

idaTSNSimulator

Generated by Doxygen 1.8.19

1 RTPS Configuration	1
1.1 Overall configuration	1
1.2 Endpoint specific	1
1.3 Stream specific	2
1.4 Wrtier specific	2
1.5 Reader specific	2
1.6 Example	2
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 ArbiterAVB Class Reference	9
5.1.1 Detailed Description	10
5.1.2 Constructor & Destructor Documentation	10
5.1.2.1 ArbiterAVB()	10
5.1.2.2 ~ArbiterAVB()	11
5.1.3 Member Function Documentation	11
5.1.3.1 arbitrateBufferStatusList()	11
5.1.3.2 calculateClassInformationCredits()	11
5.1.3.3 calculateCredits()	11
5.1.3.4 calculateCreditValues()	11
5.1.3.5 calculatedRechargeTime()	13
5.1.3.6 calculateTransmissionDelay()	13
5.1.3.7 calculateTransmissionSpeed()	13
5.1.3.8 checkAndCastBufferControlMessage()	14
5.1.3.9 checkAndCastEthernetFrame()	14
5.1.3.10 collectCreditsStatistics()	14
5.1.3.11 controlArbitrationBufferStatusList()	15
5.1.3.12 dealWithControlMessage()	15
5.1.3.13 dealWithEthernetFrame()	15
5.1.3.14 dealWithExternalMessage()	15
5.1.3.15 dealWithInternalMessage()	16
5.1.3.16 debugStatus()	16
5.1.3.17 debugStatusInitialization()	16
5.1.3.18 handleMessage()	16
5.1.3.19 initialize()	16
5.1.3.20 initializeAVBClassInformation()	17

5.1.3.21 loadAVBClassInformationFromFile()	17
5.1.3.22 prepareBufferStatusList()	17
5.1.3.23 prepareClassInformations()	17
5.1.3.24 sendBufferAuthorizedBuffer()	17
5.1.3.25 sendBufferRequest()	18
5.1.3.26 sendFrameOut()	18
5.1.3.27 setRechargeTime()	18
5.1.3.28 settriggerCreditRecharged()	18
5.1.3.29 settriggerDelayEnd()	18
5.1.3.30 updateClassInformationsStatus()	19
5.1.4 Member Data Documentation	19
5.1.4.1 arbiterIsBusy	19
5.1.4.2 avbTableVector	19
5.1.4.3 channelTransmissionRate	19
5.1.4.4 classInformations	19
5.1.4.5 config	19
5.1.4.6 configFile	20
5.1.4.7 creditValueStatisticStreamA	20
5.1.4.8 creditValueStatisticStreamB	20
5.1.4.9 currentBufferStatusList	20
5.1.4.10 egressPortIndex	20
5.1.4.11 ethernetFrameWaitForSend	20
5.1.4.12 macAddress	20
5.1.4.13 numberPriorities	20
5.1.4.14 parentChannelOut	21
5.1.4.15 parentGateOut	21
5.1.4.16 parentModul	21
5.1.4.17 triggerSelf	21
5.2 arbiterAVBClass Struct Reference	21
5.2.1 Detailed Description	21
5.2.2 Member Data Documentation	21
5.2.2.1 classCreditValue	22
5.2.2.2 classFrameWaiting	22
5.2.2.3 classIsShaped	22
5.2.2.4 classStatus	22
5.2.2.5 idleSlope	22
5.2.2.6 lastArbitrationTimeStamp	22
5.2.2.7 sendSlope	22
5.2.2.8 timerIsScheduled	22
5.3 ArbiterIEEE802_1Q Class Reference	23
5.3.1 Detailed Description	24
5.3.2 Constructor & Destructor Documentation	24

5.3.2.1 ArbiterIEEE802_1Q()	24
5.3.2.2 ~ArbiterIEEE802_1Q()	24
5.3.3 Member Function Documentation	24
5.3.3.1 arbitrateBufferStatusList()	24
5.3.3.2 calculateTransmissionDelay()	24
5.3.3.3 calculateTransmissionSpeed()	25
5.3.3.4 controlArbitrationBufferStatusList()	25
5.3.3.5 dealWithControlMessage()	25
5.3.3.6 dealWithEthernetFrame()	25
5.3.3.7 dealWithExternalMessage()	26
5.3.3.8 dealWithInternalMessage()	26
5.3.3.9 handleMessage()	26
5.3.3.10 initialize()	26
5.3.3.11 prepareBufferStatusList()	27
5.3.3.12 sendBufferAuthorizedBuffer()	27
5.3.3.13 sendBufferRequest()	27
5.3.3.14 sendFrameOut()	27
5.3.3.15 settriggerDelayEnd()	27
5.3.4 Member Data Documentation	28
5.3.4.1 arbiterIsBusy	28
5.3.4.2 channelTransmissionRate	28
5.3.4.3 currentBufferStatusList	28
5.3.4.4 ethernetFrameWaitForSend	28
5.3.4.5 numberPriorities	28
5.3.4.6 parentChannelOut	28
5.3.4.7 parentGateOut	29
5.3.4.8 parentModul	29
5.3.4.9 triggerSelf	29
5.4 ArbiterIEEE802_1Qbv Class Reference	29
5.4.1 Detailed Description	31
5.4.2 Constructor & Destructor Documentation	31
5.4.2.1 ArbiterIEEE802_1Qbv()	32
5.4.2.2 ~ArbiterIEEE802_1Qbv()	32
5.4.3 Member Function Documentation	32
5.4.3.1 arbitrateBufferStatusList()	32
5.4.3.2 calculateGuardBandDurationFromPayloadSize()	32
5.4.3.3 calculateTimeModulo()	34
5.4.3.4 calculateTransmissionDelay()	34
5.4.3.5 controlArbitrationBufferStatusList()	35
5.4.3.6 convertDoubleIntoTime()	35
5.4.3.7 dealWithControlMessage()	35
5.4.3.8 dealWithEthernetFrame()	36

5.4.3.9 dealWithExternalMessage()	36
5.4.3.10 dealWithInternalMessage()	36
5.4.3.11 dealWithSegmentTimerMessage()	37
5.4.3.12 dealWithTriggerSelf()	37
5.4.3.13 getCriticalClasses()	37
5.4.3.14 getCriticalIntervalDuration()	38
5.4.3.15 getInitialStateAndCalculateTriggerTimer()	38
5.4.3.16 getNextState()	38
5.4.3.17 getPeriodDuration()	39
5.4.3.18 getPortSpeed()	39
5.4.3.19 getSegmentDurationFromCurrentState()	39
5.4.3.20 getTimeOffset()	40
5.4.3.21 handleMessage()	40
5.4.3.22 initialize()	40
5.4.3.23 initializeTimerMessage()	40
5.4.3.24 prepareBufferStatusList()	40
5.4.3.25 printInitializationConfiguration()	41
5.4.3.26 sendBufferAuthorizedBuffer()	41
5.4.3.27 sendBufferRequest()	41
5.4.3.28 sendFrameOut()	42
5.4.3.29 setEthPacketSizeFromPayload()	42
5.4.3.30 setTriggerDelayEnd()	42
5.4.3.31 setUpVectorCriticalPriorities()	42
5.4.4 Member Data Documentation	43
5.4.4.1 arbiterIsBusy	43
5.4.4.2 criticalGuardBandSegmentDuration	43
5.4.4.3 criticalIntervalDuration	43
5.4.4.4 criticalPriorityClasses	43
5.4.4.5 criticalSendingSegmentDuration	44
5.4.4.6 currentBufferStatusList	44
5.4.4.7 currentState	44
5.4.4.8 ethernetFrameWaitForTransmission	44
5.4.4.9 frameStoredInBuffer	44
5.4.4.10 isCritical	44
5.4.4.11 maximumCriticalPayloadSize	45
5.4.4.12 maximumNoncriticalPayloadSize	45
5.4.4.13 nextState	45
5.4.4.14 noncriticalGuardBandSegmentDuration	45
5.4.4.15 noncriticalIntervalDuration	45
5.4.4.16 noncriticalSendingSegmentDuration	45
5.4.4.17 numberSupportedPriorities	46
5.4.4.18 periodDuration	46

5.4.4.19 portIndex	46
5.4.4.20 segmentTriggerTimerMessage	46
5.4.4.21 switchMAC	46
5.4.4.22 timeOffset	46
5.4.4.23 triggerSelf	47
5.4.4.24 triggerTimer	47
5.5 ArbiterIEEE802_3 Class Reference	47
5.5.1 Constructor & Destructor Documentation	48
5.5.1.1 ArbiterIEEE802_3()	48
5.5.1.2 ~ArbiterIEEE802_3()	48
5.5.2 Member Function Documentation	48
5.5.2.1 arbitrateBufferStatusList()	48
5.5.2.2 calculateTransmissionDelay()	49
5.5.2.3 calculateTransmissionSpeed()	49
5.5.2.4 controlArbitrationBufferStatusList()	49
5.5.2.5 dealWithControlMessage()	49
5.5.2.6 dealWithEthernetFrame()	50
5.5.2.7 dealWithExternalMessage()	50
5.5.2.8 dealWithInternalMessage()	50
5.5.2.9 debugStatusInitialization()	50
5.5.2.10 handleMessage()	51
5.5.2.11 initialize()	51
5.5.2.12 prepareBufferStatusList()	51
5.5.2.13 sendBufferAuthorizedBuffer()	51
5.5.2.14 sendBufferRequest()	51
5.5.2.15 sendFrameOut()	52
5.5.2.16 settriggerDelayEnd()	52
5.5.3 Member Data Documentation	52
5.5.3.1 arbiterIsBusy	52
5.5.3.2 channelTransmissionRate	52
5.5.3.3 currentBufferStatusList	52
5.5.3.4 ethernetFrameWaitForSend	52
5.5.3.5 numberPriorities	53
5.5.3.6 parentChannelOut	53
5.5.3.7 parentGateOut	53
5.5.3.8 parentModul	53
5.5.3.9 rememberLastWinner	53
5.5.3.10 triggerSelf	53
5.6 BuffersModule Class Reference	53
5.6.1 Detailed Description	54
5.6.2 Constructor & Destructor Documentation	55
5.6.2.1 BuffersModule()	55

5.6.2.2 ~BuffersModule()	55
5.6.3 Member Function Documentation	55
5.6.3.1 bufferControlLevelRequest()	55
5.6.3.2 castMessageIntoBufferControlMessage()	55
5.6.3.3 castMessageIntoEthernetFrame()	56
5.6.3.4 dealWithBufferControl()	56
5.6.3.5 dealWithEthernetFrame()	56
5.6.3.6 dealWithExternalMessage()	57
5.6.3.7 deleteBuffers()	57
5.6.3.8 deleteFramesInBuffers()	57
5.6.3.9 finish()	57
5.6.3.10 getBufferControlInstruction()	57
5.6.3.11 getFrameBufferInformation()	58
5.6.3.12 getMaximumBufferSize()	58
5.6.3.13 getNumberSupportedPriorities()	59
5.6.3.14 handleBufferControlInstruction()	59
5.6.3.15 handleMessage()	59
5.6.3.16 initialize()	59
5.6.3.17 isFristArrivingFrameWithPriority()	60
5.6.3.18 isMessageControlMessage()	60
5.6.3.19 sendBufferControlMessage()	60
5.6.3.20 sendEthernetFrameFromBuffer()	61
5.6.3.21 setInitinalUserInputs()	61
5.6.3.22 setUpBufferInformation()	61
5.6.3.23 setUpFrameBuffers()	62
5.6.3.24 storeEthernetFrameInBuffer()	62
5.6.3.25 triggerUpdateMessageNewPriorityArrival()	62
5.6.4 Member Data Documentation	62
5.6.4.1 frameBuffers	62
5.6.4.2 maximumBufferSize	62
5.6.4.3 numberSupportedPriorities	63
5.7 ClassifierPriority Class Reference	63
5.7.1 Detailed Description	63
5.7.2 Constructor & Destructor Documentation	63
5.7.2.1 ClassifierPriority()	63
5.7.2.2 ~ClassifierPriority()	64
5.7.3 Member Function Documentation	64
5.7.3.1 handleMessage()	64
5.7.3.2 initialize()	64
5.7.4 Member Data Documentation	64
5.7.4.1 framePriority	64
5.7.4.2 numberPriorities	64

5.7.4.3 parentModul	64
5.8 ConfigurationReader Class Reference	65
5.8.1 Constructor & Destructor Documentation	65
5.8.1.1 ~ConfigurationReader()	65
5.8.1.2 ConfigurationReader()	65
5.8.2 Member Function Documentation	66
5.8.2.1 getAVBTableEntriesFromInputFile()	66
5.8.2.2 getInstance()	66
5.8.2.3 getMACTableEntriesFromInputFile()	66
5.8.2.4 isCommentLine()	66
5.8.2.5 openConfigurationFile()	66
5.8.2.6 readOutInputFile()	66
5.8.2.7 splitFileLineAndGetEntry()	67
5.8.3 Member Data Documentation	67
5.8.3.1 avbTableVector	67
5.8.3.2 fileLine	67
5.8.3.3 inputFile	67
5.8.3.4 instance	67
5.8.3.5 macTableVector	67
5.9 DelayUnit Class Reference	67
5.9.1 Detailed Description	68
5.9.2 Constructor & Destructor Documentation	68
5.9.2.1 DelayUnit()	68
5.9.2.2 ~DelayUnit()	68
5.9.3 Member Function Documentation	68
5.9.3.1 handleMessage()	68
5.9.3.2 initialize()	69
5.10 E2ELatencyStatistics Class Reference	69
5.10.1 Detailed Description	69
5.10.2 Constructor & Destructor Documentation	69
5.10.2.1 E2ELatencyStatistics()	69
5.10.2.2 ~E2ELatencyStatistics()	69
5.10.3 Member Function Documentation	70
5.10.3.1 addE2ELatency()	70
5.10.3.2 finish()	70
5.10.4 Member Data Documentation	70
5.10.4.1 histogramE2Estatistics	70
5.10.4.2 vectorE2Estatistics	70
5.11 EgressPort Class Reference	71
5.11.1 Detailed Description	71
5.11.2 Constructor & Destructor Documentation	71
5.11.2.1 EgressPort()	71

5.11.2.2 ~EgressPort()	71
5.12 Endpoint Class Reference	71
5.12.1 Detailed Description	72
5.12.2 Constructor & Destructor Documentation	72
5.12.2.1 Endpoint()	72
5.12.2.2 ~Endpoint()	72
5.12.3 Member Function Documentation	72
5.12.3.1 calculateRtpsMsgSize()	72
5.12.3.2 getNextUniqueRtpsMsgId()	73
5.12.4 Member Data Documentation	73
5.12.4.1 entityId	73
5.12.4.2 entityKind	73
5.12.4.3 rtpsMsgSequenceld	73
5.13 ethernetHeaderInformation Struct Reference	73
5.13.1 Member Data Documentation	74
5.13.1.1 payloadLength	74
5.13.1.2 priority	74
5.13.1.3 ready	74
5.14 EthMsgCreator Class Reference	74
5.14.1 Detailed Description	75
5.14.2 Constructor & Destructor Documentation	75
5.14.2.1 EthMsgCreator()	75
5.14.2.2 ~EthMsgCreator()	75
5.14.3 Member Function Documentation	75
5.14.3.1 finish()	75
5.14.3.2 getNextUniqueFrameId()	76
5.14.3.3 handleMessage()	76
5.14.3.4 initialize()	76
5.14.4 Member Data Documentation	76
5.14.4.1 enableFineGrainedLatencyStats	76
5.14.4.2 uniqueFrameId	76
5.15 ForwardingLogic Class Reference	77
5.15.1 Detailed Description	77
5.15.2 Constructor & Destructor Documentation	78
5.15.2.1 ForwardingLogic()	78
5.15.2.2 ~ForwardingLogic()	78
5.15.3 Member Function Documentation	78
5.15.3.1 dealWithMulticast()	78
5.15.3.2 dealWithUnicast()	79
5.15.3.3 errorHandling()	79
5.15.3.4 forwardingByStreamID()	79
5.15.3.5 getMACTable()	80

5.15.3.6	getOnlyEntriesForThisSwitch()	80
5.15.3.7	getPortIDFromNextHOP()	80
5.15.3.8	handleMessage()	81
5.15.3.9	initialize()	81
5.15.3.10	isConnectedModuleEndNode()	81
5.15.3.11	isHopMACAddressEqualEndNodeMAC()	82
5.15.3.12	isHopMACAddressEqualSwitchMAC()	82
5.15.3.13	isValidNumberOfEntries()	83
5.15.3.14	portIDInvalid()	83
5.15.3.15	printSimulationOutputs()	83
5.15.4	Member Data Documentation	84
5.15.4.1	ethernetFrame	84
5.15.4.2	macTable	84
5.15.4.3	macTableFile	84
5.15.4.4	parentSwitch	84
5.15.4.5	streamID	84
5.15.4.6	switchIndex	84
5.15.4.7	switchMAC	85
5.16	FrameBuffer Class Reference	85
5.16.1	Detailed Description	85
5.16.2	Constructor & Destructor Documentation	86
5.16.2.1	FrameBuffer()	86
5.16.2.2	~FrameBuffer()	86
5.16.3	Member Function Documentation	86
5.16.3.1	deleteFramesInBuffer()	86
5.16.3.2	freeSpaceInBuffer()	86
5.16.3.3	getFrameFillLevel()	87
5.16.3.4	getFramePriority()	87
5.16.3.5	getNextFrameLength()	87
5.16.3.6	popNextFrame()	88
5.16.3.7	pushFrameToBack()	88
5.16.3.8	updateFreeBufferSize()	88
5.16.4	Member Data Documentation	88
5.16.4.1	bufferSize	89
5.16.4.2	droppedBytesInBuffer	89
5.16.4.3	droppedBytesVector	89
5.16.4.4	droppedFramesInBuffer	89
5.16.4.5	droppedFramesVector	89
5.16.4.6	frameBuffer	89
5.16.4.7	frameNumberInBuffer	89
5.16.4.8	frameNumberVector	89
5.16.4.9	framePriority	90

5.16.4.10 storedBytesInBuffer	90
5.16.4.11 storedBytesVector	90
5.17 FramesLostStatistics Class Reference	90
5.17.1 Detailed Description	90
5.17.2 Constructor & Destructor Documentation	90
5.17.2.1 FramesLostStatistics()	91
5.17.2.2 ~FramesLostStatistics()	91
5.17.3 Member Function Documentation	91
5.17.3.1 addLostFrame()	91
5.17.3.2 addLostFrameTrace()	91
5.17.3.3 finish()	91
5.17.4 Member Data Documentation	92
5.17.4.1 histogramLostFramesPerStream	92
5.17.4.2 vectorLostFramesPerStream	92
5.17.4.3 vectorLostFramesTracePerStream	92
5.18 Frer_ID Struct Reference	92
5.18.1 Member Data Documentation	92
5.18.1.1 port_ids	92
5.19 HistogramStatistics2DMap Class Reference	92
5.19.1 Detailed Description	93
5.19.2 Constructor & Destructor Documentation	93
5.19.2.1 HistogramStatistics2DMap()	93
5.19.2.2 ~HistogramStatistics2DMap()	94
5.19.3 Member Function Documentation	94
5.19.3.1 addIntegerResult()	94
5.19.3.2 addLatency()	94
5.19.3.3 finish()	95
5.19.4 Member Data Documentation	95
5.19.4.1 modulePrefix	95
5.19.4.2 prefix	95
5.19.4.3 storedMeasurements	95
5.19.4.4 submodulePrefix	95
5.19.4.5 valuePrefix	96
5.20 HistogramStatistics3DMap Class Reference	96
5.20.1 Detailed Description	97
5.20.2 Constructor & Destructor Documentation	97
5.20.2.1 HistogramStatistics3DMap()	97
5.20.2.2 ~HistogramStatistics3DMap()	97
5.20.3 Member Function Documentation	97
5.20.3.1 addInteger()	97
5.20.3.2 addLatency() ^[1/2]	98
5.20.3.3 addLatency() ^[2/2]	98

5.20.3.4 finish()	99
5.20.3.5 getEntry() [1/2]	99
5.20.3.6 getEntry() [2/2]	99
5.20.4 Member Data Documentation	100
5.20.4.1 empty	100
5.20.4.2 modulePrefix	100
5.20.4.3 prefix	100
5.20.4.4 storedMeasurements	100
5.20.4.5 submodulePrefix	100
5.20.4.6 valuePrefix	101
5.21 HistorySampleEntry Class Reference	101
5.21.1 Detailed Description	102
5.21.2 Constructor & Destructor Documentation	102
5.21.2.1 HistorySampleEntry()	102
5.21.2.2 ~HistorySampleEntry()	102
5.21.3 Member Data Documentation	102
5.21.3.1 complete	102
5.21.3.2 deadlineViolation	102
5.21.3.3 generationTime	102
5.21.3.4 globalStreamId	103
5.21.3.5 historySizeExceeded	103
5.21.3.6 lifespanExpired	103
5.21.3.7 numberFragments	103
5.21.3.8 readerEntityId	103
5.21.3.9 readerParticipantEntityId	103
5.21.3.10 rtpsMsgLatencies	104
5.21.3.11 sampleFragmentArray	104
5.21.3.12 sampleSequenceNumber	104
5.21.3.13 sampleSize	104
5.21.3.14 subscriberReceiveTime	104
5.21.3.15 writerArrivalTime	104
5.21.3.16 writerEntityId	104
5.21.3.17 writerFirstInjectionTime	105
5.21.3.18 writerParticipantEntityId	105
5.22 IngressPort Class Reference	105
5.22.1 Constructor & Destructor Documentation	106
5.22.1.1 IngressPort()	106
5.22.1.2 ~IngressPort()	106
5.22.2 Member Function Documentation	106
5.22.2.1 checkAndCastEthernetFrame()	106
5.22.2.2 dealWithEthernetFrame()	106
5.22.2.3 dealWithExternalMessage()	107

5.22.2.4	dealWithInternalMessage()	107
5.22.2.5	handleMessage()	107
5.22.2.6	initialize()	108
5.22.3	Member Data Documentation	108
5.22.3.1	ingressPortID	108
5.22.3.2	linkUtilizationStatistics	108
5.22.3.3	macAddress	108
5.22.3.4	numBytes	108
5.22.3.5	sampleTimePeriod	108
5.22.3.6	triggerSelf	108
5.23	JoinReaders Class Reference	109
5.23.1	Detailed Description	109
5.23.2	Constructor & Destructor Documentation	109
5.23.2.1	JoinReaders()	109
5.23.2.2	~JoinReaders()	109
5.23.3	Member Function Documentation	109
5.23.3.1	finish()	110
5.23.3.2	handleMessage()	110
5.23.3.3	initialize()	110
5.24	LinkTransmissionStatistics Class Reference	110
5.24.1	Detailed Description	110
5.24.2	Constructor & Destructor Documentation	110
5.24.2.1	LinkTransmissionStatistics()	111
5.24.2.2	~LinkTransmissionStatistics()	111
5.24.3	Member Function Documentation	111
5.24.3.1	ethernetFrameFinishedSending()	111
5.24.3.2	ethernetFrameStartedSending()	111
5.24.3.3	finish()	112
5.24.4	Member Data Documentation	112
5.24.4.1	activationPeriod	112
5.25	lostFrameCounter Struct Reference	112
5.25.1	Member Data Documentation	112
5.25.1.1	lostFrames	112
5.25.1.2	newestFrameID	113
5.26	Participant Class Reference	113
5.26.1	Detailed Description	113
5.26.2	Constructor & Destructor Documentation	113
5.26.2.1	Participant()	114
5.26.2.2	~Participant()	114
5.26.3	Member Function Documentation	114
5.26.3.1	finish()	114
5.26.3.2	handleMessage()	114

5.26.3.3 initialize()	114
5.26.4 Member Data Documentation	114
5.26.4.1 nbrReaders	115
5.26.4.2 nbrWriters	115
5.26.4.3 participantId	115
5.26.4.4 readerIds	115
5.26.4.5 writerIds	115
5.27 Reader Class Reference	115
5.27.1 Detailed Description	117
5.27.2 Constructor & Destructor Documentation	117
5.27.2.1 Reader()	117
5.27.2.2 ~Reader()	117
5.27.3 Member Function Documentation	117
5.27.3.1 checkCompletion()	117
5.27.3.2 checkLifespan()	118
5.27.3.3 cleanHistoryCache()	118
5.27.3.4 finish()	118
5.27.3.5 getCache()	118
5.27.3.6 handleMessage()	119
5.27.3.7 initialize()	119
5.27.3.8 insertSampleInCache()	119
5.27.3.9 registerReader()	119
5.27.3.10 relevantNewSample()	120
5.27.3.11 sendSampleToSink()	120
5.27.3.12 updateCache()	120
5.27.4 Member Data Documentation	121
5.27.4.1 bestEffort	121
5.27.4.2 deadline	121
5.27.4.3 entityId	121
5.27.4.4 frameLatencyInjectionToReception	121
5.27.4.5 frameLatencyInjectionToReceptionVec	121
5.27.4.6 historySizeQoS	122
5.27.4.7 lifespan	122
5.27.4.8 readerHistoryCache	122
5.27.4.9 registerEvent	122
5.27.4.10 SampleLatencySinceGenerationStat	122
5.27.4.11 SampleLatencySinceGenerationStatVec	122
5.27.4.12 SampleLatencySinceSampleInjectionStat	123
5.27.4.13 SampleLatencySinceSampleInjectionStatVec	123
5.27.4.14 SampleLatencySinceWriterArrivalStat	123
5.27.4.15 SampleLatencySinceWriterArrivalStatVec	123
5.27.4.16 sourceMac	123

5.27.4.17 streamEthernetPriority	123
5.27.4.18 transportPriority	124
5.28 ReaderHistory Class Reference	124
5.28.1 Detailed Description	124
5.28.2 Constructor & Destructor Documentation	124
5.28.2.1 ReaderHistory()	124
5.28.2.2 ~ReaderHistory()	124
5.28.3 Member Data Documentation	125
5.28.3.1 cache	125
5.28.3.2 maxNbrStreams	125
5.28.3.3 nbrStreams	125
5.29 Rtps Class Reference	125
5.29.1 Detailed Description	126
5.29.2 Constructor & Destructor Documentation	126
5.29.2.1 Rtps()	126
5.29.2.2 ~Rtps()	126
5.29.3 Member Function Documentation	126
5.29.3.1 initialize()	126
5.30 RtpsToEthernetAdapter Class Reference	126
5.30.1 Detailed Description	127
5.30.2 Constructor & Destructor Documentation	127
5.30.2.1 RtpsToEthernetAdapter()	127
5.30.2.2 ~RtpsToEthernetAdapter()	127
5.30.3 Member Function Documentation	127
5.30.3.1 finish()	128
5.30.3.2 getNextFrameSequenceld()	128
5.30.3.3 handleMessage()	128
5.30.3.4 initialize()	128
5.30.3.5 sendRtpsPacket()	128
5.30.4 Member Data Documentation	128
5.30.4.1 frameSequenceld	129
5.31 SampleFragment Class Reference	129
5.31.1 Detailed Description	129
5.31.2 Constructor & Destructor Documentation	129
5.31.2.1 SampleFragment()	129
5.31.2.2 ~SampleFragment()	130
5.31.3 Member Data Documentation	130
5.31.3.1 acked	130
5.31.3.2 fragmentSequenceNumber	130
5.31.3.3 fragmentSize	130
5.31.3.4 received	130
5.31.3.5 sampleReference	130

5.31.3.6 send	131
5.31.3.7 sendTime	131
5.32 SampleLatencyStatistics Class Reference	131
5.32.1 Detailed Description	131
5.32.2 Constructor & Destructor Documentation	131
5.32.2.1 SampleLatencyStatistics()	131
5.32.2.2 ~SampleLatencyStatistics()	132
5.32.3 Member Function Documentation	132
5.32.3.1 addMessage()	132
5.32.3.2 finish()	132
5.32.3.3 report()	132
5.32.4 Member Data Documentation	133
5.32.4.1 sampleLatencyStatisticsHistogram	133
5.32.4.2 sampleLatencyStatisticsVector	133
5.33 SampleMsgCreator Class Reference	133
5.33.1 Detailed Description	134
5.33.2 Constructor & Destructor Documentation	134
5.33.2.1 SampleMsgCreator()	134
5.33.2.2 ~SampleMsgCreator()	134
5.33.3 Member Function Documentation	134
5.33.3.1 finish()	134
5.33.3.2 getNextSampleSequenceNumberId()	134
5.33.3.3 handleMessage()	134
5.33.3.4 initialize()	135
5.33.4 Member Data Documentation	135
5.33.4.1 sampleSequenceNumberId	135
5.34 SampleSink Class Reference	135
5.34.1 Detailed Description	136
5.34.2 Member Function Documentation	136
5.34.2.1 handleMessage()	136
5.34.2.2 initialize()	136
5.34.3 Member Data Documentation	136
5.34.3.1 myMac	136
5.35 Shaper Class Reference	136
5.35.1 Constructor & Destructor Documentation	137
5.35.1.1 Shaper()	137
5.35.1.2 ~Shaper()	137
5.35.2 Member Function Documentation	137
5.35.2.1 finish()	137
5.35.2.2 handleMessage()	138
5.35.2.3 initialize()	138
5.35.2.4 send_from_queue()	138

