

Simple TDMA implementation for OMNeT++

Generated by Doxygen 1.8.19

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 Class Documentation	5
3.1 AbstractFrame Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 isEnd	5
3.1.2.2 isStart	6
3.1.2.3 seqNum	6
3.1.2.4 size	6
3.1.2.5 ts	6
3.2 AbstractPacket Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Member Data Documentation	7
3.2.2.1 size	7
3.3 Rlc Class Reference	7
3.3.1 Detailed Description	8
3.3.2 Constructor & Destructor Documentation	8
3.3.2.1 ~Rlc()	8
3.3.3 Member Function Documentation	8
3.3.3.1 getBufferStatus()	8
3.3.3.2 requestFrame()	8
3.3.4 Member Data Documentation	9
3.3.4.1 rlcCore	9
3.4 RlcCore Class Reference	9
3.4.1 Detailed Description	10
3.4.2 Constructor & Destructor Documentation	10
3.4.2.1 RlcCore()	10
3.4.2.2 ~RlcCore()	10
3.4.3 Member Function Documentation	10
3.4.3.1 addFrame()	10
3.4.3.2 addPacket()	11
3.4.3.3 getCompletePackets()	11
3.4.3.4 getFrame()	11
3.4.3.5 hasFrame()	11
3.4.4 Member Data Documentation	12
3.4.4.1 packetQueue	12
3.4.4.2 receivedFrames	12
3.5 TdmaMac Class Reference	12

3.6 TdmaScheduler Class Reference	13
Index	15

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AbstractFrame	5
AbstractPacket	6
AckingMac	
TdmaMac	12
cSimpleModule	
TdmaScheduler	13
LayeredProtocolBase	
Rlc	7
RlcCore	9

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[AbstractFrame](#)

The [AbstractFrame](#) is a data structure to decouple packet representation from the internal OM \leftrightarrow NeT++/INET data structure 5

[AbstractPacket](#)

The [AbstractPacket](#) is a data structure to decouple packet representation from the internal O \leftrightarrow MNeT++/INET data structure 6

[Rlc](#)

[Rlc](#) is the OMNeT++ wrapper implemenatation of the RLC layer. It borrows its functionality from the [RlcCore](#) class 7

[RlcCore](#)

[RlcCore](#) is a pure C++ implementation of the fragmentation/reassembly logic 9

[TdmaMac](#) 12

[TdmaScheduler](#) 13

Chapter 3

Class Documentation

3.1 AbstractFrame Struct Reference

The [AbstractFrame](#) is a data structure to decouple packet representation from the internal OMNeT++/INET data structure.

```
#include <RlcCore.h>
```

Public Attributes

- int [size](#)
- int [seqNum](#)
- bool [isEnd](#)
- bool [isStart](#)
- double [ts](#)

3.1.1 Detailed Description

The [AbstractFrame](#) is a data structure to decouple packet representation from the internal OMNeT++/INET data structure.

3.1.2 Member Data Documentation

3.1.2.1 isEnd

```
bool AbstractFrame::isEnd
```

indicates whether a packets ends with this frame

3.1.2.2 isStart

```
bool AbstractFrame::isStart
```

indicates whether a new packet starts with this frame

3.1.2.3 seqNum

```
int AbstractFrame::seqNum
```

sequence number

3.1.2.4 size

```
int AbstractFrame::size
```

size of the frame

3.1.2.5 ts

```
double AbstractFrame::ts
```

timestamp to be used to discard very old frames which could not be reassembled

The documentation for this struct was generated from the following file:

- `src/rlc/RlcCore.h`

3.2 AbstractPacket Struct Reference

The [AbstractPacket](#) is a data structure to decouple packet representation from the internal OMNeT++/INET data structure.

```
#include <RlcCore.h>
```

Public Attributes

- int [size](#)

3.2.1 Detailed Description

The [AbstractPacket](#) is a data structure to decouple packet representation from the internal OMNeT++/INET data structure.

