同一个RSU内的车辆分入不同的簇之中

## 分簇

为了减少车辆冲突的概率，我们在RSU的配合下，对RSU覆盖范围内的车辆进行分簇，在城区场景下，依据RSU所处的位置，可以分为两种：其一位于十字路口，其二位于道路中央。这两种类型的RSU所分得的簇的数量是不同的，位于十字路口的RSU共有8个簇，位于道路中央的RSU共有四个簇，详见图 3‑1。



图 3‑1 城区分簇示意图

将RSU分为多个簇后，我们可以将RSU的一个时域调度单元，划分为多个时段，每个簇内的车辆只能在该簇所处于的时段内才能进行数据包的发送，以位于道路中央的RSU为例，该RSU的时域调度单元被分为了4个时段，时段的长度与该簇内车辆的数目成正比。另外，为了保证紧急事件的高时效性，与高优先级，我们单独划分出一个频段用于传输紧急事件，而该频段在时域上不进行划分，即只要紧急事件触发，在任何时段都可以进行接入，保证了紧急性事件具有较低的时延特性，详见图 3‑2



图 3‑2 RSU时域调度单元划分示意图

## 频域资源划分

目前LTEV2X系统级平台只能支持固定的资源块的选择，而不能根据包长的大小以及事件时延的需求来自由选择占用的RB数量，我们根据不同的业务类型设计了几种不同的Pattern(一个Pattern为1个或多个RB)。于是车辆所能选择的最小可调用单元(时频单元)从RB提升为为Pattern。

我们将频域大致划分为三个部分，详见图 3‑3。其中，灰色部分为紧急性业务的可占用频段，最小可占用的时频资源单位是PatternA，一个PattenrA对应了3个连续的RB，并且在时域上没有依据簇来进行划分，即任何时候，只要紧急事件触发，就可以在灰色部分频段进行接入；绿色部分为周期性业务的可用频段，最小可占用的时频资源单位是PatternB，一个PatternB对应了5个连续的RB；黄色部分为数据业务的可用频段，最小可占用的时频资源单位是PatternC，一个PatternC对应了10个连续的RB。对于PatternB和PatternC在时域上呈现了不同的色度，同一种色度表示同一个簇的可调用时间段，簇内的车辆只能在该簇对应的可调用时段中才能竞争对应的PatternB或PatternC(PatternB或PatternC依据事件类型而定)。



图 3‑3 频域资源划分

## 资源预留机制

在我们的LTEV2X系统级仿真平台中，我们对已经成功接入(接入的TTI时刻无碰撞)的车辆采用了Pattern预留机制：当一个车辆成功接入后，将在该车辆传输完毕之前，或者发生位置更新导致所在RSU或者所在簇发生了变动之前，持续发送参考信号，来标明该Pattern在该簇对应的时段已被占用，避免同一簇内其他车辆尝试在该Pattern上进行接入，造成不必要的冲突。

对于每个Pattern，预留一小部分频段用于传输参考信号，每一个簇单独占据一个频段，用于表示该簇所在的可调度时段内该Pattern是否被占用，详见图 3‑4。从上到下依次为用于发送簇1-簇4的参考信号频段，位于簇1的车辆B，在TTI=26时尝试进行接入，接入前车辆B在簇1所对应的参考信号频段进行参考信号检测，发现该时段内该Pattern尚未被占用，(假设车辆B在可用Pattern的选择中选取了当前Pattern，并且接入后无冲突)于是车辆B成功接入该Pattern，并且在簇1对应的可调用时段内，车辆B可以在该Pattern内进行数据包的发送。随后在TTI=51时刻，同样位于簇1的车辆D要在该时段内选择相应的Pattern来进行数据包的发送，在发送前车辆D在簇1所对应的参考信号频段进行参考信号检测，发现该频段能检测到参考信号功率，于是车辆D将该Pattern排除在可用Pattern之外，不会在该Pattern上尝试接入。



图 3‑4 冲突避免机制

另外，对于同一可用Pattern，同时可能会被多个车辆选用，此时便会产生冲突，在冲突发生后，便终止此次传输，在下一个可调用时段再次进行资源的竞争

## 冲突避让机制

我们采用载波监听多路访问/冲突检测方法（CSMA/CD）。控制过程包含四个处理内容：侦听、发送、检测、冲突处理：

* 侦听：UE在发送数据前先侦听一下是否有数据正在传送（线路是否忙）。若“忙”则进入后述的“退避”处理程序，进而进一步反复进行侦听工作。若“闲”，则进行数据发送；
* 发送：当确定要发送数据后，进行数据的发送；
* 数据发送后，也可能发生数据碰撞。因此，要对数据边发送，边接收，以判断是否冲突了；
* 当确认发生冲突后，进入冲突处理程序。有两种冲突情况：
* 若在侦听中发现线路忙，则等待一个延时后再次侦听，若仍然忙，则继续延迟等待，一直到可以发送为止。每次延时的时间不一致，由退避算法确定延时值；
* 若发送过程中发现数据碰撞，先发送阻塞信息，强化冲突，再进行侦听工作，以待下次重新发送；

