LTE-V2X System Simulation 项目

系统级仿真平台建设

业务模型与控制单元设计文档

北京邮电大学移动通信实验室

**目录**

[1. 概述 3](#_Toc468959379)

[2. 概要设计 3](#_Toc468959380)

[2.1. 车联网的业务和消息 3](#_Toc468959381)

[2.2. 业务模型 4](#_Toc468959382)

[2.2.1. V2V 4](#_Toc468959383)

[2.2.2. V2I 5](#_Toc468959384)

[2.3. 性能评估 5](#_Toc468959385)

[2.3.1. 时延统计 6](#_Toc468959386)

[2.3.2. 包传输率 6](#_Toc468959387)

[2.3.3. 包接收率 6](#_Toc468959388)

[2.3.4. 系统吞吐量 7](#_Toc468959389)

[3. 模块设计 7](#_Toc468959390)

[3.1. 整体流程 7](#_Toc468959391)

[3.2. 吞吐量统计 8](#_Toc468959392)

[3.3. 生成数据包 9](#_Toc468959393)

[3.4. 释放资源 9](#_Toc468959394)

[4. 数据结构 9](#_Toc468959395)

[4.1. 概述 9](#_Toc468959396)

[4.2. 详细参数 9](#_Toc468959397)

[参考文献 15](#_Toc468959398)

[附录A 修改历史 16](#_Toc468959399)

# 概述

本文档主要分为四个章节，其中第二章介绍LTE-V2X系统级仿真平台中业务模型和控制单元，其主要包含三部分：

* 简要介绍了仿真中涉及的车联网业务和消息分类
* 业务模型部分：分别介绍了对于V2V, V2I的业务模型
* 性能评估部分：介绍了几类性能指标，例如时延统计，包传输率，包接收率，吞吐量统计，并简要说明了参考的计算方法

第三章则详细介绍了业务模型和控制单元所包含的各个模块的设计，包括各模块功能描述和流程设计；第四章介绍了各个模块的所包含的参数设定和仿真结果。

# 概要设计

本章节主要介绍仿真中涉及到的业务和消息分类、几类业务模型和控制单元的总体描述、模块划分和整体流程。

## 车联网的业务和消息

车联网应用可以支持的这些业务大体可以分为两类，安全相关的和非安全相关的。安全相关的业务可以包括交通事故预防、交通事故和交通拥堵原因调查。非安全相关的业务可以包括所有与道路交通安全无关的个人数据业务，包括驾驶员和乘车人在车内可能需要的各类业务，例如导航、网页浏览、网络游戏和微博等娱乐业务。

其中安全相关的业务还可以进一步分为周期性的安全消息和事件触发的紧急安全消息。

* 周期性安全消息：例如信标消息，发送模式主要是广播，通过周期性的广播车辆可以将自己当前的状态通知给周围的其他车辆，因为车辆的状态信息对于周围所有的邻居车辆都十分重要。这类消息主要包含车辆的速度、方向和位置等信息。产生这种数据包的频率为5-10HZ。数据包长度在字节200-400bytes。
* 事件触发型的紧急安全消息：一般用于道路上突发的安全事故或者安全隐患的广播。一旦在道路上发现安全隐患，具有广播能力的车联网终端须要马上发送紧急安全消息通知附近车辆。事件触发的安全消息可能是通知车内异常情况，包括发送机、制动和动力系统故障，也可以通知路上的异常情况，例如碰撞警告、急刹车和低能见度等情况。这类数据包的大小略小于周期性安全消息，长度为100-300bytes。

## 业务模型

业务模型主要是计算传输数据包的大小，主要通过业务traffic的函数实现。在LTE-V2X系统平台中可以采用同样的模式，同时根据自身情况进行修改。

在TR 36.885协议中规定业务模型有两种：事件触发型机制和时间触发型机制。在仿真平台构建中，由于采用的是事件驱动机制，因此业务模型只考虑事件触发型机制。仿真中的每个车辆都要根据此模型产生消息。

协议的相关规范设定如下：

### V2V

对于时间触发型（周期性）的交通情况下，消息的产生周期已被严格的定义如下：

|  |  |  |
| --- | --- | --- |
| 车辆撒点情景 | 车辆绝对速度（km/h） | 消息产生周期（ms） |
| 高速公路1 | 140 | 100 |
| 高速公路2 | 70 | 100 |
| 城区1 | 60 | 100 |
| 城区2 | 15 | 100 |
| 城区3 | 15 | 500 |

