DMDL Manual and its GNU Radio API Reference

Peng Wang

Draft Version 0.1, Thursday, 25 January 2018

Table of Contents

1 Introduction 3

2 Decomposite MAC Description Language 4

2.1 Command 4

2.2 Blocks 4

2.3 Port List....................................5

2.3.1 Labels....................................7

2.3.2 Time Related Blocks 8

2.4 Frame Related Blocks....................................11

2.4.1 Interface Related Blocks....................................19

2.4.2 Auxiliary Blocks 21

3 GR-DMDL – DMDL in GNU Radio....................................26

3.1 How to build GR-DMDL 26

3.2 Commands in GR-DMDL....................................26

3.3 Blocks in GR-DMDL 27

3.4 Calibration 27

3.4.1 Testbed Configurations 27

3.4.2 Waveform measurement 27

3.4.3 Single frame transmission 28

3.4.4 Handshaking transmission....................................30

3.5 Examples and Results....................................32

1 Introduction

This user manual explains how you can take advantage of Decomposite MAC Description Language (DMDL) to design your own communication protocols and how you can run your protocol using its GNU Radio based implementation, GR-DMDL. DMDL following the modular design philosophy, in which the atomic elements are the elementary MAC function components selected from a large-scale survey on different types of MAC that we have conducted. In DMDL, a MAC protocol is modeled as a hierarchical finite state machine with synchronous data flow model. MAC protocols are composed as directed graph with the MAC function components at the vertices. A token-like controlling unit is defined to pass control signals and data elements among MAC function components via the directed connections. DMDL is platform independent thus can be used not only the designing tool but also the guide to implement powerful and convenient prototyping tools on other SDR platforms.

GR-DMDL is an exemplary implementation of Decomposite MAC Description Language (DMDL). GR-DMDL is aimed for rapid development and modeling of MAC protocols. It is implemented as a plug-in of GNU Radio, so it has the full access to all GNU Radio blocks and library and it is able to access all Software Defined Radios (SDR) supported by GNU Radio. Protocol designers and developers can use it to implement many types of MAC protocols including 802.11 DCF. DMDL is an open source project which can be download from GitHub[*].

This document contains two parts. In Chapter 2, we introduce DMDL and describe its syntax, provides definitions to all its components and gives examples. It is worth noting that DMDL itself does not require any specific way of implementation. Developers may select the way to implement blocks according to the development environment. In Chapter 3 we introduce the GR-DMDL including examples, performance demonstrations and analysis. It is also a brief manual to show how to download, compile and use GR-DMDL. Protocol designers who would like to use GR-DMDL to fast implement their protocols and conduct measurements may jump over Chapter 2 but at least read Chapter 3. Chapter 2 is the guide for tool makers who would like to implement their own version of DMDL on other platforms. However, it is also informative for protocol designers because it shows the detail of how the blocks and signaling systems are defined.

[*]https://github.com/joqoko/gr-dmdl.git

2 Decomposite MAC Description Language

DMDL is a graphical MAC designing tool aimed at rapid designing and modeling of MAC protocols. This chapter can be used as a guidance to design MAC protocol directly use DMDL, or a directive standard for developers to implement DMDL in any Software Defined Radio (SDR) environment.

2.1 Command

Command is container used in DMDL to convey control signal and data among blocks. A command may contain arbitrary number of items. An empty command may be also useful because the generation or even the existence of a command also carries information, e.g., it can be used to activate a timer. Commands are passed from blocks to blocks via ports. DMDL has no restrict on what information should be carried by a command for a block. But the logic inside the block may require specific information or data. Therefore, it is the block developer to add appropriate warning or error feedback into their code if the required information is not included in the arriving command. On the protocol designers' side, be familiar with API of blocks before connecting may significantly shorten the time used for debugging and improve the readability of the protocol. Due to the graphical nature of DMDL, commands are iconized. Designers may highlight the information in the command by filling text or figures of the information. Examples of commands used in this manual are shown in Figure 1. It is worth noting that a command may containing any combination of information which may not be able to be completely displayed.

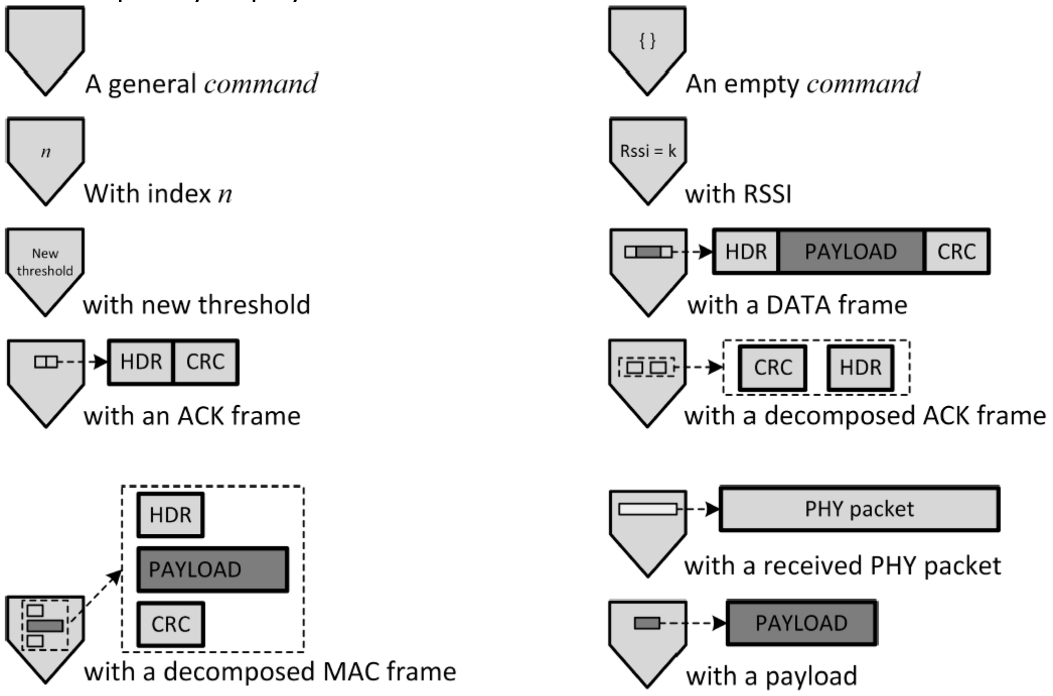


Figure 1. Examples of commands.

2.2 Blocks

Blocks are the elementary semantics of DMDL which are the actions or processes in MAC layer or responses to MAC related events. In DMDL, a typical block is shown in Figure 2. The block name is equivalent to the function name in conventional programming languages which briefly indicates the purpose and major usage of the block. The Properties are the configurations that users can set for the blocks before the execution. Two types of port are defined in DMDL, namely the input port and the output port. The input ports are the destination of directed edges while the output ports are the source. A block has to have at least one port regardless of their types so that it can be connected to the other parts of the protocol. A command is always passed from an output port to input ports. An output port can connect to multiple input ports, and vice versa. Both the arrival and departure of command in input ports and output ports can be asynchronous, i.e., a command can arrive at different input ports at different time. Exceptions are the time related blocks which will be elaborated later in the section. A command can also be passed from an output port to input ports of the same block.

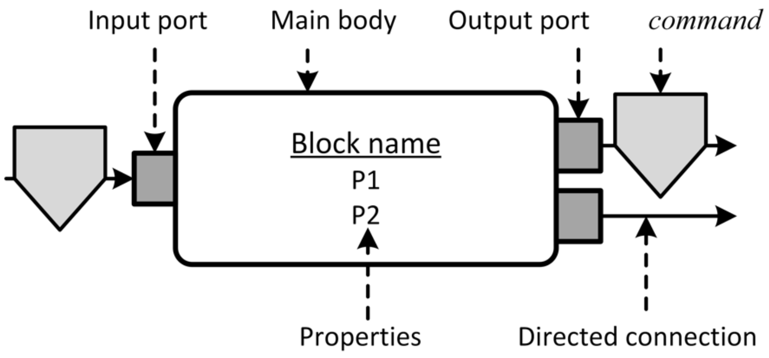


Figure 2. Block architecture.

Developers are welcome to add their new blocks into the DMDL library. A detailed guidance of how to add new blocks are also given in this section. For more information please contact us via pengwang@kth.se or https://github.com/joqoko.

2.3 Port List

Ports in DMDL are defined according to the functionalities and independently from blocks. Blocks may have various number of ports. In DMDL, ports are marked by two colors. A port marked by grey ▨ means it is a mandatory port of the block. Developers must implement it in their code. A port with white background ☐ is an optional port that developer may not implement it or add some controlling properties so that users may hide it if the port is not in use to avoid cable salad problem. Developers should name the new defined ports with different names. It is highly recommended the developers to first read the port list and then reuse the existing ports in designing customized blocks if the same functionalities are required to achieve the highest compatibility.

Ports	Type	Description
☐ A HALT	In	The command arrived at this port halts the ongoing operation, e.g., to halt the countdown of the timer. Warning information should be provided if there is

		no ongoing operation is running when a command arrives.
$\boxed{\text{B}}$ **B**EGIN	In	The port to activate the block.
$\boxed{\text{C}}$ PHY PA**C**KET	In/Out	The port for sending and receiving PHY packet.
$\boxed{\text{D}}$ **D**EQUEUE	In	Command arrives in this port let the block pop a command from the built-in queue. If there is no built-in queue, the port should not be used.
$\boxed{\text{E}}$ **E**ND	In	A general port used to pass the input command to the following blocks if the command itself is not changed and indicate the operation of the current block is finished.
$\boxed{\text{F}}$ **F**AIL	Out	This port is used to export the command which is failed in being checked by the internal logic of the block, such as the address check or CRC check.
$\boxed{\text{G}}$ RECONFI**G**URE	In	The port is used to receive command containing new configurations of the block.
$\boxed{\text{H}}$ FLUS**H**	In	Flush the built-in queue. If there is no built-in queue, the port should not be used.
$\boxed{\text{I}}$ CONF**I**RM	In	Used for blocks which expecting the second input that are confirm or acknowledge the first input. For example, the received Clear-To-Send (CTS) frame should arrive at this port to response the waiting Ready-To-Send (RTS) frame.
$\boxed{\text{K}}$ PIC**K**	Out	If a block has actions of selection such as pick up commands from multiple inputs, this port should be used to export the selected command.
$\boxed{\text{L}}$ FU**L**L	Out	This port is used for a block with a built-in queue to indicate that the queue is full.
$\boxed{\text{M}}$ EMPTY	Out	To indicate the built-in queue or other registers are have no command buffered.
$\boxed{\text{N}}$ **N**OT FULL	Out	This port is used for a block with a built-in queue to indicate that the queue is not full.
$\boxed{\text{O}}$ SIGNAL/NOISE RATI**O**	In/Out	To convey the commands that contains the signal to noise ratio of the received PHY packet.
$\boxed{\text{P}}$ **P**ASS	Out	Used by blocks that need to make decision such as the CRC check or address check. If the check is passed, the block should export the command from this port.
$\boxed{\text{Q}}$ EN**Q**UEUE	In	For blocks with built-in queue to insert new command.
$\boxed{\text{R}}$ **R**SSI	In	Reset the inner counter or state transit to the default settings.