**退避机制**：当出现线路冲突时，如果冲突的各站点都采用同样的退避间隔时间，则很容易产生二次、三次的碰撞。因此，要求各个站点的退避间隔时间具有差异性。这要求通过退避算法来实现。

我们采用截断的二进制指数退避算法（退避算法之一）：

当一个站点发现线路忙时，要等待一个延时时间T，然后再进行侦听工作。延时时间T以以下算法决定：



其中，t表示最小时隙，K表示此刻已经侦听的次数，表示最大时隙。该表达式的含义是：在范围内随机选择整数，作为本次侦听前等待的时隙数目。当超过时，作特殊处理。



图 3‑5 冲突避让机制

## 详细流程

整个资源竞争的流程详见3.6



图 3‑6 RRM\_TDM\_DRA流程

# RRM\_RR

LTEV2X仿真平台实现了轮询调度模块，轮询调度以RSU为单位，在RSU的配合下，对该RSU覆盖范围内的车辆进行调度，统一分配时频资源。

## 频域资源划分

整个LTEV2X所占用的总带宽大约为50个RB，同样地，我们将用户的最小可调用单元从RB提升为Pattern，一个Pattern表示10个连续的RB，在不同的Pattern上并行进行轮询调度，最大程度提升系统的吞吐率



图 4‑1 RRM\_RR模式Pattern划分

## 优先级维护

对于紧急性业务与非紧急性业务我们给予了不同的优先级，对于LTEV2X仿真平台来说，优先级可以通过维护一个优先级链表来完成，详见图 4‑2。Head与Point1之间的是紧急事件，Point1与Point2之间是非紧急事件，紧急事件会插入到Point1位置之后，而非紧急事件则插入到Point2之后，这样就保证了紧急事件的高优先级



图 4‑2 优先级链表维护示意图

## 详细流程

轮询调度详细流程见图 4‑3。



图 4‑3 RRM\_RR流程

# RRM\_ICC\_DRA

## 概述

LTEV2X仿真平台采用的RRM\_ICC\_DRA分布式方案包含了三种策略

* Collision avoidance based on sensing (P1)：冲突避免机制，车辆在自由选择资源块进行数据包传输前，会先检测可用的资源块，通过测量对应频段上的参考信号功率来判断该资源块是否被占用。当车辆成功占用资源块后，在该车辆剩余的传输时间段内，会在该频段上持续发送参考信号，以表明持续占有该资源块
* Enhanced random resource selection (P2)：将频域资源划分为多个资源块，车辆将在空闲的资源块上自由选择
* Location-based resource allocation (P3)：在RSU的配合下，以地理位置以及道路(有向道路)为单位进行地理位置的划分，将同一个RSU内的车辆分入不同的簇之中

## 分簇

为了减少车辆冲突的概率，我们在RSU的配合下，对RSU覆盖范围内的车辆进行分簇，在城区场景下，依据RSU所处的位置，可以分为两种：其一位于十字路口，其二位于道路中央。这两种类型的RSU所分得的簇的数量是不同的，位于十字路口的RSU共有8个簇，位于道路中央的RSU共有四个簇，详见图 5‑1。



图 5‑1 城区分簇示意图

将RSU分为多个簇后，与RRM\_TDM\_DRA不同，我们将每个簇作为一个独立的调度单元，不再进行时域上的划分，而是同时同频进行资源竞争。

## 冲突避让机制以及优先级维护

RRM\_ICC\_DRA采用与RRM\_TDM\_DRA相同的冲突避让机制，其机制在3.5小节中已经详细说明，这里就不再赘述。

值得说明的是，与RRM\_TDM\_DRA不同，RRM\_ICC\_DRA并没有采用针对不同业务类型进行频域资源划分，因此该方案的优先级体现在冲突避让机制的参数配置上，为了划分事件的优先等级，我们通过根据事件分配不同的以及，优先级较大的事件，例如紧急事件，会具有较小的t以及较小的，通过这种策略可以有效地降低冲突时高等级事件的避让时间与避让次数