5.35.3 Member Data Documentation	138
5.35.3.1 nextMsgReady	138
5.35.3.2 rtpsMsgQueue	138
5.35.3.3 shaperEnabled	138
5.35.3.4 shapingDuration	139
5.36 Sink Class Reference	139
5.36.1 Detailed Description	139
5.36.2 Member Function Documentation	140
5.36.2.1 checkDestinationsMAC()	140
5.36.2.2 checkForMissingFrame()	140
5.36.2.3 finish()	140
5.36.2.4 handleMessage()	140
5.36.2.5 initialize()	140
5.36.2.6 processEthernetFrameAtDestination()	141
5.36.3 Member Data Documentation	141
5.36.3.1 lostFrameCounters	141
5.36.3.2 macAddress	141
5.36.3.3 trackMissingDataMap	141
5.37 StatisticManager Class Reference	141
5.37.1 Detailed Description	142
5.37.2 Constructor & Destructor Documentation	142
5.37.2.1 ~StatisticManager()	142
5.37.2.2 StatisticManager()	142
5.37.3 Member Function Documentation	143
5.37.3.1 finishAll()	143
5.37.3.2 getDelayStatistics()	143
5.37.3.3 getFramesLostStatistics()	143
5.37.3.4 getInstance()	143
5.37.3.5 getLinkTransmissionStatistics()	144
5.37.3.6 getSampleLatencyStatistics()	144
5.37.3.7 getTimeAwareShaperStatistics()	144
5.37.4 Member Data Documentation	144
5.37.4.1 framesLostStats	144
5.37.4.2 hasFinishedAll	145
5.37.4.3 linkTransmission	145
5.37.4.4 pathDelayStats	145
5.37.4.5 sampleLatencyStats	145
5.37.4.6 tsnTASStatistics	145
5.38 TimeAwareShaperSegmentStatistic Class Reference	145
5.38.1 Detailed Description	146
5.38.2 Constructor & Destructor Documentation	146
5.38.2.1 TimeAwareShaperSegmentStatistic()	146

5.38.2.2 ~TimeAwareShaperSegmentStatistic()	146
5.38.3 Member Function Documentation	146
5.38.3.1 criticalGuardbandSegmentStarts()	146
5.38.3.2 criticalSendingSegmentStarts()	147
5.38.3.3 finish()	147
5.38.3.4 noncriticalGuardbandSegmentStarts()	147
5.38.3.5 noncriticalSendingSegmentStarts()	147
5.38.4 Member Data Documentation	148
5.38.4.1 tasPortTimeSegments	148
5.39 TrafficSource Class Reference	148
5.39.1 Detailed Description	149
5.39.2 Constructor & Destructor Documentation	149
5.39.2.1 TrafficSource()	149
5.39.2.2 ~TrafficSource()	149
5.39.3 Member Function Documentation	149
5.39.3.1 handleMessage()	149
5.39.3.2 initialize()	150
5.39.3.3 scheduleNextBurst()	150
5.39.3.4 scheduleNextIntraBurst()	150
5.39.3.5 sendPacket()	150
5.39.4 Member Data Documentation	150
5.39.4.1 generatorEnable	151
5.39.4.2 globalStreamId	151
5.39.4.3 interBurstMsg	151
5.39.4.4 intraBurstMsg	151
5.39.4.5 pendingBursts	151
5.39.4.6 pendingIntraBurstSize	151
5.40 UdpIpStack Class Reference	152
5.40.1 Detailed Description	152
5.40.2 Constructor & Destructor Documentation	152
5.40.2.1 UdpIpStack()	152
5.40.2.2 ~UdpIpStack()	152
5.40.3 Member Function Documentation	152
5.40.3.1 handleMessage()	152
5.40.3.2 initialize()	153
5.41 ValidationHandler Class Reference	153
5.41.1 Member Function Documentation	153
5.41.1.1 ThrowError()	153
5.41.1.2 validateInputBounds()	153
5.42 VectorStatistics2DMap Class Reference	154
5.42.1 Detailed Description	154
5.42.2 Constructor & Destructor Documentation	154

5.42.2.1 VectorStatistics2DMap()	154
5.42.2.2 ~VectorStatistics2DMap()	155
5.42.3 Member Function Documentation	155
5.42.3.1 accumulateInteger()	155
5.42.3.2 addInteger()	155
5.42.3.3 addLatency()	156
5.42.3.4 finish()	156
5.42.3.5 getEntry()	156
5.42.4 Member Data Documentation	157
5.42.4.1 modulePrefix	157
5.42.4.2 prefix	157
5.42.4.3 storedMeasurements	157
5.42.4.4 submodulePrefix	157
5.42.4.5 valuePrefix	157
5.43 VectorStatistics3DMap Class Reference	157
5.43.1 Constructor & Destructor Documentation	158
5.43.1.1 VectorStatistics3DMap()	158
5.43.1.2 ~VectorStatistics3DMap()	159
5.43.2 Member Function Documentation	159
5.43.2.1 accumulateInteger()	159
5.43.2.2 addInteger()	159
5.43.2.3 addLatency() [1/2]	160
5.43.2.4 addLatency() [2/2]	160
5.43.2.5 finish()	161
5.43.2.6 getEntry() [1/2]	161
5.43.2.7 getEntry() [2/2]	161
5.43.3 Member Data Documentation	161
5.43.3.1 empty	161
5.43.3.2 modulePrefix	161
5.43.3.3 prefix	161
5.43.3.4 storedMeasurements	162
5.43.3.5 submodulePrefix	162
5.43.3.6 valuePrefix	162
5.44 Writer Class Reference	162
5.44.1 Detailed Description	163
5.44.2 Constructor & Destructor Documentation	163
5.44.2.1 Writer()	164
5.44.2.2 ~Writer()	164
5.44.3 Member Function Documentation	164
5.44.3.1 addNextFragmentToSendList()	164
5.44.3.2 checkLifespanOfSamples()	164
5.44.3.3 createRtpsMsgFromFragment()	164

5.44.3.4 finish()	164
5.44.3.5 handleMessage()	165
5.44.3.6 initialize()	165
5.44.3.7 registerWriter()	165
5.44.3.8 sendMessage()	165
5.44.4 Member Data Documentation	165
5.44.4.1 bestEffort	165
5.44.4.2 currentSampleSequenceNumber	166
5.44.4.3 destinationEntityId	166
5.44.4.4 destinationMac	166
5.44.4.5 entityId	166
5.44.4.6 fragmentSize	166
5.44.4.7 globalStreamId	166
5.44.4.8 historyCache	167
5.44.4.9 historySizeQoS	167
5.44.4.10 lifespan	167
5.44.4.11 registerEvent	167
5.44.4.12 send_queue	167
5.44.4.13 sendEvent	167
5.44.4.14 shapingDuration	168
5.44.4.15 sourceMac	168
5.44.4.16 streamEthernetPriority	168
5.44.4.17 transportPriority	168
6 File Documentation	169
6.1 ArbiterAVB.cc File Reference	169
6.2 ArbiterAVB.h File Reference	169
6.2.1 Enumeration Type Documentation	170
6.2.1.1 arbiterClassStatus	170
6.2.1.2 avbTablePostition	170
6.2.2 Function Documentation	170
6.2.2.1 Define_Module()	170
6.3 ArbiterIEEE802_1Q.cc File Reference	170
6.4 ArbiterIEEE802_1Q.h File Reference	171
6.4.1 Function Documentation	171
6.4.1.1 Define_Module()	171
6.5 ArbiterIEEE802_1Qbv.cc File Reference	171
6.6 ArbiterIEEE802_1Qbv.h File Reference	171
6.6.1 Enumeration Type Documentation	172
6.6.1.1 timeSegments	172
6.6.2 Function Documentation	172
6.6.2.1 Define_Module()	172

6.7 ArbiterIEEE802_3.cc File Reference	172
6.8 ArbiterIEEE802_3.h File Reference	173
6.8.1 Function Documentation	173
6.8.1.1 Define_Module()	173
6.9 BuffersModule.cc File Reference	173
6.10 BuffersModule.h File Reference	173
6.10.1 Function Documentation	174
6.10.1.1 Define_Module()	174
6.11 ClassifierPriority.cc File Reference	174
6.12 ClassifierPriority.h File Reference	174
6.12.1 Function Documentation	174
6.12.1.1 Define_Module()	175
6.13 ConfigurationReader.cc File Reference	175
6.14 ConfigurationReader.h File Reference	175
6.14.1 Macro Definition Documentation	175
6.14.1.1 COMMENT_POSITION	175
6.14.1.2 COMMENT_SYMBOL	176
6.14.2 Typedef Documentation	176
6.14.2.1 configVector	176
6.15 ddsSampleLatencyStatistics.cc File Reference	176
6.16 ddsSampleLatencyStatistics.h File Reference	176
6.17 DelayUnit.cc File Reference	176
6.17.1 Function Documentation	176
6.17.1.1 Define_Module()	177
6.18 DelayUnit.h File Reference	177
6.19 E2ELatencyStatistics.cc File Reference	177
6.20 E2ELatencyStatistics.h File Reference	177
6.21 EgressPort.cc File Reference	177
6.22 EgressPort.h File Reference	177
6.22.1 Function Documentation	178
6.22.1.1 Define_Module()	178
6.23 Endpoint.cc File Reference	178
6.24 Endpoint.h File Reference	178
6.25 EthernetHeaderInformation.h File Reference	178
6.26 EthernetMacros.h File Reference	179
6.26.1 Macro Definition Documentation	179
6.26.1.1 _CRC_	179
6.26.1.2 _CRC_INTERPACKET_GAP_	180
6.26.1.3 _eth_check_for_max_payload_	180
6.26.1.4 _eth_check_for_padding_	180
6.26.1.5 _eth_framesize_from_payload_	180
6.26.1.6 _eth_packetsize_from_payload_	180

6.26.1.7	<code>_FRAME_OVERHEAD_</code>	180
6.26.1.8	<code>_INTERPACKET_GAP_</code>	180
6.26.1.9	<code>_MAX_ETH_PAYLOAD_</code>	181
6.26.1.10	<code>_MIN_ETH_FRAME_SIZE_</code>	181
6.26.1.11	<code>_MIN_ETH_PACKET_SIZE_</code>	181
6.26.1.12	<code>_MIN_ETH_PACKET_SIZE_PREEMPTION_</code>	181
6.26.1.13	<code>_MIN_ETH_PAYLOAD_</code>	181
6.26.1.14	<code>_MIN_ETH_PAYLOAD_UNTAGGED_</code>	181
6.26.1.15	<code>_PACKET_OVERHEAD_</code>	181
6.26.1.16	<code>_PACKET_OVERHEAD_FIRST_PREEMPTION_</code>	181
6.26.1.17	<code>_PACKET_OVERHEAD_INTERMEDIATE_PREEMPTION_</code>	182
6.26.1.18	<code>_PAYLOAD_INTERMEDIATE_PREEMPTION_</code>	182
6.26.1.19	<code>_std_eth_colour_</code>	182
6.26.2	Enumeration Type Documentation	182
6.26.2.1	<code>_eth_egress_helper_message_type_</code>	182
6.27	EthMsgCreator.cc File Reference	182
6.28	EthMsgCreator.h File Reference	182
6.28.1	Function Documentation	183
6.28.1.1	<code>Define_Module()</code>	183
6.29	ForwardingLogic.cc File Reference	183
6.30	ForwardingLogic.h File Reference	183
6.30.1	Enumeration Type Documentation	184
6.30.1.1	<code>errorType</code>	184
6.30.1.2	<code>macTableEntriesPositon</code>	184
6.30.1.3	<code>simulationOutputType</code>	184
6.30.2	Function Documentation	185
6.30.2.1	<code>Define_Module()</code>	185
6.31	FrameBuffer.cc File Reference	185
6.32	FrameBuffer.h File Reference	185
6.32.1	Enumeration Type Documentation	185
6.32.1.1	<code>bufferUpdate</code>	185
6.33	FramesLostStatistics.cc File Reference	186
6.34	FramesLostStatistics.h File Reference	186
6.35	GlobalMacros.h File Reference	186
6.35.1	Macro Definition Documentation	187
6.35.1.1	<code>_CAN2CAN_LOOKUP_</code>	187
6.35.1.2	<code>_CAN2ETH_LOOKUP_</code>	187
6.35.1.3	<code>_ETH2CAN_LOOKUP_</code>	187
6.35.1.4	<code>_ETH2ETH_LOOKUP_</code>	187
6.35.1.5	<code>_max_</code>	187
6.35.1.6	<code>_min_</code>	188
6.35.1.7	<code>_msg_black_</code>	188

6.35.1.8 _msg_blue_	188
6.35.1.9 _msg_cyan_	188
6.35.1.10 _msg_green_	188
6.35.1.11 _msg_magenta_	188
6.35.1.12 _msg_red_	188
6.35.1.13 _msg_white_	189
6.35.1.14 _msg_yellow_	189
6.35.2 Typedef Documentation	189
6.35.2.1 csvVector	189
6.35.3 Function Documentation	189
6.35.3.1 simtime_modulo()	189
6.35.3.2 truncate_min_max()	189
6.36 HistogramStatistics2DMap.cc File Reference	189
6.37 HistogramStatistics2DMap.h File Reference	190
6.38 HistogramStatistics3DMap.cc File Reference	190
6.39 HistogramStatistics3DMap.h File Reference	190
6.40 HistoryEntry.h File Reference	190
6.41 IngressPort.cc File Reference	190
6.42 IngressPort.h File Reference	191
6.42.1 Function Documentation	191
6.42.1.1 Define_Module()	191
6.43 JoinReaders.cc File Reference	191
6.43.1 Function Documentation	191
6.43.1.1 Define_Module()	191
6.44 JoinReaders.h File Reference	192
6.45 LinkTransmissionStatistics.cc File Reference	192
6.46 LinkTransmissionStatistics.h File Reference	192
6.47 Participant.cc File Reference	192
6.48 Participant.h File Reference	192
6.48.1 Function Documentation	193
6.48.1.1 Define_Module()	193
6.49 Reader.cc File Reference	193
6.50 Reader.h File Reference	193
6.50.1 Function Documentation	193
6.50.1.1 Define_Module()	194
6.51 README_RTPS.md File Reference	194
6.52 Rtps.cc File Reference	194
6.52.1 Function Documentation	194
6.52.1.1 Define_Module()	194
6.53 Rtps.h File Reference	194
6.54 RtpsMacros.h File Reference	194
6.54.1 Macro Definition Documentation	195

6.54.1.1 _IP_HEADER_SIZE_	195
6.54.1.2 _MAX_RTPS_PAYLOAD_	195
6.54.1.3 _MIN_RTPS_PAYLOAD_	196
6.54.1.4 _rtps_add_udp_ip_overhead_	196
6.54.1.5 _rtps_check_for_max_payload_	196
6.54.1.6 _RTPS_DATAFRAG_STATIC_SIZE_	196
6.54.1.7 _RTPS_HEADER_SIZE_	196
6.54.1.8 _RTPS_HEARTBEATFRAG_SIZE_	196
6.54.1.9 _RTPS_INFO_DST_SIZE_	196
6.54.1.10 _RTPS_INFO_TS_SIZE_	197
6.54.1.11 _RTPS_MAX_FRAG_SIZE_	197
6.54.1.12 _RTPS_MAX_MSG_SIZE_	197
6.54.1.13 _RTPS_NACKFRAG_STATIC_SIZE_	197
6.54.1.14 _UDP_HEADER_SIZE_	197
6.54.2 Enumeration Type Documentation	197
6.54.2.1 rtpsEntityKind	197
6.55 RtpsToEthernetAdapter.cc File Reference	198
6.55.1 Function Documentation	198
6.55.1.1 Define_Module()	198
6.56 RtpsToEthernetAdapter.h File Reference	198
6.56.1 Macro Definition Documentation	198
6.56.1.1 _adapter_src_verbose_	199
6.57 SampleMsgCreator.cc File Reference	199
6.58 SampleMsgCreator.h File Reference	199
6.58.1 Function Documentation	199
6.58.1.1 Define_Module()	199
6.59 SampleSink.cc File Reference	199
6.59.1 Function Documentation	200
6.59.1.1 Define_Module()	200
6.60 SampleSink.h File Reference	200
6.60.1 Macro Definition Documentation	200
6.60.1.1 _eth_sampleSink_verbose_	200
6.61 Shaper.cc File Reference	200
6.61.1 Function Documentation	201
6.61.1.1 Define_Module()	201
6.62 Shaper.h File Reference	201
6.63 Sink.cc File Reference	201
6.64 Sink.h File Reference	201
6.64.1 Function Documentation	202
6.64.1.1 Define_Module()	202
6.65 StatisticManager.cc File Reference	202
6.66 StatisticManager.h File Reference	202

6.67 TimeAwareShaperSegmentStatistic.cc File Reference	202
6.68 TimeAwareShaperSegmentStatistic.h File Reference	203
6.68.1 Enumeration Type Documentation	203
6.68.1.1 tasTimeSegments	203
6.69 TrafficSource.cc File Reference	203
6.70 TrafficSource.h File Reference	203
6.70.1 Macro Definition Documentation	204
6.70.1.1 _burst_src_verbose_	204
6.70.2 Function Documentation	204
6.70.2.1 Define_Module()	204
6.71 UdpIpStack.cc File Reference	204
6.72 UdpIpStack.h File Reference	204
6.72.1 Function Documentation	205
6.72.1.1 Define_Module()	205
6.73 ValidationHandler.h File Reference	205
6.74 VectorStatistics2DMap.cc File Reference	205
6.75 VectorStatistics2DMap.h File Reference	205
6.76 VectorStatistics3DMap.cc File Reference	206
6.77 VectorStatistics3DMap.h File Reference	206
6.78 Writer.cc File Reference	206
6.79 Writer.h File Reference	206
6.79.1 Function Documentation	207
6.79.1.1 Define_Module()	207

Chapter 1

RTPS Configuration

The following parameters have to be configured

- For a multicast transmission reader entityIds should have the same Id even if they are in different participants!
- * Therefore search an entityId, which is not used in each of the receiving participants

1.1 Overall configuration

- `**middleware.delayUnit[*].duration = normal(xms, yms)` (Distribution of the initial delay of a generated Sample before it is handled in the RTPS middleware)
- `**middleware.delayUnit[*].durationMin = xms` (The minimum initial delay of a generated Sample before it is handled in the RTPS middleware)
- `**middleware.delayUnit[*].durationMax = xms` (The maximum initial delay of a generated Sample before it is handled in the RTPS middleware)
- `**middleware.shaperEnabled = true` (Shaping over all outgoing messages of an RTPS middleware component)
- `**middleware.shapingDuration = 10us` (Shaping duration over all outgoing messages of an RTPS middleware component)
- `**middleware.writer[*].history = 10` (The history size of a writer entity)
- `**middleware.reader[*].history = 10` (The history size of a reader entity)

1.2 Endpoint specific

- `**Endpoint.middleware.participantId = x` (globally unique participant Id)
- `**Endpoint.numMwStreams = x` (The number of middleware writers)
- `**Endpoint.numMwSinks = x` (The number of middleware readers)

1.3 Stream specific

- `**Endpoint.sampleGenerator[<>].sampleSize = normal(xB,yB)` (The size distribution of a sample)
- `**Endpoint.sampleGenerator[<>].sampleSizeMin = xB` (The minimum size of a sample)
- `**Endpoint.sampleGenerator[<>].sampleSizeMax = xB` (The maximum size of a sample)
- `**Endpoint.sampleGenerator[<>].period = normal(xs,ys)` (The period distribution of a sample i.e. the distance between two consecutive sample generations)
- `**Endpoint.sampleGenerator[<>].periodMin = xs` (The minimum period)
- `**Endpoint.sampleGenerator[<>].periodMax = xs` (The maximum period)
- `**Endpoint.sampleGenerator[<>].globalStreamId = <>` (The unique global stream Id)

1.4 Wrtier specific

- `**Endpoint.middleware.writer[<>].globalStreamId = <>` (The unique global stream ID of the writer corresponding to the connected generator)
- `**Endpoint.middleware.writer[<>].streamEthernetPriority = <>` (The Ethernet priority)
- `**Endpoint.middleware.writer[<>].sourceMac = <>` (The own egress port to send the messages out)
- `**Endpoint.middleware.writer[<>].entityId = <>` (The own entity Id which has to be unique for the participant)
- `**Endpoint.middleware.writer[<>].destinationMac = <>` (The remote (ingress) port MAC adress to send the messages to)
- `**Endpoint.middleware.writer[<>].destinationEntityId = <>` (The remote entity Id – Note that for multicast all readers have to be configured with this adress)
- `**Endpoint.middleware.writer[<>].fragmentSize = <>` (The fragment size)
- `**Endpoint.middleware.writer[<>].shaper.shapingDuration = <>` (The frame shaping duration of the writer for its sample transmission)

1.5 Reader specific

- `**Endpoint.middleware.reader[<>].entityId = X` (The readers entityId which has to be unique for the RTPS participant)
- `**Endpoint.middleware.reader[<>].sourceMac = X` (The own egress prot to send the messages out – backchannel for retransmissions)

1.6 Example

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

arbiterAVBClass	21
cModule	
EgressPort	71
ConfigurationReader	65
cSimpleModule	
ArbiterAVB	9
ArbiterIEEE802_1Q	23
ArbiterIEEE802_1Qbv	29
ArbiterIEEE802_3	47
BuffersModule	53
ClassifierPriority	63
DelayUnit	67
EthMsgCreator	74
ForwardingLogic	77
IngressPort	105
JoinReaders	109
Participant	113
Reader	115
Rtps	125
RtpsToEthernetAdapter	126
SampleMsgCreator	133
SampleSink	135
Shaper	136
Sink	139
TrafficSource	148
UdpIpStack	152
Writer	162
E2ELatencyStatistics	69
Endpoint	71
Reader	115
Writer	162
ethernetHeaderInformation	73
FrameBuffer	85
FramesLostStatistics	90
Frer_ID	92

HistogramStatistics2DMap	92
HistogramStatistics3DMap	96
HistorySampleEntry	101
LinkTransmissionStatistics	110
lostFrameCounter	112
ReaderHistory	124
SampleFragment	129
SampleLatencyStatistics	131
StatisticManager	141
TimeAwareShaperSegmentStatistic	145
ValidationHandler	153
VectorStatistics2DMap	154
VectorStatistics3DMap	157

Chapter 3

Class Index

3.1 Class List

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

ArbiterAVB	9
arbiterAVBClass	21
ArbiterIEEE802_1Q	23
ArbiterIEEE802_1Qbv	29
ArbiterIEEE802_3	47
BuffersModule	53
ClassifierPriority	63
ConfigurationReader	65
DelayUnit	67
E2ELatencyStatistics	69
EgressPort	71
Endpoint	71
ethernetHeaderInformation	73
EthMsgCreator	74
ForwardingLogic	77
FrameBuffer	85
FramesLostStatistics	90
Frer_ID	92
HistogramStatistics2DMap	92
HistogramStatistics3DMap	96
HistorySampleEntry	101
IngressPort	105
JoinReaders	109
LinkTransmissionStatistics	110
lostFrameCounter	112
Participant	113
Reader	115
ReaderHistory	124
Rtps	125
RtpsToEthernetAdapter	126
SampleFragment	129
SampleLatencyStatistics	131
SampleMsgCreator	133
SampleSink	135
Shaper	136

Sink	139
StatisticManager	141
TimeAwareShaperSegmentStatistic	145
TrafficSource	148
UdpIpStack	152
ValidationHandler	153
VectorStatistics2DMap	154
VectorStatistics3DMap	157
Writer	162

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

ArbiterAVB.cc	169
ArbiterAVB.h	169
ArbiterIEEE802_1Q.cc	170
ArbiterIEEE802_1Q.h	171
ArbiterIEEE802_1Qbv.cc	171
ArbiterIEEE802_1Qbv.h	171
ArbiterIEEE802_3.cc	172
ArbiterIEEE802_3.h	173
BuffersModule.cc	173
BuffersModule.h	173
ClassifierPriority.cc	174
ClassifierPriority.h	174
ConfigurationReader.cc	175
ConfigurationReader.h	175
ddsSampleLatencyStatistics.cc	176
ddsSampleLatencyStatistics.h	176
DelayUnit.cc	176
DelayUnit.h	177
E2ELatencyStatistics.cc	177
E2ELatencyStatistics.h	177
EgressPort.cc	177
EgressPort.h	177
Endpoint.cc	178
Endpoint.h	178
EthernetHeaderInformation.h	178
EthernetMacros.h	179
EthMsgCreator.cc	182
EthMsgCreator.h	182
ForwardingLogic.cc	183
ForwardingLogic.h	183
FrameBuffer.cc	185
FrameBuffer.h	185
FramesLostStatistics.cc	186
FramesLostStatistics.h	186
GlobalMacros.h	186

HistogramStatistics2DMap.cc	189
HistogramStatistics2DMap.h	190
HistogramStatistics3DMap.cc	190
HistogramStatistics3DMap.h	190
HistoryEntry.h	190
IngressPort.cc	190
IngressPort.h	191
JoinReaders.cc	191
JoinReaders.h	192
LinkTransmissionStatistics.cc	192
LinkTransmissionStatistics.h	192
Participant.cc	192
Participant.h	192
Reader.cc	193
Reader.h	193
Rtps.cc	194
Rtps.h	194
RtpsMacros.h	194
RtpsToEthernetAdapter.cc	198
RtpsToEthernetAdapter.h	198
SampleMsgCreator.cc	199
SampleMsgCreator.h	199
SampleSink.cc	199
SampleSink.h	200
Shaper.cc	200
Shaper.h	201
Sink.cc	201
Sink.h	201
StatisticManager.cc	202
StatisticManager.h	202
TimeAwareShaperSegmentStatistic.cc	202
TimeAwareShaperSegmentStatistic.h	203
TrafficSource.cc	203
TrafficSource.h	203
UdplpStack.cc	204
UdplpStack.h	204
ValidationHandler.h	205
VectorStatistics2DMap.cc	205
VectorStatistics2DMap.h	205
VectorStatistics3DMap.cc	206
VectorStatistics3DMap.h	206
Writer.cc	206
Writer.h	206

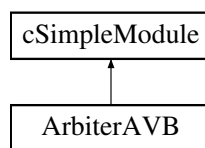
Chapter 5

Class Documentation

5.1 ArbiterAVB Class Reference

```
#include <ArbiterAVB.h>
```

Inheritance diagram for ArbiterAVB:



Public Member Functions

- [ArbiterAVB](#) ()
- virtual [~ArbiterAVB](#) ()
- void [initialize](#) ()
- void [handleMessage](#) (cMessage *message)

Private Member Functions

- void [calculateTransmissionSpeed](#) ()
- void [initializeAVBClassInformation](#) ()
- void [loadAVBClassInformationFromFile](#) ()
- void [dealWithInternalMessage](#) (cMessage *message)
- void [dealWithExternalMessage](#) (cMessage *message)
- EthernetFrame * [checkAndCastEthernetFrame](#) (cMessage *message)
- BufferControlMessage * [checkAndCastBufferControlMessage](#) (cMessage *message)
- void [dealWithEthernetFrame](#) (cMessage *message)
- void [dealWithControlMessage](#) (cMessage *controlMessage)
- void [controlArbitrationBufferStatusList](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- void [prepareClassInformations](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- void [calculateClassInformationCredits](#) ()
- void [calculateCredits](#) (int priority)
- int [calculateCreditValues](#) (int priority)

- `std::vector< ethernetHeaderInformation * > prepareBufferStatusList` (`std::vector< ethernetHeaderInformation * > BufferStatusList`)
- `int arbitrateBufferStatusList` (`std::vector< ethernetHeaderInformation * > BufferStatusList`)
- `void sendBufferAuthorizedBuffer` (`int authorizedBuffer`)
- `void sendBufferRequest` ()
- `simtime_t calculateTransmissionDelay` (`EthernetFrame *ethernetFrame`)
- `void setRechargeTime` ()
- `void settriggerDelayEnd` (`simtime_t transmissionPathDelay`)
- `void settriggerCreditRecharged` (`simtime_t calculatedRechargeTime`, `int priority`)
- `simtime_t calculatedRechargeTime` (`int priority`)
- `void sendFrameOut` ()
- `void updateClassInformationsStatus` (`int authorizedBuffer`)
- `void collectCreditsStatistics` (`std::vector< ethernetHeaderInformation * > BufferStatusList`)
- `void debugStatus` ()
- `void debugStatusInitialization` ()

Private Attributes

- `std::vector< ethernetHeaderInformation * > currentBufferStatusList`
- `int numberPriorities`
- `cModule * parentModul`
- `cGate * parentGateOut`
- `cChannel * parentChannelOut`
- `double channelTransmissionRate`
- `int egressPortIndex`
- `int macAddress`
- `configVector avbTableVector`
- `cOutVector creditValueStatisticStreamA`
- `cOutVector creditValueStatisticStreamB`
- `csvVector config`
- `std::string configFile`
- `bool arbiterIsBusy`
- `EthernetFrame * ethernetFrameWaitForSend`
- `cMessage * triggerSelf = new cMessage()`
- `std::vector< arbiterAVBClass * > classInformations`

5.1.1 Detailed Description

[ArbiterAVB](#) is an arbiter which arbitrate the stored Frames in [BuffersModule](#). Arbiter will select based on the CBS algorithm the next sending Frame and store it for the length of the transmission delay

5.1.2 Constructor & Destructor Documentation

5.1.2.1 ArbiterAVB()

```
ArbiterAVB::ArbiterAVB ( )
```

Constructor [ArbiterAVB](#), nothing to do here

5.1.2.2 ~ArbiterAVB()

```
ArbiterAVB::~~ArbiterAVB ( ) [virtual]
```

Destructor [ArbiterAVB](#), nothing to do here

5.1.3 Member Function Documentation

5.1.3.1 arbitrateBufferStatusList()

```
int ArbiterAVB::arbitrateBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

Arbiter which choose out of the Buffer Status List the current winner for next transmission

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.1.3.2 calculateClassInformationCredits()

```
void ArbiterAVB::calculateClassInformationCredits ( ) [private]
```

choose for which priority the credits have to be updated

5.1.3.3 calculateCredits()

```
void ArbiterAVB::calculateCredits (
    int priority ) [private]
```

calculate the total credit values of that priority

Parameters

<i>priority</i>	
-----------------	--

5.1.3.4 calculateCreditValues()

```
int ArbiterAVB::calculateCreditValues (
    int priority ) [private]
```

calculate the current credit value based on the time difference of sending the frame

Parameters

<i>priority</i>	
-----------------	--

5.1.3.5 calculatedRechargeTime()