此外，对于周期性交通，对信息的大小进行假设如下：一个300-bytes的信息，加上四个190-bytes的信息，以及一个在产生300-bytes信息的时间间隔里在车辆之中随机产生的信息。需要注意的是，在进行性能分析时，信息的大小可以不必考虑。

对于事件触发型的业务模型，每个车辆每秒的事件到达率是遵循泊松分布的，事件到达率X的值由公司自己确定。一旦事件被触发，在100ms的时间内会产生上述的六个信息，同时，对于物理层的事件触发型业务，对信息大小的工作假设就为800bytes。

### V2I

* V2I/I2V Traffic Model 1:

和V2V统计的性能指标基本相同，延迟需求也为100ms，且每个信息之间相互独立，通信范围规定为V2V情形的一半

* V2I/I2V Traffic Model 2:

信息产生频率为1或者0.1Hz，且延迟要求为>100ms（例如1000ms）信息大小固定值为300Bytes，且V2I的信息是附加在V2V的信息之上的，此时公司应该提供在评估中假定的通信范围的取值，且该值应该比model 1要大

注：I2V只存在于城市的道路岔口场景中，因为在高速公路时I2V可以看作以100m的间隔在道路中间均匀分布的情形，且I2V的信息产生点的位置坐标和UE类型的RSU相同

## 性能评估

在物理层的性能评估部分，主要会涉及到如下性能评估参数：

* 平均传输时延
* 包传输率
* 包接收率
* 信道吞吐量

\*可能考虑的参数：包内接收Packet Inter-Reception (PIR)，即从节点A到节点B两个不同包传输的两个成功接收情况的时间间隔

最后数据统计的部分，将对仿真平台的上述性能评估参数的结果进行统计和说明。

### 时延统计

常见的时延种类有四种：

* 传播时延：第一个比特从发送节点到接收节点在传输链路上经历的时间
* 发送时延（也叫传输时延）：发送节点在传输链路上开始发送分钟的第一个比特至发送完该分组的最后一个比特所需要的时间
* 处理时延：从一个节点的输入端到达该节点的输出端所经历的时延
* 排队时延：传输或处理前的等待时间

一般的时延测量方法如下：

Sender

被测网络

Receiver

Start time End time

图 2‑1网络传输时延测试原理

由上图可得，被测系统两个特定节点间的网络传输时延可被唯一确定，网络传输时延D=End time – Start time.

对于一个网络的两个特定时间节点间的实际网络延迟可以被完全确定。但仍需解决时间的同步问题。

### 包传输率

包传输率Packet delivery ratio (PDR) 可以定义为在标记的车辆传输范围内的所有车辆成功接收标记车辆广播发送的消息的几率。即正确传输数据包的统计度量，进一步表明了选择路径能力和路径稳定性。

注：丢包准则——自动丢弃传输时间超过所设定的最大传输时间（T\_drop）的文件，可得丢包率。

### 包接收率

包接收率Packet Reception Ratio (PRR) 可以定义为成功接收的UE或者车辆数量/一个包内位置在（a, b）范围内的UE或者车辆总数。

另外，包接收率PRR可分两类：第一类是PRR的累积分布函数（CDF），即可以随机设定b为城区50m道路且车辆速度为15km/h（只适用于系统级仿真）；第二类是平均PRR值。

### 系统吞吐量

一般化的用户吞吐量可以定义为成功传输数据大小（文件大小）/ 下载该数据或者文件所需要的时间。而从系统的角度，在一个产生包的间隔内，感知到这个信道的所有车辆总共又会产生数据包，因此需要考虑总体的系统吞吐量。

# 模块设计

本章节详细介绍了LTE-V2X业务模型与控制单元的各个模块，首先介绍了业务模型的整体流程，并进行逐模块介绍。

## 整体流程



图 3‑1业务模型流程图

平台仿真中，业务模型主要为广播业务模型，对于周期性的安全消息的业务模型的主体流程可以简化为上图所示；而对于事件触发型的紧急安全消息，第一步则需要按照泊松分布产生数据包并发送，其他流程不变。

本模型主要流程如下，首先初始化参数，包括描述广播时间间隔的随机数类型和参数，广播数据包大小服从的随机数类型和参数，然后调度发送，调度业务结束，释放资源。下面以上各部分进行详细介绍。