## 详细流程

RRM\_ICC\_DRA详细流程见图 5‑2



图 5‑2 RRM\_ICC\_DRA流程

# 数据结构

## 概述

为了充分地模块化无线资源管理单元，我们使用了继承结构来实现这一目标。首先，定义了接口类RRM，相关的定义放在了RRM.h中，该RRM类定义了该模块下的共有数据类型，以及定义了该模块的所有可调用接口

我们通过将分布式资源管理类型RRM\_TDM\_DRA，集中式轮询调度类型RRM\_RR继承接口类RRM，并相应地做出了各种具体的实现

## 详细参数

下面分别列出接口类RRM以及三个实现类RRM\_TDM\_DRA、RRM\_RR、RRM\_ICC\_DRA的重要数据成员以及成员方法

表格 6‑1 RRM数据成员表

|  |  |
| --- | --- |
| 数据成员 | 含义 |
| System\* m\_Context | 指向System的指针 |
| RRM\_RSU\*\* m\_RSUAry | RRM视图下的RSU容器 |
| RRM\_VeUE\*\* m\_VeUEAry | RRM视图下的VeUE容器 |

表格 6‑2 RRM成员函数表

|  |  |
| --- | --- |
| 成员函数 | 含义 |
| virtual void initialize() = 0 | 初始化RRM视图下的RSU VeUE容器 |
| virtual void cleanWhenLocationUpdate() = 0 | 当发生位置更新时，清除缓存的调度相关信息 |
| virtual void schedule() = 0 | 流程总控函数 |

表格 6‑3 RRM\_TDM\_DRA数据成员表

|  |  |
| --- | --- |
| 数据成员 | 含义 |
| std::list<int> m\_SwitchEventIdList | 用于存放进行RSU切换的车辆，暂时保存的作用 |
| std::vector<std::vector<std::list<int>>> m\_InterferenceVec | 用于存放指定车辆指定Pattern的干扰列表(只保留RSU内簇间干扰)   * 外层下标为VeUEId * 内层下标为PatternIdx(绝对量) |
| int m\_ThreadNum | 多线程总数 |
| std::vector<std::thread> m\_Threads | 多线程容器 |
| std::vector<std::pair<int, int>> m\_ThreadsRSUIdRange | 分配给每个线程的RSUId范围 |

表格 6‑4 RRM\_TDM\_DRA成员函数表

|  |  |
| --- | --- |
| 成员函数 | 含义 |
| void informationClean() | 资源分配信息清空 |
| void groupSizeBasedTDM(bool t\_ClusterFlag) | 基于簇大小的时分复用 |
| void uniformTDM(bool t\_ClusterFlag) | 均匀分配给每个簇时域资源 |
| void updateAccessEventIdList(bool t\_ClusterFlag) | 更新等待链表 |
| void processEventList() | 处理事件链表，刻触发的事件放入RSU的等待链表中 |
| void processScheduleInfoTableWhenLocationUpdate() | 地理位置更新时，处理调度表 |
| void processWaitEventIdListWhenLocationUpdate() | 地理位置更新时，处理等待链表 |
| void processSwitchListWhenLocationUpdate() | 地理位置更新时，处理转接表 |
| void processWaitEventIdList() | 处理等待链表，生成接入链表 |
| void selectRBBasedOnP123() | 基于P1/P2/P3的资源选择 |
| void delaystatistics() | 时延统计 |
| void conflictListener() | 帧听冲突，冲突后避让 |
| void transimitPreparation() | 计算干扰信道响应 |
| void transimitStart() | 模拟传输开始，更新调度信息，累计吞吐量 |
| void transimitEnd() | 模拟传输结束，更新状态 |
| void writeScheduleInfo() | 记录调度信息日志 |
| void writeTTILogInfo(int t\_TTI, EventLogType t\_EventLogType, int t\_EventId, int t\_FromRSUId, int t\_FromClusterIdx, int t\_FromPatternIdx, int t\_ToRSUId, int t\_ToClusterIdx, int t\_ToPatternIdx, std::string t\_Description); | 以TTI为主键记录日志 |
| void writeClusterPerformInfo() | 写入分簇信息的日志 |

表格 6‑5 RRM\_RR成员表

|  |  |
| --- | --- |
| 数据成员 | 含义 |
| std::list<int> m\_SwitchEventIdList | 用于存放进行RSU切换的车辆，暂时保存的作用 |
| std::vector<std::vector<std::list<int>>> m\_InterferenceVec | 用于存放指定车辆指定Pattern的干扰列表(只保留RSU内簇间干扰)   * 外层下标为VeUEId * 内层下标为PatternIdx(绝对量) |
| int m\_ThreadNum | 多线程总数 |
| std::vector<std::thread> m\_Threads | 多线程容器 |
| std::vector<std::pair<int, int>> m\_ThreadsRSUIdRange | 分配给每个线程的RSUId范围 |