```
simtime_t ArbiterAVB::calculatedRechargeTime (
    int priority ) [private]
```

calculated recharge Time

Parameters

<i>priority</i>	
-----------------	--

Returns

simtime_t

5.1.3.6 calculateTransmissionDelay()

```
simtime_t ArbiterAVB::calculateTransmissionDelay (
    EthernetFrame * ethernetFrame ) [private]
```

Calculate based on Ethernet frame length transmission delay

Parameters

<i>ethernetFrame</i>	
----------------------	--

Returns

simtime_t

5.1.3.7 calculateTransmissionSpeed()

```
void ArbiterAVB::calculateTransmissionSpeed ( ) [private]
```

Calculate Transmission seed of channel SafeGuard prevent for choose nullpointer as channel

5.1.3.8 checkAndCastBufferControlMessage()

```
BufferControlMessage * ArbiterAVB::checkAndCastBufferControlMessage (
    cMessage * message ) [private]
```

check if message is an BufferControlMessage and return it

Parameters

<i>message</i>	
----------------	--

Returns

BufferControlMessage*

5.1.3.9 checkAndCastEthernetFrame()

```
EthernetFrame * ArbiterAVB::checkAndCastEthernetFrame (
    cMessage * message ) [private]
```

check if message is an Ethernet Frame and return it

Parameters

<i>message</i>	this is a comment about the input variable
----------------	--

Returns

pointer to EthernetFrame

5.1.3.10 collectCreditsStatistics()

```
void ArbiterAVB::collectCreditsStatistics (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

This function record Credit Values to cOutVector structures If more than two Priorities exist AVB will work and than statistics are collected. Just Stream A and Stream B are collected, BE is ignored

Parameters

<i>BufferStatusList</i>	with informations about AVB Arbiter
-------------------------	-------------------------------------

5.1.3.11 controlArbitrationBufferStatusList()

```
void ArbiterAVB::controlArbitrationBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

control the arbitration steps. First prepare the bufferList and calculate the credits(recharging if nessecary). Based on new credit value arbitrate and collect statistics. Last but not least the arbitration result is send to BuffersMoudles.

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.1.3.12 dealWithControlMessage()

```
void ArbiterAVB::dealWithControlMessage (
    cMessage * controlMessage ) [private]
```

Select which type of Control Message is received

Parameters

<i>controlMessage</i>	
-----------------------	--

5.1.3.13 dealWithEthernetFrame()

```
void ArbiterAVB::dealWithEthernetFrame (
    cMessage * message ) [private]
```

deal with incoming Ethernet Frames

Parameters

<i>message</i>	
----------------	--

5.1.3.14 dealWithExternalMessage()

```
void ArbiterAVB::dealWithExternalMessage (
    cMessage * message ) [private]
```

deal with incoming external Messages like Control Messages or Ethernet Frames

Parameters

<i>message</i>	incomming external message
----------------	----------------------------

5.1.3.15 dealWithInternalMessage()

```
void ArbiterAVB::dealWithInternalMessage (
    cMessage * message ) [private]
```

Deals with internal messages and process timer timeouts. Timer trigger start of transmission to next router

Parameters

<i>message</i>	incoming message
----------------	------------------

5.1.3.16 debugStatus()

```
void ArbiterAVB::debugStatus ( ) [private]
```

shows the credit values and current status of priorities

5.1.3.17 debugStatusInitialization()

```
void ArbiterAVB::debugStatusInitialization ( ) [private]
```

shows the status of Arbiter initializations class

5.1.3.18 handleMessage()

```
void ArbiterAVB::handleMessage (
    cMessage * message )
```

Classification of incoming messages into internal and external sources

5.1.3.19 initialize()

```
void ArbiterAVB::initialize ( )
```

Initialize Arbiter and load informations form Parent Start calculation of transmission speed

5.1.3.20 initializeAVBClassInformation()

```
void ArbiterAVB::initializeAVBClassInformation ( ) [private]
```

Calculate Transmission seed of channel SafeGuard prevent for choose nullpointer as channel

5.1.3.21 loadAVBClassInformationFromFile()

```
void ArbiterAVB::loadAVBClassInformationFromFile ( ) [private]
```

load Class information from AVB configuration file

5.1.3.22 prepareBufferStatusList()

```
std::vector< ethernetHeaderInformation * > ArbiterAVB::prepareBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

Arbiter which choose out of the Buffer Status List the current winner for next transmission

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.1.3.23 prepareClassInformations()

```
void ArbiterAVB::prepareClassInformations (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

checks if new Frames arrived in BufferModule

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.1.3.24 sendBufferAuthorizedBuffer()

```
void ArbiterAVB::sendBufferAuthorizedBuffer (
    int authorizedBuffer ) [private]
```

Send Buffer the result of arbitration, send the authorized Buffer to transmit next frame

Parameters

<i>authorizedBuffer</i>	
-------------------------	--

5.1.3.25 sendBufferRequest()

```
void ArbiterAVB::sendBufferRequest ( ) [private]
```

Send a Request for Buffer to answer the current Buffer Status

5.1.3.26 sendFrameOut()

```
void ArbiterAVB::sendFrameOut ( ) [private]
```

send Frame out after Transmission time Delay is past

5.1.3.27 setRechargeTime()

```
void ArbiterAVB::setRechargeTime ( ) [private]
```

start recharge calculation of each priority with negative credit value

5.1.3.28 settriggerCreditRecharged()

```
void ArbiterAVB::settriggerCreditRecharged (
    simtime_t calculatedRechargeTime,
    int priority ) [private]
```

set the trigger for credit recharge of one priority

Parameters

<i>calculatedRechargeTime</i> , <i>priority</i>	
---	--

5.1.3.29 settriggerDelayEnd()

```
void ArbiterAVB::settriggerDelayEnd (
    simtime_t transmissionPathDelay ) [private]
```

set the trigger for sending frame out

Parameters

<i>transmissionPathDelay</i>	
------------------------------	--

5.1.3.30 updateClassInformationsStatus()

```
void ArbiterAVB::updateClassInformationsStatus (
    int authorizedBuffer ) [private]
```

update class information status of each priority

Parameters

<i>authorizedBuffer</i>	
-------------------------	--

5.1.4 Member Data Documentation

5.1.4.1 arbiterIsBusy

```
bool ArbiterAVB::arbiterIsBusy [private]
```

5.1.4.2 avbTableVector

```
configVector ArbiterAVB::avbTableVector [private]
```

5.1.4.3 channelTransmissionRate

```
double ArbiterAVB::channelTransmissionRate [private]
```

5.1.4.4 classInformations

```
std::vector<arbiterAVBClass*> ArbiterAVB::classInformations [private]
```

5.1.4.5 config

```
csvVector ArbiterAVB::config [private]
```

5.1.4.6 configFile

```
std::string ArbiterAVB::configFile [private]
```

5.1.4.7 creditValueStatisticStreamA

```
cOutVector ArbiterAVB::creditValueStatisticStreamA [private]
```

5.1.4.8 creditValueStatisticStreamB

```
cOutVector ArbiterAVB::creditValueStatisticStreamB [private]
```

5.1.4.9 currentBufferStatusList

```
std::vector<ethernetHeaderInformation*> ArbiterAVB::currentBufferStatusList [private]
```

5.1.4.10 egressPortIndex

```
int ArbiterAVB::egressPortIndex [private]
```

5.1.4.11 ethernetFrameWaitForSend

```
EthernetFrame* ArbiterAVB::ethernetFrameWaitForSend [private]
```

5.1.4.12 macAddress

```
int ArbiterAVB::macAddress [private]
```

5.1.4.13 numberPriorities

```
int ArbiterAVB::numberPriorities [private]
```

5.1.4.14 parentChannelOut

```
cChannel* ArbiterAVB::parentChannelOut [private]
```

5.1.4.15 parentGateOut

```
cGate* ArbiterAVB::parentGateOut [private]
```

5.1.4.16 parentModul

```
cModule* ArbiterAVB::parentModul [private]
```

5.1.4.17 triggerSelf

```
cMessage* ArbiterAVB::triggerSelf = new cMessage() [private]
```

5.2 arbiterAVBClass Struct Reference

```
#include <ArbiterAVB.h>
```

Public Attributes

- int [classCreditValue](#)
- bool [classFrameWaiting](#)
- bool [classIsShaped](#)
- int [classStatus](#)
- double [sendSlope](#)
- double [idleSlope](#)
- simtime_t [lastArbitrationTimeStamp](#)
- bool [timerIsScheduled](#)

5.2.1 Detailed Description

General Struct for Information about Arbiter Classes

5.2.2 Member Data Documentation

5.2.2.1 classCreditValue

```
int arbiterAVBClass::classCreditValue
```

5.2.2.2 classFrameWaiting

```
bool arbiterAVBClass::classFrameWaiting
```

5.2.2.3 classIsShaped

```
bool arbiterAVBClass::classIsShaped
```

5.2.2.4 classStatus

```
int arbiterAVBClass::classStatus
```

5.2.2.5 idleSlope

```
double arbiterAVBClass::idleSlope
```

5.2.2.6 lastArbitrationTimeStamp

```
simtime_t arbiterAVBClass::lastArbitrationTimeStamp
```

5.2.2.7 sendSlope

```
double arbiterAVBClass::sendSlope
```

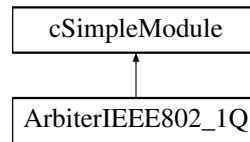
5.2.2.8 timerIsScheduled

```
bool arbiterAVBClass::timerIsScheduled
```


5.3 ArbiterIEEE802_1Q Class Reference

```
#include <ArbiterIEEE802_1Q.h>
```

Inheritance diagram for ArbiterIEEE802_1Q:



Public Member Functions

- [ArbiterIEEE802_1Q](#) ()
- virtual [~ArbiterIEEE802_1Q](#) ()
- void [initialize](#) ()
- void [handleMessage](#) (cMessage *message)

Private Member Functions

- void [dealWithInternalMessage](#) (cMessage *message)
- void [dealWithExternalMessage](#) (cMessage *message)
- void [dealWithEthernetFrame](#) (cMessage *frame)
- void [dealWithControlMessage](#) (cMessage *controlMessage)
- void [controlArbitrationBufferStatusList](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- std::vector< [ethernetHeaderInformation](#) * > [prepareBufferStatusList](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- int [arbitrateBufferStatusList](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- void [sendBufferAuthorizedBuffer](#) (int authorizedBuffer)
- void [sendBufferRequest](#) ()
- simtime_t [calculateTransmissionDelay](#) (EthernetFrame *ethernetFrame)
- void [calculateTransmissionSpeed](#) ()
- void [settriggerDelayEnd](#) (simtime_t transmissionPathDelay)
- void [sendFrameOut](#) ()

Private Attributes

- std::vector< [ethernetHeaderInformation](#) * > [currentBufferStatusList](#)
- int [numberPriorities](#)
- cModule * [parentModul](#)
- cGate * [parentGateOut](#)
- cChannel * [parentChannelOut](#)
- double [channelTransmissionRate](#)
- bool [arbiterIsBusy](#)
- EthernetFrame * [ethernetFrameWaitForSend](#)
- cMessage * [triggerSelf](#) = new cMessage()

5.3.1 Detailed Description

[ArbiterIEEE802_1Q](#) is an arbiter which arbitrate the stored frames in [BuffersModule](#) Based on the static priority non preemption algorithm frames are selected Each frame will be stored for the transmission delay inside the arbiter and after the time it is send out

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ArbiterIEEE802_1Q()

```
ArbiterIEEE802_1Q::ArbiterIEEE802_1Q ( )
```

Constructor ArbiterIEEE802.1Q, nothing to do here

5.3.2.2 ~ArbiterIEEE802_1Q()

```
ArbiterIEEE802_1Q::~~ArbiterIEEE802_1Q ( ) [virtual]
```

Destructor ArbiterIEEE802.1Q, nothing to do here

5.3.3 Member Function Documentation

5.3.3.1 arbitrateBufferStatusList()

```
int ArbiterIEEE802_1Q::arbitrateBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

Arbiter which choose the next winning Frame for next transmission out of the Buffer Status List

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.3.3.2 calculateTransmissionDelay()

```
simtime_t ArbiterIEEE802_1Q::calculateTransmissionDelay (
    EthernetFrame * ethernetFrame ) [private]
```

Calculate based on Ethernet frame length transmission delay

Parameters

<i>ethernetFrame</i>	
----------------------	--

Returns

simtime_t

5.3.3.3 calculateTransmissionSpeed()

```
void ArbiterIEEE802_1Q::calculateTransmissionSpeed ( ) [private]
```

Calculate Transmission seed of channel SafeGuard prevent for choose nullpointer as channel

5.3.3.4 controlArbitrationBufferStatusList()

```
void ArbiterIEEE802_1Q::controlArbitrationBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

control the arbitration result and initialized recharging if nessecary

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.3.3.5 dealWithControlMessage()

```
void ArbiterIEEE802_1Q::dealWithControlMessage (
    cMessage * controlMessage ) [private]
```

Process incoming Control Messages Check what kind of Control Message is received. Buffer Level Update or Buffer Level Response. The Buffer Level Updates are processed if Arbiter is not busy

Parameters

<i>controlMessage</i>	
-----------------------	--

5.3.3.6 dealWithEthernetFrame()

```
void ArbiterIEEE802_1Q::dealWithEthernetFrame (
    cMessage * frame ) [private]
```

Process incoming Ethernet frames and store them. Only one Frame is allowed. Start transmission delay process chain

Parameters

<i>frame</i>	
--------------	--

5.3.3.7 dealWithExternalMessage()

```
void ArbiterIEEE802_1Q::dealWithExternalMessage (
    cMessage * message ) [private]
```

Deals with external incoming messages and check if messages was external Ethernet Frame or Control Buffer Message

Parameters

<i>message</i>	incoming external message
----------------	---------------------------

5.3.3.8 dealWithInternalMessage()

```
void ArbiterIEEE802_1Q::dealWithInternalMessage (
    cMessage * message ) [private]
```

Deals with internal messages and process timer timeouts. Timer trigger start of transmission to next router

Parameters

<i>message</i>	incoming message
----------------	------------------

5.3.3.9 handleMessage()

```
void ArbiterIEEE802_1Q::handleMessage (
    cMessage * message )
```

Classification of incoming messages into internal and external sources

5.3.3.10 initialize()

```
void ArbiterIEEE802_1Q::initialize ( )
```

Initialize Arbiter and load informations form Parent Start calculation of transmission speed

5.3.3.11 prepareBufferStatusList()

```
std::vector< ethernetHeaderInformation * > ArbiterIEEE802_1Q::prepareBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

prepare the ready value of BufferStatusList caused by frame length if length = 0 frame is not ready

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.3.3.12 sendBufferAuthorizedBuffer()

```
void ArbiterIEEE802_1Q::sendBufferAuthorizedBuffer (
    int authorizedBuffer ) [private]
```

Send Buffer the result of arbitration, send the authorized Buffer to transmit next frame

Parameters

<i>authorizedBuffer</i>	
-------------------------	--

5.3.3.13 sendBufferRequest()

```
void ArbiterIEEE802_1Q::sendBufferRequest ( ) [private]
```

Send a request to Buffer to check the current Buffer status fill level

5.3.3.14 sendFrameOut()

```
void ArbiterIEEE802_1Q::sendFrameOut ( ) [private]
```

send Frame out after Transmission time Delay is past

5.3.3.15 settriggerDelayEnd()

```
void ArbiterIEEE802_1Q::settriggerDelayEnd (
    simtime_t transmissionPathDelay ) [private]
```

set the trigger for sending frame out

Parameters

<i>transmissionPathDelay</i>	
------------------------------	--

5.3.4 Member Data Documentation

5.3.4.1 arbiterIsBusy

```
bool ArbiterIEEE802_1Q::arbiterIsBusy [private]
```

5.3.4.2 channelTransmissionRate

```
double ArbiterIEEE802_1Q::channelTransmissionRate [private]
```

5.3.4.3 currentBufferStatusList

```
std::vector<ethernetHeaderInformation*> ArbiterIEEE802_1Q::currentBufferStatusList [private]
```

5.3.4.4 ethernetFrameWaitForSend

```
EthernetFrame* ArbiterIEEE802_1Q::ethernetFrameWaitForSend [private]
```

5.3.4.5 numberPriorities

```
int ArbiterIEEE802_1Q::numberPriorities [private]
```

5.3.4.6 parentChannelOut

```
cChannel* ArbiterIEEE802_1Q::parentChannelOut [private]
```

5.3.4.7 parentGateOut

```
cGate* ArbiterIEEE802_1Q::parentGateOut [private]
```

5.3.4.8 parentModul

```
cModule* ArbiterIEEE802_1Q::parentModul [private]
```

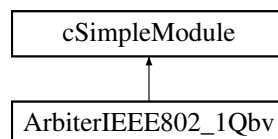
5.3.4.9 triggerSelf

```
cMessage* ArbiterIEEE802_1Q::triggerSelf = new cMessage() [private]
```

5.4 ArbiterIEEE802_1Qbv Class Reference

```
#include <ArbiterIEEE802_1Qbv.h>
```

Inheritance diagram for ArbiterIEEE802_1Qbv:



Public Member Functions

- [ArbiterIEEE802_1Qbv](#) ()
- virtual [~ArbiterIEEE802_1Qbv](#) ()
- virtual void [initialize](#) (void) override
- virtual void [handleMessage](#) (cMessage *message) override

Private Member Functions

- `simtime_t` [getPeriodDuration](#) ()
- `simtime_t` [getCriticalIntervalDuration](#) ()
- `simtime_t` [convertDoubleIntoTime](#) (double valueToBeConverted)
- `simtime_t` [calculateGuardBandDurationFromPayloadSize](#) (int payload)
- `long` [getPortSpeed](#) ()
- `int` [setEthPacketSizeFromPayload](#) (int payload)
- `std::vector< int >` [getCriticalClasses](#) ()
- `std::vector< bool >` [setUpVectorCriticalPriorities](#) (`std::vector< int >` criticalPriorities)
- `simtime_t` [getTimeOffset](#) ()
- `simtime_t` [calculateTimeModulo](#) (`simtime_t` numeratorTime, `simtime_t` denominatorTime)
- `int` [getInitialStateAndCalculateTriggerTimer](#) (`simtime_t` offset)
- `int` [getNextState](#) (int currentState)
- `void` [initializeTimerMessage](#) ()
- `void` [printInitializationConfiguration](#) ()
- `void` [dealWithInternalMessage](#) (`cMessage *message`)
- `void` [dealWithSegmentTimerMessage](#) ()
- `simtime_t` [getSegmentDurationFromCurrentState](#) (int state)
- `void` [dealWithTriggerSelf](#) ()
- `void` [sendFrameOut](#) ()
- `void` [sendBufferRequest](#) ()
- `void` [dealWithExternalMessage](#) (`cMessage *message`)
- `void` [dealWithControlMessage](#) (`cMessage *controlMessage`)
- `void` [controlArbitrationBufferStatusList](#) (`std::vector< ethernetHeaderInformation * >` bufferStatusList)
- `std::vector< ethernetHeaderInformation * >` [prepareBufferStatusList](#) (`std::vector< ethernetHeaderInformation * >` bufferStatusList)
- `int` [arbitrateBufferStatusList](#) (`std::vector< ethernetHeaderInformation * >` bufferStatusList)
- `void` [sendBufferAuthorizedBuffer](#) (int authorizedBuffer)
- `void` [dealWithEthernetFrame](#) (`cMessage *frame`)
- `simtime_t` [calculateTransmissionDelay](#) (`EthernetFrame *ethernetFrame`)
- `void` [setTriggerDelayEnd](#) (`simtime_t` transmissionPathDelay)

Private Attributes

- `int` [currentState](#)
the state is equal the current time segment
- `int` [nextState](#)
the next state is equal to the next time segment after the current
- `int` [maximumCriticalPayloadSize](#)
set by user -> defines the maximum payload number in bytes for critical priorities frames
- `int` [maximumNoncriticalPayloadSize](#)
set by user -> defines the maximum payload number in bytes for noncritical priorities frames
- `int` [numberSupportedPriorities](#)
number of supported priorities (= default 8 priorities)
- `int` [switchMAC](#)
MAC address of the switch.
- `int` [portIndex](#)
index of the [EgressPort](#) via which the frame is forwarded
- `simtime_t` [periodDuration](#)
set by user -> defines the duration of the period (= default 5 ms)
- `simtime_t` [criticalIntervalDuration](#)

- set by user -> defines the duration of the critical interval*
- simtime_t [criticalSendingSegmentDuration](#)
critical interval is split into sending and guard band segment
- simtime_t [criticalGuardBandSegmentDuration](#)
guard band segment is as long as a critical frame needs for transmission
- simtime_t [noncriticalIntervalDuration](#)
non critical interval duration is the time remaining from the subtraction of period and criticalInterval
- simtime_t [noncriticalSendingSegmentDuration](#)
non critical interval duration is split into sending and guard band segment
- simtime_t [noncriticalGuardBandSegmentDuration](#)
non critical guard band segment is as long as a non critical frame needs for transmission
- std::vector< int > [criticalPriorityClasses](#)
integer vector which stores the critical priorities (= default ("7 6"))
- simtime_t [timeOffset](#)
timer, which allows to start the simulation in any time segment (= default 0 -> critical timing segment)
- simtime_t [triggerTimer](#)
timer to trigger a a self message
- cMessage * [segmentTriggerTimerMessage](#)
cMessage timer for changing states (time segments)
- cMessage * [triggerSelf](#) = new cMessage()
cMessage timer for trigger Ethernet frame transmission
- EthernetFrame * [ethernetFrameWaitForTransmission](#)
arbiter stores only one Ethernet frame which must wait for its transmission to simulate transmission rate
- std::vector< [ethernetHeaderInformation](#) * > [currentBufferStatusList](#)
- std::vector< bool > [isCritical](#)
boolean vector indicating which of the priorities is critical (= true) or not (= false)
- bool [arbiterIsBusy](#)
variable that indicates that the arbitration is already in progress
- bool [frameStoredInBuffer](#)

5.4.1 Detailed Description

[ArbiterIEEE802_1Qbv](#) is an [Arbiter\(\)](#) which arbitrate the stored Ethernet frames in [BuffersModule](#). It is also known as Time Aware [Shaper](#) (TAS).

TAS is a circuit-switched arbitration. The period is divided into two intervals

1. critical interval
2. noncritical interval Each interval is also divided into the following segments: *.1 sending segment *.2 guard band segment Only Ethernet frames with critical priorities are sent in the critical interval of the sending segment. In IEEE802_1Qbv, traffic classes with priorities 6 and 7 are classified as critical (set as default). For simulation purpose it is also possible to set other classes as critical as well. The guard band segment is only as long for an Ethernet frame to complete its transmission. No Ethernet frames start their transmission in this segment. In noncritical sending segment only non critical priority classes are sent (default priority 5-0).

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ArbiterIEEE802_1Qbv()

```
ArbiterIEEE802_1Qbv::ArbiterIEEE802_1Qbv ( )
```

Class constructor - nothing to do here

5.4.2.2 ~ArbiterIEEE802_1Qbv()

```
ArbiterIEEE802_1Qbv::~~ArbiterIEEE802_1Qbv ( ) [virtual]
```

Class destructor

Cancel and delete the self messages triggerSelf and segmentTriggerTimerMessage.

5.4.3 Member Function Documentation

5.4.3.1 arbitrateBufferStatusList()

```
int ArbiterIEEE802_1Qbv::arbitrateBufferStatusList (
    std::vector< ethernetHeaderInformation * > bufferStatusList ) [private]
```

Arbitrate the [ethernetHeaderInformation](#) vector.

After the preparation of the [ethernetHeaderInformation](#) vector an Ethernet frame is selected for transmission. It is first checked at which state (time segment) the simulation is. Critical sending segment -> only Ethernet frames with critical priorities are transmit. Critical guard band segment -> no Ethernet frame are transmit. Noncritical sending segment -> only noncritical Ethernet frames (best effort) are transmit. If several messages of different noncritical or critical priorities are in the FrameBuffers, the selection is made according to static priority non preemptive.

Parameters

<i>bufferStautsList</i>	a vector containing pointer ethernetHeaderInformation with set boolean variable ready
-------------------------	---

Returns

integer value of the priority which specifies the [FrameBuffer](#) from which the first Ethernet frame is forwarded

5.4.3.2 calculateGuardBandDurationFromPayloadSize()

```
simtime_t ArbiterIEEE802_1Qbv::calculateGuardBandDurationFromPayloadSize (
    int payload ) [private]
```

Calculate the duration of the guard band depending on the payload size of an Ethernet frame.

The period is divided into critical and noncritical interval. Each interval itself is also divided into sending and guard band segments. The duration of the guard band segment depends on the maximum payload size of the Ethernet frame. The guard band is so long that one Ethernet frame can complete its transmission. But: no frame starts transmission in the guard band!

Parameters

<i>payload</i>	integer of the Ethernet frame payload size
----------------	--

Returns

the duration of the guard band

5.4.3.3 calculateTimeModulo()

```
simtime_t ArbiterIEEE802_1Qbv::calculateTimeModulo (
    simtime_t numeratorTime,
    simtime_t denominatorTime ) [private]
```

Calculates the time remaining after dividing two times.

This is required for setting the initial time segment with which the arbitration. It is not possible to divide `simtime_t` values and get the remainder. Therefore the denominator is subtracted from numerator until denominator is larger than numerator. The rest is returned.

Parameters

<i>numeratorTime</i>	<code>simtime_t</code> which is to divided
<i>denominatorTime</i>	<code>simtime_t</code> to be divided with

Returns

the rest of the division

5.4.3.4 calculateTransmissionDelay()

```
simtime_t ArbiterIEEE802_1Qbv::calculateTransmissionDelay (
    EthernetFrame * ethernetFrame ) [private]
```

Calculate the transmission time of the Ethernet frame.

To set up a timer message that simulates the transmission time of the Ethernet frame it must first be calculated. This requires the Ethernet frame payload length and the port speed. (Port speed is obtained from [getPortSpeed\(\)](#)).

Parameters

<i>ethernetFrame</i>	pointer of an EthernetFrame message
----------------------	-------------------------------------

Returns

the transmission time of the Ethernet frame

5.4.3.5 controlArbitrationBufferStatusList()

```
void ArbiterIEEE802_1Qbv::controlArbitrationBufferStatusList (
    std::vector< ethernetHeaderInformation * > bufferStatusList ) [private]
```

Control the arbitration of the received [ethernetHeaderInformation](#) vector from the BufferControlMessage.

With the received [ethernetHeaderInformation](#) vector the arbitration phase starts. The [ethernetHeaderInformation](#) vector contains for each [FrameBuffer](#) in [BuffersModule](#) the payload of the first Ethernet frame in the [FrameBuffer](#) as well as the priority of the the corresponding buffer. The arbitration phase is divided in two parts: prepareBuffer↔StatusList(...) and arbitrateBufferStatusList

Parameters

<i>bufferStatusList</i>	a vector containing pointer ethernetHeaderInformation
-------------------------	---

5.4.3.6 convertDoubleIntoTime()

```
simtime_t ArbiterIEEE802_1Qbv::convertDoubleIntoTime (
    double valueToBeConverted ) [private]
```

Convert a double value into simtime_t

Time values are entered as double values. Double has to be converted into time values to work with simtime_t.

Parameters

<i>valueToBeConverted</i>	a double value which has to be converted
---------------------------	--

Returns

converted simtime_t value

5.4.3.7 dealWithControlMessage()

```
void ArbiterIEEE802_1Qbv::dealWithControlMessage (
    cMessage * controlMessage ) [private]
```

Deal with the external message BufferControlMessage.

Arriving external message is a `BufferControlMessage`. `BufferControlMessages` are messages to control the `BuffersModule()`. There are two different flags that the `Arbiter()` must handle. `BUFFER_CONTROL_UPDATE` is a update message. It always arrives if a Ethernet frame with new priority is stored into a `FrameBuffer()`. `BUFFER_LEVEL_RESPONSE` is the answer of the `BUFFER_LEVEL_REQUEST`. First the `Arbiter` sends a request to the `BuffersModule()` to get the fill level of the `FrameBuffers()`. Than the `BuffersModule` answers with the `BUFFER_LEVEL_RESPONSE` containing a `ethernetHeaderInformation` vector.

Parameters

<i>controlMessage</i>	cMessage pointer to a <code>BufferControlMessage</code>
-----------------------	---

5.4.3.8 dealWithEthernetFrame()

```
void ArbiterIEEE802_1Qbv::dealWithEthernetFrame (
    cMessage * frame ) [private]
```

Deal with external message Ethernet frame.

Arriving external message is an Ethernet frame. The frame is temporarily stored in variable `ethernetFrameWaitForTransmission` until the transmission time is over. For simulation the transmission time a timer message `TriggerSelf` is scheduled.

Parameters

<i>frame</i>	cMessage pointer of an Ethernet Frame
--------------	---------------------------------------

5.4.3.9 dealWithExternalMessage()

```
void ArbiterIEEE802_1Qbv::dealWithExternalMessage (
    cMessage * message ) [private]
```

Deal with external message (Ethernet frame and `BufferControlMessage`)

The arbiter has to deal with internal (self) messages and external (`BufferControlMessage` and Ethernet frame) messages. `BufferControlMessages` are messages to control the `BuffersModule()`. The arriving Ethernet frame is the message that was selected by the arbitration and is now forwarded.

Parameters

<i>message</i>	cMessage pointer to arriving external message
----------------	---

5.4.3.10 dealWithInternalMessage()

```
void ArbiterIEEE802_1Qbv::dealWithInternalMessage (
```

```
cMessage * message ) [private]
```

Deal with internal messages (self messages).

The arbiter has to deal with internal (self) messages and external (bufferControl and Ethernet frame) messages. The internal messages simulates timer functions. Two internal messages must be distinguished. The segmentTrigger↔TimerMessage triggers always a self message after changing the state. The triggerSelf self message simulates the transmission of an Ethernet frame. It is triggered when the transmission time has expired and the Ethernet frame is transmit.