S **S**TOP	In	This port is used to end the ongoing execution in the block if any. Warning information should be able to be displayed if there is no ongoing operation is running when a command arrives.
T CS **T**HRESHOLD	In/Out	To convey commands with the threshold of Carrier Sensing (CS) among blocks.
U DEQ**U**EUED	Out	This port is used by a block with a built-in queue to export the popped commands.
V **V**irtual CS	In	Designed for passing the duration of virtual CS if it is used in the protocol.
X E**X**HAUSTED	Out	This port is used by a block with a built-in queue to indicate the following block that the queue is empty.
Y STA**Y**	Out	If a block has actions of selection such as pick up commands from multiple inputs, this port should be used to export the unselected command.
R **R**SSI	In/Out	To convey commands with Received Signal Strength Indication (RSSI) information among blocks.
DATA **DATA** FRAME	In/Out	To convey DATA frames between blocks.
ACK **ACK** FRAME	In/Out	To convey ACK frames between blocks.
BCO **B**EA**CO**N FRAME	In/Out	To convey BEACON frames between blocks.
RTS **RTS** FRAME	In/Out	To convey RTS frames between blocks.
CTS **CTS** FRAME	In/Out	To convey CTS frames between blocks.
AMS **AMS**DU FRAME	In/Out	To convey Aggregated MAC Service DATA Unit (AMSDU) frames between block.
AMP **AMP**DU FRAME	In/Out	To convey Aggregated MAC Protocol DATA Unit (AMPDU) frames between block.
BACK **B**lock **ACK** FRAME	In/Out	To convey Block ACK frames between blocks.

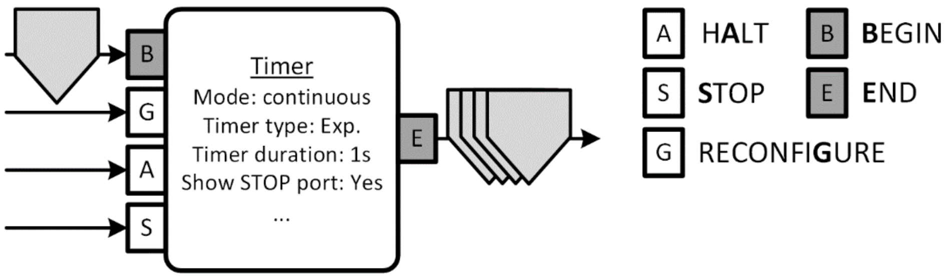
2.3.1 Labels

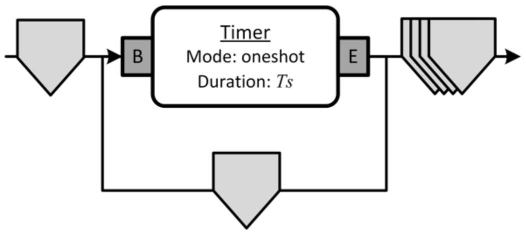
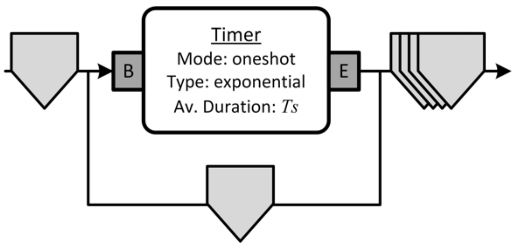
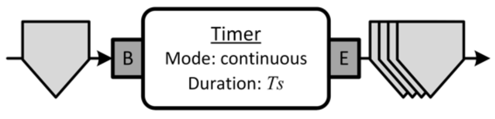
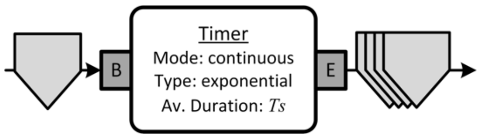
It is not straightforward to have a tree-structured categorization for all blocks, so blocks are introduced roughly according to their functionalities and purposes in this section. Blocks are the containers of MAC functionalities. To gain a better understanding of blocks, they are labeled according to their purposes and data format they processed. The labels currently in use are as follows,

Labels	Description
TIME	The execution of the block may take time, e.g., timer.

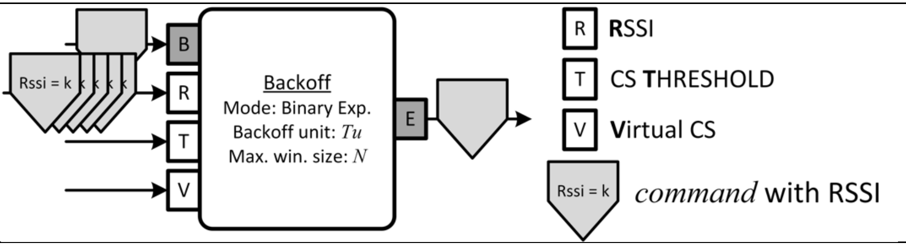
MAC	The block may process MAC frames or execute MAC layer logics.
PHY	The block may process PHY packets or contain PHY layer computations.
QUEUE	The block has at least one built-in queue.
REG	The block may store commands for later usage, but not a queue.
POLY	The block may have polymorphic representations. The properties and the number of ports may vary. User may find the switch in the properties of the block.
MATH	The block includes mathematical calculations and may export command containing pure numerical result.
API	The block may contain external APIs such as hardware interfaces or network layer protocol interfaces.

2.3.2 Time Related Blocks

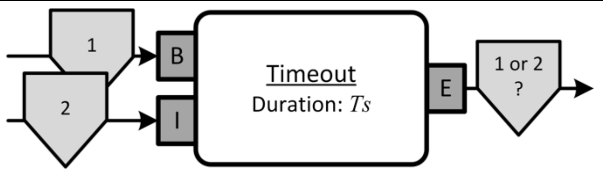
Timer		
Executing the fundamental timing function of DMDL		
TIME REG POLY		
Complete Form	 B G A S Timer Mode: continuous Timer type: Exp. Timer duration: 1s Show STOP port: Yes ... E A HALT B BEGIN S STOP E END G RECONFIGURE	
Ports	B BEGIN	Any command arrives at this port activates the timer.
	G RECONFIGURE	This port is used to change the duration of the timer. The arriving command must contain a numerical variable which can define the new duration.
	A HALT	Holds the countdown of the timer when the timer is executing. Warning information should be displayed if the timer is inactivated when a command arrives. Timer can be

		resumed by receiving command from this port or $\boxed{\text{B}}$ **B**EGIN
	$\boxed{\text{S}}$ **S**TOP	To kill the timer when it is running. Warning information should be displayed if the timer is inactivated when a command arrives.
	$\boxed{\text{E}}$ **E**ND	Exports a command when the timer is expired. In general, the command which activated the timer should be forwarded.
Properties	Mode	The mode of the timer. It is recommended to have at least two modes: oneshot and periodic. In the oneshot mode, the timer is only countdown one time and export the trigger command one time. In the Periodic mode, the timer periodically forwards the corresponding command from the port.
	Timer Type	It is an optional property. Developers may define different types such as exponential or other distributions so that users can use them to reduce the complicity of their design.
	Duration	Sets the waiting time of the timer. It may vary according to the Time type.
Examples		
		 B Timer Mode: oneshot Type: exponential Av. Duration: *Ts* E
		 B Timer Mode: continuous Type: exponential Av. Duration: *Ts* E

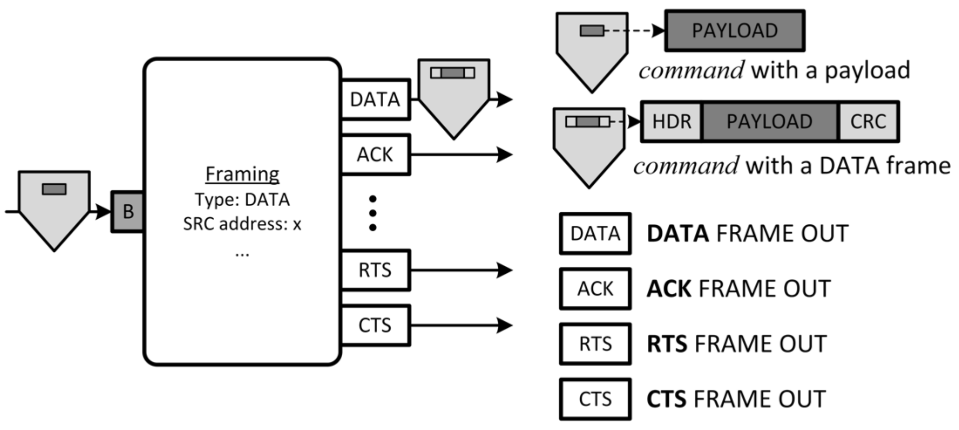
Backoff
To space out repeated retransmissions or retransmission attempts of the same frame.