## 吞吐量统计

根据ACK/NACK进行文件处理。若是ACK，则累加当前用户接收文件大小，并根据已接收文件大小判断是否接收完成。当接收完成则将文件的大小累加到吞吐量中，并设置相关变量表示该呼叫请求在当前TTI内结束；否则需要判断是否超时。

若是NACK，则判断是否达到最大重传次数或者超过最大时延。需要注意的是，处理中要考虑到当前用户传输为两个码字时码字间的相互影响。

具体流程如下图所示：



图 3‑2文件处理流程图

## 生成数据包

生成数据包模块，可根据计算得出的数据包大小，统计当前呼叫发起后已发送文件的大小。在以下两种情况下，置发送数据包大小为零：

* 用户文件传输失败，文件丢包；
* 用户在当前呼叫结束前已发送完成；

## 释放资源

主要功能为根据用户文件下载情况，对完成下载和丢包用户执行结束处理，释放系统资源。

# 数据结构

## 概述

业务模型与控制单元主要包括事件类Event的定义，以及模块类型TMC的定义。

## 详细参数

表格 4‑1 Event数据成员表

|  |  |
| --- | --- |
| 数据成员 | 含义 |
| const int m\_EventId | 每个事件都有唯一的Id，Id从0开始编号 |
| const int m\_VeUEId | 该事件对应的用户的Id |
| const int m\_TriggerTTI | 事件触发的TTI时刻 |
| const MessageType m\_MessageType | 该消息的类型 |
| const int m\_PackageNum | 消息的数据包总数 |
| m\_TransimitPackageNum | 实际传输的数据包总包数 |
| int m\_PackageLossNum | 实际发生丢包的数据包总包数 |
| const std::vector<int> m\_BitNumPerPackage | 每个包的bit数量 |
| bool m\_IsFinished | 标记所有包是否传输完毕 |
| int m\_CurrentPackageIdx | 标记即将要传输的bit所在的包序号 |
| int m\_RemainBitNum; | currentPackageIdx所指向的package剩余待传输的bit数 |
| std::vector<bool> m\_PackageIsLoss | 记录每个数据包是否丢包  一个包只能被丢一次，因此用布尔值来标记，而不是用丢包次数来标记 |
| std::vector<double> m\_PackageTransimitDistance | 记录包传输完毕后，车辆距离信宿的距离   * 注意，丢包也算是包传输完毕，只不过传输完毕后无法解码而已 * 包含m\_PackageLossDistance * 累积状态，reset不重置 |
| std::vector<double> m\_PackageLossDistance | 丢包时距离信宿的距离，用于分析PDR之用   * 当没有发生RSU切换时，丢包次数不会超过总包数 * 但是当发生RSU切换时，由于需要重置，因此传输的总包数会大于总包数 * 此时丢包的总包数会大于总包数 * 累积状态，reset不重置 * 于是对于给定一个距离d * m\_PackageLossDistance中该值出现的次数为n1 * m\_PackageTransimitDistance中该值出现的次数为n2 * n1/n2就是该距离下的一个丢包率 |
| const int m\_InitialWindowSize | 回退窗初始大小 |
| const int m\_MaxWindowSize | 回退窗最大大小 |
| int m\_CurWindowSize | 回退窗当前大小。初始化为初始窗大小，调用一次退避后变为两倍或者最大窗大小 |
| int m\_WithdrawalTime | 还需退避多少TTI |
| std::list<std::string> m\_LogTrackList | 记录该事件的所有日志 |
| int m\_ConflictNum | 冲突次数 |
| int m\_SendDelay | 发送时延 |
| int m\_QueueDelay | 排队时延 |
| int m\_ProcessDelay | 处理时延，从一个节点的输入端到达该节点的输出端所经历的时延 |