Parameters

<i>message</i>	pointer to a self message, segmentTriggerTimerMessage or triggerSelf
----------------	--

5.4.3.11 dealWithSegmentTimerMessage()

```
void ArbiterIEEE802_1Qbv::dealWithSegmentTimerMessage ( ) [private]
```

Deal with internal message segmentTriggerTimerMessage.

A segmentTriggerTimerMessage is arrived. It indicates the finishing of a time segment. The current state is updated as well as the next state. The segmentTriggerTimerMessage is updated and scheduled.

5.4.3.12 dealWithTriggerSelf()

```
void ArbiterIEEE802_1Qbv::dealWithTriggerSelf ( ) [private]
```

Deal with internal message triggerSelf.

A triggerSelf message is arrived. It simulates the transmission time of the Ethernet frame. Its arriving indicates that the transmission time of the stored Ethernet Frame is finished.

5.4.3.13 getCriticalClasses()

```
std::vector< int > ArbiterIEEE802_1Qbv::getCriticalClasses ( ) [private]
```

Get the critical priority classes.

To transmit Ethernet frames in critical sending segment critical class priorities has to be set. Per default 7 and 6 are critical priorities. They are set as string in .ini file. They are converted as integer and stored into a integer vector.

Returns

an integer vector containing the critical priorities

5.4.3.14 getCriticalIntervalDuration()

```
simtime_t ArbiterIEEE802_1Qbv::getCriticalIntervalDuration ( ) [private]
```

Get the duration of the critical interval

The period is divided into critical and noncritical interval. The duration of the critical interval has to be set by the user. Per default it is set to 3 ms.

Returns

the duration of the critical interval

5.4.3.15 getInitialStateAndCalculateTriggerTimer()

```
int ArbiterIEEE802_1Qbv::getInitialStateAndCalculateTriggerTimer (
    simtime_t offset ) [private]
```

Get the state (time segment) with which the arbitration starts.

Depending on the time offset the initial state is set. If no time offset state is set (=0) the initial state is always the critical sending segment.

Returns

integer value of the initial state.

5.4.3.16 getNextState()

```
int ArbiterIEEE802_1Qbv::getNextState (
    int currentState ) [private]
```

Get the next state of the current state.

There are four states. The state transitions are fixed. From Critical sending to critical guard band to noncritical sending to noncritical guard band back to critical sending. For a correct arbitration, the next state must be set from the current state.

Parameters

<i>currentState</i>	integer value of the current state (time segment)
---------------------	---

Returns

integer value of the next state (time segment)

5.4.3.17 getPeriodDuration()

```
simtime_t ArbiterIEEE802_1Qbv::getPeriodDuration ( ) [private]
```

Get the duration of the period

The period duration is set by the user in .ini file and defines the the time length of a single arbitration run. Per default it is set to 5 ms.

Returns

the duration of the period

5.4.3.18 getPortSpeed()

```
long ArbiterIEEE802_1Qbv::getPortSpeed ( ) [private]
```

Calculate the Port speed and get the value.

For calculating the duration of the guard band the port speed is required. The guard band is as long as the transmission of an Ethernet frame. The Transmission is calculated by packetSize (in bit) divided by port speed.

Returns

the port speed

5.4.3.19 getSegmentDurationFromCurrentState()

```
simtime_t ArbiterIEEE802_1Qbv::getSegmentDurationFromCurrentState (
    int state ) [private]
```

Get the duration time of the segment from the current state

To schedule the segmentTriggerTimerMessage the duration of the current state is required.

Parameters

<i>state</i>	integer value of the state from which the duration time is required
--------------	---

Returns

the time of the state (time segment)

5.4.3.20 getTimeOffset()

```
simtime_t ArbiterIEEE802_1Qbv::getTimeOffset ( ) [private]
```

Gets the time offset.

If the time offset is not set, the arbitration always starts in the critical sending segment. To start in another segment the offset is used. This can be set larger than the period but does not change the timing of the arbitration. Arbitration always starts at time 0.

Returns

the time offset

5.4.3.21 handleMessage()

```
void ArbiterIEEE802_1Qbv::handleMessage (
    cMessage * message ) [override], [virtual]
```

Handle arriving messages - internal and external messages

There are two types of messages to handle. Internal messages are self messages that simulates time triggers. TAS two time trigger are required. The first one is the segmentTriggerTimerMessage. It is always triggered when the duration of a time segment is up. The second one is triggerSelf. It simulates the transmission delay from an Ethernet frame. There are also two types of external messages to handle. An arriving Ethernet frame is temporarily stored until its transmission delay is up. Then it is transmit to the next module. A BufferControlMessage contains information about the fill level of the FrameBuffers() in BufferModule().

5.4.3.22 initialize()

```
void ArbiterIEEE802_1Qbv::initialize (
    void ) [override], [virtual]
```

Initialize the time aware shaper (TAS).

For the initialization: Get the period and critical interval duration. Get the maximum payload size of critical and noncritical Ethernet frames. Calculate the guard band and sending segments. Set up the segmentTriggerTimer↵ Message to simulate the time states of TAS. Get the current state and the next state of TAS.

5.4.3.23 initializeTimerMessage()

```
void ArbiterIEEE802_1Qbv::initializeTimerMessage ( ) [private]
```

Initialize the timer message segmentTriggerTimerMessage to control the Arbitration.

To change the state under correct timing conditions a self message segmentTriggerTimerMessage is initialized. It is always sent, if the duration of a segment is finished.

5.4.3.24 prepareBufferStatusList()

```
std::vector< ethernetHeaderInformation * > ArbiterIEEE802_1Qbv::prepareBufferStatusList (
    std::vector< ethernetHeaderInformation * > bufferStatusList ) [private]
```

Preparation of the ethernetHeaderInformation vector for arbitration.

EthernetHeaderInformation is a struct containing the priority of a FrameBuffer() and the payload size of the first Ethernet frame in a FrameBuffer(). If no Ethernet frame is stored in a FrameBuffer() it is indicated with a payload size 0. If an Ethernet frame is in the buffer (payload size > 0), the boolean variable is ready from EthernetHeader↵ Information is set true.

Parameters

<i>bufferStatusList</i>	a vector containing pointer ethernetHeaderInformation
-------------------------	---

Returns

a vector containing pointer [ethernetHeaderInformation](#) with set boolean variable ready

5.4.3.25 printInitializationConfiguration()

```
void ArbiterIEEE802_1Qbv::printInitializationConfiguration ( ) [private]
```

Print all initialization configurations.

Print the period duration, the critical interval duration, non critical interval duration, the critical sending and guard band segment duration as well as the noncritical sending and guard band segment duration. It is also the initial state and the current state printed.

5.4.3.26 sendBufferAuthorizedBuffer()

```
void ArbiterIEEE802_1Qbv::sendBufferAuthorizedBuffer (
    int authorizedBuffer ) [private]
```

Send a authorization BufferControlMessage to [BuffersModule](#).

By arbitration the priority of the [FrameBuffer](#) is selected that transmit its Ethernet frame. This is told to the [BuffersModule\(\)](#) via BufferControlMessage with the flag SEND_AUTHORISATION.

Parameters

<i>authorizedBuffer</i>	integer value of the FrameBuffers priority
-------------------------	--

5.4.3.27 sendBufferRequest()

```
void ArbiterIEEE802_1Qbv::sendBufferRequest ( ) [private]
```

Send a request to the buffer to get information about stored Ethernet frames in buffer.

After finishing the transmission of the waiting Ethernet frame a new Ethernet frame can start its transmission. To get the information about stored Ethernet frames in the buffer a BufferControlMessage with the flag BUFFER_CONTROL_REQUEST is transmit to the [BuffersModule\(\)](#).

5.4.3.28 sendFrameOut()

```
void ArbiterIEEE802_1Qbv::sendFrameOut ( ) [private]
```

Transmit the Ethernet frames that waits for its transmission

A triggerSelf message is arrived. It indicates that the transmission time of the Ethernet frame is finished. Now the frame is sent.

5.4.3.29 setEthPacketSizeFromPayload()

```
int ArbiterIEEE802_1Qbv::setEthPacketSizeFromPayload (
    int payload ) [private]
```

Set the packet size of an Ethernet frame.

From the payload the packet size of an Ethernet frame is set. The frame overhead (22 byte), packet overhead (8 byte) and interpacket gap (12 byte) are added to the payload. It is required for calculating the transmission duration of an Ethernet frame.

Returns

the packet size of an Ethernet frame in byte depending on the payload

5.4.3.30 setTriggerDelayEnd()

```
void ArbiterIEEE802_1Qbv::setTriggerDelayEnd (
    simtime_t transmissionPathDelay ) [private]
```

Set up timer message TriggerSelf

To simulate the transmission time of the Ethernet frame an self message TriggerSelf is set up. It is scheduledAt the above calculated transmission time.

Parameters

<i>transmissionPathDelay</i>	the transmission time of the Ethernet frame from one module to the next hop
------------------------------	---

5.4.3.31 setUpVectorCriticalPriorities()

```
std::vector< bool > ArbiterIEEE802_1Qbv::setUpVectorCriticalPriorities (
    std::vector< int > criticalPriorities ) [private]
```

Set up a boolean vector for all supported priorities to indicate their criticality.

For arbitration it is essential to know which priority is critical and which is not. Therefore a boolean vector is implemented. It contains for each priority an entry. True indicates that the priority is critical and false that it is noncritical.

Parameters

<i>criticalPriorities</i>	integer vector containing the critical priorities
---------------------------	---

Returns

boolean vector indicating which of the priorities is critical (= true) or not (= false)

5.4.4 Member Data Documentation**5.4.4.1 arbiterIsBusy**

```
bool ArbiterIEEE802_1Qbv::arbiterIsBusy [private]
```

variable that indicates that the arbitration is already in progress

5.4.4.2 criticalGuardBandSegmentDuration

```
simtime_t ArbiterIEEE802_1Qbv::criticalGuardBandSegmentDuration [private]
```

guard band segment is as long as a critical frame needs for transmission

5.4.4.3 criticalIntervalDuration

```
simtime_t ArbiterIEEE802_1Qbv::criticalIntervalDuration [private]
```

set by user -> defines the duration of the critical interval

5.4.4.4 criticalPriorityClasses

```
std::vector<int> ArbiterIEEE802_1Qbv::criticalPriorityClasses [private]
```

integer vector which stores the critical priorities (= default ("7 6"))

5.4.4.5 criticalSendingSegmentDuration

```
simtime_t ArbiterIEEE802_1Qbv::criticalSendingSegmentDuration [private]
```

critical interval is split into sending and guard band segment

5.4.4.6 currentBufferStatusList

```
std::vector<ethernetHeaderInformation*> ArbiterIEEE802_1Qbv::currentBufferStatusList [private]
```

5.4.4.7 currentState

```
int ArbiterIEEE802_1Qbv::currentState [private]
```

the state is equal the current time segment

5.4.4.8 ethernetFrameWaitForTransmission

```
EthernetFrame* ArbiterIEEE802_1Qbv::ethernetFrameWaitForTransmission [private]
```

arbiter stores only one Ethernet frame which must wait for its transmission to simulate transmission rate

5.4.4.9 frameStoredInBuffer

```
bool ArbiterIEEE802_1Qbv::frameStoredInBuffer [private]
```

5.4.4.10 isCritical

```
std::vector<bool> ArbiterIEEE802_1Qbv::isCritical [private]
```

boolean vector indicating which of the priorities is critical (= true) or not (= false)

5.4.4.11 maximumCriticalPayloadSize

```
int ArbiterIEEE802_1Qbv::maximumCriticalPayloadSize [private]
```

set by user -> defines the maximum payload number in bytes for critical priorities frames

5.4.4.12 maximumNoncriticalPayloadSize

```
int ArbiterIEEE802_1Qbv::maximumNoncriticalPayloadSize [private]
```

set by user -> defines the maximum payload number in bytes for noncritical priorities frames

5.4.4.13 nextState

```
int ArbiterIEEE802_1Qbv::nextState [private]
```

the next state is equal to the next time segment after the current

5.4.4.14 noncriticalGuardBandSegmentDuration

```
simtime_t ArbiterIEEE802_1Qbv::noncriticalGuardBandSegmentDuration [private]
```

non critical guard band segment is as long as a non critical frame needs for transmission

5.4.4.15 noncriticalIntervalDuration

```
simtime_t ArbiterIEEE802_1Qbv::noncriticalIntervalDuration [private]
```

non critical interval duration is the time remaining from the subtraction of period and criticalInterval

5.4.4.16 noncriticalSendingSegmentDuration

```
simtime_t ArbiterIEEE802_1Qbv::noncriticalSendingSegmentDuration [private]
```

non critical interval duration is split into sending and guard band segment

5.4.4.17 numberSupportedPriorities

```
int ArbiterIEEE802_1Qbv::numberSupportedPriorities [private]
```

number of supported priorities (= default 8 priorities)

5.4.4.18 periodDuration

```
simtime_t ArbiterIEEE802_1Qbv::periodDuration [private]
```

set by user -> defines the duration of the period (= default 5 ms)

5.4.4.19 portIndex

```
int ArbiterIEEE802_1Qbv::portIndex [private]
```

index of the [EgressPort](#) via which the frame is forwarded

5.4.4.20 segmentTriggerTimerMessage

```
cMessage* ArbiterIEEE802_1Qbv::segmentTriggerTimerMessage [private]
```

cMessage timer for changing states (time segments)

5.4.4.21 switchMAC

```
int ArbiterIEEE802_1Qbv::switchMAC [private]
```

MAC address of the switch.

5.4.4.22 timeOffset

```
simtime_t ArbiterIEEE802_1Qbv::timeOffset [private]
```

timer, which allows to start the simulation in any time segment (= default 0 -> critical timing segment)

5.4.4.23 triggerSelf

```
cMessage* ArbiterIEEE802_1Qbv::triggerSelf = new cMessage() [private]
```

cMessage timer for trigger Ethernet frame transmission

5.4.4.24 triggerTimer

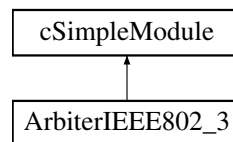
```
simtime_t ArbiterIEEE802_1Qbv::triggerTimer [private]
```

timer to trigger a self message

5.5 ArbiterIEEE802_3 Class Reference

```
#include <ArbiterIEEE802_3.h>
```

Inheritance diagram for ArbiterIEEE802_3:



Public Member Functions

- [ArbiterIEEE802_3 \(\)](#)
- virtual [~ArbiterIEEE802_3 \(\)](#)
- void [initialize \(\)](#)
- void [handleMessage](#) (cMessage *message)

Private Member Functions

- void [dealWithInternalMessage](#) (cMessage *message)
- void [dealWithExternalMessage](#) (cMessage *message)
- void [dealWithEthernetFrame](#) (cMessage *frame)
- void [dealWithControlMessage](#) (cMessage *controlMessage)
- std::vector< [ethernetHeaderInformation](#) * > [prepareBufferStatusList](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- void [controlArbitrationBufferStatusList](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- int [arbitrateBufferStatusList](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)
- void [sendBufferAuthorizedBuffer](#) (int authorizedBuffer)
- void [sendBufferRequest](#) ()
- simtime_t [calculateTransmissionDelay](#) (EthernetFrame *ethernetFrame)
- void [calculateTransmissionSpeed](#) ()
- void [settriggerDelayEnd](#) (simtime_t transmissionPathDelay)
- void [sendFrameOut](#) ()
- void [debugStatusInitialization](#) (std::vector< [ethernetHeaderInformation](#) * > BufferStatusList)

Private Attributes

- `std::vector< ethernetHeaderInformation * >` `currentBufferStatusList`
- `int` `numberPriorities`
- `cModule *` `parentModul`
- `cGate *` `parentGateOut`
- `cChannel *` `parentChannelOut`
- `double` `channelTransmissionRate`
- `bool` `arbiterIsBusy`
- `EthernetFrame *` `ethernetFrameWaitForSend`
- `cMessage *` `triggerSelf` = `new cMessage()`
- `int` `rememberLastWinner`

5.5.1 Constructor & Destructor Documentation

5.5.1.1 ArbiterIEEE802_3()

```
ArbiterIEEE802_3::ArbiterIEEE802_3 ( )
```

Constructor ArbiterIEEE802.1Q, nothing to do here

5.5.1.2 ~ArbiterIEEE802_3()

```
ArbiterIEEE802_3::~~ArbiterIEEE802_3 ( ) [virtual]
```

Destructor ArbiterIEEE802.1Q, nothing to do here

5.5.2 Member Function Documentation

5.5.2.1 arbitrateBufferStatusList()

```
int ArbiterIEEE802_3::arbitrateBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

Arbiter which choose the next winning Frame for next transmission out of the Buffer Status List

Parameters

<i>BufferStatusList</i>	is <code>std::vector<ethernetHeaderInformation*></code>
-------------------------	---

5.5.2.2 calculateTransmissionDelay()

```
simtime_t ArbiterIEEE802_3::calculateTransmissionDelay (
    EthernetFrame * ethernetFrame ) [private]
```

Calculate based on Ethernet frame length transmission delay

Parameters

<i>ethernetFrame</i>	
----------------------	--

Returns

simtime_t

5.5.2.3 calculateTransmissionSpeed()

```
void ArbiterIEEE802_3::calculateTransmissionSpeed ( ) [private]
```

Calculate Transmission seed of channel SafeGuard prevent for choose nullpointer as channel

5.5.2.4 controlArbitrationBufferStatusList()

```
void ArbiterIEEE802_3::controlArbitrationBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

This function controls the arbiter and call the steps which are necessary to prepare and arbitrate frames

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.5.2.5 dealWithControlMessage()

```
void ArbiterIEEE802_3::dealWithControlMessage (
    cMessage * controlMessage ) [private]
```

Process incoming Control Messages Check what kind of Control Message is received. Buffer Level Update or Buffer Level Response. The Buffer Level Updates are processed if Arbiter is not busy

Parameters

<i>controlMessage</i>	
-----------------------	--

5.5.2.6 dealWithEthernetFrame()

```
void ArbiterIEEE802_3::dealWithEthernetFrame (
    cMessage * frame ) [private]
```

Process incoming Ethernet frames and store them. Only one Frame is allowed. Start transmission delay process chain

Parameters

<i>frame</i>	
--------------	--

5.5.2.7 dealWithExternalMessage()

```
void ArbiterIEEE802_3::dealWithExternalMessage (
    cMessage * message ) [private]
```

Deals with external incoming messages and check if messages was external Ethernet Frame or Control Buffer Message

Parameters

<i>message</i>	incoming external message
----------------	---------------------------

5.5.2.8 dealWithInternalMessage()

```
void ArbiterIEEE802_3::dealWithInternalMessage (
    cMessage * message ) [private]
```

Deals with internal messages and process timer timeouts. Timer trigger start of transmission to next router

Parameters

<i>message</i>	incoming message
----------------	------------------

5.5.2.9 debugStatusInitialization()

```
void ArbiterIEEE802_3::debugStatusInitialization (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

5.5.2.10 handleMessage()

```
void ArbiterIEEE802_3::handleMessage (
    cMessage * message )
```

Classification of incoming messages into internal and external sources

5.5.2.11 initialize()

```
void ArbiterIEEE802_3::initialize (
    void )
```

Initialize Arbiter and load informations form Parent Start calculation of transmission speed

5.5.2.12 prepareBufferStatusList()

```
std::vector< ethernetHeaderInformation * > ArbiterIEEE802_3::prepareBufferStatusList (
    std::vector< ethernetHeaderInformation * > BufferStatusList ) [private]
```

prepare the ready value of BufferStatusList caused by frame length if length = 0 frame is not ready

Parameters

<i>BufferStatusList</i>	is std::vector<ethernetHeaderInformation*>
-------------------------	--

5.5.2.13 sendBufferAuthorizedBuffer()

```
void ArbiterIEEE802_3::sendBufferAuthorizedBuffer (
    int authorizedBuffer ) [private]
```

Send Buffer the result of arbitration, send the authorized Buffer to transmit next frame

Parameters

<i>authorizedBuffer</i>	
-------------------------	--

5.5.2.14 sendBufferRequest()

```
void ArbiterIEEE802_3::sendBufferRequest ( ) [private]
```

Send a request to Buffer to check the current Buffer status fill level

5.5.2.15 sendFrameOut()

```
void ArbiterIEEE802_3::sendFrameOut ( ) [private]
```

send Frame out after Transmission time Delay is past

5.5.2.16 settriggerDelayEnd()

```
void ArbiterIEEE802_3::settriggerDelayEnd (
    simtime_t transmissionPathDelay ) [private]
```

set the trigger for sending frame out

Parameters

<i>transmissionPathDelay</i>	
------------------------------	--

5.5.3 Member Data Documentation

5.5.3.1 arbiterIsBusy

```
bool ArbiterIEEE802_3::arbiterIsBusy [private]
```

5.5.3.2 channelTransmissionRate

```
double ArbiterIEEE802_3::channelTransmissionRate [private]
```

5.5.3.3 currentBufferStatusList

```
std::vector<ethernetHeaderInformation*> ArbiterIEEE802_3::currentBufferStatusList [private]
```

5.5.3.4 ethernetFrameWaitForSend

```
EthernetFrame* ArbiterIEEE802_3::ethernetFrameWaitForSend [private]
```

5.5.3.5 numberPriorities

```
int ArbiterIEEE802_3::numberPriorities [private]
```

5.5.3.6 parentChannelOut

```
cChannel* ArbiterIEEE802_3::parentChannelOut [private]
```

5.5.3.7 parentGateOut

```
cGate* ArbiterIEEE802_3::parentGateOut [private]
```

5.5.3.8 parentModul

```
cModule* ArbiterIEEE802_3::parentModul [private]
```

5.5.3.9 rememberLastWinner

```
int ArbiterIEEE802_3::rememberLastWinner [private]
```

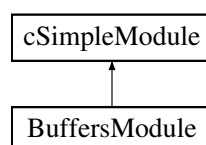
5.5.3.10 triggerSelf

```
cMessage* ArbiterIEEE802_3::triggerSelf = new cMessage() [private]
```

5.6 BuffersModule Class Reference

```
#include <BuffersModule.h>
```

Inheritance diagram for BuffersModule:



Public Member Functions

- [BuffersModule](#) ()
- virtual [~BuffersModule](#) ()
- virtual void [finish](#) () override
- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override

Private Member Functions

- void [setInitialUserInputs](#) ()
- int [getNumberSupportedPriorities](#) ()
- int [getMaximumBufferSize](#) ()
- void [setUpFrameBuffers](#) ()
- void [dealWithExternalMessage](#) (cMessage *message)
- bool [isMessageControlMessage](#) (cMessage *message)
- void [dealWithBufferControl](#) (cMessage *message)
- void [dealWithEthernetFrame](#) (cMessage *message)
- BufferControlMessage * [castMessageIntoBufferControlMessage](#) (cMessage *message)
- int [getBufferControlInstruction](#) (BufferControlMessage *controlMessage)
- void [handleBufferControlInstruction](#) (int instruction, BufferControlMessage *controlMessage)
- void [bufferControlLevelRequest](#) ()
- void [sendEthernetFrameFromBuffer](#) (BufferControlMessage *controlMessage)
- void [triggerUpdateMessageNewPriorityArrival](#) ()
- EthernetFrame * [castMessageIntoEthernetFrame](#) (cMessage *message)
- bool [isFristArrivingFrameWithPriority](#) (int priority)
- void [storeEthernetFrameInBuffer](#) (int priority, EthernetFrame *ethernetFrame)
- std::vector< [ethernetHeaderInformation](#) * > [setUpBufferInformation](#) ()
- [ethernetHeaderInformation](#) * [getFrameBufferInformation](#) (int priority)
- void [sendBufferControlMessage](#) (std::vector< [ethernetHeaderInformation](#) * > bufferInformation, message↔ InformationConstant status)
- void [deleteFramesInBuffers](#) ()
- void [deleteBuffers](#) ()

Private Attributes

- int [numberSupportedPriorities](#)
- std::vector< [FrameBuffer](#) * > [frameBuffers](#)
- int [maximumBufferSize](#)

5.6.1 Detailed Description

[BuffersModule\(\)](#) is part of the [EgressPort\(\)](#) as well as the interface [Arbiter\(\)](#) and interface [Classifier\(\)](#). The [BuffersModule\(\)](#) consists several [FrameBuffers](#). The number of [FrameBuffers\(\)](#) in [BufferModule\(\)](#) depends on the number of supported priorities. For each priority a [FrameBuffer\(\)](#) is created.

An incoming Ethernet frame is stored into the [FrameBuffer\(\)](#) with the same priority depending on its priority. When an Ethernet frame appears the first time, [BuffersModule\(\)](#) sends a control message with status [BUFFER_LEVEL↔_UPDATE](#) to the [Arbiter\(\)](#). The [Arbiter\(\)](#) then starts the arbitration and informs via control Message [SEND_AUT↔HORISATION](#) the [BuffersModule\(\)](#) which frame to be send.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 BuffersModule()

```
BuffersModule::BuffersModule ( )
```

Constructor [BuffersModule](#), nothing to do here

5.6.2.2 ~BuffersModule()

```
BuffersModule::~~BuffersModule ( ) [virtual]
```

Destructor [BuffersModule](#), delete every stored Ethernet frame and every generated [FrameBuffer](#).

5.6.3 Member Function Documentation

5.6.3.1 bufferControlLevelRequest()

```
void BuffersModule::bufferControlLevelRequest ( ) [private]
```

Response of BUFFER_CONTROL_REQUEST from Arbiter

5.6.3.2 castMessageIntoBufferControlMessage()

```
BufferControlMessage * BuffersModule::castMessageIntoBufferControlMessage (
    cMessage * message ) [private]
```

Cast cMessage into a BufferControlMessage.

Incoming external message is cast into a BufferControlMessage. It is also checked if the message exists, otherwise an error is thrown.

Parameters

<i>message</i>	Pointer to cMessage that will be cast into BufferControlMessage
----------------	---

Returns

The pointer to the cast BufferControlMessage

5.6.3.3 castMessageIntoEthernetFrame()

```
EthernetFrame * BuffersModule::castMessageIntoEthernetFrame (  
    cMessage * message )    [private]
```

Cast cMessage into an EthernetFrame.

Incoming external message is cast into an EthernetFrame. It is also checked if the message exists, otherwise an error is thrown.

Parameters

<i>message</i>	Pointer to cMessage that will be cast into an EthernetFrame
----------------	---

Returns

The pointer to the cast EthernetFrame

5.6.3.4 dealWithBufferControl()

```
void BuffersModule::dealWithBufferControl (  
    cMessage * message )    [private]
```

Deal with new arrived message as BufferControlMessage.

New arrived message is a BufferControlMessage from the arbiter. Cast the cMessage into a BufferControlMessage, get the control instruction from the arbiter and handle this.

Parameters

<i>message</i>	Pointer to type cMessage, is a BufferControlMessage
----------------	---

5.6.3.5 dealWithEthernetFrame()

```
void BuffersModule::dealWithEthernetFrame (  
    cMessage * message )    [private]
```

Deal with new arrived message as EthernetFrame.

New arrived message is an EthernetFrame. Cast cMessage into an EthernetFrame. Get the priority and checked whether Ethernet frames with the same priority are already stored in buffer. Store the arrived Ethernet frame in the buffer and trigger an update message to the Arbiter if the buffer is empty.

Parameters

<i>message</i>	Pointer to type cMessage, is an EthernetFrame
----------------	---

5.6.3.6 dealWithExternalMessage()

```
void BuffersModule::dealWithExternalMessage (
    cMessage * message ) [private]
```

Deal with an external incoming message

[BuffersModule](#) has only to deal with external messages. It has to distinguished two types of external messages. The first message type is the EthernetFrame and the second one the BufferControlMessage.

Parameters

<i>message</i>	Pointer to type cMessage, can be EthernetFrame or BufferControlMessage
----------------	--

5.6.3.7 deleteBuffers()

```
void BuffersModule::deleteBuffers ( ) [private]
```

Delete every remaining [FrameBuffer](#) for every priority.

After call finish or the class destructor clean up the memory. Every remaining [FrameBuffer](#) has to be deleted to avoid warnings and errors.

5.6.3.8 deleteFramesInBuffers()

```
void BuffersModule::deleteFramesInBuffers ( ) [private]
```

Delete every remaining frame in the buffers after finishing the program

After call finish or the class destructor clean up the memory. Every remaining frame in the buffers has to be deleted to avoid warnings and errors.

5.6.3.9 finish()

```
void BuffersModule::finish ( ) [override], [virtual]
```

Called after finishing the simulation. Delete every stored Ethernet frame in the buffer.

5.6.3.10 getBufferControlInstruction()

```
int BuffersModule::getBufferControlInstruction (
    BufferControlMessage * controlMessage ) [private]
```

Get the control instructions for [BuffersModule](#) from BufferControlMessage

BufferControlMessages are messages between arbiter and [BuffersModule](#). There a two instructions from arbiter that the [BuffersModule](#) has to handle.

Parameters

<i>controlMessage</i>	Pointer to BufferControlMessage containing instructions
-----------------------	---

Returns

an integer from enum messageInformationConstant indicating the instruction how to deal with the message

5.6.3.11 getFrameBufferInformation()

```
ethernetHeaderInformation * BuffersModule::getFrameBufferInformation (
    int priority ) [private]
```

Get the information of one buffer.

Relevant information are the priority of the buffer and the frame length of the first frame in the buffer. If no frame is stored the frame length is set to 0. This indicates the arbiter that no frame of this priority is stored.

Parameters

<i>Integer</i>	value of the buffer priority
----------------	------------------------------

Returns

Pointer to struct [ethernetHeaderInformation](#) containing the priority and first frame length

5.6.3.12 getMaximumBufferSize()