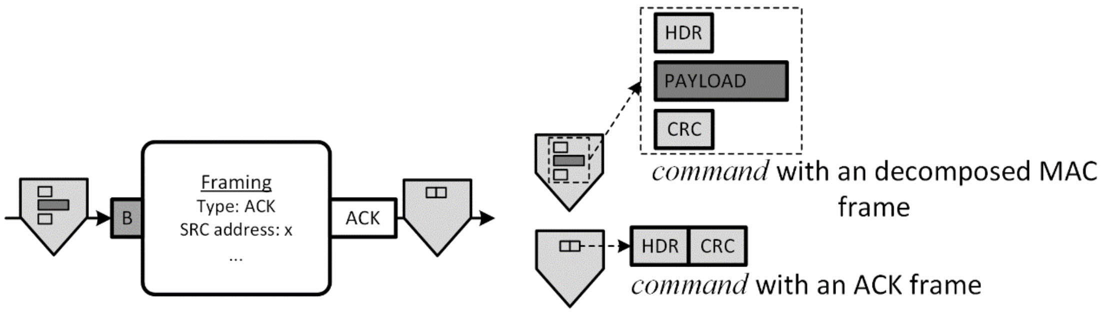
Complete Form	 Rssi = k B R T V Backoff Mode: Binary Exp. Backoff unit: *Tu* Max. win. size: *N* E R RSSI T CS THRESHOLD V Virtual CS Rssi = k *command* with RSSI	
TIME REG POLY		
Ports	B **B**EGIN	A command with a valid MAC frame arrives at this port activates backoff.
	R **R**SSI	A port continuously receives RSSI value from the receiving to get the channel information.
	T CS **T**HRESHOLD	The CS threshold may vary and depend on SDR devices. Therefore, a port is designed to update the CS threshold from other blocks, which enables the backoff block to track the channel conditions in a dynamic way.
	V **V**irtual CS	A port to receive the waiting time of Virtual CS. The information normally generated by blocks in the receiving chain that capture a non-destined
	E **E**ND	A command with the reschedule MAC frame is exported from this port when the backoff ends.
Properties	Mode	For user to choose different types of backoff, including but not limited to binary exponential backoff, random backoff and constant backoff (normally used as a base line).
	Backoff Unit	The time unit used in backoff.
	Max. win. size	The maximal backoff window size. Used in some backoff algorithms.

Timeout
To perform general timeout function in a MAC protocol
POLY

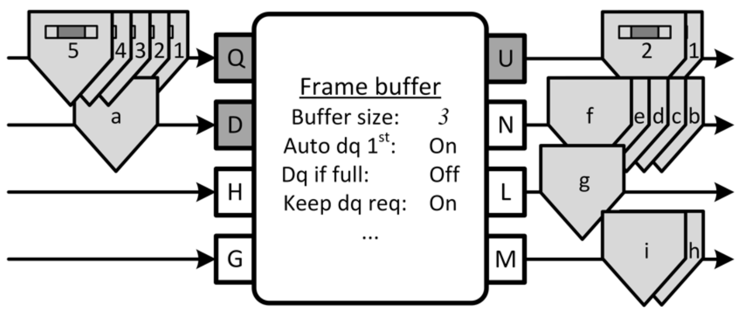
Complete Form	 1 2 B I Timeout Duration: *Ts* E 1 or 2 ? I CONFIRM	
Ports	B **B**EGIN	The input port for the first frame that is waiting for response, e.g., the DATA frame or the RTS frame.
	I CONFIRM	The input port for the response frame, e.g., the ACK frame and the CTS frame.
	E **E**ND	The output of the block. To simplify the connection of the block and avoid
Properties	Duration	The duration set for the timeout timer. A response comes out of the duration is defined as a failed response.

2.4 Frame Related Blocks

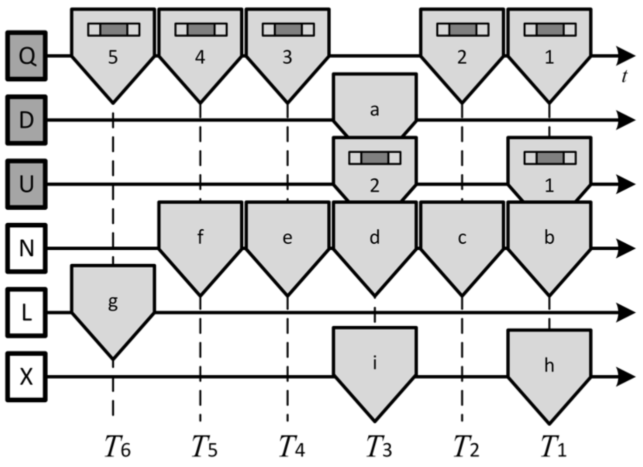
Framing		
Generate different type of MAC frames		
MAC POLY		
Complete Form	 B Framing Type: DATA SRC address: x ... DATA ACK RTS CTS PAYLOAD *command* with a payload HDR PAYLOAD CRC *command* with a DATA frame DATA **DATA** FRAME OUT; ACK **ACK** FRAME OUT; RTS **RTS** FRAME OUT; CTS **CTS** FRAME OUT	
Ports (extendable)	B **B**EGIN	The input port for command containing information for generating MAC frames. For example, to generate a DATA frame the coming command must have the data payload.
	DATA **DATA** FRAME	Output port for created DATA frame.
	ACK **ACK** FRAME	Output port for created ACK frame.

	BCO **BEACON** FRAME	Output port for created BEACON frame.
	RTS **RTS** FRAME	Output port for created RTS frame.
	CTS **CTS** FRAME	Output port for created CTS frame.
	AMS **AMS**DU FRAME	Output port for created AMSDU frame.
	AMP **AMP**DU FRAME	Output port for created AMPDU frame.
	BACK **B**lock **ACK** FRAME	Output port for created BACK frame.
Properties	Type	Type of generated frame.
	SRC address	The address of the node that should be inserted into the corresponding field of the MAC frame header.
Examples		
		

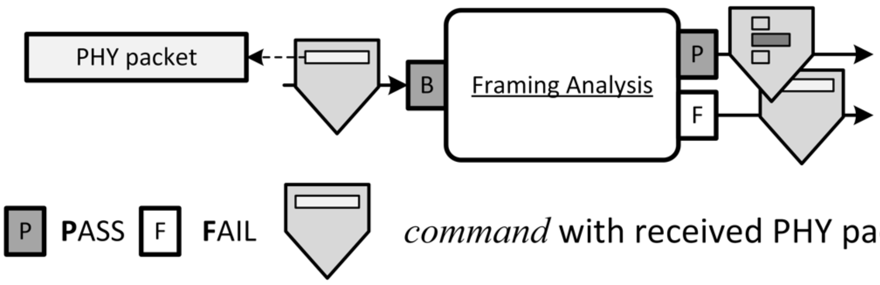
B
Framing
Type: ACK
SRC address: x
...
ACK
HDR
PAYLOAD
CRC
command with an decomposed MAC frame
HDR
CRC
command with an ACK frame

Buffer	
Frame and command buffer with basic logics of queueing operations.	
POLY	
Complete Form	 Q ENQUEUE, H FLUSH, L FULL, N NOT FULL, D DEQUEUE, U DEQUEUED, M EMPTY

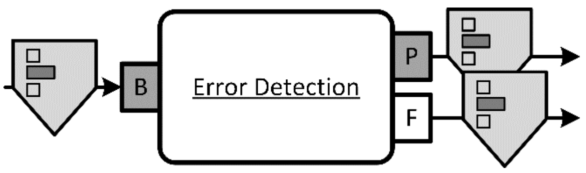
Ports	Q EN**Q**UEUE	Command arrives at this port is pushed in the queue.
	D **D**EQUEUE	A command arrives at this port lets the queue to pop out one command from the queue (not the arrived one).
	H FLUS**H**	The queue is flushed if one command is arrived at this port.
	G RECONFI**G**URE	Users may use this port to reconfigure the length of the queue.
	U DEQ**U**EUED	Commands are popped out from this port.
	N **N**OT FULL	An empty command is exported to Indicate the buffer is not full.
	L FU**L**L	An empty command is exported to Indicate the buffer is full.
	M EMPTY	An empty command is exported to Indicate the buffer is empty.
Properties	Buffer size	Define the length of the command queue.
	Auto dq 1st	In many cases, frames are popped out of a queue because the previous one is either successfully sent or failed. However, the way to pop out the first is problematic or requires extra logics. Set this property true would automatically pop out the first one when it pushed in.
	Dq if full	Pop out one command when a new command is pushed in, and the queue is full.
	Keep dq req	It allows the buffer to remember the dequeue request comes from port D **D**EQUEUE but the queue is empty. If it is set to TRUE, then the next command will be immediately popped out when it arrives. Otherwise the earlier arrived dequeue request is discarded.
Examples		

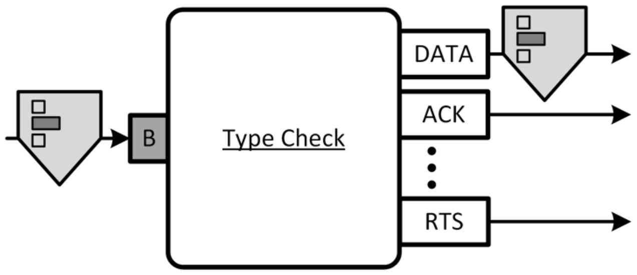


If the properties of the buffer block are set as shown in the Complete Form block and the input commands are following the order shown above, the outputs of the block is also shown as the above figure.

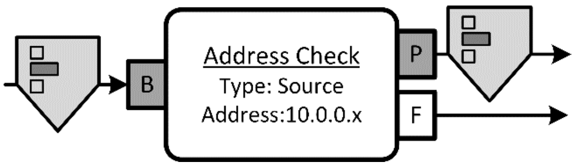
Frame Analysis		
Decode frame header information from raw PHY payload		
MATH MAC POLY		
Complete Form	 PHY packet B Framing Analysis P F P PASS F FAIL *command* with received PHY packet	
Ports	B **B**EGIN	This block tries to cut the MAC header and error detection/correction bytes from the PHY packet of imported command.
	P **P**ASS	If the imported command contains a valid MAC frame, then the MAC frame with the information from the MAC header are exported from this port.
	F **F**AIL	If the imported command does not contain a valid MAC frame, then the MAC frame with a signal indicating failure should be export from this port.

Error Detection

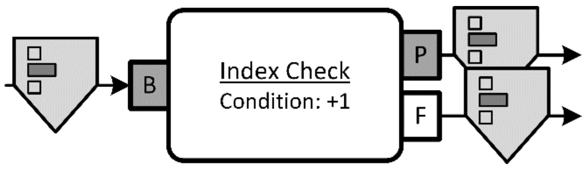
The block to perform error detection, e.g., Cyclic Redundancy Check (CRC), and make decisions.		
MAC POLY		
Complete Form		
Ports	B **BEGIN**	Receives commands with MAC payload and error detection code.
	P **PASS**	If the imported MAC frame passes the check, then the imported command is exported from this port.
	F **FAIL**	If the imported MAC frame is proved invalid, then the imported command is exported from this port.