表格 6‑6 RRM\_RR函数表

|  |  |
| --- | --- |
| 成员函数 | 含义 |
| void informationClean() | 资源分配信息清空 |
| void updateAccessEventIdList(bool t\_ClusterFlag) | 更新等待链表 |
| void processEventList() | 处理事件链表，刻触发的事件放入RSU的等待链表中 |
| void processWaitEventIdListWhenLocationUpdate() | 地理位置更新时，处理等待链表 |
| void processSwitchListWhenLocationUpdate() | 地理位置更新时，处理转接表 |
| void processWaitEventIdList() | 处理等待链表，生成接入链表 |
| void roundRobin() | 轮询调度，分配当前TTI的资源(就是更新ScheduleTable) |
| void delaystatistics() | 时延统计 |
| void transimitPreparation() | 计算干扰信道响应 |
| void transimitStart() | 模拟传输开始，更新调度信息，累计吞吐量 |
| void transimitEnd() | 模拟传输结束，更新状态 |
| void writeScheduleInfo() | 记录调度信息日志 |
| void writeTTILogInfo(int t\_TTI, EventLogType t\_EventLogType, int t\_EventId, int t\_FromRSUId, int t\_FromClusterIdx, int t\_FromPatternIdx, int t\_ToRSUId, int t\_ToClusterIdx, int t\_ToPatternIdx, std::string t\_Description); | 以TTI为主键记录日志 |
| void writeClusterPerformInfo() | 写入分簇信息的日志 |

表格 6‑7 RRM\_ICC\_DRA数据成员表

|  |  |
| --- | --- |
| 数据成员 | 含义 |
| std::list<int> m\_SwitchEventIdList | 用于存放进行RSU切换的车辆，暂时保存的作用 |
| std::vector<std::vector<std::list<int>>> m\_InterferenceVec | 用于存放指定车辆指定Pattern的干扰列表(只保留RSU内簇间干扰)   * 外层下标为VeUEId * 内层下标为PatternIdx(绝对量) |
| int m\_ThreadNum | 多线程总数 |
| std::vector<std::thread> m\_Threads | 多线程容器 |
| std::vector<std::pair<int, int>> m\_ThreadsRSUIdRange | 分配给每个线程的RSUId范围 |

表格 6‑8 RRM\_ICC\_DRA成员函数表

|  |  |
| --- | --- |
| 成员函数 | 含义 |
| void informationClean() | 资源分配信息清空 |
| void updateAccessEventIdList(bool t\_ClusterFlag) | 更新等待链表 |
| void processEventList() | 处理事件链表，刻触发的事件放入RSU的等待链表中 |
| void processScheduleInfoTableWhenLocationUpdate() | 地理位置更新时，处理调度表 |
| void processWaitEventIdListWhenLocationUpdate() | 地理位置更新时，处理等待链表 |
| void processSwitchListWhenLocationUpdate() | 地理位置更新时，处理转接表 |
| void processWaitEventIdList() | 处理等待链表，生成接入链表 |
| void selectRBBasedOnP123() | 基于P1/P2/P3的资源选择 |
| void delaystatistics() | 时延统计 |
| void conflictListener() | 帧听冲突，冲突后避让 |
| void transimitPreparation() | 计算干扰信道响应 |
| void transimitStart() | 模拟传输开始，更新调度信息，累计吞吐量 |
| void transimitEnd() | 模拟传输结束，更新状态 |
| void writeScheduleInfo() | 记录调度信息日志 |
| void writeTTILogInfo(int t\_TTI, EventLogType t\_EventLogType, int t\_EventId, int t\_FromRSUId, int t\_FromClusterIdx, int t\_FromPatternIdx, int t\_ToRSUId, int t\_ToClusterIdx, int t\_ToPatternIdx, std::string t\_Description); | 以TTI为主键记录日志 |
| void writeClusterPerformInfo() | 写入分簇信息的日志 |

# 参考文献

1. 3GPP TR 36.885: “Study on LTE-based V2X Services”.
2. R1-160307, “UE autonomous resource selection” ,Huawei, HiSilicon, RAN1#84, Feb. 15 – 19, 2016.

# 附录A 修改历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修改历史 | | | | |
| 日期 | 姓名 | 备注 | 旧版本号 | 新版本号 |
| 2015.6.22 | 贺辰枫、饶鑫 | 起草 |  | 1.1 |
| 2016.9.14 | 贺辰枫 | 修改 | 1.1 | 1.2 |
| 2016.12.7 | 贺辰枫 | 修改 | 1.2 | 2.1 |