```
int BuffersModule::getMaximumBufferSize ( ) [private]
```

Get the size of the buffer, which indicates the storage capacity

The maximum buffer size indicates how many frames can be stored in the buffer depending on the frame size. By default the maximum buffer size is set to 4096 Byte.

Returns

The integer value of the maximum buffer size in Byte.

5.6.3.13 `getNumberSupportedPriorities()`

```
int BuffersModule::getNumberSupportedPriorities ( ) [private]
```

Get the number of supported priorities.

Number of priorities is set by user in .ini file during the set up of the network. By default the number is set to 8 priorities with values between 0 and 7. The number is needed to set up the number of buffers. Each priority has its own frame buffer.

Returns

The integer value of the number of priorities.

5.6.3.14 `handleBufferControlInstruction()`

```
void BuffersModule::handleBufferControlInstruction (
    int instruction,
    BufferControlMessage * controlMessage ) [private]
```

Handle the instructions from the BufferControlMessage.

There are two different instructions that the [BuffersModule](#) has to handle. The first instruction is BUFFER_CONTROL_REQUEST. The arbiter sends a BufferControlMessage to [BuffersModule](#) to get the fill level of the buffers. The second instruction is SEND_AUTHORISATION. The [BuffersModule](#) gets the rights from the arbiter to send an EthernetFrame.

Parameters

<i>instruction</i>	An integer value indicating the instruction depending on messageInformationConstant
<i>controlMessage</i>	Pointer to BufferControlMessage

5.6.3.15 `handleMessage()`

```
void BuffersModule::handleMessage (
    cMessage * msg ) [override], [virtual]
```

Only external messages to handle. There are two different types of external messages. The first one is an EthernetFrame the second one is a BufferControlMessage from the arbiter.

5.6.3.16 `initialize()`

```
void BuffersModule::initialize (
    void ) [override], [virtual]
```

Initialize [BuffersModule](#). First get the user inputs. Then set up the Buffer. For each priority there is a buffer and a maximum size.

5.6.3.17 isFristArrivingFrameWithPriority()

```
bool BuffersModule::isFristArrivingFrameWithPriority (
    int priority ) [private]
```

Check whether an EthernetFrame with its priority is already stored in the buffer.

The arbiter has to be informed for new arriving EthernetFrames. Therefore it is checked whether an EthernetFrame is the first one of its priority. If it is the first one return true and if EthernetFrames with the same priority are already stored in the buffer return false.

Parameters

<i>priority</i>	Integer value of the EthernetFrame priority
-----------------	---

Returns

true, if no EthernetFrame is stored in the Buffer otherwise false.

5.6.3.18 isMessageControlMessage()

```
bool BuffersModule::isMessageControlMessage (
    cMessage * message ) [private]
```

Set true if message is a control message and false otherwise.

[BuffersModule](#) has to deal with two different message types. Message type BufferControlMessage arrives [BuffersModule](#) from "control" gate. Message type EthernetFrame arrives [BuffersModule](#) from "in" gate. To distinguished the message type check the gate at which the message arrivals.

Parameters

<i>message</i>	Pointer new arrived cMessage
----------------	------------------------------

Returns

true if message type is BufferControlMessage

5.6.3.19 sendBufferControlMessage()

```
void BuffersModule::sendBufferControlMessage (
    std::vector< ethernetHeaderInformation * > bufferInformation,
    messageInformationConstant status ) [private]
```

A BufferControlMessage is sent to the arbiter.

BufferControlMessage is sent to the arbiter. The message contains information about each buffer. The priority and the length of the first frame in each buffer is provided for the arbiter. The status of BufferControlMessage depends on the send instruction from arbiter or if a frame with a new priority arrived.

Parameters

<i>bufferInformation</i>	Vector containing for each buffer priority and frist frame lenght
<i>status</i>	Integer value status of the BufferControlMessage from type messageInformationConstant

5.6.3.20 sendEthernetFrameFromBuffer()

```
void BuffersModule::sendEthernetFrameFromBuffer (
    BufferControlMessage * controlMessage ) [private]
```

Response of SEND_AUTHORISATION from Arbiter. Contains the authorization to send an Ethernet frame.

Parameters

<i>controlMessage</i>	
-----------------------	--

5.6.3.21 setInitinalUserInputs()

```
void BuffersModule::setInitinalUserInputs ( ) [private]
```

Set the initial user inputs which are made in ini file.

To set up the buffers module the inputs from ini-file are taken. The number of supported priorities is taken to set up one [FrameBuffer](#) for each priority. The maximum size of the buffers indicates the storage capacity of each buffer.

5.6.3.22 setUpBufferInformation()

```
std::vector< ethernetHeaderInformation * > BuffersModule::setUpBufferInformation ( ) [private]
```

Set up the information about each buffer for the arbiter.

The [BuffersModule](#) has to provide information the arbiter with information about each buffer so that the arbiter can decide which Ethernet frame to send. The relevant informations are the priority of the buffer and the frame length of the first frame in the buffer.

Returns

A vector containing the priority and frame length of first frame in each buffer

5.6.3.23 `setUpFrameBuffers()`

```
void BuffersModule::setUpFrameBuffers ( ) [private]
```

Set up a frame buffer for each priority.

Depending on the number of supported priorities and the maximum buffer size a frame buffer of class [FrameBuffer](#) is set up.

5.6.3.24 `storeEthernetFrameInBuffer()`

```
void BuffersModule::storeEthernetFrameInBuffer (
    int priority,
    EthernetFrame * ethernetFrame ) [private]
```

Arriving EthernetFrame is stored in buffer with its priority.

Parameters

<i>priority</i>	Integer value of the EthernetFrame priority
<i>ethernetFrame</i>	Pointer to EthernetFrame that is stored in buffer

5.6.3.25 `triggerUpdateMessageNewPriorityArrival()`

```
void BuffersModule::triggerUpdateMessageNewPriorityArrival ( ) [private]
```

Sends a control message to Arbiter() when an EthernetFrame with new priority appears.

On arrival of an Ethernet frame with a priority that is not yet sorted in a [FrameBuffer\(\)](#), an control message with status BUFFER_LEVEL_UPDATE is transmit to the Arbiter(). Thus the Arbiter() is always kept up to date.

5.6.4 Member Data Documentation

5.6.4.1 `frameBuffers`

```
std::vector<FrameBuffer*> BuffersModule::frameBuffers [private]
```

5.6.4.2 `maximumBufferSize`

```
int BuffersModule::maximumBufferSize [private]
```

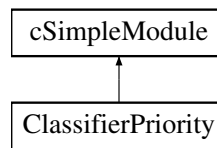

5.6.4.3 numberSupportedPriorities

```
int BuffersModule::numberSupportedPriorities [private]
```

5.7 ClassifierPriority Class Reference

```
#include <ClassifierPriority.h>
```

Inheritance diagram for ClassifierPriority:



Public Member Functions

- [ClassifierPriority](#) ()
- virtual [~ClassifierPriority](#) ()
- void [initialize](#) ()
- void [handleMessage](#) (cMessage *message)

Private Attributes

- cModule * [parentModul](#)
- int [numberPriorities](#)
- int [framePriority](#)

5.7.1 Detailed Description

The Classifier classify Frames based on the Ethernet priority and forward them in one of the priority asserted channels

5.7.2 Constructor & Destructor Documentation

5.7.2.1 ClassifierPriority()

```
ClassifierPriority::ClassifierPriority ( )
```

Constructor of the class, nothing to do

5.7.2.2 ~ClassifierPriority()

```
ClassifierPriority::~ClassifierPriority ( ) [virtual]
```

Destructor of the class, nothing to do

5.7.3 Member Function Documentation

5.7.3.1 handleMessage()

```
void ClassifierPriority::handleMessage (
    cMessage * message )
```

Process incoming Ethernet frames and send them based on the priority out
sort frames based on priority to class

5.7.3.2 initialize()

```
void ClassifierPriority::initialize (
    void )
```

Initialize the global Variables

5.7.4 Member Data Documentation

5.7.4.1 framePriority

```
int ClassifierPriority::framePriority [private]
```

5.7.4.2 numberPriorities

```
int ClassifierPriority::numberPriorities [private]
```

5.7.4.3 parentModul

```
cModule* ClassifierPriority::parentModul [private]
```

5.8 ConfigurationReader Class Reference

```
#include <ConfigurationReader.h>
```

Public Member Functions

- virtual [~ConfigurationReader](#) ()
- [configVector](#) [getMACTableEntriesFromInputFile](#) (std::string inputConfigurationFile)
- [configVector](#) [getAVBTableEntriesFromInputFile](#) (std::string inputConfigurationFile)

Static Public Member Functions

- static [ConfigurationReader](#) * [getInstance](#) ()

Private Member Functions

- [ConfigurationReader](#) ()
- void [openConfigurationFile](#) (std::string inputConfigurationFile)
- [configVector](#) [readOutInputFile](#) ()
- bool [isCommentLine](#) (std::string line)
- std::vector< int > [splitFileLineAndGetEntry](#) ()

Private Attributes

- std::ifstream [inputFile](#)
- std::string [fileLine](#)
- [configVector](#) [macTableVector](#)
- [configVector](#) [avbTableVector](#)

Static Private Attributes

- static [ConfigurationReader](#) * [instance](#) = 0

5.8.1 Constructor & Destructor Documentation

5.8.1.1 ~ConfigurationReader()

```
ConfigurationReader::~~ConfigurationReader ( ) [virtual]
```

5.8.1.2 ConfigurationReader()

```
ConfigurationReader::ConfigurationReader ( ) [private]
```

5.8.2 Member Function Documentation

5.8.2.1 getAVBTableEntriesFromInputFile()

```
configVector ConfigurationReader::getAVBTableEntriesFromInputFile (
    std::string inputConfigurationFile )
```

5.8.2.2 getInstance()

```
ConfigurationReader * ConfigurationReader::getInstance ( ) [static]
```

5.8.2.3 getMACTableEntriesFromInputFile()

```
configVector ConfigurationReader::getMACTableEntriesFromInputFile (
    std::string inputConfigurationFile )
```

5.8.2.4 isCommentLine()

```
bool ConfigurationReader::isCommentLine (
    std::string line ) [private]
```

5.8.2.5 openConfigurationFile()

```
void ConfigurationReader::openConfigurationFile (
    std::string inputConfigurationFile ) [private]
```

5.8.2.6 readOutInputFile()

```
configVector ConfigurationReader::readOutInputFile ( ) [private]
```

5.8.2.7 splitFileLineAndGetEntry()

```
std::vector< int > ConfigurationReader::splitFileLineAndGetEntry ( ) [private]
```

5.8.3 Member Data Documentation

5.8.3.1 avbTableVector

```
configVector ConfigurationReader::avbTableVector [private]
```

5.8.3.2 fileLine

```
std::string ConfigurationReader::fileLine [private]
```

5.8.3.3 inputFile

```
std::ifstream ConfigurationReader::inputFile [private]
```

5.8.3.4 instance

```
ConfigurationReader * ConfigurationReader::instance = 0 [static], [private]
```

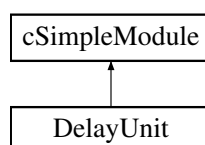
5.8.3.5 macTableVector

```
configVector ConfigurationReader::macTableVector [private]
```

5.9 DelayUnit Class Reference

```
#include <DelayUnit.h>
```

Inheritance diagram for DelayUnit:



Public Member Functions

- [DelayUnit](#) ()
- virtual [~DelayUnit](#) ()

Protected Member Functions

- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override

5.9.1 Detailed Description

Adds artificial delay to the delivery of the sample to the writer to model runtime overoverhead

5.9.2 Constructor & Destructor Documentation

5.9.2.1 DelayUnit()

```
DelayUnit::DelayUnit ( ) [inline]
```

Default constructor

5.9.2.2 ~DelayUnit()

```
DelayUnit::~~DelayUnit ( ) [virtual]
```

Default destructor

5.9.3 Member Function Documentation

5.9.3.1 handleMessage()

```
void DelayUnit::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Delays and forwards incoming messages

Parameters

<i>msg</i>	The message to delay and forward
------------	----------------------------------

5.9.3.2 initialize()

```
void DelayUnit::initialize ( ) [override], [protected], [virtual]
```

Nothing to initialize

5.10 E2ELatencyStatistics Class Reference

```
#include <E2ELatencyStatistics.h>
```

Public Member Functions

- [E2ELatencyStatistics](#) ()
- virtual [~E2ELatencyStatistics](#) ()
- int [addE2ELatency](#) (int receiverID, int globalStreamID, simtime_t delay)
- void [finish](#) ()

Private Attributes

- [VectorStatistics2DMap](#) * [vectorE2EStatistics](#) = 0
- [HistogramStatistics2DMap](#) * [histogramE2EStatistics](#) = 0

5.10.1 Detailed Description

This class is gathering E2E latency statistics per streamID. As streams are virtual entities (no module) they must be gathered by [StatisticManager](#).

5.10.2 Constructor & Destructor Documentation

5.10.2.1 E2ELatencyStatistics()

```
E2ELatencyStatistics::E2ELatencyStatistics ( )
```

default constructor, nothing to do

5.10.2.2 ~E2ELatencyStatistics()

```
E2ELatencyStatistics::~~E2ELatencyStatistics ( ) [virtual]
```

default destructor, nothing to do

5.10.3 Member Function Documentation

5.10.3.1 addE2ELatency()

```
int E2ELatencyStatistics::addE2ELatency (
    int receiverID,
    int globalStreamID,
    simtime_t delay )
```

stores the end-to-end delay of at the receiver of the stream ID

Parameters

<i>receiverID</i>	the end node ID (must be unique)
<i>globalStreamID</i>	the stream ID of the received message
<i>delay</i>	the end-to-end delay (receive time - start time)

Returns

1 if success, see error codes within code

5.10.3.2 finish()

```
void E2ELatencyStatistics::finish ( )
```

Finishes statistics gathering for End-2-End latencies Vectors will be deleted Histograms: are firstly recorded (save to file) and next deleted

5.10.4 Member Data Documentation

5.10.4.1 histogramE2Estatistics

```
HistogramStatistics2DMap* E2ELatencyStatistics::histogramE2Estatistics = 0 [private]
```

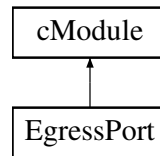
5.10.4.2 vectorE2EStatistics

```
VectorStatistics2DMap* E2ELatencyStatistics::vectorE2EStatistics = 0 [private]
```


5.11 EgressPort Class Reference

```
#include <EgressPort.h>
```

Inheritance diagram for EgressPort:



Public Member Functions

- [EgressPort](#) ()
- virtual [~EgressPort](#) ()

5.11.1 Detailed Description

In this [EgressPort](#) Module nothing happend here

5.11.2 Constructor & Destructor Documentation

5.11.2.1 EgressPort()

```
EgressPort::EgressPort ( )
```

Constructor [EgressPort](#), nothing to do here

5.11.2.2 ~EgressPort()

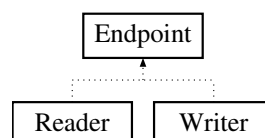
```
EgressPort::~EgressPort ( ) [virtual]
```

Destructor [EgressPort](#), nothing to do here

5.12 Endpoint Class Reference

```
#include <Endpoint.h>
```

Inheritance diagram for Endpoint:



Public Member Functions

- [Endpoint](#) ()
- [~Endpoint](#) ()
- void [calculateRtpsMsgSize](#) (RTPSEthMessage *rtpsMsg)
- unsigned long long [getNextUniqueRtpsMsgId](#) ()

Public Attributes

- int [entityId](#)
The unique entity Id.
- int [entityKind](#)
The entity kind.
- unsigned long long [rtpsMsgSequenceId](#)
RTPS message sequence ID counter for unique frame identification of RTPS messages regarding one entity.

5.12.1 Detailed Description

Superclass of RTPS writers and readers. Implements basic functionalities used by both.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Endpoint()

```
Endpoint::Endpoint ( ) [inline]
```

The default constructor

5.12.2.2 ~Endpoint()

```
Endpoint::~~Endpoint ( ) [inline]
```

5.12.3 Member Function Documentation

5.12.3.1 calculateRtpsMsgSize()

```
void Endpoint::calculateRtpsMsgSize (
    RTPSEthMessage * rtpsMsg )
```

Calculates the rtps message size of a message and sets the corresponding parameters

Parameters

<i>rtpsMsg</i>	Represents the relevant message to calculate the size for.
----------------	--

5.12.3.2 getNextUniqueRtpsMsgId()

```
unsigned long long Endpoint::getNextUniqueRtpsMsgId ( )
```

Return and increment next unique RTPS message ID

5.12.4 Member Data Documentation**5.12.4.1 entityId**

```
int Endpoint::entityId
```

The unique entity Id.

5.12.4.2 entityKind

```
int Endpoint::entityKind
```

The entity kind.

5.12.4.3 rtpsMsgSequenceId

```
unsigned long long Endpoint::rtpsMsgSequenceId
```

RTPS message sequence ID counter for unique frame identification of RTPS messages regarding one entity.

5.13 ethernetHeaderInformation Struct Reference

```
#include <EthernetHeaderInformation.h>
```

Public Attributes

- int [payloadLength](#)
- int [priority](#)
- bool [ready](#)

5.13.1 Member Data Documentation

5.13.1.1 payloadLength

```
int ethernetHeaderInformation::payloadLength
```

5.13.1.2 priority

```
int ethernetHeaderInformation::priority
```

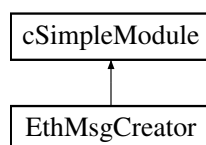
5.13.1.3 ready

```
bool ethernetHeaderInformation::ready
```

5.14 EthMsgCreator Class Reference

```
#include <EthMsgCreator.h>
```

Inheritance diagram for EthMsgCreator:



Public Member Functions

- [EthMsgCreator](#) ()
- virtual [~EthMsgCreator](#) ()

Protected Member Functions

- virtual void [initialize](#) () override
- virtual void [finish](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override
- int64_t [getNextUniqueFrameId](#) ()

Protected Attributes

- bool [enableFineGrainedLatencyStats](#)
Enables detailed statistics of the path an Ethernet message takes.
- int64_t [uniqueFrameId](#)
The frame sequence ID unique for a given globalStreamId.

5.14.1 Detailed Description

The EthMsg Creator instantiates and forwards a raw Ethernet packet with a given size after stimulation from the preceding module.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 EthMsgCreator()

```
EthMsgCreator::EthMsgCreator ( )
```

Default constructor.

5.14.2.2 ~EthMsgCreator()

```
EthMsgCreator::~EthMsgCreator ( ) [virtual]
```

The Destructor

5.14.3 Member Function Documentation

5.14.3.1 finish()

```
void EthMsgCreator::finish ( ) [override], [protected], [virtual]
```

Overridden method, called at the end of the simulation

5.14.3.2 getNextUniqueFrameId()

```
int64_t EthMsgCreator::getNextUniqueFrameId ( ) [inline], [protected]
```

This method calculates unique IDs for Ethernet messages. It returns the current value and increases it thereafter.

5.14.3.3 handleMessage()

```
void EthMsgCreator::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Overridden method, triggered by a traffic source and creates one Ethernet message each time it is called. The message is passed to the Stack and the Ethernet network.

Parameters

<i>msg</i>	each incoming message is interpreted as a stimulation
------------	---

5.14.3.4 initialize()

```
void EthMsgCreator::initialize ( ) [override], [protected], [virtual]
```

Overridden method, initializes the module parameters

5.14.4 Member Data Documentation

5.14.4.1 enableFineGrainedLatencyStats

```
bool EthMsgCreator::enableFineGrainedLatencyStats [protected]
```

Enables detailed statistics of the path an Ethernet message takes.

5.14.4.2 uniqueFrameId

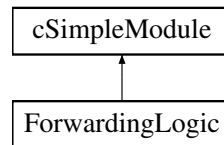
```
int64_t EthMsgCreator::uniqueFrameId [protected]
```

The frame sequence ID unique for a given globalStreamId.

5.15 ForwardingLogic Class Reference

```
#include <ForwardingLogic.h>
```

Inheritance diagram for ForwardingLogic:



Public Member Functions

- [ForwardingLogic](#) ()
- virtual [~ForwardingLogic](#) ()
- virtual void [initialize](#) () override
- void [handleMessage](#) (cMessage *message) override

Private Member Functions

- [configVector](#) [getMACTable](#) ()
- [configVector](#) [getOnlyEntriesForThisSwitch](#) ([configVector](#) macTableEntries)
- bool [forwardingByStreamID](#) (EthernetFrame *[ethernetFrame](#))
- bool [dealWithUnicast](#) (EthernetFrame *[ethernetFrame](#))
- bool [dealWithMulticast](#) (EthernetFrame *[ethernetFrame](#))
- void [isValidNumberOfEntries](#) (int macTableRow)
- int [getPortIDFromNextHOP](#) (int hopMACAddress)
- bool [isConnectedModuleEndNode](#) (cGate *nextConnectedModuleGate)
- bool [isHopMACAddressEqualEndNodeMAC](#) (int gateNumber, int hopMACAddress, cModule *connectedModule)
- bool [isHopMACAddressEqualSwitchMAC](#) (int hopMACAddress, cModule *connectedModule)
- bool [portIDInvalid](#) (int portID)
- void [printSimulationOutputs](#) ([simulationOutputType](#) type)
- void [errorHandling](#) ([errorType](#) type)

Private Attributes

- int [switchIndex](#)
- int [switchMAC](#)
- int [streamID](#)
- cModule * [parentSwitch](#)
- std::string [macTableFile](#)
- [configVector](#) [macTable](#)
- EthernetFrame * [ethernetFrame](#)

5.15.1 Detailed Description

Module [ForwardingLogic](#)() is part of [Switch](#)() module as well as the [IngressPort](#)() and [EgressPortModule](#)(). The task of the [ForwardingLogic](#)() is to get from the MAC table the right gate index to forward an Ethernet frame.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 ForwardingLogic()

```
ForwardingLogic::ForwardingLogic ( )
```

constructor: nothing to do here

default constructor

5.15.2.2 ~ForwardingLogic()

```
ForwardingLogic::~~ForwardingLogic ( ) [virtual]
```

destructor: nothing to do here

default destructor

5.15.3 Member Function Documentation

5.15.3.1 dealWithMulticast()

```
bool ForwardingLogic::dealWithMulticast (
    EthernetFrame * ethernetFrame ) [private]
```

Transmit the Ethernet frame to several end nodes (multicast).

An Ethernet frame is transmitted to several end nodes by global stream ID and connected module MAC (next hop). It is iterated over all MAC table entries and checked if the streamID is equal to stream ID of the Ethernet frame. With the same stream ID the port of the next connected Module is determined. By choosing multicast it is known that several entries with the same stream ID exist. The Ethernet frame is forwarded to all entries with the same stream ID until there are no more entries in the MAC table are left.

Parameters

<i>ethernetFrame</i>	pointer to Ethernet frame containing the information about global stream ID, among others
----------------------	---

Returns

true, if the next portID is found to which the Ethernet frame is forwarded

5.15.3.2 dealWithUnicast()

```
bool ForwardingLogic::dealWithUnicast (
    EthernetFrame * ethernetFrame ) [private]
```

Transmit the Ethernet frame to only one end node (unicast).

An Ethernet frame is transmitted only to one end node by global stream ID and connected module MAC (next hop). It is iterated over all MAC table entries and checked if the stream ID entry is equal to stream ID of Ethernet frame. With the same stream ID the port of the next connected Module is determined. Since only one end node receives the Ethernet frame, the MAC Table entry search is aborted after transmitting the frame.

Parameters

<i>ethernetFrame</i>	pointer to Ethernet Frame containing the information about global stream ID, among others
----------------------	---

Returns

true, if the next portID is found to which the Ethernet frame is forwarded

5.15.3.3 errorHandling()

```
void ForwardingLogic::errorHandling (
    errorType type ) [private]
```

A collection of output streams to handle (throw) errors.

Error 1 INCORRECT_MESSAGE_TYPE: cMessage cannot cast the Ethernet frame Error 2 UNKNOWN_STREAM_ID: stream ID in .config-file does not correspond to stream ID of the Ethernet frame Error 3 INVALID_NUMBER_OF_MAC_TABLE_ENTRIES: the number of entries in the mac Table set by user is invalid Error 4 INVALID_PORT_ID: There is no Port for given next connected module MAC address

Parameters

<i>type</i>	value of enum errorType.
-------------	--------------------------

5.15.3.4 forwardingByStreamID()

```
bool ForwardingLogic::forwardingByStreamID (
    EthernetFrame * ethernetFrame ) [private]
```

A incoming Ethernet frame is forwarded by global and unique streamID

The global stream ID is obtained from an Ethernet frame. The Port ID of the next module (next Hop) is determined to which the Ethernet frame is forwarded is determined from MAC Table. A distinction is made as to whether the frame is transmitted unicast (only to one end node) or multicast (to several end nodes)

Parameters

<i>ethernetFrame</i>	pointer to a Ethernet frame containing the information about global stream ID, among others
----------------------	---

Returns

true, if the next portID is found to which the Ethernet frame is forwarded

5.15.3.5 getMACTable()

```
configVector ForwardingLogic::getMACTable ( ) [private]
```

Get the all MAC Table entries.

MAC Table is defined as a .config-file by user. All MAC address of the network are entered in .config.-file.

Returns

the MAC Table which is defined in the .config file as vector of vectors of integers

5.15.3.6 getOnlyEntriesForThisSwitch()

```
configVector ForwardingLogic::getOnlyEntriesForThisSwitch (
    configVector macTableEntries ) [private]
```

Get only the MAC Table entries which affects this switch.

A given MAC Table configVector is searched iteratively according to entries that are only intended for it.

Parameters

<i>macTableEntries</i>	a configVector that contains the MACTable which is set by user and generated form ConfigurationReader
------------------------	---

Returns

a MACTable which includes only entries intended for a specific switch

5.15.3.7 getPortIDFromNextHOP()

```
int ForwardingLogic::getPortIDFromNextHOP (
    int hopMACAddress ) [private]
```

Get the port ID of the next connected module (hop)

Forwarding by stream ID requires the MAC address of the next connected module (hop). The module MAC address is set in the .config-file by user. With this MAC address the Port ID within the switch is determined. All connected gates in the switch and the corresponding MAC address of the connected modules are searched until the hop MAC address of .config-file is found. If no the hop MAC address does not exist, a -1 is returned.

Parameters

<i>hopMACAddress</i>	integer value of the connected module MAC (next hop) get from configVector macTable
----------------------	---

Returns

the portID of the next connected module or -1 if the entry is not found

5.15.3.8 handleMessage()

```
void ForwardingLogic::handleMessage (
    cMessage * message ) [override]
```

Handle an incoming message. Cast it to an Ethernet frame and check if it is forwarded only to one end node (unicast) or several nodes (multicast)

5.15.3.9 initialize()

```
void ForwardingLogic::initialize (
    void ) [override], [virtual]
```

The initialization of the switch. The MAC address of the switch is set, the MAC Table for this switch is generated and some outputs are made.

5.15.3.10 isConnectedModuleEndNode()

```
bool ForwardingLogic::isConnectedModuleEndNode (
    cGate * nextConnectedModuleGate ) [private]
```

Is true if the next connected module is an EndNode otherwise false.

It is important to distinguish Switch() and EndNode() modules. The port MAC address of a Switch() is always the same as the switch MAC address. The port MAC address of EndNodes() can differ from the MAC address of the EndNode().

Parameters

<i>nextConnectedModuleGate</i>	Pointer to cGate of the next connected module
--------------------------------	---

Returns

true if the next connected module is an EndNode() otherwise false

5.15.3.11 isHopMACAddressEqualEndNodeMAC()

```
bool ForwardingLogic::isHopMACAddressEqualEndNodeMAC (
    int gateNumber,
    int hopMACAddress,
    cModule * connectedModule ) [private]
```

Returns true if EndNode() MAC address and hop MAC address is equal, otherwise returns false.

The MAC addresses from EndNode() module and EndNode ports can be differ. Therefore, all submodules of type [IngressPort\(\)](#) from the EndNode() are checked to get the port MAC address. The MAC address is compared to the nextHopMAC address. With the same MAC the gate index from the switch must be checked. The reason for this is the support of multiple EndNode ports. To transmit the Ethernet frame over the correct channel, the gateNumber must be equal to the gateIndex.

Parameters

<i>gateNumber</i>	Integer value, contains the gate index of switch
<i>hopMACAddress</i>	Integer value of the next hop MAC address
<i>connectedModule</i>	Pointer to cModule of the next connected module

Returns

true if EndNode() MAC address and hop MAC address are equal

5.15.3.12 isHopMACAddressEqualSwitchMAC()

```
bool ForwardingLogic::isHopMACAddressEqualSwitchMAC (
    int hopMACAddress,
    cModule * connectedModule ) [private]
```

Returns true if the Switch() MAC address is equal to the next hop MAC address, otherwise false.

The MAC address from switch ports and switch module is the same. With connectedModules the MAC address of the connected switch is obtained. These MAC address is compared to the hopMACAddress.

Parameters

<i>hopMACAddress</i>	Integer value of next hop MAC address
<i>connectedModule</i>	Pointer to cModule of the next connected Module

Returns

true if Switch() MAC address is equal to hop MAC address

5.15.3.13 isValidNumberOfEntries()

```
void ForwardingLogic::isValidNumberOfEntries (
    int macTableRow ) [private]
```

Checks if the number of entries in the MAC table row is valid

The number of in the is 3. First entry corresponds to the switch MAC address, second entry to global stream ID and the third to the MAC address of the next connected module (hop). With an higher number of entries an error is thrown otherwise the procedure continues.

Parameters

<i>macTableEntries</i>	is a integer value of a specific row entry in the configVector
------------------------	--

5.15.3.14 portIDInvalid()

```
bool ForwardingLogic::portIDInvalid (
    int portID ) [private]
```

Checks if the portID is valid.

If the MAC address of the next connected module (hop) does not exist a -1 is returned. This functions checks if portID is -1 one. If this is the case, an error is thrown.

Parameters

<i>portID</i>	integer value of the MAC address of the next connected Module
---------------	---

Returns

true, if the value is lower than 0 are greater than the number of connected gates in the switch, otherwise false

5.15.3.15 printSimulationOutputs()