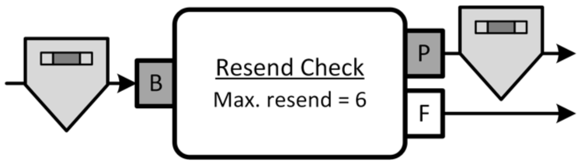
Type Check		
The block checks the frame type of the MAC frame in the imported command and exports them according to their types.		
MAC POLY		
Complete Form		
Ports (extendable)	B **BEGIN**	Receives commands with MAC header information and payload.
	DATA **DATA** FRAME	Output port for DATA frame.
	ACK **ACK** FRAME	Output port for ACK frame.
	BCO **BEACON** FRAME	Output port for BEACON frame.
	RTS **RTS** FRAME	Output port for RTS frame.
	CTS **CTS** FRAME	Output port for CTS frame.
	AMS **AMS**DU FRAME	Output port for AMSDU frame.

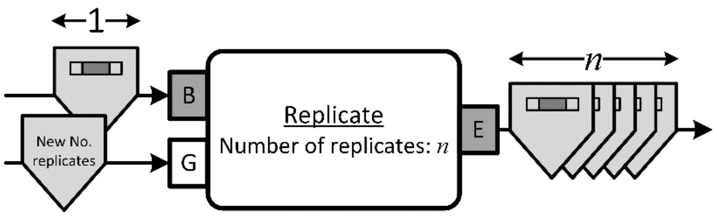
	AMP **AMP**DU FRAME	Output port for AMPDU frame.
	BACK **B**lock **ACK** FRAME	Output port for BACK frame.

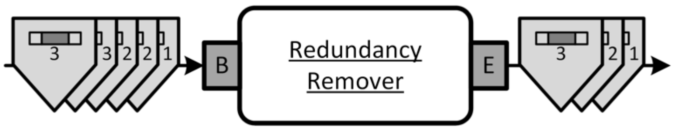
Address Check		
The block to compare the address of the arriving frame and the preset address.		
MAC POLY		
Complete Form	 B Address Check Type: Source Address:10.0.0.x P F	
Ports	B **B**EGIN	Receives commands with MAC payload and address information.
	P **P**ASS	If the imported MAC frame passed the address check, then the imported command is exported from this port.
	F **F**AIL	If the imported MAC frame failed in the address check, then the imported command is exported from this port.
Properties	Type	Select the types of the address check, e.g., comparing the predefined address with the source address of the arriving MAC frame or the destination address of the arriving MAC frame.
	Address	The address which is used to compare with the arriving frame.
Examples (We may need one here)		

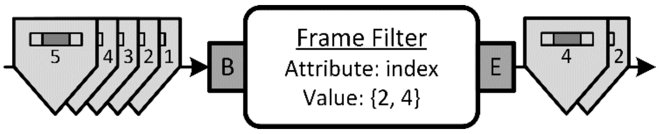
Index Check (need to revise the code)
Check the frame index of two successively arrived frames.
MATH MAC REG POLY

Complete Form		
Ports	B **B**EGIN	Port to receive commands with MAC frames.
	P **P**ASS	If the imported MAC frame passed the address check, then the imported command is exported from this port.
	F **F**AIL	If the imported MAC frame failed in the address check, then the imported command is exported from this port.
Properties	Condition	Users may define a function here to check the index of adjacent frames. For example, it can be + 1, where the block compares the index of the arrived frame X and the index of the previous one Y. If X = Y+1, then the later frame is sent out from the port P PASS . Otherwise form the port F FAIL

Resend Check		
Check the number of retransmissions of a frame.		
MAC POLY		
Complete Form		
Ports	B **B**EGIN	Receives commands with MAC frames and number of retransmissions.
	P **P**ASS	If the imported MAC frame passed the address check, then the imported command is exported from this port.
	F **F**AIL	If the imported MAC frame failed in the address check, then the imported command is exported from this port.
Properties	Max. resend	Maximal number of the retransmissions.

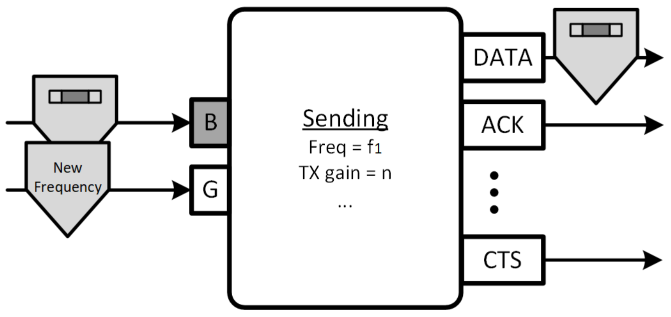
Replicate		
Create and export multiple copies of the imported command.		
MAC POLY		
Complete Form	 1 New No. replicates B G Replicate Number of replicates: n E n	
Ports	$\boxed{\text{B}}$ **B**EGIN	Receives commands with MAC frames and number of retransmissions.
	$\boxed{\text{G}}$ RECONFI**G**URE	Resets the number of replicates.
	$\boxed{\text{E}}$ **E**ND	Exports the original command and its copies.
Properties	Number of replicates	Set the default number of replicates.

Redundancy Remover		
Remove multiple copies of the same frame. Only the first one is getting through.		
MAC POLY		
Complete Form	 3 3 2 1 B Redundancy Remover E 3 2 1	
Ports	$\boxed{\text{B}}$ **B**EGIN	Receives commands with MAC frames with valid frame index.
	$\boxed{\text{E}}$ **E**ND	Exports commands sequence without redundancy.

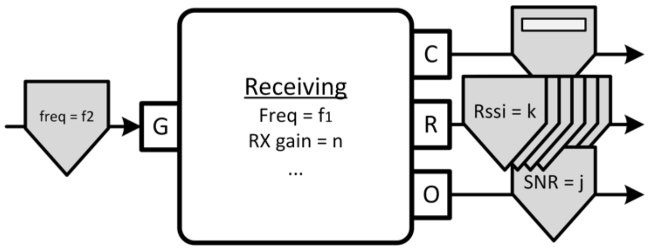
Attribute Filter (need to be expended later)	
Filtering frames based on their frame information.	
MAC POLY	
Complete Form	 5 4 3 2 1 B Frame Filter Attribute: index Value: {2, 4} E 4 2

Ports	B BEGIN	Receives commands with MAC frames with valid frame index.
	E END	Exports commands sequence with specific selected index.
Properties	Attribute	Specifies the attribute to selected.
	E END	Assigns the targeted values of the given attribute..

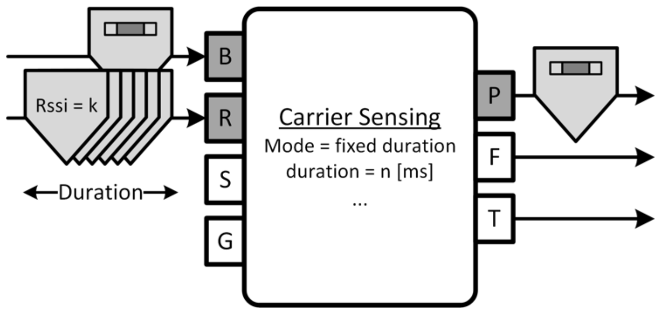
2.4.1 Interface Related Blocks

Sending (Need to be implemented)		
Sends MAC frames over the air, including all PHY layer operations on the transmitter side and also the hardware API. The transmitted frames are also export from the block so that the later part of the protocol can use them.		
MAC PHY POLY API		
Complete Form	 New Frequency B G Sending Freq = f1 TX gain = n ... DATA ACK CTS	
Ports	B BEGIN	Receives commands with MAC frames which are ready to send.
	G RECONFIGURE	Reconfigures the PHY layer functionalities and the radio parameters.
	DATA DATA FRAME	Output port for the sent DATA frame.
	ACK ACK FRAME	Output port for the sent ACK frame.
	BCO BEACON FRAME	Output port for the sent BEACON frame.
	RTS RTS FRAME	Output port for the sent RTS frame.
	CTS CTS FRAME	Output port for the sent CTS frame.
	AMS AMSDU FRAME	Output port for the sent AMSDU frame.
	AMP AMPDU FRAME	Output port for the sent AMPDU frame.

	BACK Block **ACK** FRAME	Output port for the sent BACK frame.
Properties	Frequency	Sets the carrier frequencies
	TX gain	Transmitting gain
	…	…

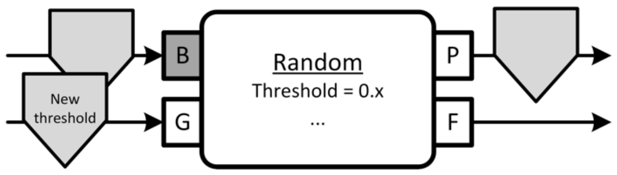
Receiving (Need to be implemented)		
Receives PHY packet over the air, including all PHY operations on the receiver side and also the hardware API. This block also supposed to provide channel condition and the status of receiving to other part of the protocol.		
MAC PHY POLY API		
Complete Form	 freq = f2 G Receiving Freq = f1 RX gain = n … C R O Rssi = k SNR = j C PHY PACKET O SIGNAL TO NOISE RATIO	
	G RECONFIGURE	Reconfigures the PHY layer functionalities and the radio parameters.
	C PHY PACKET	Output port for the successfully received PHY packet.
	ACK **ACK** FRAME	Output port for the sent ACK frame.
	BCO **BEACON** FRAME	Output port for the sent BEACON frame.
Properties	Frequency	Sets the carrier frequencies
	RX gain	Sets the receiving gain
	…	…

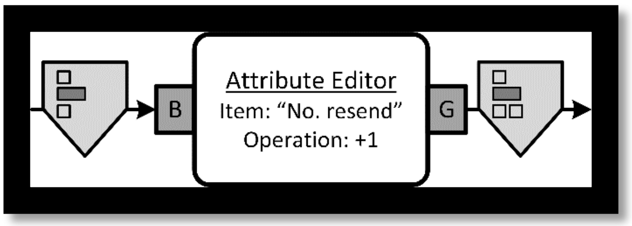
Carrier Sensing
The block to perform carrier sensing.
MAC REG POLY