3.2.2 Member Data Documentation

3.2.2.1 size

```
int AbstractPacket::size
```

size of the packet

The documentation for this struct was generated from the following file:

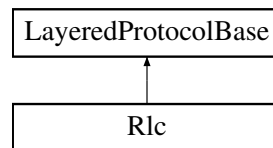
- src/rlc/RlcCore.h

3.3 Rlc Class Reference

[Rlc](#) is the OMNeT++ wrapper implementation of the RLC layer. It borrows its functionality from the [RlcCore](#) class.

```
#include <Rlc.h>
```

Inheritance diagram for Rlc:



Public Member Functions

- int [getBufferStatus](#) ()
- void [requestFrame](#) (int size)

Protected Member Functions

- [~Rlc](#) ()
- virtual void **initialize** (int stage) override
- virtual void **sendUp** (cMessage *message)
- virtual void **sendDown** (cMessage *message)
- virtual void **handleMessageWhenDown** (cMessage *msg) override
- virtual void **handleStartOperation** (LifecycleOperation *operation) override
- virtual void **handleStopOperation** (LifecycleOperation *operation) override
- virtual void **handleCrashOperation** (LifecycleOperation *operation) override
- virtual bool **isInitializeStage** (int stage) override
- virtual bool **isModuleStartStage** (int stage) override
- virtual bool **isModuleStopStage** (int stage) override
- virtual bool **isUpperMessage** (cMessage *message) override
- virtual bool **isLowerMessage** (cMessage *message) override
- virtual void **handleUpperPacket** (Packet *packet) override
- virtual void **handleLowerPacket** (Packet *packet) override
- virtual void **handleSelfMessage** (cMessage *message) override

Protected Attributes

- [RlcCore](#) * `rlcCore` = nullptr
- int `upperLayerInGateId` = -1
- int `upperLayerOutGateId` = -1
- int `lowerLayerInGateId` = -1
- int `lowerLayerOutGateId` = -1

3.3.1 Detailed Description

[Rlc](#) is the OMNeT++ wrapper implementation of the RLC layer. It borrows its functionality from the [RlcCore](#) class.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 ~Rlc()

```
Rlc::~~Rlc ( ) [protected]
```

Standard destructor

3.3.3 Member Function Documentation

3.3.3.1 getBufferStatus()

```
int Rlc::getBufferStatus ( )
```

This function returns the current buffer status in bytes

Returns

buffer fill level in bytes

3.3.3.2 requestFrame()

```
void Rlc::requestFrame (
    int size )
```

Called from the MAC layer to instruct the RLC layer to send down a frame in the required size

Parameters

<i>size</i>	of the frame in bytes
-------------	-----------------------

3.3.4 Member Data Documentation

3.3.4.1 rlcCore

```
RlcCore* Rlc::rlcCore = nullptr [protected]
```

Reference to the [RlcCore](#) instance

The documentation for this class was generated from the following files:

- `src/rlc/Rlc.h`
- `src/rlc/Rlc.cc`

3.4 RlcCore Class Reference

[RlcCore](#) is a pure C++ implementation of the fragmentation/reassembly logic.

```
#include <RlcCore.h>
```

Public Member Functions

- [RlcCore](#) ()
- [~RlcCore](#) ()
- void [addFrame](#) ([AbstractFrame](#) frame)
- void [addPacket](#) ([AbstractPacket](#) packet)
- bool [hasFrame](#) ()
- [AbstractFrame](#) [getFrame](#) (int size)
- vector< [AbstractPacket](#) > [getCompletePackets](#) ()

Protected Attributes

- vector< [AbstractFrame](#) > [receivedFrames](#)
- vector< [AbstractPacket](#) > [packetQueue](#)

3.4.1 Detailed Description

[RlcCore](#) is a pure C++ implementation of the fragmentation/reassembly logic.

More to come...

Author

Konrad Fuger, TUHH ComNets

Date

August 2020

3.4.2 Constructor & Destructor Documentation

3.4.2.1 RlcCore()

```
RlcCore::RlcCore ( )
```

Standard Constructor

3.4.2.2 ~RlcCore()

```
RlcCore::~~RlcCore ( )
```

Standard Destructor

3.4.3 Member Function Documentation

3.4.3.1 addFrame()

```
void RlcCore::addFrame (
    AbstractFrame frame )
```

Adds a received frame to the

Parameters

AbstractFrame	frame: The frame to be stored
-------------------------------	-------------------------------

3.4.3.2 addPacket()

```
void RlcCore::addPacket (
    AbstractPacket packet )
```

Adds a packet received from the network layer

Parameters

<i>AbstractPacket</i>	packet: The packet received
-----------------------	-----------------------------

3.4.3.3 getCompletePackets()

```
vector<AbstractPacket> RlcCore::getCompletePackets ( )
```

returns all complete packets

Returns

A list of all successfully reassembled packets

3.4.3.4 getFrame()

```
AbstractFrame RlcCore::getFrame (
    int size )
```

returns a frame in the required size

Parameters

<i>int</i>	size: Size of the required frame.
------------	-----------------------------------