```
void ForwardingLogic::printSimulationOutputs (
    simulationOutputType type ) [private]
```

A collection of stream outputs which are shown during the simulation.

It contains only information for the user. Type 1: INIT_OUTPUT print the MAC address of the initialized switch as well as the number of loaded elements Type 2: MAC_TABLE_CONFIGURATION_INFO is printed if there is no entry in MAC table for a switch.

Parameters

<i>type</i>	value of enum simulationOutputType.
-------------	-------------------------------------

5.15.4 Member Data Documentation

5.15.4.1 ethernetFrame

EthernetFrame* ForwardingLogic::ethernetFrame [private]

5.15.4.2 macTable

configVector ForwardingLogic::macTable [private]

5.15.4.3 macTableFile

std::string ForwardingLogic::macTableFile [private]

5.15.4.4 parentSwitch

cModule* ForwardingLogic::parentSwitch [private]

5.15.4.5 streamID

int ForwardingLogic::streamID [private]

5.15.4.6 switchIndex

int ForwardingLogic::switchIndex [private]

5.15.4.7 switchMAC

```
int ForwardingLogic::switchMAC [private]
```

5.16 FrameBuffer Class Reference

```
#include <FrameBuffer.h>
```

Public Member Functions

- [FrameBuffer](#) (int priority, int maximumBufferSize)
- virtual [~FrameBuffer](#) ()
- int [getFramePriority](#) ()
- int [getFrameFillLevel](#) ()
- int [getNextFrameLength](#) ()
- void [pushFrameToBack](#) (EthernetFrame *frame)
- EthernetFrame * [popNextFrame](#) ()
- EthernetFrame * [deleteFramesInBuffer](#) ()

Private Member Functions

- int [updateFreeBufferSize](#) ([bufferUpdate](#) status, int frameLength)
- bool [freeSpaceInBuffer](#) (int frameLength)

Private Attributes

- int [framePriority](#)
- int [bufferSize](#)
- int [frameNumberInBuffer](#)
- int [storedBytesInBuffer](#)
- int [droppedFramesInBuffer](#)
- int [droppedBytesInBuffer](#)
- cOutVector [frameNumberVector](#)
- cOutVector [storedBytesVector](#)
- cOutVector [droppedFramesVector](#)
- cOutVector [droppedBytesVector](#)
- std::list< EthernetFrame * > [frameBuffer](#)

5.16.1 Detailed Description

Class [FrameBuffer](#) is one buffer in which are frames (e.g. Ethernet frames) are stored. Only frames of the same priority are in one buffer. The Buffer size (capacity in byte) depends on the module BufferModuls.ned

[FrameBuffer](#) is similar to a queue. New arriving frames are stored at the end of the buffers. Always the first element is selected for transmission.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 `FrameBuffer()`

```
FrameBuffer::FrameBuffer (
    int priority,
    int maximumBufferSize )
```

Constructor of `FrameBuffer`. By creating a new object `FrameBuffer` it is necessary to set the priority and the size (Byte) of the buffer

Parameters

<i>priority</i>	integer value of the <code>FrameBuffer</code> priority depending on frame priority
<i>maximumBufferSize</i>	integer value that defines the buffer size (set per default to 4096 byte)

5.16.2.2 `~FrameBuffer()`

```
FrameBuffer::~~FrameBuffer ( ) [virtual]
```

Destructor of `FrameBuffer` -> empty, nothing to do here

5.16.3 Member Function Documentation

5.16.3.1 `deleteFramesInBuffer()`

```
EthernetFrame * FrameBuffer::deleteFramesInBuffer ( )
```

This function is called by destructor `~BuffersModule()` to delete every remaining frame in the buffer.

After finishing the simulation every remaining Ethernet frame in a buffer must be deleted. In contrast to `popNextFrame()` no statistics are collected here. Due to the statistics `popNextFrame()` is improper for the destructor.

Returns

Pointer to the first stored Ethernet frame in buffer.

5.16.3.2 `freeSpaceInBuffer()`

```
bool FrameBuffer::freeSpaceInBuffer (
    int frameLenght ) [private]
```

checks if there is enough space in the buffer to store a frame

Parameters

<i>frameLength</i>	Integer value, indicates the length of the frame to be stored
--------------------	---

Returns

returns true if the bufferSize is larger than frame length otherwise false

5.16.3.3 getFrameFillLevel()

```
int FrameBuffer::getFrameFillLevel ( )
```

Get the number of frames stored in the [FrameBuffer](#)

Returns

The integer value number of frames stored in the buffer

5.16.3.4 getFramePriority()

```
int FrameBuffer::getFramePriority ( )
```

Get the priority of a [FrameBuffer](#) and thus the stored frame priority

Returns

The integer value priority of a [FrameBuffer](#)

5.16.3.5 getNextFrameLength()

```
int FrameBuffer::getNextFrameLength ( )
```

Get the frame length of the first frame in the buffer

Returns

The length of the first frame in the buffer (in byte)

5.16.3.6 popNextFrame()

```
EthernetFrame * FrameBuffer::popNextFrame ( )
```

Remove the first frame of the [FrameBuffer](#). To simulate "first in, first out" (FIFO) frames are only be removed from the front of the buffer

Returns

Pointer to first EthernetFrame in Buffer.

5.16.3.7 pushFrameToBack()

```
void FrameBuffer::pushFrameToBack (
    EthernetFrame * frame )
```

Add a new frame into the [FrameBuffer](#). Frame buffer is similar to a queue. Frames can only be added at the end of the buffer

Parameters

<i>frame</i>	Pointer of EthernetFrame that is added to FrameBuffer
--------------	---

5.16.3.8 updateFreeBufferSize()

```
int FrameBuffer::updateFreeBufferSize (
    bufferUpdate status,
    int frameLength ) [private]
```

Update the number of free space in buffer.

By adding or removing a frame the frame buffer size has to be updated. The frame length of a new arrived frame decreases the buffer size. The frame length of a removed frame increases the buffer size

Parameters

<i>status</i>	Specifies whether a frame is added or removed
<i>frameLength</i>	integer value indicating the added/removed frame length

Returns

integer value of the updated buffer size (in byte)

5.16.4 Member Data Documentation

5.16.4.1 bufferSize

```
int FrameBuffer::bufferSize [private]
```

5.16.4.2 droppedBytesInBuffer

```
int FrameBuffer::droppedBytesInBuffer [private]
```

5.16.4.3 droppedBytesVector

```
cOutVector FrameBuffer::droppedBytesVector [private]
```

5.16.4.4 droppedFramesInBuffer

```
int FrameBuffer::droppedFramesInBuffer [private]
```

5.16.4.5 droppedFramesVector

```
cOutVector FrameBuffer::droppedFramesVector [private]
```

5.16.4.6 frameBuffer

```
std::list<EthernetFrame*> FrameBuffer::frameBuffer [private]
```

One [FrameBuffer](#) is implemented as list but acts like a queue (first in, first out)

5.16.4.7 frameNumberInBuffer

```
int FrameBuffer::frameNumberInBuffer [private]
```

5.16.4.8 frameNumberVector

```
cOutVector FrameBuffer::frameNumberVector [private]
```

5.16.4.9 framePriority

```
int FrameBuffer::framePriority [private]
```

5.16.4.10 storedBytesInBuffer

```
int FrameBuffer::storedBytesInBuffer [private]
```

5.16.4.11 storedBytesVector

```
cOutVector FrameBuffer::storedBytesVector [private]
```

5.17 FramesLostStatistics Class Reference

```
#include <FramesLostStatistics.h>
```

Public Member Functions

- [FramesLostStatistics](#) ()
- virtual [~FramesLostStatistics](#) ()
- void [finish](#) ()
- void [addLostFrame](#) (int receiverID, int globalStreamID, int counter)
- void [addLostFrameTrace](#) (int receiverID, int globalStreamID, int counter)

Private Attributes

- [VectorStatistics2DMap](#) * [vectorLostFramesPerStream](#) = 0
- [VectorStatistics2DMap](#) * [vectorLostFramesTracePerStream](#) = 0
- [HistogramStatistics2DMap](#) * [histogramLostFramesPerStream](#) = 0

5.17.1 Detailed Description

This class is gathering statistics for the frame loss per streamID. As streams are virtual entities (no module) they must be gathered by [StatisticManager](#).

5.17.2 Constructor & Destructor Documentation

5.17.2.1 FramesLostStatistics()

```
FramesLostStatistics::FramesLostStatistics ( )
```

default constructor, nothing to do

5.17.2.2 ~FramesLostStatistics()

```
FramesLostStatistics::~~FramesLostStatistics ( ) [virtual]
```

default destructor, nothing to do

5.17.3 Member Function Documentation

5.17.3.1 addLostFrame()

```
void FramesLostStatistics::addLostFrame (
    int receiverID,
    int globalStreamID,
    int counter )
```

stores the number of lost frames at the receiver by the stream with streamID

Parameters

<i>receiverID</i>	the end node ID (must be unique)
<i>globalStreamID</i>	the stream ID of the received message
<i>counter</i>	number of lost frames

5.17.3.2 addLostFrameTrace()

```
void FramesLostStatistics::addLostFrameTrace (
    int receiverID,
    int globalStreamID,
    int counter )
```

5.17.3.3 finish()

```
void FramesLostStatistics::finish ( )
```

5.17.4 Member Data Documentation

5.17.4.1 histogramLostFramesPerStream

```
HistogramStatistics2DMap* FramesLostStatistics::histogramLostFramesPerStream = 0 [private]
```

5.17.4.2 vectorLostFramesPerStream

```
VectorStatistics2DMap* FramesLostStatistics::vectorLostFramesPerStream = 0 [private]
```

5.17.4.3 vectorLostFramesTracePerStream

```
VectorStatistics2DMap* FramesLostStatistics::vectorLostFramesTracePerStream = 0 [private]
```

5.18 Frer_ID Struct Reference

```
#include <GlobalMacros.h>
```

Public Attributes

- int [port_ids](#) [2]

5.18.1 Member Data Documentation

5.18.1.1 port_ids

```
int Frer_ID::port_ids[2]
```

5.19 HistogramStatistics2DMap Class Reference

```
#include <HistogramStatistics2DMap.h>
```

Public Member Functions

- [HistogramStatistics2DMap](#) (const char *[prefix](#), const char *[modulePrefix](#), const char *[submodulePrefix](#))
- virtual [~HistogramStatistics2DMap](#) ()
- void [finish](#) ()
- void [addIntegerResult](#) (int moduleID, int submoduleID, int value)
- void [addLatency](#) (int moduleID, int submoduleID, simtime_t value)

Private Attributes

- std::map< int, std::map< int, cDoubleHistogram * > * > * [storedMeasurements](#) = 0
histogram containing the stored results
- const char * [prefix](#)
prefix is a string which stores information allowing to indentify the type of statistics which is gathered e.g. E2E latency, packet drop etc.
- const char * [modulePrefix](#)
modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID
- const char * [submodulePrefix](#)
submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID
- const char * [valuePrefix](#)
value prefix is a string allowing to identify the type of stored data

5.19.1 Detailed Description

Helper class used for storing the histogram statistics for objects which do not have physical modules e.g. streams or DDS samples with two parameters describing each entry e.g. moduleID and submoduleID

5.19.2 Constructor & Destructor Documentation

5.19.2.1 HistogramStatistics2DMap()

```
HistogramStatistics2DMap::HistogramStatistics2DMap (
    const char * prefix,
    const char * modulePrefix,
    const char * submodulePrefix )
```

Constructor of the class Stores parameters which will be later used for the creation of the string which is used to identify results.

Parameters

<i>prefix</i>	
<i>modulePrefix</i>	
<i>submodulePrefix</i>	creates the statistic collector
<i>prefix</i>	what name should be added to the statistic Histogram?
<i>is</i>	this collection using the stream ID or the stream priority

5.19.2.2 ~HistogramStatistics2DMap()

```
HistogramStatistics2DMap::~HistogramStatistics2DMap ( ) [virtual]
```

Destructor, nothing to do

5.19.3 Member Function Documentation

5.19.3.1 addIntegerResult()

```
void HistogramStatistics2DMap::addIntegerResult (
    int moduleID,
    int submoduleID,
    int value )
```

Function stores the integer measurement results into the statistics histogram

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. streamID
<i>value</i>	- the measured value, must be an integer

5.19.3.2 addLatency()

```
void HistogramStatistics2DMap::addLatency (
    int moduleID,
    int submoduleID,
    simtime_t value )
```

Function stores the simulation latency measurement results into the statistics histogram

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. streamID
<i>value</i>	- the measured value, must be an simtime_t

stores the given time associated with the switchMAC, portID and priority

5.19.3.3 finish()

```
void HistogramStatistics2DMap::finish ( )
```

Finish function is used to clear & save the statistics at the end of simulation

this calls recordAs on the histograms with the name given in add_simtime/add_int if there are any values stored, delete them (free memory and reset pointers)

5.19.4 Member Data Documentation

5.19.4.1 modulePrefix

```
const char* HistogramStatistics2DMap::modulePrefix [private]
```

modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID

5.19.4.2 prefix

```
const char* HistogramStatistics2DMap::prefix [private]
```

prefix is a string which stores information allowing to indentify the type of statistics which is gathered e.g. E2E latency, packet drop etc.

5.19.4.3 storedMeasurements

```
std::map<int, std::map<int, cDoubleHistogram*>*>* HistogramStatistics2DMap::storedMeasurements  
= 0 [private]
```

histogram containing the stored results

5.19.4.4 submodulePrefix

```
const char* HistogramStatistics2DMap::submodulePrefix [private]
```

submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID

5.19.4.5 valuePrefix

```
const char* HistogramStatistics2DMap::valuePrefix [private]
```

value prefix is a string allowing to identify the type of stored data

5.20 HistogramStatistics3DMap Class Reference

```
#include <HistogramStatistics3DMap.h>
```

Public Member Functions

- [HistogramStatistics3DMap](#) (const char *prefix, const char *modulePrefix, const char *submodulePrefix, const char *valuePrefix)
- virtual [~HistogramStatistics3DMap](#) ()
- void [finish](#) ()
- void [addInteger](#) (int moduleID, int submoduleID, int valueID, int value)
- void [addLatency](#) (int moduleID, int submoduleID, int valueID, simtime_t value)
- void [addLatency](#) (int moduleID, int submoduleID, int valueID, const std::string &valueIDStr, simtime_t value)

Private Member Functions

- cDoubleHistogram * [getEntry](#) (int moduleID, int submoduleID, int valueID)
- cDoubleHistogram * [getEntry](#) (int moduleID, int submoduleID, int valueID, const std::string &valueIDStr)

Private Attributes

- const char * [prefix](#) = 0
prefix is a string which stores information allowing to identify the type of statistics which is gathered e.g. E2E latency, packet drop etc.
- const char * [modulePrefix](#) = 0
modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID
- const char * [submodulePrefix](#) = 0
submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID
- const char * [valuePrefix](#) = 0
value prefix is a string allowing to identify the type of stored data
- std::map< int, std::map< int, std::map< int, cDoubleHistogram * > * > * > * [storedMeasurements](#) = 0
histogram containing the stored results, three dimensional

Static Private Attributes

- static const std::string [empty](#)
helper variable used when there is no valueID

5.20.1 Detailed Description

Helper class used for storing histogram statistics for objects which do not have physical modules e.g. streams or DDS samples with three parameters describing each entry e.g. moduleID and submoduleID and Value ID for instance port, buffer queue, streamID

5.20.2 Constructor & Destructor Documentation

5.20.2.1 HistogramStatistics3DMap()

```
HistogramStatistics3DMap::HistogramStatistics3DMap (
    const char * prefix,
    const char * modulePrefix,
    const char * submodulePrefix,
    const char * valuePrefix )
```

Constructor of the class Stores parameters which will be later used for the creation of the string which is used to identify results.

Parameters

<i>prefix</i>	
<i>modulePrefix</i>	
<i>submodulePrefix</i>	
<i>valuePrefix</i>	

5.20.2.2 ~HistogramStatistics3DMap()

```
HistogramStatistics3DMap::~HistogramStatistics3DMap ( ) [virtual]
```

Destructor, nothing to do

5.20.3 Member Function Documentation

5.20.3.1 addInteger()

```
void HistogramStatistics3DMap::addInteger (
    int moduleID,
    int submoduleID,
    int valueID,
    int value )
```

Function stores the integer measurement results into the statistics histogram

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. bufferID
<i>valueID</i>	- id of the stream/streams for which statistics has been taken e.g. priority, streamID
<i>value</i>	- the measured value, must be an integer

5.20.3.2 addLatency() [1/2]

```
void HistogramStatistics3DMap::addLatency (
    int moduleID,
    int submoduleID,
    int valueID,
    const std::string & valueIDStr,
    simtime_t value )
```

Function stores the integer measurement results into the statistics histogram

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. bufferID
<i>valueID</i>	- id of the stream/streams for which statistics has been taken e.g. priority, streamID
<i>valueIDStr</i>	- substitutes the valueID if defined
<i>value</i>	- the measured value, must be an simtime_t

5.20.3.3 addLatency() [2/2]

```
void HistogramStatistics3DMap::addLatency (
    int moduleID,
    int submoduleID,
    int valueID,
    simtime_t value )
```

Function stores the integer measurement results into the statistics histogram

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. bufferID
<i>valueID</i>	- id of the stream/streams for which statistics has been taken e.g. priority, streamID
<i>value</i>	- the measured value, must be an simtime_t

5.20.3.4 finish()

```
void HistogramStatistics3DMap::finish ( )
```

Finish function is used to clear & save the statistics at the end of simulation

5.20.3.5 getEntry() [1/2]

```
cDoubleHistogram * HistogramStatistics3DMap::getEntry (
    int moduleID,
    int submoduleID,
    int valueID ) [private]
```

Function which returns an entry identified by parameters valueIDStr is defined to be empty

Parameters

<i>moduleID</i>	- module where measurement has been takes e.g. portID
<i>submoduleID-</i>	module for which measurement has been take e.g. bufferID
<i>valueID</i>	- additional parameter e.g. streamID

Returns

5.20.3.6 getEntry() [2/2]

```
cDoubleHistogram* HistogramStatistics3DMap::getEntry (
    int moduleID,
    int submoduleID,
    int valueID,
    const std::string & valueIDStr ) [private]
```

Function which returns an entry identified by parameters

Parameters

<i>moduleID</i>	- module where measurement has been takes e.g. portID
<i>submoduleID-</i>	module for which measurement has been take e.g. bufferID
<i>valueID</i>	- additional parameter e.g. streamID
<i>valueIDStr</i>	- string identification, substitutes valueID if defined

Returns

5.20.4 Member Data Documentation

5.20.4.1 empty

```
const string HistogramStatistics3DMap::empty [static], [private]
```

helper variable used when there is no valueID

5.20.4.2 modulePrefix

```
const char* HistogramStatistics3DMap::modulePrefix = 0 [private]
```

modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID

5.20.4.3 prefix

```
const char* HistogramStatistics3DMap::prefix = 0 [private]
```

prefix is a string which stores information allowing to identify the type of statistics which is gathered e.g. E2E latency, packet drop etc.

5.20.4.4 storedMeasurements

```
std::map<int, std::map<int, std::map<int, cDoubleHistogram*>*>*> HistogramStatistics3DMap←  
::storedMeasurements = 0 [private]
```

histogram containing the stored results, thre dimmensional

5.20.4.5 submodulePrefix

```
const char* HistogramStatistics3DMap::submodulePrefix = 0 [private]
```

submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID

5.20.4.6 valuePrefix

```
const char* HistogramStatistics3DMap::valuePrefix = 0 [private]
```

value prefix is a string allowing to identify the type of stored data

5.21 HistorySampleEntry Class Reference

```
#include <HistoryEntry.h>
```

Public Member Functions

- [HistorySampleEntry \(\)](#)
- [~HistorySampleEntry \(\)](#)

Public Attributes

- int [sampleSize](#)
Information on sample granularity.
- int [sampleSequenceNumber](#)
- int [globalStreamId](#)
The writers global stream ID.
- int [readerEntityId](#)
- int [readerParticipantEntityId](#)
- int [writerEntityId](#)
- int [writerParticipantEntityId](#)
- int [numberFragments](#)
Fragment information storage.
- [SampleFragment](#) ** [sampleFragmentArray](#)
- bool [complete](#)
Evaluation specific.
- bool [lifespanExpired](#)
Invalidates the entry after the lifespan expired.
- bool [historySizeExceeded](#)
Invalidates the entry when too many samples are in the cache.
- bool [deadlineViolation](#)
Deadline violation corresponding to latency budget.
- [simtime_t](#) [generationTime](#)
- [simtime_t](#) [writerArrivalTime](#)
Sample creation time (arrival in RTPS component)
- [simtime_t](#) [writerFirstInjectionTime](#)
Injection time of the first rtps message into the network.
- [simtime_t](#) [subscriberReceiveTime](#)
Completely received at the reader.
- [std::list< simtime_t >](#) [rtpsMsgLatencies](#)
Latencies of the individual rtps_messages.

5.21.1 Detailed Description

Represents a sample in the history cache

5.21.2 Constructor & Destructor Documentation

5.21.2.1 HistorySampleEntry()

```
HistorySampleEntry::HistorySampleEntry ( ) [inline]
```

The default constructor

5.21.2.2 ~HistorySampleEntry()

```
HistorySampleEntry::~HistorySampleEntry ( ) [inline]
```

The default destructor

5.21.3 Member Data Documentation

5.21.3.1 complete

```
bool HistorySampleEntry::complete
```

Evaluation specific.

5.21.3.2 deadlineViolation

```
bool HistorySampleEntry::deadlineViolation
```

Deadline violation corresponding to latency budget.

5.21.3.3 generationTime

```
simtime_t HistorySampleEntry::generationTime
```

Statistic related Sample generation time (in generator)

5.21.3.4 globalStreamId

```
int HistorySampleEntry::globalStreamId
```

The writers global stream ID.

5.21.3.5 historySizeExeeded

```
bool HistorySampleEntry::historySizeExeeded
```

Invalidates the entry when too many samples are in the cache.

5.21.3.6 lifespanExpired

```
bool HistorySampleEntry::lifespanExpired
```

Invalidates the entry after the lifespan expired.

5.21.3.7 numberFragments

```
int HistorySampleEntry::numberFragments
```

Fragment information storage.

5.21.3.8 readerEntityId

```
int HistorySampleEntry::readerEntityId
```

5.21.3.9 readerParticipantEntityId

```
int HistorySampleEntry::readerParticipantEntityId
```

5.21.3.10 rtpsMsgLatencies

```
std::list<simtime_t> HistorySampleEntry::rtpsMsgLatencies
```

Latencies of the individual rtps_messages.

5.21.3.11 sampleFragmentArray

```
SampleFragment** HistorySampleEntry::sampleFragmentArray
```

5.21.3.12 sampleSequenceNumber

```
int HistorySampleEntry::sampleSequenceNumber
```

5.21.3.13 sampleSize

```
int HistorySampleEntry::sampleSize
```

Information on sample granularity.

5.21.3.14 subscriberReceiveTime

```
simtime_t HistorySampleEntry::subscriberReceiveTime
```

Completely received at the reader.

5.21.3.15 writerArrivalTime

```
simtime_t HistorySampleEntry::writerArrivalTime
```

Sample creation time (arrival in RTPS component)

5.21.3.16 writerEntityId

```
int HistorySampleEntry::writerEntityId
```

5.21.3.17 writerFirstInjectionTime

```
simtime_t HistorySampleEntry::writerFirstInjectionTime
```

Injection time of the first rtps message into the network.

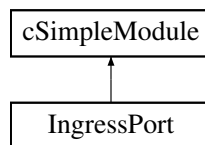
5.21.3.18 writerParticipantEntityId

```
int HistorySampleEntry::writerParticipantEntityId
```

5.22 IngressPort Class Reference

```
#include <IngressPort.h>
```

Inheritance diagram for IngressPort:



Public Member Functions

- [IngressPort](#) ()
- virtual [~IngressPort](#) ()
- void [initialize](#) ()
- void [handleMessage](#) (cMessage *message)

Private Member Functions

- void [dealWithInternalMessage](#) (cMessage *message)
- void [dealWithExternalMessage](#) (cMessage *message)
- EthernetFrame * [checkAndCastEthernetFrame](#) (cMessage *message)
- void [dealWithEthernetFrame](#) (EthernetFrame *ethernetFrame)

Private Attributes

- int [macAddress](#)
- int [ingressPortID](#)
- simtime_t [sampleTimePeriod](#)
- cMessage * [triggerSelf](#) = new cMessage()
- int [numBytes](#)
- cOutVector [linkUtilizationStatistics](#)

5.22.1 Constructor & Destructor Documentation

5.22.1.1 IngressPort()

```
IngressPort::IngressPort ( )
```

Constructor [IngressPort](#), nothing to do here

Constructor

5.22.1.2 ~IngressPort()

```
IngressPort::~~IngressPort ( ) [virtual]
```

Destructor [IngressPort](#) Cancel and delete self trigger Messages from type cMessage

Destructor

5.22.2 Member Function Documentation

5.22.2.1 checkAndCastEthernetFrame()

```
EthernetFrame * IngressPort::checkAndCastEthernetFrame (
    cMessage * message ) [private]
```

check if incoming Message is an Ethernet Frame and if so return the Frame

Parameters

<i>message</i>	Pointer to type cMessage
----------------	--------------------------

Returns

message Pointer of type EthernetFrame

5.22.2.2 dealWithEthernetFrame()

```
void IngressPort::dealWithEthernetFrame (
    EthernetFrame * ethernetFrame ) [private]
```

Sends out the incoming Ethernet Frame

Parameters

<i>message</i>	Pointer of type EthernetFrame
----------------	-------------------------------

5.22.2.3 dealWithExternalMessage()

```
void IngressPort::dealWithExternalMessage (
    cMessage * message ) [private]
```

deal with incoming external Messages and check if it is an Ethernet Frame

Parameters

<i>message</i>	Pointer of type cMessage
----------------	--------------------------

5.22.2.4 dealWithInternalMessage()

```
void IngressPort::dealWithInternalMessage (
    cMessage * message ) [private]
```

deal with internal self Messages Take next snapshot for statistic values if Message is triggerSelf

Parameters

<i>message</i>	Pointer to type cMessage
----------------	--------------------------

5.22.2.5 handleMessage()

```
void IngressPort::handleMessage (
    cMessage * message )
```

deal with incoming Messages and select between Messages from outside e.g. Ethernet Messages or self Messages

Parameters

<i>message</i>	Pointer to type cMessage, can be EthernetFrame or self Messages
----------------	---

5.22.2.6 initialize()

```
void IngressPort::initialize (  
    void )
```

initialization of variables from ned files Initialize vector for statistics and set self trigger first time

5.22.3 Member Data Documentation

5.22.3.1 ingressPortID

```
int IngressPort::ingressPortID [private]
```

5.22.3.2 linkUtilizationStatistics

```
cOutVector IngressPort::linkUtilizationStatistics [private]
```

5.22.3.3 macAddress

```
int IngressPort::macAddress [private]
```

5.22.3.4 numBytes

```
int IngressPort::numBytes [private]
```

5.22.3.5 sampleTimePeriod

```
simtime_t IngressPort::sampleTimePeriod [private]
```

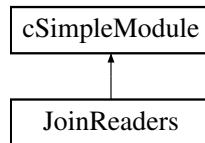
5.22.3.6 triggerSelf

```
cMessage* IngressPort::triggerSelf = new cMessage() [private]
```

5.23 JoinReaders Class Reference

```
#include <JoinReaders.h>
```

Inheritance diagram for JoinReaders:



Public Member Functions

- [JoinReaders](#) ()
- virtual [~JoinReaders](#) ()

Protected Member Functions

- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override
- virtual void [finish](#) () override

5.23.1 Detailed Description

Implementation of a join. All input messages are directly forwarded to the single output.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 JoinReaders()

```
JoinReaders::JoinReaders ( ) [inline]
```

The default constructor.

5.23.2.2 ~JoinReaders()

```
JoinReaders::~~JoinReaders ( ) [virtual]
```

The default destructor

5.23.3 Member Function Documentation

5.23.3.1 finish()

```
void JoinReaders::finish ( ) [override], [protected], [virtual]
```

Nothing to do after finishing.

5.23.3.2 handleMessage()

```
void JoinReaders::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

5.23.3.3 initialize()

```
void JoinReaders::initialize ( ) [override], [protected], [virtual]
```

Nothing to initialize here.

5.24 LinkTransmissionStatistics Class Reference

```
#include <LinkTransmissionStatistics.h>
```

Public Member Functions

- [LinkTransmissionStatistics](#) ()
- virtual [~LinkTransmissionStatistics](#) ()
- void [finish](#) ()
- void [ethernetFrameStartedSending](#) (int switchMAC, int portIndex, int streamID)
- void [ethernetFrameFinishedSending](#) (int switchMAC, int portIndex, int streamID)

Private Attributes

- [VectorStatistics3DMap](#) * [activationPeriod](#) = 0

5.24.1 Detailed Description

The statistic `LinkTransmissionStatistic` records the transmission time from one module to the next module.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 LinkTransmissionStatistics()

```
LinkTransmissionStatistics::LinkTransmissionStatistics ( )
```

default constructor, nothing to do here

5.24.2.2 ~LinkTransmissionStatistics()

```
LinkTransmissionStatistics::~~LinkTransmissionStatistics ( ) [virtual]
```

default destructor, nothing to do here

5.24.3 Member Function Documentation

5.24.3.1 ethernetFrameFinishedSending()

```
void LinkTransmissionStatistics::ethernetFrameFinishedSending (
    int switchMAC,
    int portIndex,
    int streamID )
```

An Ethernet frame has finished its transmission (between two modules)

It records the finishing time when an Ethernet frame start its transmission at a given port on a given switch with a given streamID. First a 0 is stored and then a 1. This is done to receive a right angle in result output.