表格 4‑2 Event成员函数表

|  |  |
| --- | --- |
| 成员函数 | 含义 |
| get\*(); | getter函数，这里都省略不解释了 |
| void increaseSendDelay() | 递增传输时延 |
| void increaseQueueDelay() | 递增等待时延 |
| void reset() | 重新发送该消息时，重置信息各个状态 |
| bool tryAcccess() | 尝试接入，并更新状态 |
| int transimit(int t\_TransimitMaxBitNum, double t\_Distance); | 更新信息状态   * transimitMaxBitNum为本次该时频资源可传输的最大bit数 * 但本次传输的实际bit数可以小于该值，并返回实际传输的bit数量 |
| void conflict() | 冲突之后更新信息状态 |
| void packetLoss(double t\_Distance) | 当前传输的包发生丢包的状态更新 |
| std::string toString() | 生成格式化的字符串 |
| std::string toLogString(int t\_NumTab) | 生成用于日志输出的格式化的字符串 |
| void addEventLog(int t\_TTI, EventLogType t\_EventLogType, int t\_FromRSUId, int t\_FromClusterIdx, int t\_FromPatternIdx, int t\_ToRSUId, int t\_ToClusterIdx, int t\_ToPatternIdx, std::string t\_Description) | 添加日志 |

表格 4‑3 TMC数据成员表

|  |  |
| --- | --- |
| 数据成员 | 含义 |
| System\* m\_Context | 指向System的指针 |
| TMC\_RSU\*\* m\_RSUAry | TMC视图下的RSU容器 |
| TMC\_VeUE\*\* m\_VeUEAry | TMC视图下的VeUE容器 |
| std::vector<Event> m\_EventVec | 事件容器，下标代表事件ID |
| std::vector<std::list<int>> m\_EventTTIList | 以TTI为下标的事件容器   * 事件触发链表，m\_EventTTIList[i]代表第i个TTI的事件表   \*/ |
| std::vector<std::list<int>> m\_EergencyVeUEIdListOfTriggerTTI | 紧急事件触发列表   * 外层下标代表TTI * 内层list存放对应TTI触发紧急事件的车辆Id |
| std::vector<std::list<int>> m\_DataVeUEIdListOfTriggerTTI | 数据事件触发列表   * 外层下标代表TTI * 内层list存放对应TTI触发数据事件的车辆Id |
| std::vector<std::vector<int>> m\_TTIRSUThroughput | 吞吐率  外层下标为TTI，内层下标为RSUId |
| std::vector<int> m\_VeUEEmergencyNum | 每辆车紧急事件触发的次数  用于验证泊松分布，仿真中并无用处 |
| std::vector<int> m\_VeUEDataNum | 每辆车数据业务事件触发的次数  用于验证泊松分布，仿真中并无用处 |
| std::vector<int> m\_TransimitSucceedEventNumPerEventType | 每类事件成功传输的数目  外层下标为事件种类 |
| std::ofstream m\_FileEventLogInfo;  std::ofstream m\_FileEventListInfo;  std::ofstream m\_FileStatisticsDescription;  std::ofstream m\_FileEmergencyDelayStatistics  std::ofstream m\_FilePeriodDelayStatistics  std::ofstream m\_FileDataDelayStatistics  std::ofstream m\_FileEmergencyPossion  std::ofstream m\_FileDataPossion  std::ofstream m\_FileEmergencyConflictNum  std::ofstream m\_FilePeriodConflictNum  std::ofstream m\_FileDataConflictNum  std::ofstream m\_FileTTIThroughput  std::ofstream m\_FileRSUThroughput  std::ofstream m\_FilePackageLoss  std::ofstream m\_FilePackageTransimit | 日志文件 |

表格 4‑4 TMC成员函数表

|  |  |
| --- | --- |
| 成员函数 | 含义 |
| System\* getContext() | 获取系统类的指针 |
| void initialize() | 初始化RSU VeUE内该单元的内部类 |
| void eventTrigger() | 生成事件链表 |
| void buildEmergencyDataEventTriggerTTI() | 生成事件链表 |
| void processStatistics() | 仿真结束后统计各种数据 |
| void writeEventListInfo() | 写入事件列表的信息 |
| void writeEventLogInfo() | 写入以事件的日志信息 |

# 参考文献

1. 3GPP TR 36.885: “Study on LTE-based V2X Services”.
2. 3GPP TR 36.814: "Further advancements for E-UTRA physical layer aspects".

# 附录A 修改历史

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 起草 | | | | | |
| 日期 | 姓名 | | 版本号 | | |
| 2016.6.14 | 秦华婵 | | 1.1 | | |
| 修改历史 | | | | | |
| 日期 | 姓名 | 备注 | | 旧版本号 | 新版本号 |
| 2016.12.8 | 贺辰枫 | 修改最后的数据结构 | | 1.1 | 2.1 |