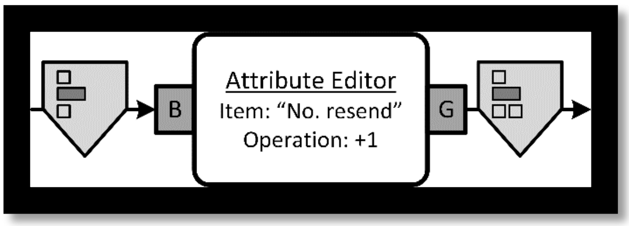
Complete Form	 Rssi = k Duration B R S G Carrier Sensing Mode = fixed duration duration = n [ms] ... P F T	
Ports	B **B**EGIN	Receives commands which used to activate carrier sensing.
	P **P**ASS	If the channel is free during the carrier sensing period, then the command is exported from this port.
	F **F**AIL	If the channel is detected busy during the carrier sensing period, then the imported command is exported from this port.
	T CS **T**HRESHOLD	If the carrier sensing threshold can be adjusted dynamically and used by other blocks of the protocol, it is the port to export the new thresholds.
	R **R**SSI	Imports RSSI values from the receiving blocks.
	S **S**TOP	Stops the ongoing carrier sensing.
	G RECONFI**G**URE	Reconfigures the carrier sensing duration.
Properties	Mode	Users may select different modes of carrier sensing.
	Duration	The duration of performing carrier sensing.
	…	…

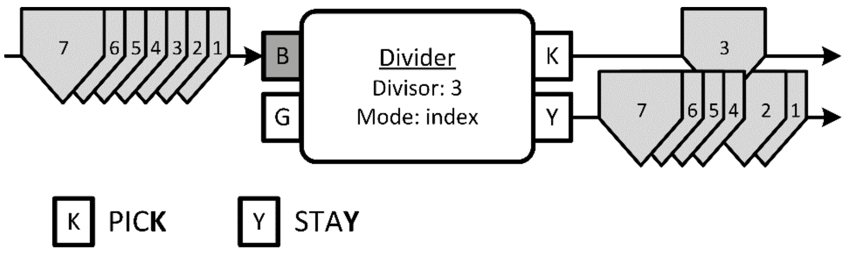
2.4.2 Auxiliary Blocks

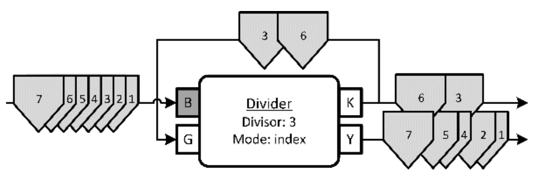
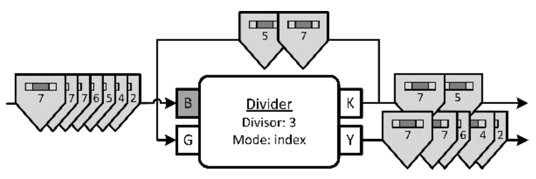
Random
Randomly forwards the arrived commands according a given probability.
MATH POLY

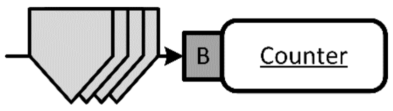
Complete Form	 New threshold B G Random Threshold = 0.x ... P F	
Ports	B **BEGIN**	Input port of the block.
	P **PASS**	The block generates a random number between 0 and 1 for every arrived command. If the number is smaller than the given threshold, the command is exported from this port.
	F **FAIL**	The block generates a random number between 0 and 1 for every arrived command. If the number is smaller than the given threshold, the command is exported from this port.
	G RECONFIGURE	Reset the threshold for comparison.
Properties	Threshold	The constant probability to compare with the generated random number.

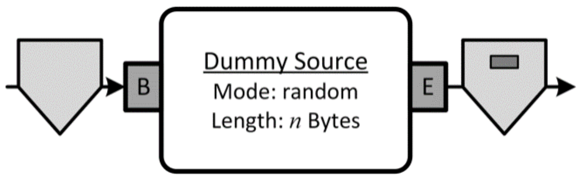
Attribute Editor		
Edits an attribute of the arrived command.		
MATH POLY		
Complete Form	 B Attribute Editor Item: "No. resend" Operation: +1 G	
Ports	B **BEGIN**	Input port of the block.
	E **END**	Commands are exported from this port after being modified.
Properties	Attribute	Users can choose the attribute by name which they want to change the value in it.
	Operation	Users may specify the operation they want to apply to the selected attribute.

Attribute Splitter		
Find an attribute of the arrived command and export it separately.		
MATH POLY		
Complete Form	 B Attribute Editor Item: “No. resend” Operation: +1 G	
Ports	B **BEGIN**	Input port of the block.
	E **END**	Commands with the selected attribute are exported from this port after being modified.
	K PIC**K**	Commands with all attributes are exported from this port after being modified.
Properties	Attribute	Users can choose the attribute by name which they want to export separately.

Divider		
Divides a command sequence based on the order of arrival. Everything arriving command is assigned with an index increasing by 1. The arrived commands are forwarded from different ports according to their divisibility by the predefined divisor.		
MATH REG POLY		
Complete Form	 7 6 5 4 3 2 1 B G Divider Divisor: 3 Mode: index K Y 3 7 6 5 4 2 1 K PICK Y STAY	
Ports	B **BEGIN**	Input port of the block to accept the command sequences.
	G RECONFI**G**URE	Changes the divisor.
	P **P**ASS	If the index of the command is divisible by the divisor, the command is passed from this port.

	F **FAIL**	If the index of the command is not divisible by the divisor, the command is passed from this port.
Properties	Divisor	The divisor used in categorizing arrived commands.
	Mode	The block is able to divide frames according to the frame index or the order of arrival.
Examples		
		

Counter		
Counts the number of frames arrived.		
MATH REG API		
Complete Form		
Ports	B **BEGIN**	Input port of the block to accept the command sequences.

Dummy Source		
Generate MAC payload to emulate the upper layer traffic.		
MATH REG POLY		
Complete Form		
Ports	B **BEGIN**	Triggers to generate MAC payload are imported from this port.
	S **STOP**	To stop the generation if the block is in continuous mode.

	$\boxed{E}$ **E**ND	Commands with generated MAC payload are exported from this port.
Properties	Mode	The mode that how the block generates payload.
	Length	Sets the length of the generated payload.

3 GR-DMDL – DMDL in GNU Radio

3.1 How to build GR-DMDL

GR-DMDL is primarily developed on Ubuntu 14.04 and 16.04. The compatibility of other platforms and other versions of Linux is not well tested. In order to install GR-DMDL in GNU Radio and use it to operate USRPs, GNU Radio and UHD are required to be installed first. GR-DMDL supports almost all Ettus Research USRP™ hardware, including all motherboards and daughter boards. Abbreviated instructions are duplicated below.

1. Ensure that you have satisfied the external dependencies for both UHD and GNU Radio. Please check them from their official website.

2. Install GNU Radio and UHD before installing GR-DMDL.
   a. Install UHD from https://github.com/EttusResearch/uhd
   b. Install GNU Radio from https://github.com/gnuradio/gnuradio#pybombs
2. Checkout the latest code of GR-DMDL:

   - `$ git clone https://github.com/joqoko/gr-dmdl.git`

3. Build with CMake:

   - `$ cd gr-dmdl`
   - `$ mkdir build`
   - `$ cd build`
   - `$ cmake ../`
   - `$ make`
   - `$ sudo make install`
   - `$ sudo ldconfig`

After Install GR-DMDL, you may find several new categories in the Block Tree Panel of GNU Radio Companion (GRC) including dmdl and dmdl.simphy as shown in Figure 3. Category dmdl contains all blocks defined in DMDL and dmdl.simphy contains PHY related blocks. Part of the blocks of dmdl.simphy is derived from project gr-inets [1].

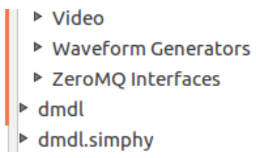


Figure 3. dmdl categories in GNU Radio

3.2 Commands in GR-DMDL

Due to the limitation of GRC, *command* of DMDL is not visualized. It is implemented as the dictionary container which is a Polymorphic Type of GNU Radio.

3.3 Blocks in GR-DMDL

Most of the default GR-DMDL blocks are written in C++ to shorten processing delay in the host. Users are free to develop their own Python written blocks along with default GR-DMDL blocks. In order to use Sending and Receiving blocks, user need to **first compile** the following hierarchical blocks using GRC.

- `grc/sending_hier.grc`
- `grc/receiving_hier.grc`

Most of the Blocks in GR-DMDL follow their definition in DMDL. In the meanwhile, several new blocks are also defined to enable fast experimental data collection and processing. Users may find examples of the new blocks in the **example** folder. Two specific properties are developed for all blocks called *develop_mode* and *block_id* which are used to output debugging information of the block in the console panel of GRC. Switching on/off optional ports are also implemented so that developers and designers are able to design their protocol in a clear and straightforward way.

3.4 Calibration

To gain a better understanding of GR-DMDL and its performance with USRPs, the basic timing performances are calibrated. It is worth noting that the results are platform dependent. The configuration of host PC, the method used to connect host PC and USRPs play an important role in the measurements.

3.4.1 Testbed Configurations

Host PC
- Intel i7-3770 CPU @ 3.40GHz
- 16 GB RAM (8 GB 2Rx8 PC3 12800U × 2)
- SAMSUNG SSD SM841 512 GB

USRP
- USRP2 and USRP 2920
- SBX daughterboard

Ethernet
- Intel PRO/1000 GT Desktop Adapter

3.4.2 Waveform measurement

In this measurement, we measured all critical time points in transmitting a single DATA frame. The GRC code is shown in Figure 14 and Figure 15[†]. The results are plotted in Figure 5 and Figure 6. The tail effect is the key factor to determine the transmitting-receiving switching duration.

[†] Blocks names may slightly vary in different versions of DMDL. Please contact the author in case of confusion.

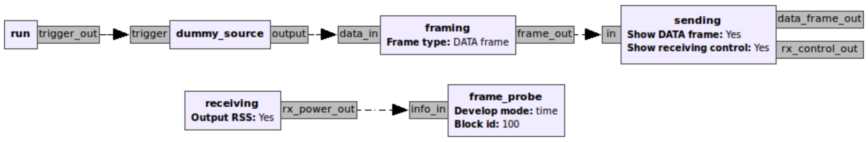


Figure 4. GRC code used in Frame waveform measurement.