Parameters

<i>switchMAC</i>	integer value of the switch MAC address to identifies the switch
<i>portIndex</i>	integer value of a specific port in the switch.
<i>streamID</i>	integer value of the streamID which identifies the Ethernet frame

5.24.3.2 ethernetFrameStartedSending()

```
void LinkTransmissionStatistics::ethernetFrameStartedSending (
    int switchMAC,
    int portIndex,
    int streamID )
```

An Ethernet frame has started its transmission (between two modules)

It records the start time when an Ethernet frame start its transmission at a given port on a given switch with a given streamID. First a 0 is stored and then a 1. This is done to receive a right angle in result output.

Parameters

<i>switchMAC</i>	integer value of the switch MAC address to identifies the switch
<i>portIndex</i>	integer value of a specific port in the switch.
<i>streamID</i>	integer value of the streamID which identifies the Ethernet frame

5.24.3.3 finish()

```
void LinkTransmissionStatistics::finish ( )
```

is called by finishing the simulation.

5.24.4 Member Data Documentation**5.24.4.1 activationPeriod**

```
VectorStatistics3DMap* LinkTransmissionStatistics::activationPeriod = 0 [private]
```

5.25 lostFrameCounter Struct Reference

```
#include <GlobalMacros.h>
```

Public Attributes

- int [newestFrameID](#)
- `std::unordered_set< int >` [lostFrames](#)

5.25.1 Member Data Documentation**5.25.1.1 lostFrames**

```
std::unordered_set<int> lostFrameCounter::lostFrames
```

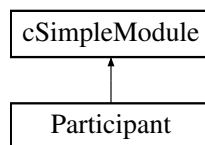
5.25.1.2 newestFrameID

```
int lostFrameCounter::newestFrameID
```

5.26 Participant Class Reference

```
#include <Participant.h>
```

Inheritance diagram for Participant:



Public Member Functions

- [Participant](#) ()
- virtual [~Participant](#) ()

Protected Member Functions

- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override
- virtual void [finish](#) () override

Private Attributes

- int [participantId](#)
The globally unique ID of the participant.
- int [nbrWriters](#)
Store the number of reader and writer locally.
- int [nbrReaders](#)
- int * [writerIds](#)
Registry for reader IDs to gate-index mappings.
- int * [readerIds](#)
Registry for writer IDs to gate-index mappings.

5.26.1 Detailed Description

Dispatches the messages and pass them either to the correct reader/writer for receiving or to the shaper for sending

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Participant()

```
Participant::Participant ( ) [inline]
```

Default constructor

5.26.2.2 ~Participant()

```
Participant::~~Participant ( ) [virtual]
```

Default destructor

5.26.3 Member Function Documentation

5.26.3.1 finish()

```
void Participant::finish ( ) [override], [protected], [virtual]
```

Deletes all stored context

5.26.3.2 handleMessage()

```
void Participant::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Handles and dispatches (forwards) incoming messages

Parameters

<i>msg</i>	The incoming msg to forward, based on its origin and content
------------	--

5.26.3.3 initialize()

```
void Participant::initialize ( ) [override], [protected], [virtual]
```

Initialization to load the module parameters

5.26.4 Member Data Documentation

5.26.4.1 nbrReaders

```
int Participant::nbrReaders [private]
```

5.26.4.2 nbrWriters

```
int Participant::nbrWriters [private]
```

Store the number of reader and writer locally.

5.26.4.3 participantId

```
int Participant::participantId [private]
```

The globally unique ID of the participant.

5.26.4.4 readerIds

```
int* Participant::readerIds [private]
```

Registry for writer IDs to gate-index mappings.

5.26.4.5 writerIds

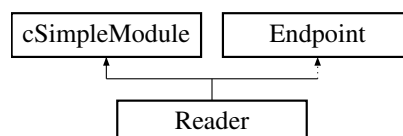
```
int* Participant::writerIds [private]
```

Registry for reader IDs to gate-index mappings.

5.27 Reader Class Reference

```
#include <Reader.h>
```

Inheritance diagram for Reader:



Public Member Functions

- [Reader](#) ()
- virtual [~Reader](#) ()

Protected Member Functions

- virtual void [finish](#) () override
- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override
- void [registerReader](#) ()
- std::list< [HistorySampleEntry](#) * > * [getCache](#) (int sourceParticipantId, int writerId)
- void [updateCache](#) (RTPSEthMessage *rtpsMsg, std::list< [HistorySampleEntry](#) * > *historyCache)
- bool [relevantNewSample](#) (int sequenceNumber, int writerEntityId, int sourceParticipantId, std::list< [HistorySampleEntry](#) * > *historyCache)
- void [insertSampleInCache](#) ([HistorySampleEntry](#) *sample)
- void [checkCompletion](#) (std::list< [HistorySampleEntry](#) * > *historyCache)
- void [checkLifespan](#) (std::list< [HistorySampleEntry](#) * > *historyCache)
- void [cleanHistoryCache](#) (std::list< [HistorySampleEntry](#) * > *historyCache)
- void [sendSampleToSink](#) ([HistorySampleEntry](#) *entry)

Private Attributes

- cMessage * [registerEvent](#)
Self message to register the reader at the participant.
- int [historySizeQoS](#)
The size of the history cache.
- bool [bestEffort](#)
Configuration of best effort or reliable communication.
- simtime_t [lifespan](#)
The lifespan of the samples.
- simtime_t [deadline](#)
The deadline of the samples.
- int [transportPriority](#)
The transport priority QoS parameter.
- [ReaderHistory](#) * [readerHistoryCache](#)
The history cache.
- cHistogram [SampleLatencySinceGenerationStat](#)
Histogram for sample Latency from sample generation.
- cOutVector [SampleLatencySinceGenerationStatVec](#)
Vector for sample Latency from sample generation.
- cHistogram [SampleLatencySinceWriterArrivalStat](#)
Histogram for sample Latency from arrival at writer.
- cOutVector [SampleLatencySinceWriterArrivalStatVec](#)
Vector for sample Latency from arrival at writer.
- cHistogram [SampleLatencySinceSampleInjectionStat](#)
Histogram for sample Latency from first network injection of the first packet.
- cOutVector [SampleLatencySinceSampleInjectionStatVec](#)
Vector for sample Latency from first network injection of the first packet.
- cHistogram [frameLatencyInjectionToReception](#)
Frame network latency histogram.

- cOutVector [frameLatencyInjectionToReceptionVec](#)
Frame network latency vector.
- int [sourceMac](#)
The own MAC address.
- int [entityId](#)
The own entity ID.
- int [streamEthernetPriority](#)
The Ethernet Priority.

Additional Inherited Members

5.27.1 Detailed Description

The RTPS [Reader](#) entity subscribes to remote writers and delivers completely received samples to the application/sink

5.27.2 Constructor & Destructor Documentation

5.27.2.1 Reader()

```
Reader::Reader ( ) [inline]
```

The default constructor

5.27.2.2 ~Reader()

```
Reader::~~Reader ( ) [virtual]
```

The default destructor

5.27.3 Member Function Documentation

5.27.3.1 checkCompletion()

```
void Reader::checkCompletion (
    std::list< HistorySampleEntry * > * historyCache ) [protected]
```

Check all samples in the cache corresponding to a writer for completion.

Parameters

<i>historyCache</i>	The history Cache for which all samples are checked for completion.
---------------------	---

5.27.3.2 checkLifespan()

```
void Reader::checkLifespan (
    std::list< HistorySampleEntry * > * historyCache ) [protected]
```

Check the lifespan of all samples in the cache corresponding to a writer.

Parameters

<i>historyCache</i>	The history Cache for which all samples are checked for lifespan expiration.
---------------------	--

5.27.3.3 cleanHistoryCache()

```
void Reader::cleanHistoryCache (
    std::list< HistorySampleEntry * > * historyCache ) [protected]
```

Remove samples from the history cache after its maximum size has been exceeded.

Parameters

<i>historyCache</i>	The history cache to clean from completed, or exceeded entries.
---------------------	---

5.27.3.4 finish()

```
void Reader::finish ( ) [override], [protected], [virtual]
```

Record the histograms here.

5.27.3.5 getCache()

```
std::list< HistorySampleEntry * > * Reader::getCache (
    int sourceParticipantId,
    int writerId ) [protected]
```

Get the partial history Cache corresponding to the referenced writer

Parameters

<i>source</i> ↔ <i>ParticipantId</i>	The ID of the participant related to the writer.
<i>writerId</i>	The ID of the writer itself. @ret Returns the cache corresponding to the specified unique writer.

5.27.3.6 handleMessage()

```
void Reader::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Handles incoming messages

Parameters

<i>msg</i>	Receives RTPSEthMessages and evaluates them.
------------	--

5.27.3.7 initialize()

```
void Reader::initialize ( ) [override], [protected], [virtual]
```

Initialize all relevant parameters and register the reader at the participant.

5.27.3.8 insertSampleInCache()

```
void Reader::insertSampleInCache (
    HistorySampleEntry * sample ) [protected]
```

Inserts a sample to the correct place in the cache (ordered by the sequence number).

Parameters

<i>sample</i>	The sample to be placed.
---------------	--------------------------

5.27.3.9 registerReader()

```
void Reader::registerReader ( ) [protected]
```

Register the reader at the participant for message dispatching.

5.27.3.10 relevantNewSample()

```
bool Reader::relevantNewSample (
    int sequenceNumber,
    int writerEntityId,
    int sourceParticipantId,
    std::list< HistorySampleEntry * > * historyCache ) [protected]
```

Checks if the received RTPS message contains data from a sample which is not yet stored in the cache

Parameters

<i>sequenceNumber</i>	The sequence number of the received sample.
<i>writerEntityId</i>	The entity ID of the samples writer.
<i>sourceParticipantId</i>	The ID of the participant the sample writer is placed in.
<i>historyCache</i>	The history Cache related to the samples writer at the reader. @ret True if the sample is relevant.

5.27.3.11 sendSampleToSink()

```
void Reader::sendSampleToSink (
    HistorySampleEntry * entry ) [protected]
```

Creates a sample message from the completed sample Entry and sends it to the sample sink.

Parameters

<i>entry</i>	The sample entry to convert to a sample message.
--------------	--

5.27.3.12 updateCache()

```
void Reader::updateCache (
    RTPSEthMessage * rtpsMsg,
    std::list< HistorySampleEntry * > * historyCache ) [protected]
```

Update the cache after receiving a new RTPS message

Parameters

<i>rtpsMsg</i>	The received RTPS message to be to update the corresponding cache with.
<i>historyCache</i>	The cache to update.

5.27.4 Member Data Documentation

5.27.4.1 bestEffort

```
bool Reader::bestEffort [private]
```

Configuration of best effort or reliable communication.

5.27.4.2 deadline

```
simtime_t Reader::deadline [private]
```

The deadline of the samples.

5.27.4.3 entityId

```
int Reader::entityId [private]
```

The own entity ID.

5.27.4.4 frameLatencyInjectionToReception

```
cHistogram Reader::frameLatencyInjectionToReception [private]
```

Frame network latency histogram.

5.27.4.5 frameLatencyInjectionToReceptionVec

```
cOutVector Reader::frameLatencyInjectionToReceptionVec [private]
```

Frame network latency vector.

5.27.4.6 historySizeQoS

```
int Reader::historySizeQoS [private]
```

The size of the history cache.

5.27.4.7 lifespan

```
simtime_t Reader::lifespan [private]
```

The lifespan of the samples.

5.27.4.8 readerHistoryCache

```
ReaderHistory* Reader::readerHistoryCache [private]
```

The history cache.

5.27.4.9 registerEvent

```
cMessage* Reader::registerEvent [private]
```

Self message to register the reader at the participant.

5.27.4.10 SampleLatencySinceGenerationStat

```
cHistogram Reader::SampleLatencySinceGenerationStat [private]
```

Histogram for sample Latency from sample generation.

5.27.4.11 SampleLatencySinceGenerationStatVec

```
cOutVector Reader::SampleLatencySinceGenerationStatVec [private]
```

Vector for sample Latency from sample generation.

5.27.4.12 SampleLatencySinceSampleInjectionStat

```
cHistogram Reader::SampleLatencySinceSampleInjectionStat [private]
```

Histogram for sample Latency from first network injection of the first packet.

5.27.4.13 SampleLatencySinceSampleInjectionStatVec

```
cOutVector Reader::SampleLatencySinceSampleInjectionStatVec [private]
```

Vector for sample Latency from first network injection of the first packet.

5.27.4.14 SampleLatencySinceWriterArrivalStat

```
cHistogram Reader::SampleLatencySinceWriterArrivalStat [private]
```

Histogram for sample Latency from arrival at writer.

5.27.4.15 SampleLatencySinceWriterArrivalStatVec

```
cOutVector Reader::SampleLatencySinceWriterArrivalStatVec [private]
```

Vector for sample Latency from arrival at writer.

5.27.4.16 sourceMac

```
int Reader::sourceMac [private]
```

The own MAC address.

5.27.4.17 streamEthernetPriority

```
int Reader::streamEthernetPriority [private]
```

The Ethernet Priority.

5.27.4.18 transportPriority

```
int Reader::transportPriority [private]
```

The transport priority QoS parameter.

5.28 ReaderHistory Class Reference

```
#include <HistoryEntry.h>
```

Public Member Functions

- [ReaderHistory](#) (int size)
- [~ReaderHistory](#) ()

Public Attributes

- int [nbrStreams](#)
Number of incoming streams.
- int [maxNbrStreams](#)
The maximum history size (maximum number of writers to subscribe)
- std::list< [HistorySampleEntry](#) * > ** [cache](#)
The cache.

5.28.1 Detailed Description

This class represents the history cache for the reader, which can subscribe to multiple writer streams

5.28.2 Constructor & Destructor Documentation

5.28.2.1 ReaderHistory()

```
ReaderHistory::ReaderHistory (  
    int size ) [inline]
```

The constructor

@ param size Initializes the history with a given size

5.28.2.2 ~ReaderHistory()

```
ReaderHistory::~ReaderHistory ( ) [inline]
```

The default constructor

5.28.3 Member Data Documentation

5.28.3.1 cache

```
std::list<HistorySampleEntry*>** ReaderHistory::cache
```

The cache.

5.28.3.2 maxNbrStreams

```
int ReaderHistory::maxNbrStreams
```

The maximum history size (maximum number of writers to subscribe)

5.28.3.3 nbrStreams

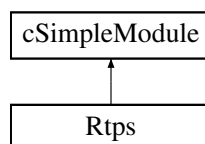
```
int ReaderHistory::nbrStreams
```

Number of incoming streams.

5.29 Rtps Class Reference

```
#include <Rtps.h>
```

Inheritance diagram for Rtps:



Public Member Functions

- [Rtps](#) ()
- virtual [~Rtps](#) ()

Protected Member Functions

- virtual void [initialize](#) () override

5.29.1 Detailed Description

The module containing the RTPS entities

5.29.2 Constructor & Destructor Documentation

5.29.2.1 Rtps()

```
Rtps::Rtps ( ) [inline]
```

The default constructor

5.29.2.2 ~Rtps()

```
Rtps::~~Rtps ( ) [virtual]
```

The default destructor

5.29.3 Member Function Documentation

5.29.3.1 initialize()

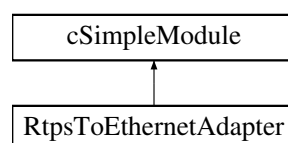
```
void Rtps::initialize ( ) [override], [protected], [virtual]
```

The default initialization method

5.30 RtpsToEthernetAdapter Class Reference

```
#include <RtpsToEthernetAdapter.h>
```

Inheritance diagram for RtpsToEthernetAdapter:



Public Member Functions

- [RtpsToEthernetAdapter](#) ()
- virtual [~RtpsToEthernetAdapter](#) ()

Public Attributes

- unsigned long long [frameSequenceld](#)
Frame sequence ID counter for unique frame identification of the middleware.

Protected Member Functions

- void [sendRtpsPacket](#) (RTPSEthMessage *packet)
- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override
- virtual void [finish](#) () override
- unsigned long long [getNextFrameSequenceld](#) ()

5.30.1 Detailed Description

The adapter fills the EthernetFrame parameters to the RTPSEthMessage, from which it is derived.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 RtpsToEthernetAdapter()

```
RtpsToEthernetAdapter::RtpsToEthernetAdapter ( )
```

Default constructor

5.30.2.2 ~RtpsToEthernetAdapter()

```
RtpsToEthernetAdapter::~~RtpsToEthernetAdapter ( ) [virtual]
```

Default destructor

5.30.3 Member Function Documentation

5.30.3.1 finish()

```
void RtpsToEthernetAdapter::finish ( ) [override], [protected], [virtual]
```

Returns the number of send frames

5.30.3.2 getNextFrameSequenceId()

```
unsigned long long RtpsToEthernetAdapter::getNextFrameSequenceId ( ) [protected]
```

Return and increment the frame sequence ID

5.30.3.3 handleMessage()

```
void RtpsToEthernetAdapter::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Initializes the relevant context of the derived EthernetFrame to the packet.

Parameters

<i>msg</i>	represents the outgoing msg to adapt for EthernetFrame compliance.
------------	--

5.30.3.4 initialize()

```
void RtpsToEthernetAdapter::initialize ( ) [override], [protected], [virtual]
```

Empty initialization

5.30.3.5 sendRtpsPacket()

```
void RtpsToEthernetAdapter::sendRtpsPacket (
    RTPSEthMessage * packet ) [protected]
```

Put the information from the middleware to configure the Ethernet message.

Parameters

<i>packet</i>	the outgoing packet to send and to configure for Ethernet transmission.
---------------	---

5.30.4 Member Data Documentation

5.30.4.1 frameSequenceld

```
unsigned long long RtpsToEthernetAdapter::frameSequenceId
```

Frame sequence ID counter for unique frame identification of the middleware.

5.31 SampleFragment Class Reference

```
#include <HistoryEntry.h>
```

Public Member Functions

- [SampleFragment](#) ()
- [~SampleFragment](#) ()

Public Attributes

- int [fragmentSequenceNumber](#)
The fragment sequence number.
- int [fragmentSize](#)
The size of the fragment.
- [HistorySampleEntry](#) * [sampleReference](#)
Reference to the corresponding sample.
- bool [send](#)
True if the fragment has already been send and has not been negatively acknowledged (writer side)
- bool [acked](#)
True if the fragment has been positively acknowledged (writer side)
- bool [received](#)
True if the fragment has been received (reader side)
- [simtime_t](#) [sendTime](#)
The last time the fragment was send by the writer.

5.31.1 Detailed Description

Represents a fragment and stores its send/receive state

5.31.2 Constructor & Destructor Documentation

5.31.2.1 SampleFragment()

```
SampleFragment::SampleFragment ( ) [inline]
```

The default constructor

5.31.2.2 ~SampleFragment()

```
SampleFragment::~~SampleFragment ( ) [inline]
```

The default destructor

5.31.3 Member Data Documentation

5.31.3.1 acked

```
bool SampleFragment::acked
```

True if the fragment has been positively acknowledged (writer side)

5.31.3.2 fragmentSequenceNumber

```
int SampleFragment::fragmentSequenceNumber
```

The fragment sequence number.

5.31.3.3 fragmentSize

```
int SampleFragment::fragmentSize
```

The size of the fragment.

5.31.3.4 received

```
bool SampleFragment::received
```

True if the fragment has been received (reader side)

5.31.3.5 sampleReference

```
HistorySampleEntry* SampleFragment::sampleReference
```

Reference to the corresponding sample.

5.31.3.6 send

```
bool SampleFragment::send
```

True if the fragment has already been send and has not been negatively acknowledged (writer side)

5.31.3.7 sendTime

```
simtime_t SampleFragment::sendTime
```

The last time the fragment was send by the writer.

5.32 SampleLatencyStatistics Class Reference

```
#include <ddsSampleLatencyStatistics.h>
```

Public Member Functions

- [SampleLatencyStatistics](#) ()
- virtual [~SampleLatencyStatistics](#) ()
- void [addMessage](#) (int participantId, int readerId, int globalStreamId, simtime_t delay)
- void [report](#) (const omnetpp::simtime_t &latency, int participantId, int readerId, int globalStreamId)
- void [finish](#) ()

Private Attributes

- [VectorStatistics3DMap](#) * [sampleLatencyStatisticsVector](#) = nullptr
- [HistogramStatistics3DMap](#) * [sampleLatencyStatisticsHistogram](#) = nullptr

5.32.1 Detailed Description

This class is gathering E2E latency statistics per DDS samples As DDS samples are virtual entities (no module) they must be gathered by [StatisticManager](#).

5.32.2 Constructor & Destructor Documentation

5.32.2.1 SampleLatencyStatistics()

```
SampleLatencyStatistics::SampleLatencyStatistics ( )
```

default constructor, nothing to do

5.32.2.2 ~SampleLatencyStatistics()

```
SampleLatencyStatistics::~~SampleLatencyStatistics ( ) [virtual]
```

default destructor, nothing to do

Destructor.

5.32.3 Member Function Documentation

5.32.3.1 addMessage()

```
void SampleLatencyStatistics::addMessage (
    int participantId,
    int readerId,
    int globalStreamId,
    simtime_t delay )
```

stores the end to end statistics for the samples

Parameters

<i>participantId</i>	- DDS participant ID
<i>readerId</i>	- DDS reader ID
<i>globalStreamId</i>	- id of the stream
<i>delay</i>	- end to end latency for the sample

5.32.3.2 finish()

```
void SampleLatencyStatistics::finish ( )
```

Finishes statistics gathering for End-2-End latencies Vectors will be deleted Histograms: are firstly recorded (save to file) and next deleted

5.32.3.3 report()

```
void SampleLatencyStatistics::report (
    const omnetpp::simtime_t & latency,
    int participantId,
    int readerId,
    int globalStreamId )
```

Stores the results into the vectors Helper function

Parameters

<i>latency</i>	
<i>participantId</i>	
<i>readerId</i>	
<i>globalStreamId</i>	

5.32.4 Member Data Documentation

5.32.4.1 sampleLatencyStatisticsHistogram

```
HistogramStatistics3DMap* SampleLatencyStatistics::sampleLatencyStatisticsHistogram = nullptr
[private]
```

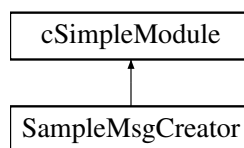
5.32.4.2 sampleLatencyStatisticsVector

```
VectorStatistics3DMap* SampleLatencyStatistics::sampleLatencyStatisticsVector = nullptr [private]
```

5.33 SampleMsgCreator Class Reference

```
#include <SampleMsgCreator.h>
```

Inheritance diagram for SampleMsgCreator:



Public Member Functions

- [SampleMsgCreator](#) ()
- virtual [~SampleMsgCreator](#) ()

Protected Member Functions

- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override
- virtual void [finish](#) () override
- int64_t [getNextSampleSequenceNumberId](#) ()

Protected Attributes

- `int64_t sampleSequenceNumberId`
The frame sequence ID unique for a given globalStreamId.

5.33.1 Detailed Description

Create and forwards an RTPS sample after stimulation from a source module

5.33.2 Constructor & Destructor Documentation

5.33.2.1 SampleMsgCreator()

```
SampleMsgCreator::SampleMsgCreator ( )
```

Empty default constructor

5.33.2.2 ~SampleMsgCreator()

```
SampleMsgCreator::~~SampleMsgCreator ( ) [virtual]
```

Empty default destructor

5.33.3 Member Function Documentation

5.33.3.1 finish()

```
void SampleMsgCreator::finish ( ) [override], [protected], [virtual]
```

Overridden method, called at the end of the simulation

5.33.3.2 getNextSampleSequenceNumberId()

```
int64_t SampleMsgCreator::getNextSampleSequenceNumberId ( ) [inline], [protected]
```

Calculates the unique sequence number of the writer streams samples.

5.33.3.3 handleMessage()

```
void SampleMsgCreator::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Overridden method, triggered by a traffic source and creates one Sample each time it is called. The sample is then passed to the middleware.

Parameters

<i>msg</i>	Each incoming message is interpreted as a stimulation
------------	---

5.33.3.4 initialize()

```
void SampleMsgCreator::initialize ( ) [override], [protected], [virtual]
```

Overwritten method, initializes module after its creation.

5.33.4 Member Data Documentation**5.33.4.1 sampleSequenceNumberId**

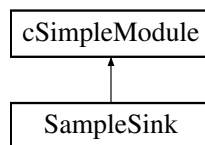
```
int64_t SampleMsgCreator::sampleSequenceNumberId [protected]
```

The frame sequence ID unique for a given globalStreamId.

5.34 SampleSink Class Reference

```
#include <SampleSink.h>
```

Inheritance diagram for SampleSink:

**Protected Member Functions**

- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override

Private Attributes

- int [myMac](#)
One MAC Address of the [Endpoint](#).

5.34.1 Detailed Description

[SampleSink](#) that gathers statistics for incoming samples.

5.34.2 Member Function Documentation

5.34.2.1 handleMessage()

```
void SampleSink::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Handles incoming messages

Parameters

<i>msg</i>	The incoming sample
------------	---------------------

5.34.2.2 initialize()

```
void SampleSink::initialize ( ) [override], [protected], [virtual]
```

5.34.3 Member Data Documentation

5.34.3.1 myMac

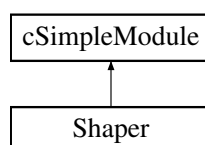
```
int SampleSink::myMac [private]
```

One MAC Address of the [Endpoint](#).

5.35 Shaper Class Reference

```
#include <Shaper.h>
```

Inheritance diagram for Shaper:



Public Member Functions

- [Shaper](#) ()
- virtual [~Shaper](#) ()

Protected Member Functions

- void [send_from_queue](#) ()
- virtual void [finish](#) () override
- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override

Private Attributes

- bool [shaperEnabled](#)
Enable/Disable the shaper.
- simtime_t [shapingDuration](#)
Duration of shaping, i.e. the time between two consecutive shaped messages.
- cMessage * [nextMsgReady](#)
- std::list< RTPSEthMessage * > [rtpsMsgQueue](#)

5.35.1 Constructor & Destructor Documentation

5.35.1.1 Shaper()

```
Shaper::Shaper ( ) [inline]
```

Default constructor

5.35.1.2 ~Shaper()

```
Shaper::~Shaper ( ) [virtual]
```

Default destructor

5.35.2 Member Function Documentation

5.35.2.1 finish()

```
void Shaper::finish ( ) [override], [protected], [virtual]
```

5.35.2.2 handleMessage()

```
void Shaper::handleMessage (
    cMessage * msg )    [override], [protected], [virtual]
```

5.35.2.3 initialize()

```
void Shaper::initialize ( )    [override], [protected], [virtual]
```

5.35.2.4 send_from_queue()

```
void Shaper::send_from_queue ( )    [protected]
```

Send the message from the queue

5.35.3 Member Data Documentation

5.35.3.1 nextMsgReady

```
cMessage* Shaper::nextMsgReady    [private]
```

5.35.3.2 rtpsMsgQueue

```
std::list<RTPSEthMessage*> Shaper::rtpsMsgQueue    [private]
```

5.35.3.3 shaperEnabled

```
bool Shaper::shaperEnabled    [private]
```

Enable/Disable the shaper.

5.35.3.4 shapingDuration

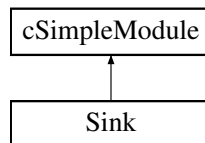
```
simtime_t Shaper::shapingDuration [private]
```

Duration of shaping, i.e. the time between two consecutive shaped messages.

5.36 Sink Class Reference

```
#include <Sink.h>
```

Inheritance diagram for Sink:



Protected Member Functions

- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override

Private Member Functions

- virtual bool [checkDestinationsMAC](#) (EthernetFrame *ethMsg)
check if frame arrived to the correct destination based on MAC
- void [processEthernetFrameAtDestination](#) (EthernetFrame *ethMsg)
gather end2end latency and return error if latency is negative
- void [checkForMissingFrame](#) (EthernetFrame *ethMsg)
check if there are lost frames evntl. gather statistics
- void [finish](#) ()
finishing statistics gathering

Private Attributes

- int [macAddress](#)
MAC address of the sink.
- std::map< int, int > [lostFrameCounters](#)
counter used to track lost frames
- std::map< int, int > [trackMissingDataMap](#)
here we check whether all data was delivered

5.36.1 Detailed Description

[Sink](#) module for the Ethernet frames (which are send directly without the DDS) Mainly used to gathers statistics.

5.36.2 Member Function Documentation

5.36.2.1 checkDestinationsMAC()

```
bool Sink::checkDestinationsMAC (
    EthernetFrame * ethMsg ) [inline], [private], [virtual]
```

check if frame arrived to the correct destination based on MAC

5.36.2.2 checkForMissingFrame()

```
void Sink::checkForMissingFrame (
    EthernetFrame * ethMsg ) [private]
```

check if there are lost frames evntl. gather statistics

5.36.2.3 finish()

```
void Sink::finish ( ) [private]
```

finishing statistics gathering

5.36.2.4 handleMessage()

```
void Sink::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Casts the msg to the EthernetFrame and processes it.

Parameters

<i>msg</i>	
------------	--

5.36.2.5 initialize()

```
void Sink::initialize ( ) [override], [protected], [virtual]
```

Constructor

5.36.2.6 processEthernetFrameAtDestination()

```
void Sink::processEthernetFrameAtDestination (
    EthernetFrame * ethMsg ) [private]
```

gather end2end latency and return error if latency is negative

5.36.3 Member Data Documentation

5.36.3.1 lostFrameCounters

```
std::map<int, int> Sink::lostFrameCounters [private]
```

counter used to track lost frames

5.36.3.2 macAddress

```
int Sink::macAddress [private]
```

MAC address of the sink.