3.4.3.5 hasFrame()

```
bool RlcCore::hasFrame ( )
```

determines whether there is still a frame to send

Returns

decision whether there is a frame

3.4.4 Member Data Documentation

3.4.4.1 packetQueue

```
vector<AbstractPacket> RlcCore::packetQueue [protected]
```

Vector to hold all received packets not transmitted yet

3.4.4.2 receivedFrames

```
vector<AbstractFrame> RlcCore::receivedFrames [protected]
```

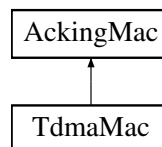
Vector to hold all received frames

The documentation for this class was generated from the following files:

- src/rlc/RlcCore.h
- src/rlc/RlcCore.cc

3.5 TdmaMac Class Reference

Inheritance diagram for TdmaMac:



Public Member Functions

- void **setSchedule** (vector< int > newSchedule)

Protected Member Functions

- void **initialize** (int stage) override
- virtual void **handleUpperPacket** (Packet *packet) override
- virtual void **handleSelfMessage** (cMessage *message) override
- virtual void **acked** (Packet *frame) override
- void **receiveSignal** (cComponent *source, simsignal_t signalID, intval_t value, cObject *details) override
- simtime_t **getNextTransmissionSlot** ()
- simtime_t **getFirstSlotInNextFrame** ()
- bool **hasGrant** ()
- bool **hasFutureGrant** ()

Protected Attributes

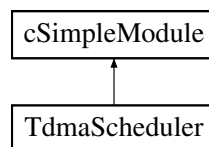
- `TdmaScheduler * scheduler` = nullptr
- `vector< int > schedule`
- `int nodeId`
- `double slotDuration`
- `int frameLength`
- `int currentTransmissionAttempts` = 0
- `int numRetries`
- `cMessage * transmissionSelfMessage` = nullptr

The documentation for this class was generated from the following files:

- `src/mac/TdmaMac.h`
- `src/mac/TdmaMac.cc`

3.6 TdmaScheduler Class Reference

Inheritance diagram for TdmaScheduler:



Public Member Functions

- `int registerClient (TdmaMac *mac, int bufferStatus)`
- `void reportBufferStatus (int nodeId, int bufferStatus)`

Protected Member Functions

- `void initialize (int stage)` override
- `void createSchedule ()`
- `virtual void handleMessage (cMessage *message)` override

Protected Attributes

- `int numNodes` = 0
- `map< int, TdmaMac * > clients`
- `map< int, int > bufferStatus`
- `vector< int > schedule`
- `double frameDuration`
- `double slotDuration`
- `int frameLength`
- `cMessage * schedulingSelfMessage` = nullptr

The documentation for this class was generated from the following files:

- `src/scheduler/TdmaScheduler.h`
- `src/scheduler/TdmaScheduler.cc`

Index

- ~Rlc
 - Rlc, [8](#)
- ~RlcCore
 - RlcCore, [10](#)
- AbstractFrame, [5](#)
 - isEnd, [5](#)
 - isStart, [5](#)
 - seqNum, [6](#)
 - size, [6](#)
 - ts, [6](#)
- AbstractPacket, [6](#)
 - size, [7](#)
- addFrame
 - RlcCore, [10](#)
- addPacket
 - RlcCore, [10](#)
- getBufferStatus
 - Rlc, [8](#)
- getCompletePackets
 - RlcCore, [11](#)
- getFrame
 - RlcCore, [11](#)
- hasFrame
 - RlcCore, [11](#)
- isEnd
 - AbstractFrame, [5](#)
- isStart
 - AbstractFrame, [5](#)
- packetQueue
 - RlcCore, [12](#)
- receivedFrames
 - RlcCore, [12](#)
- requestFrame
 - Rlc, [8](#)
- Rlc, [7](#)
 - ~Rlc, [8](#)
 - getBufferStatus, [8](#)
 - requestFrame, [8](#)
 - rlcCore, [9](#)
- RlcCore, [9](#)
 - ~RlcCore, [10](#)
 - addFrame, [10](#)
 - addPacket, [10](#)
 - getCompletePackets, [11](#)
 - getFrame, [11](#)
 - hasFrame, [11](#)
 - packetQueue, [12](#)
 - receivedFrames, [12](#)
 - RlcCore, [10](#)
- rlcCore
 - Rlc, [9](#)
- seqNum
 - AbstractFrame, [6](#)
- size
 - AbstractFrame, [6](#)
 - AbstractPacket, [7](#)
- TdmaMac, [12](#)
- TdmaScheduler, [13](#)
- ts
 - AbstractFrame, [6](#)