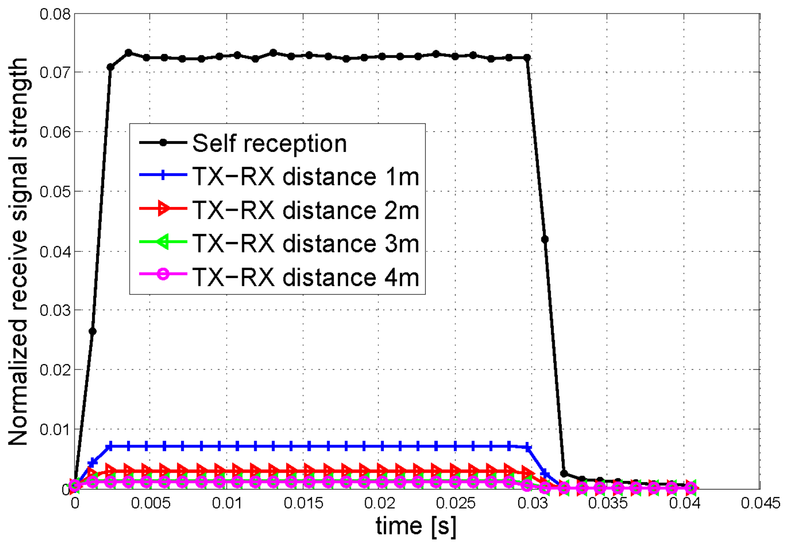


Figure 5. Waveform of DMDL frame transmission with different receiving distance.

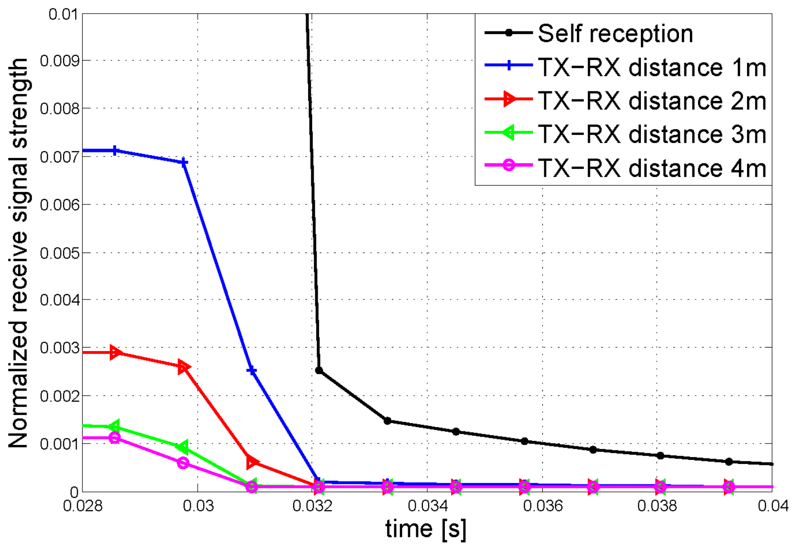
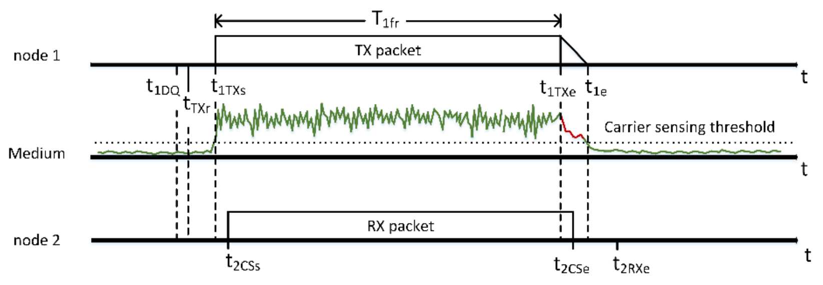


Figure 6. Tail effect of self-received signal. Zoomed from the wave figure.

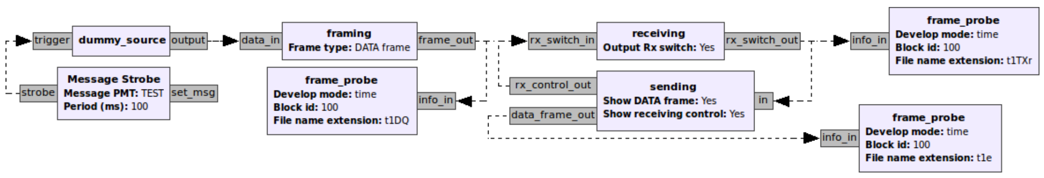
3.4.3 Single frame transmission

All critical time points evolving in transmitting a single frame at both the transmitter and receiver sides are marked in Figure 7 and measured by using the GRC code shown in Figure 8 and Figure 9, respectively. The *frame_probe* block is renamed as *probe* in later version of GR-DMDL.



t_{1DQ}: node 1 decides to transmit	t_{TXr}: node 1 TX receives the TX command	t_{1TXs}: node 1 TX starts
t_{1TXe}: node 1 finishes transmitting	t_{1e}: node 1 finishes waiting for the tail effect	t_{2CSs}: Start receiving at node 2
t_{2RXe}: node 2 RX finishes decoding	t_{2CSe}: End receiving at node 2	T_{1fr}: frame duration sent by node 1

Figure 7. Key timing parameters in single frame transmission.


trigger **dummy_source** output

Message Strobe
Message PMT: TEST
Period (ms): 100
strobe set_msg

data_in **framing**
Frame type: DATA frame
frame_out

frame_probe
Develop mode: time
Block id: 100
File name extension: t1DQ
info_in

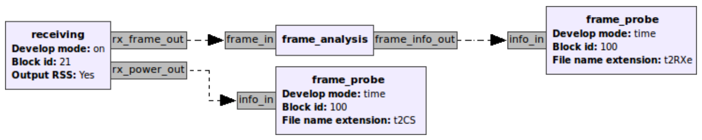
rx_switch_in **receiving**
Output Rx switch: Yes
rx_switch_out

rx_control_out
data_frame_out
sending
Show DATA frame: Yes
Show receiving control: Yes
in

info_in **frame_probe**
Develop mode: time
Block id: 100
File name extension: t1TXr

info_in **frame_probe**
Develop mode: time
Block id: 100
File name extension: t1e

Figure 8. GRC code of the transmitter in the single packet transmission.


receiving
Develop mode: on
Block id: 21
Output RSS: Yes
rx_frame_out
rx_power_out

frame_in **frame_analysis** frame_info_out

info_in **frame_probe**
Develop mode: time
Block id: 100
File name extension: t2RXe

info_in **frame_probe**
Develop mode: time
Block id: 100
File name extension: t2CS

Figure 9. GRC code of the transmitter in the single packet transmission.

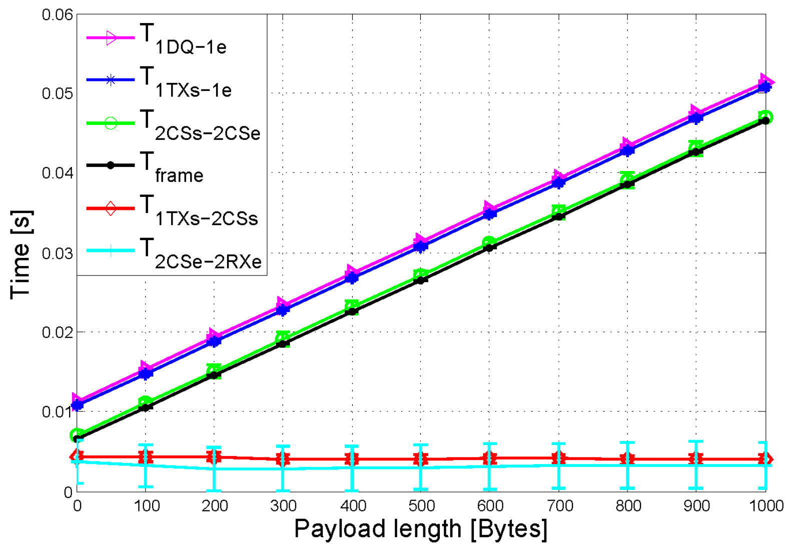


Figure 10. Frame level timing characteristics in single frame transmission measurements.

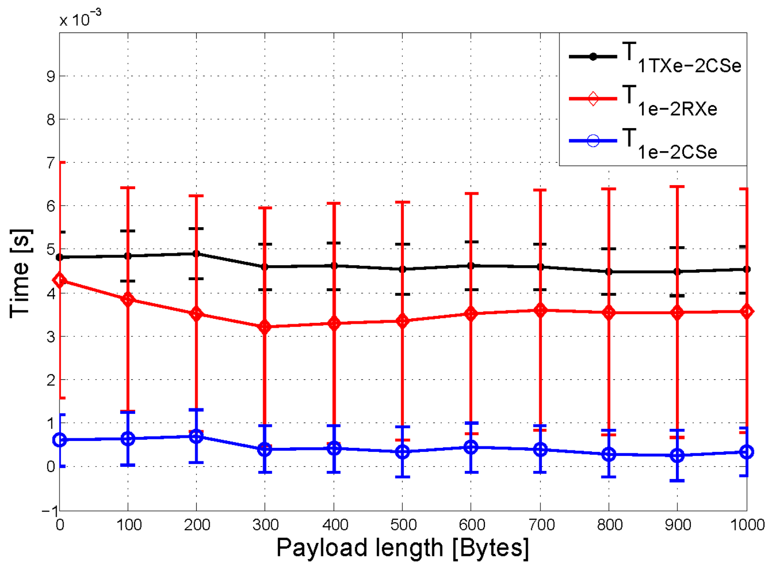


Figure 11. Delay between nodes.

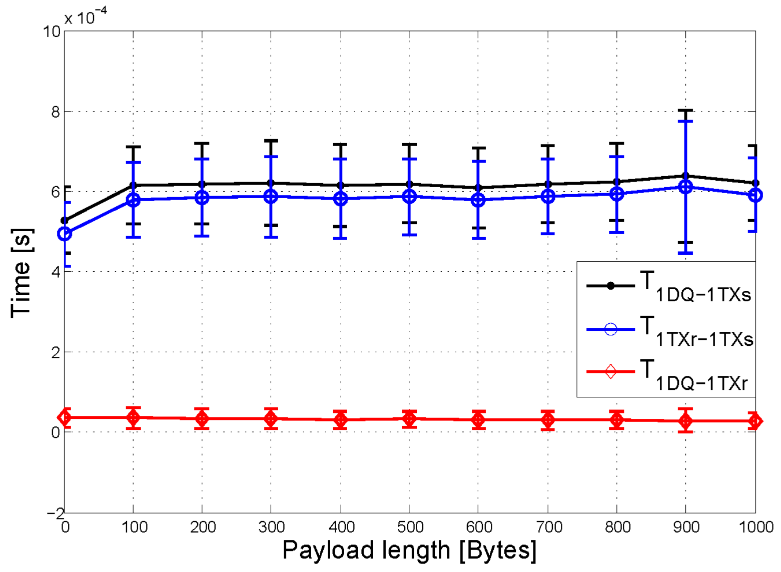
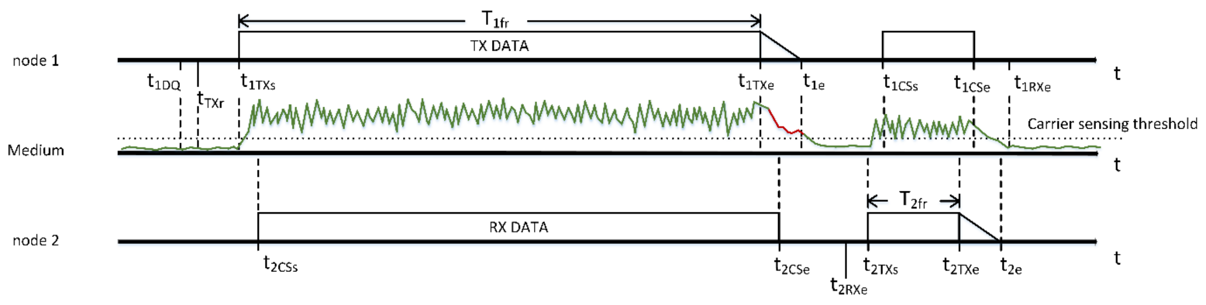


Figure 12. Delay between blocks in transmitting node.

3.4.4 Handshaking transmission

All critical time points evolving in a typical handshaking at both the transmitter and receiver sides are marked in Figure 13 and measured by using the GRC code shown in Figure 14 and Figure 15, respectively. In the measurement, the transmitter periodically generates DATA frames and the receiver keeps on receiving and sending back ACK frames. To avoid receiving self-transmitting signal, the command with a frame to transmit is first import to the receiving block to disable the receiving block before transmitting. After transmission, a special command is transferred from the sending block to receiving block to activate the receiving function.



t_{1DQ}: node 1 decides to transmit	t_{TXr}: node 1 TX receives the TX command	t_{1TXs}: node 1 TX starts	t_{1RXe}: node 1 RX finishes decoding
t_{1TXe}: node 1 finishes transmitting	t_{1e}: node 1 finishes waiting for the tail effect	t_{2CSs}: Received by node 2	t_{2e}: node 2 finishes waiting for the tail effect
t_{2RXe}: node 2 RX finishes decoding	t_{2TXs}: node 2 TX starts	t_{2CSe}: End receiving at node 2	T_{2fr}: frame duration sent by node 2 (ACK frame)
t_{2TXe}: node 2 finishes transmitting	t_{1CSe}: End receiving at node 1	t_{1CSs}: Start receiving at node 1	T_{1fr}: frame duration sent by node 1 (DATA frame)

Figure 13. Key timing parameters in handshaking.

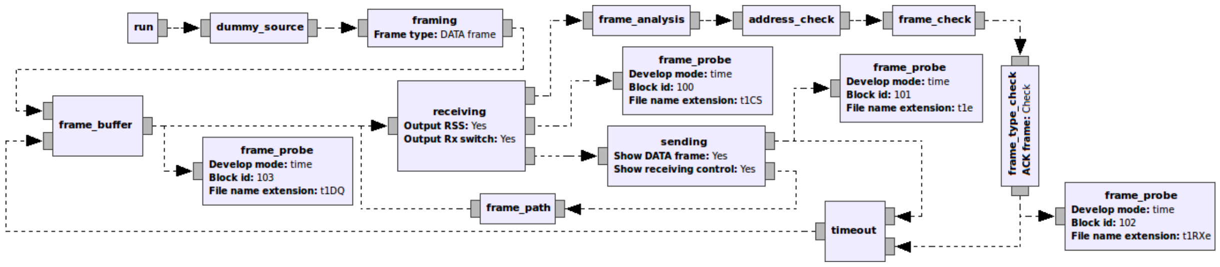

run
dummy_source
framing
Frame type: DATA frame
frame_analysis
address_check
frame_check
frame_probe
Develop mode: time
Block id: 100
File name extension: t1CS
frame_probe
Develop mode: time
Block id: 101
File name extension: t1e
frame_type_check
ACK frame: Check
frame_buffer
receiving
Output RSS: Yes
Output Rx switch: Yes
sending
Show DATA frame: Yes
Show receiving control: Yes
frame_probe
Develop mode: time
Block id: 103
File name extension: t1DQ
frame_path
timeout
frame_probe
Develop mode: time
Block id: 102
File name extension: t1RXe

Figure 14. GRC code of the transmitter in the handshaking measurements. †

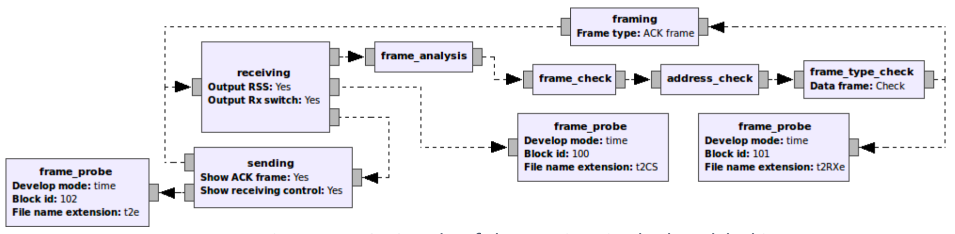

framing
Frame type: ACK frame
receiving
Output RSS: Yes
Output Rx switch: Yes
frame_analysis
frame_check
address_check
frame_type_check
Data frame: Check
frame_probe
Develop mode: time
Block id: 100
File name extension: t2CS
frame_probe
Develop mode: time
Block id: 101
File name extension: t2RXe
sending
Show ACK frame: Yes
Show receiving control: Yes
frame_probe
Develop mode: time
Block id: 102
File name extension: t2e

Figure 15. GRC code of the receiver in the handshaking measurements. †

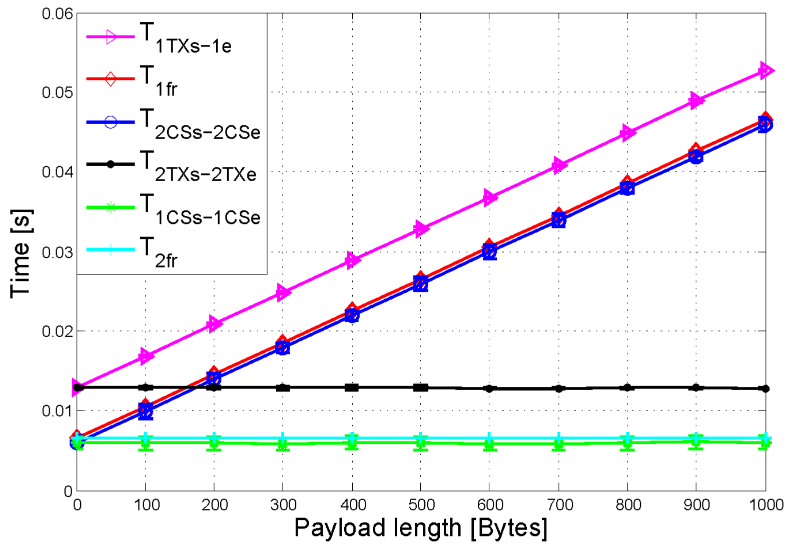


Figure 16. Frame level timing characteristics in handshaking measurements.

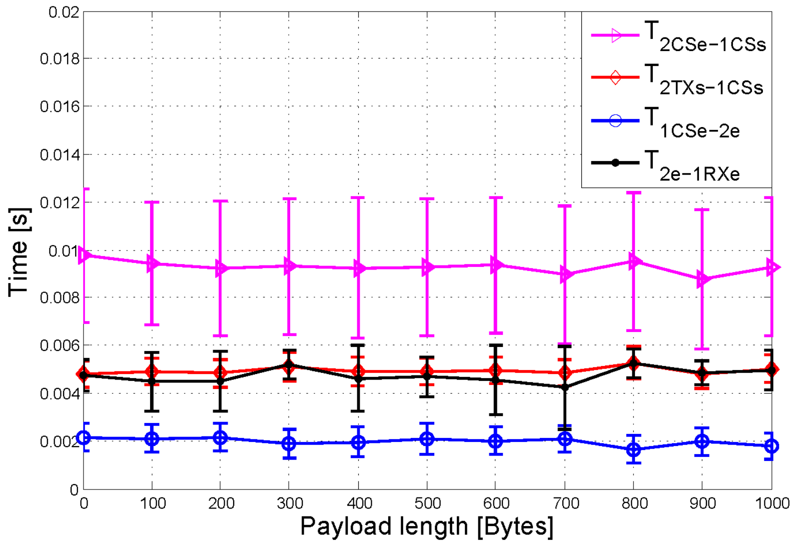


Figure 17. Inter-node timing characteristics in handshaking measurements.

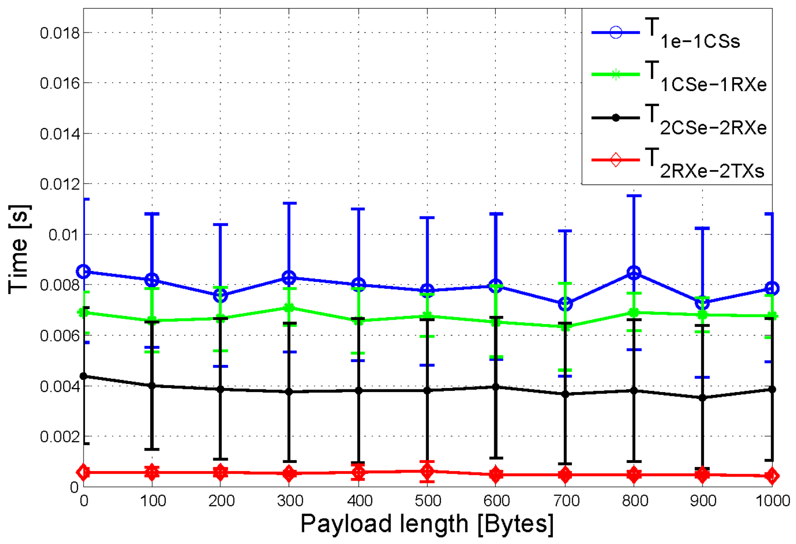


Figure 18. Intra-node timing characteristics in handshaking measurements.

We will release more data and explanations in the future publications.

3.5 Examples and Results

Examples can be found in in **example/protocol** folder. Some of the results are shown in one of our paper published in WCNC 2018 [2]. More papers are being submitted to demonstrate more details of DMDL and GR-DMDL and their performances.
In this manual, we only use ALOHA and CSMA as examples to quick demonstrate how to use DMDL to design a MAC scheme and how to carry out experiments in GNU Radio. More implemented protocols can be found in **example/protocol** folder and our WCNC paper [2].
The DMDL design of a pure ALOHA with a Poisson source is shown in Figure 19. It is worth noting that the two timers here are used to control the overall experimental time.

In DMDL:

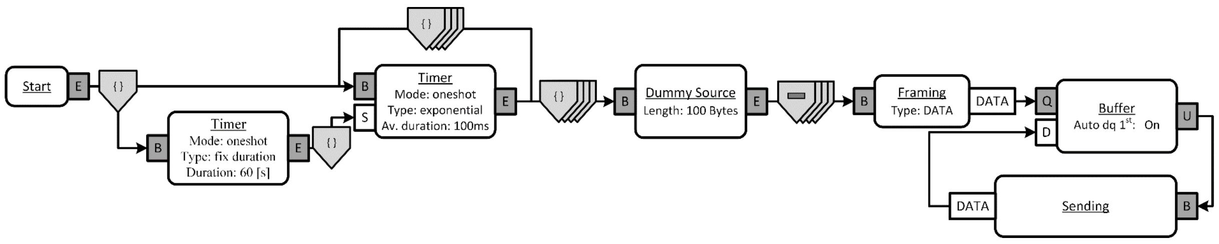


Figure 19. Pure ALOHA in DMDL.

In GR-DMDL:

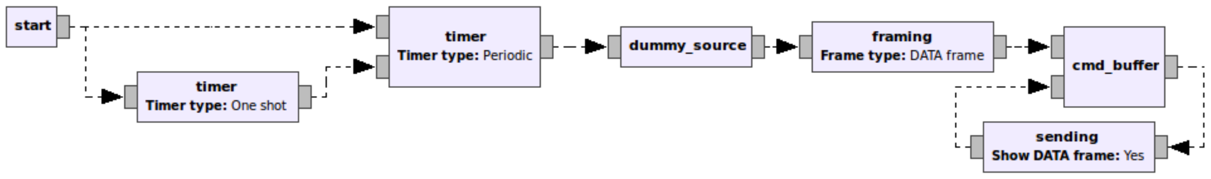


Figure 20. Pure ALOHA in GR-DMDL.

Results:

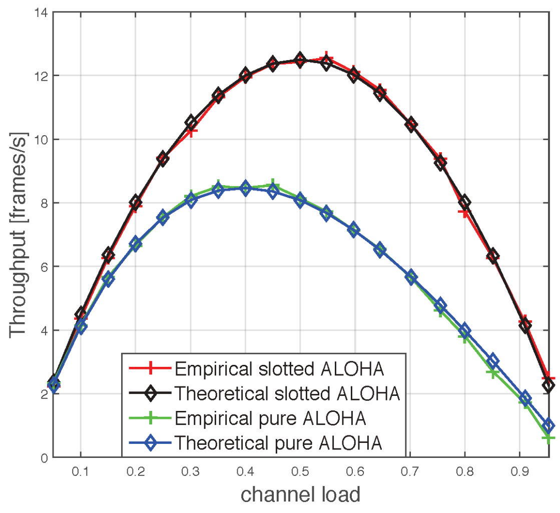


Figure 21. Performance comparison of theoretical pure ALOHA and slotted ALOHA and their GR-DMDL implementation.

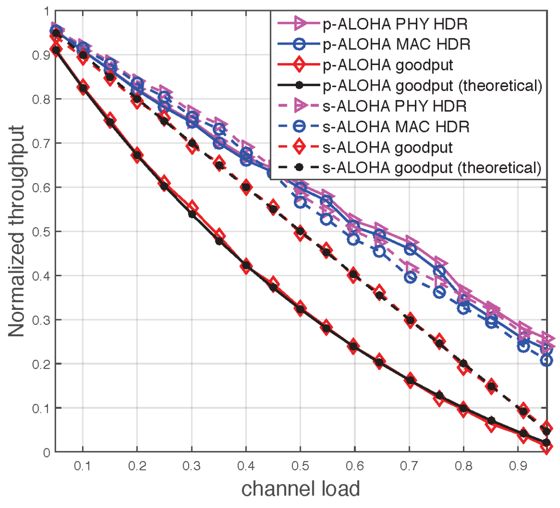


Figure 22. Normalized ALOHA throughputs.

Both Figure 21 and Figure 22 show that the results measured from GR-DMDL are in accordance with the theoretical expectations.

The DMDL design of 802.11 DCF with a Poisson source is shown in Figure 23.

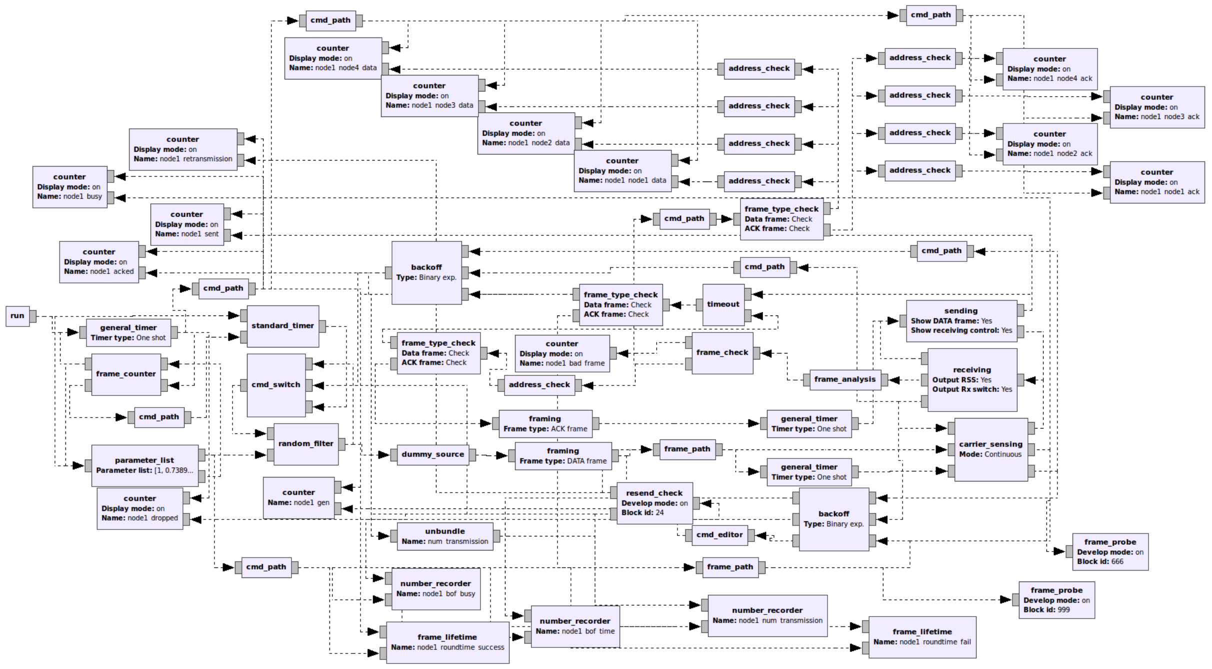


Figure 23. GRC code of the IEEE 802.11 DCF with two-way handshake.

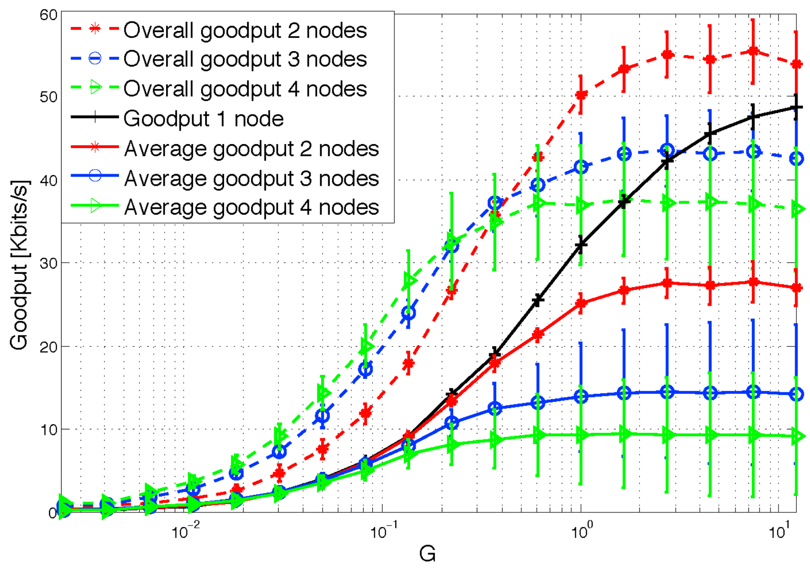


Figure 24. Up-link Goodput of networks with different number of client nodes running 802.11 DCF. Goodput is calculated by only counting the payload part of successfully acknowledged frames at each transmitter. Number of nodes is varying from 1 to 4.

We will release more examples, data and explanations in the future publications.

References

[1]. https://github.com/RWTH-iNets/gr-inets
[2]. P. Wang, M. Petrova, and P. Mähönen, DMDL: A Hierarchical Approach to Design, Visualize, and Implement MAC Protocols. In Proceedings of IEEE Wireless Communications and Networking Conference, (Bacelona, Spain), 2018.