5.36.3.3 trackMissingDataMap

```
std::map<int, int> Sink::trackMissingDataMap [private]
```

here we check whether all data was delivered

5.37 StatisticManager Class Reference

```
#include <StatisticManager.h>
```

Public Member Functions

- virtual [~StatisticManager](#) ()
- [E2ELatencyStatistics](#) * [getDelayStatistics](#) ()
- [FramesLostStatistics](#) * [getFramesLostStatistics](#) ()
- [SampleLatencyStatistics](#) * [getSampleLatencyStatistics](#) ()
- [LinkTransmissionStatistics](#) * [getLinkTransmissionStatistics](#) ()
- [TimeAwareShaperSegmentStatistic](#) * [getTimeAwareShaperStatistics](#) ()
- virtual void [finishAll](#) ()

Static Public Member Functions

- static [StatisticManager](#) & [getInstance](#) ()

Private Member Functions

- [StatisticManager](#) ()
this is singleton, make constructor private

Private Attributes

- bool [hasFinishedAll](#) = false
variable to indicate if running finish will be necessary
- [E2ELatencyStatistics](#) * [pathDelayStats](#) = 0
object to store E2E latencies
- [FramesLostStatistics](#) * [framesLostStats](#) = 0
object to store lost frames statistics
- [SampleLatencyStatistics](#) * [sampleLatencyStats](#) = 0
object to store DDS samples latencies
- [LinkTransmissionStatistics](#) * [linkTransmission](#) = 0
object to store link transmission statistics
- [TimeAwareShaperSegmentStatistic](#) * [tsnTASStatistics](#)
object to store TAS statistics

5.37.1 Detailed Description

[StatisticManager](#) introduces an virtual module for statistic gathering. It is necessary whenever we are gathering statistics for entities which does not have representation in form of an Omnet++ module. For instance, DDS samples from specific reader/writer pairs, or latencies for particular stream. Omnet++ does not support such statistics natively as framework was oriented towards average / performance optimized results.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 ~StatisticManager()

```
StatisticManager::~StatisticManager ( ) [virtual]
```

Destructor, should stay empty

5.37.2.2 StatisticManager()

```
StatisticManager::StatisticManager ( ) [private]
```

this is singleton, make constructor private

5.37.3 Member Function Documentation

5.37.3.1 finishAll()

```
void StatisticManager::finishAll ( ) [virtual]
```

finishes off all individual collections (see individual functions), frees memory and resets pointer

5.37.3.2 getDelayStatistics()

```
E2ELatencyStatistics * StatisticManager::getDelayStatistics ( )
```

Function to return an object storing the delay statistics. The statistics are stored for End-2-End communication within the network. If an object does not exist it declares a new one.

Returns

5.37.3.3 getFramesLostStatistics()

```
FramesLostStatistics * StatisticManager::getFramesLostStatistics ( )
```

Function to return an object storing the lost frames statistics. If an object does not exist it declares a new one.

Returns

5.37.3.4 getInstance()

```
static StatisticManager& StatisticManager::getInstance ( ) [inline], [static]
```

Returns the singleton instance of the collector, which allows to gather statistics from different simulation modules in one object.

5.37.3.5 getLinkTransmissionStatistics()

```
LinkTransmissionStatistics * StatisticManager::getLinkTransmissionStatistics ( )
```

Function to return an object storing the transmission time of an Ethernet frame from one module to the next module.

Returns

5.37.3.6 getSampleLatencyStatistics()

```
SampleLatencyStatistics * StatisticManager::getSampleLatencyStatistics ( )
```

Function to return an object storing the latency of whole DDS samples. Note, that each sample may be constructed from multiple frames. If an object does not exist it declares a new one.

Returns

5.37.3.7 getTimeAwareShaperStatistics()

```
TimeAwareShaperSegmentStatistic * StatisticManager::getTimeAwareShaperStatistics ( )
```

Function to return an object of the time aware shaper (TAS) segments. The period of TAS is divided in four segments. For evaluation these segments are represented as statistics.

5.37.4 Member Data Documentation

5.37.4.1 framesLostStats

```
FramesLostStatistics* StatisticManager::framesLostStats = 0 [private]
```

object to store lost frames statistics

5.37.4.2 hasFinishedAll

```
bool StatisticManager::hasFinishedAll = false [private]
```

variable to indicate if running finish will be necessary

5.37.4.3 linkTransmission

```
LinkTransmissionStatistics* StatisticManager::linkTransmission = 0 [private]
```

object to store link transmission statistics

5.37.4.4 pathDelayStats

```
E2ELatencyStatistics* StatisticManager::pathDelayStats = 0 [private]
```

object to store E2E latencies

5.37.4.5 sampleLatencyStats

```
SampleLatencyStatistics* StatisticManager::sampleLatencyStats = 0 [private]
```

object to store DDS samples latencies

5.37.4.6 tsnTASStatistics

```
TimeAwareShaperSegmentStatistic* StatisticManager::tsnTASStatistics [private]
```

object to store TAS statistics

5.38 TimeAwareShaperSegmentStatistic Class Reference

```
#include <TimeAwareShaperSegmentStatistic.h>
```

Public Member Functions

- [TimeAwareShaperSegmentStatistic](#) ()
- virtual [~TimeAwareShaperSegmentStatistic](#) ()
- void [criticalSendingSegmentStarts](#) (int switchMac, int portIndex)
- void [criticalGuardbandSegmentStarts](#) (int switchMac, int portIndex)
- void [noncriticalSendingSegmentStarts](#) (int switchMac, int portIndex)
- void [noncriticalGuardbandSegmentStarts](#) (int switchMac, int portIndex)
- virtual void [finish](#) ()

Private Attributes

- [VectorStatistics2DMap](#) * [tasPortTimeSegments](#) = 0

5.38.1 Detailed Description

Time aware shaper (IEEE802.1Qbv) is a circuit-switched arbitration. To evaluate the arbiter the TimeAwareShaper↔SegmentStatistics draws the four time segments:

1. Critical sending, 2. critical guardband, 3. noncritical sending,
2. noncritical guardband.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 TimeAwareShaperSegmentStatistic()

```
TimeAwareShaperSegmentStatistic::TimeAwareShaperSegmentStatistic ( )
```

Auto-generated constructor, nothing to do here

5.38.2.2 ~TimeAwareShaperSegmentStatistic()

```
TimeAwareShaperSegmentStatistic::~~TimeAwareShaperSegmentStatistic ( ) [virtual]
```

Auto-generated destructor, nothing to do here

5.38.3 Member Function Documentation

5.38.3.1 criticalGuardbandSegmentStarts()

```
void TimeAwareShaperSegmentStatistic::criticalGuardbandSegmentStarts (
    int switchMac,
    int portIndex )
```

This function is called when the TAS critical guardband segment starts. For each switch and port within the switch it is set.

Parameters

<i>switchMac</i>	integer value of switch MAC address to identify the switch
<i>portIndex</i>	integer value of the port index in the switch

5.38.3.2 criticalSendingSegmentStarts()

```
void TimeAwareShaperSegmentStatistic::criticalSendingSegmentStarts (
    int switchMac,
    int portIndex )
```

This function is called when the TAS critical sending segment starts. For each switch and port within the switch it is set.

Parameters

<i>switchMac</i>	integer value of switch MAC address to identify the switch
<i>portIndex</i>	integer value of the port index in the switch

5.38.3.3 finish()

```
void TimeAwareShaperSegmentStatistic::finish ( ) [virtual]
```

finishes off all individual collections (see individual functions), frees memory and resets pointer

5.38.3.4 noncriticalGuardbandSegmentStarts()

```
void TimeAwareShaperSegmentStatistic::noncriticalGuardbandSegmentStarts (
    int switchMac,
    int portIndex )
```

This function is called when the TAS noncritical guardband segment starts. For each switch and port within the switch it is set.

Parameters

<i>switchMac</i>	integer value of switch MAC address to identify the switch
<i>portIndex</i>	integer value of the port index in the switch

5.38.3.5 noncriticalSendingSegmentStarts()

```
void TimeAwareShaperSegmentStatistic::noncriticalSendingSegmentStarts (
```

```

    int switchMac,
    int portIndex )

```

This function is called when the TAS noncritical sending segment starts. For each switch and port within the switch it is set.

Parameters

<i>switchMac</i>	integer value of switch MAC address to identify the switch
<i>portIndex</i>	integer value of the port index in the switch

5.38.4 Member Data Documentation

5.38.4.1 tasPortTimeSegments

```

VectorStatistics2DMap* TimeAwareShaperSegmentStatistic::tasPortTimeSegments = 0 [private]

```

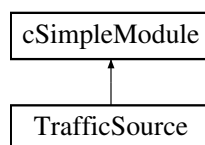
5.39 TrafficSource Class Reference

```

#include <TrafficSource.h>

```

Inheritance diagram for TrafficSource:



Public Member Functions

- [TrafficSource](#) ()
- virtual [~TrafficSource](#) ()

Protected Member Functions

- virtual void [initialize](#) ()
- virtual void [handleMessage](#) (cMessage *msg)
- void [sendPacket](#) ()
- void [scheduleNextBurst](#) (simtime_t offset)
- void [scheduleNextIntraBurst](#) ()

Private Attributes

- bool [generatorEnable](#)
Controls if the generator should be active.
- int [globalStreamId](#)
The global stream Id passed to Ethernet messages if created in the following helper module.
- cMessage * [intraBurstMsg](#)
The intra-burst self event. Signals that a burst is currently scheduled (blocking until next event)
- cMessage * [interBurstMsg](#)
The inter-burst self event. Signals that the next activation to send out is ready.
- int [pendingBursts](#)
The number of pending bursts.
- int [pendingIntraBurstSize](#)
The number of pending activations of the current burst.

5.39.1 Detailed Description

Implementation of the stimuli-generating component based on a periodic burst model

5.39.2 Constructor & Destructor Documentation

5.39.2.1 TrafficSource()

```
TrafficSource::TrafficSource ( )
```

5.39.2.2 ~TrafficSource()

```
TrafficSource::~~TrafficSource ( ) [virtual]
```

5.39.3 Member Function Documentation

5.39.3.1 handleMessage()

```
void TrafficSource::handleMessage (
    cMessage * msg ) [protected], [virtual]
```

Receiving self messages to control the output of source packets

Parameters

<i>msg</i>	A self message to start a new burst or a single transmission inside of a burst
------------	--

5.39.3.2 initialize()

```
void TrafficSource::initialize ( ) [protected], [virtual]
```

Initialize the module

5.39.3.3 scheduleNextBurst()

```
void TrafficSource::scheduleNextBurst (
    simtime_t offset ) [protected]
```

Creates the next generation message according with the specified parameters. Is only called after interBurstMsg self event. Schedules the next event in a burst.

Parameters

<i>offset</i>	Is only set the first time this method is called
---------------	--

5.39.3.4 scheduleNextIntraBurst()

```
void TrafficSource::scheduleNextIntraBurst ( ) [protected]
```

Starts a burst if no burst is currently active

5.39.3.5 sendPacket()

```
void TrafficSource::sendPacket ( ) [protected]
```

Sends a packet and reduces the bust count. Also activates pending bursts if present.

5.39.4 Member Data Documentation

5.39.4.1 generatorEnable

```
bool TrafficSource::generatorEnable [private]
```

Controls if the generator should be active.

5.39.4.2 globalStreamId

```
int TrafficSource::globalStreamId [private]
```

The global stream Id passed to Ethernet messages if created in the following helper module.

5.39.4.3 interBurstMsg

```
cMessage* TrafficSource::interBurstMsg [private]
```

The inter-burst self event. Signals that the next activation to send out is ready.

5.39.4.4 intraBurstMsg

```
cMessage* TrafficSource::intraBurstMsg [private]
```

The intra-burst self event. Signals that a burst is currently scheduled (blocking until next event)

5.39.4.5 pendingBursts

```
int TrafficSource::pendingBursts [private]
```

The number of pending bursts.

5.39.4.6 pendingIntraBurstSize

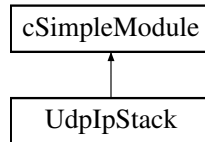
```
int TrafficSource::pendingIntraBurstSize [private]
```

The number of pending activations of the current burst.

5.40 UdpIpStack Class Reference

```
#include <UdpIpStack.h>
```

Inheritance diagram for UdpIpStack:



Public Member Functions

- [UdpIpStack\(\)](#)
- virtual [~UdpIpStack\(\)](#)
- void [initialize\(\)](#) override
- void [handleMessage\(\)](#) (cMessage *msg) override

5.40.1 Detailed Description

Passes the data from the middleware and pure Ethernet generators to the ethernet network ports. Passes the data from the Ethernet network ports to either the middleware or the pure Ethernet sink.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 UdpIpStack()

```
UdpIpStack::UdpIpStack ( )
```

Default constructor

5.40.2.2 ~UdpIpStack()

```
UdpIpStack::~~UdpIpStack ( ) [virtual]
```

Default destructor

5.40.3 Member Function Documentation

5.40.3.1 handleMessage()

```
void UdpIpStack::handleMessage (
    cMessage * msg ) [override]
```

forwarding the messages between the Ethernet network ports and the middleware and pure Ethernet generators/sink

Parameters

<i>msg</i>	The message to be forwarded
------------	-----------------------------

5.40.3.2 initialize()

```
void UdpIpStack::initialize ( ) [override]
```

Empty initialization function

5.41 ValidationHandler Class Reference

```
#include <ValidationHandler.h>
```

Static Public Member Functions

- static void [ThrowError](#) (const char *message)
- template<typename T >
static void [validateInputBounds](#) (T input, T min, T max, const char *message)

5.41.1 Member Function Documentation**5.41.1.1 ThrowError()**

```
static void ValidationHandler::ThrowError (
    const char * message ) [inline], [static]
```

5.41.1.2 validateInputBounds()

```
template<typename T >
static void ValidationHandler::validateInputBounds (
    T input,
    T min,
    T max,
    const char * message ) [inline], [static]
```

Validates the input is within the minimum and maximum bounds. Throws an exception if the input is out of bounds.

5.42 VectorStatistics2DMap Class Reference

```
#include <VectorStatistics2DMap.h>
```

Public Member Functions

- [VectorStatistics2DMap](#) (const char *[prefix](#), const char *[modulePrefix](#), const char *[submodulePrefix](#))
- virtual [~VectorStatistics2DMap](#) ()
- void [finish](#) ()
- void [addInteger](#) (int moduleID, int submoduleID, int value)
- void [accumulateInteger](#) (int moduleID, int submoduleID, int value)
- void [addLatency](#) (int moduleID, int submoduleID, simtime_t value)

Private Member Functions

- std::pair< cOutVector *, long > & [getEntry](#) (int module_ID, int submodule_ID)

Private Attributes

- std::map< int, std::map< int, std::pair< cOutVector *, long > > * > * [storedMeasurements](#) = 0
vector containing the stored results
- const char * [prefix](#)
prefix is a string which stores information allowing to indentify the type of statistics which is gathered e.g. E2E latency, packet drop etc.
- const char * [modulePrefix](#)
modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID
- const char * [submodulePrefix](#)
submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID
- const char * [valuePrefix](#)
value prefix is a string allowing to identify the type of stored data

5.42.1 Detailed Description

Helper class used for storing the vector statistics for objects which do not have physical modules e.g. streams or DDS samples with two parameters describing each entry e.g. moduleID and submoduleID

5.42.2 Constructor & Destructor Documentation

5.42.2.1 VectorStatistics2DMap()

```
VectorStatistics2DMap::VectorStatistics2DMap (
    const char * prefix,
    const char * modulePrefix,
    const char * submodulePrefix )
```

Constructor of the class Generates the string using the parameter variables which is later used to describe/identify measurement results

Parameters

<i>prefix</i>	
<i>module_fix</i>	
<i>submodule_fix</i>	

5.42.2.2 ~VectorStatistics2DMap()

```
VectorStatistics2DMap::~~VectorStatistics2DMap ( ) [virtual]
```

Destructor of the class, nothing to do here

5.42.3 Member Function Documentation**5.42.3.1 accumulateInteger()**

```
void VectorStatistics2DMap::accumulateInteger (
    int moduleID,
    int submoduleID,
    int value )
```

Function firstly accumulates the value to the last one and later stores the integer measurement results into the statistics vector

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. streamID
<i>value</i>	- the measured value, must be an integer

5.42.3.2 addInteger()

```
void VectorStatistics2DMap::addInteger (
    int moduleID,
    int submoduleID,
    int value )
```

Function stores the integer measurement results into the statistics vector

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. streamID
<i>value</i>	- the measured value, must be an integer

5.42.3.3 addLatency()

```
void VectorStatistics2DMap::addLatency (
    int moduleID,
    int submpduleID,
    simtime_t value )
```

Function stores the simulation latency measurement results into the statistics vector

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. streamID
<i>value</i>	- the measured value, must be an simtime_t

5.42.3.4 finish()

```
void VectorStatistics2DMap::finish ( )
```

Finish function is used to save the statistics at the end of simulation

5.42.3.5 getEntry()

```
pair< cOutVector *, long > & VectorStatistics2DMap::getEntry (
    int module_ID,
    int submodule_ID ) [private]
```

Function which returns an entry identified by parameters

Parameters

<i>module_ID</i>	- module where measurement has been takes e.g. bufferID
<i>submodule_ID</i>	- module for which measurmenet has been take e.g. streamID

Returns

5.42.4 Member Data Documentation

5.42.4.1 modulePrefix

```
const char* VectorStatistics2DMap::modulePrefix [private]
```

modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID

5.42.4.2 prefix

```
const char* VectorStatistics2DMap::prefix [private]
```

prefix is a string which stores information allowing to indentify the type of statistics which is gathered e.g. E2E latency, packet drop etc.

5.42.4.3 storedMeasurements

```
std::map<int, std::map<int, std::pair<cOutVector*, long> >*>* VectorStatistics2DMap::storedMeasurements = 0 [private]
```

vector containing the stored results

5.42.4.4 submodulePrefix

```
const char* VectorStatistics2DMap::submodulePrefix [private]
```

submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID

5.42.4.5 valuePrefix

```
const char* VectorStatistics2DMap::valuePrefix [private]
```

value prefix is a string allowing to identify the type of stored data

5.43 VectorStatistics3DMap Class Reference

```
#include <VectorStatistics3DMap.h>
```

Public Member Functions

- [VectorStatistics3DMap](#) (const char *[prefix](#), const char *[modulePrefix](#), const char *[submodulePrefix](#), const char *[valuePrefix](#))
- virtual [~VectorStatistics3DMap](#) ()
- void [finish](#) ()
- void [addInteger](#) (int moduleID, int submoduleID, int valueID, int value)
- void [accumulateInteger](#) (int moduleID, int submoduleID, int valueID, int value)
- void [addLatency](#) (int moduleID, int submoduleID, int valueID, simtime_t value)
- void [addLatency](#) (int moduleID, int submoduleID, int valueID, const std::string &valueIDStr, simtime_t value)

Private Member Functions

- pair< cOutVector *, long > & [getEntry](#) (int moduleID, int submoduleID, int valueID)
- pair< cOutVector *, long > & [getEntry](#) (int moduleID, int submoduleID, int valueID, const std::string &valueIDStr)

Private Attributes

- const char * [prefix](#) = 0
prefix is a string which stores information allowing to identify the type of statistics which is gathered e.g. E2E latency, packet drop etc.
- const char * [modulePrefix](#) = 0
modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID
- const char * [submodulePrefix](#) = 0
submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID
- const char * [valuePrefix](#) = 0
value prefix is a string allowing to identify the type of stored data
- map< int, map< int, map< int, pair< cOutVector *, long > > * > * > * [storedMeasurements](#)
histogram containing the stored results, three dimensional

Static Private Attributes

- static const std::string [empty](#)

5.43.1 Constructor & Destructor Documentation

5.43.1.1 VectorStatistics3DMap()

```
VectorStatistics3DMap::VectorStatistics3DMap (
    const char * prefix,
    const char * modulePrefix,
    const char * submodulePrefix,
    const char * valuePrefix )
```

Constructor of the class Stores parameters which will be later used for the creation of the string which is used to identify results.

Parameters

<i>prefix</i>	
<i>modulePrefix</i>	
<i>submodulePrefix</i>	
<i>valuePrefix</i>	

5.43.1.2 ~VectorStatistics3DMap()

```
VectorStatistics3DMap::~~VectorStatistics3DMap ( ) [virtual]
```

Destructor, nothing to do

5.43.2 Member Function Documentation**5.43.2.1 accumulateInteger()**

```
void VectorStatistics3DMap::accumulateInteger (
    int moduleID,
    int submoduleID,
    int valueID,
    int value )
```

Function stores the integer measurement results into the statistics vector

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. bufferID
<i>valueID</i>	- id of the stream/streams for which statistics has been taken e.g. priority, streamID
<i>value</i>	- the measured value, must be an integer

5.43.2.2 addInteger()

```
void VectorStatistics3DMap::addInteger (
    int moduleID,
    int submoduleID,
    int valueID,
    int value )
```

Function stores the integer measurement results into the statistics vector

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. bufferID
<i>valueID</i>	- id of the stream/streams for which statistics has been taken e.g. priority, streamID
<i>value</i>	- the measured value, must be an integer

5.43.2.3 addLatency() [1/2]

```
void VectorStatistics3DMap::addLatency (
    int moduleID,
    int submoduleID,
    int valueID,
    const std::string & valueIDStr,
    simtime_t value )
```

Function stores the integer measurement results into the statistics vector

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. bufferID
<i>valueID</i>	- id of the stream/streams for which statistics has been taken e.g. priority, streamID
<i>valueIDStr</i>	- substitutes the valueID if defined
<i>value</i>	- the measured value, must be an simtime_t

5.43.2.4 addLatency() [2/2]

```
void VectorStatistics3DMap::addLatency (
    int moduleID,
    int submoduleID,
    int valueID,
    simtime_t value )
```

Function firstly accumulates the value to the last one and later stores the integer measurement results into the statistics vector

Parameters

<i>moduleID</i>	- id of the module where the measurement has been taken
<i>submoduleID</i>	- id of the module for which the statistics has been taken e.g. bufferID
<i>valueID</i>	- id of the stream/streams for which statistics has been taken e.g. priority, streamID
<i>value</i>	- the measured value, must be an simtime_t

5.43.2.5 finish()

```
void VectorStatistics3DMap::finish ( )
```

Finish function is used to clear & save the statistics at the end of simulation

5.43.2.6 getEntry() [1/2]

```
pair< cOutVector *, long > & VectorStatistics3DMap::getEntry (
    int moduleID,
    int submoduleID,
    int valueID ) [private]
```

5.43.2.7 getEntry() [2/2]

```
pair<cOutVector*, long>& VectorStatistics3DMap::getEntry (
    int moduleID,
    int submoduleID,
    int valueID,
    const std::string & valueIDStr ) [private]
```

5.43.3 Member Data Documentation

5.43.3.1 empty

```
const string VectorStatistics3DMap::empty [static], [private]
```

5.43.3.2 modulePrefix

```
const char* VectorStatistics3DMap::modulePrefix = 0 [private]
```

modulePrefix is a string which denotes the module where statistics are gathered e.g. sink with a selected ID

5.43.3.3 prefix

```
const char* VectorStatistics3DMap::prefix = 0 [private]
```

prefix is a string which stores information allowing to identify the type of statistics which is gathered e.g. E2E latency, packet drop etc.

5.43.3.4 storedMeasurements

```
map<int, map<int, map<int, pair<cOutVector*, long> >>*> VectorStatistics3DMap::stored←
Measurements [private]
```

histogram containing the stored results, thre dimmensional

5.43.3.5 submodulePrefix

```
const char* VectorStatistics3DMap::submodulePrefix = 0 [private]
```

submodule is a string which prefix denotes for which component the statistics are gathered, it could be streamID or buffer ID

5.43.3.6 valuePrefix

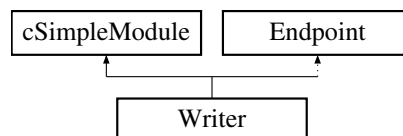
```
const char* VectorStatistics3DMap::valuePrefix = 0 [private]
```

value prefix is a string allowing to identify the type of stored data

5.44 Writer Class Reference

```
#include <Writer.h>
```

Inheritance diagram for Writer:



Public Member Functions

- [Writer](#) ()
- virtual [~Writer](#) ()

Protected Member Functions

- virtual void [initialize](#) () override
- virtual void [handleMessage](#) (cMessage *msg) override
- virtual void [finish](#) () override
- void [registerWriter](#) ()
- bool [addNextFragmentToSendList](#) ()
- void [sendMessage](#) ()
- void [checkLifespanOfSamples](#) ()
- RTPSEthMessage * [createRtpsMsgFromFragment](#) (SampleFragment *sampleFragment)

Private Attributes

- cMessage * [sendEvent](#)
Send event self message.
- cMessage * [registerEvent](#)
Self event to trigger the registration of the writer at the participant.
- int [sourceMac](#)
The own MAC address.
- int [destinationMac](#)
The MAC address of the remote reader(s)
- int [entityId](#)
The own entity ID.
- int [destinationEntityId](#)
The entity ID of the remote reader(s)
- int [streamEthernetPriority](#)
The Ethernet Priority indicator.
- int [globalStreamId](#)
The writers global stream ID.
- simtime_t [shapingDuration](#)
Shaping duration for messages of the individual writer.
- int [historySizeQoS](#)
The size of the history cache.
- simtime_t [lifespan](#)
The lifespan duration.
- bool [bestEffort](#)
Choose best effort or reliable communication.
- int [fragmentSize](#)
The fragment size.
- int [transportPriority](#)
The transport priority of the writer.
- std::list< [HistorySampleEntry](#) * > [historyCache](#)
The history cache queue of the writer.
- std::list< [SampleFragment](#) * > [send_queue](#)
The send queue.
- int [currentSampleSequenceNumber](#)
The current sample sequence number.

Additional Inherited Members

5.44.1 Detailed Description

The RTPS [Writer](#) entity sends generated samples to its subscribed readers based on the streamID routing mechanism implemented in the network.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 Writer()

```
Writer::Writer ( )
```

The default constructor

5.44.2.2 ~Writer()

```
Writer::~~Writer ( ) [virtual]
```

The default destructor

5.44.3 Member Function Documentation

5.44.3.1 addNextFragmentToSendList()

```
bool Writer::addNextFragmentToSendList ( ) [protected]
```

Choose the next fragment to send and add it in the send queue

5.44.3.2 checkLifespanOfSamples()

```
void Writer::checkLifespanOfSamples ( ) [protected]
```

Check for all samples if their lifespan have been expired

5.44.3.3 createRtpsMsgFromFragment()

```
RTPSEthMessage * Writer::createRtpsMsgFromFragment (
    SampleFragment * sampleFragment ) [protected]
```

Create an RTPS message from a sample fragment in order to send it

Parameters

<i>sampleFragment</i>	The fragment to send
-----------------------	----------------------

5.44.3.4 finish()

```
void Writer::finish ( ) [override], [protected], [virtual]
```

Nothing to finish here

5.44.3.5 handleMessage()

```
void Writer::handleMessage (
    cMessage * msg ) [override], [protected], [virtual]
```

Handles incoming messages, which are Samples from the application (generator), RTPS messages or self messages.

Parameters

<i>msg</i>	The incoming message.
------------	-----------------------

5.44.3.6 initialize()

```
void Writer::initialize ( ) [override], [protected], [virtual]
```

5.44.3.7 registerWriter()

```
void Writer::registerWriter ( ) [protected]
```

Register the writer at the participant

5.44.3.8 sendMessage()

```
void Writer::sendMessage ( ) [protected]
```

When this method is called and the sendEvent is not scheduled, than a new fragment is send out if one is pending

5.44.4 Member Data Documentation

5.44.4.1 bestEffort

```
bool Writer::bestEffort [private]
```

Choose best effort or reliable communication.

5.44.4.2 currentSampleSequenceNumber

```
int Writer::currentSampleSequenceNumber [private]
```

The current sample sequence number.

5.44.4.3 destinationEntityId

```
int Writer::destinationEntityId [private]
```

The entity ID of the remote reader(s)

5.44.4.4 destinationMac

```
int Writer::destinationMac [private]
```

The MAC address of the remote reader(s)

5.44.4.5 entityId

```
int Writer::entityId [private]
```

The own entity ID.

5.44.4.6 fragmentSize

```
int Writer::fragmentSize [private]
```

The fragment size.

5.44.4.7 globalStreamId

```
int Writer::globalStreamId [private]
```

The writers global stream ID.

5.44.4.8 historyCache

```
std::list<HistorySampleEntry*> Writer::historyCache [private]
```

The history cache queue of the writer.

5.44.4.9 historySizeQoS

```
int Writer::historySizeQoS [private]
```

The size of the history cache.

5.44.4.10 lifespan

```
simtime_t Writer::lifespan [private]
```

The lifespan duration.

5.44.4.11 registerEvent

```
cMessage* Writer::registerEvent [private]
```

Self event to trigger the registration of the writer at the participant.

5.44.4.12 send_queue

```
std::list<SampleFragment*> Writer::send_queue [private]
```

The send queue.

5.44.4.13 sendEvent

```
cMessage* Writer::sendEvent [private]
```

Send event self message.

5.44.4.14 **shapingDuration**

```
simtime_t Writer::shapingDuration [private]
```

Shaping duration for messages of the individual writer.

5.44.4.15 **sourceMac**

```
int Writer::sourceMac [private]
```

The own MAC address.

5.44.4.16 **streamEthernetPriority**

```
int Writer::streamEthernetPriority [private]
```

The Ethernet Priority indicator.

5.44.4.17 **transportPriority**

```
int Writer::transportPriority [private]
```

The transport priority of the writer.

Chapter 6

File Documentation

6.1 ArbiterAVB.cc File Reference

```
#include "ArbiterAVB.h"
```

6.2 ArbiterAVB.h File Reference

```
#include <omnetpp.h>
#include "../common/ConfigurationReader.h"
#include "../messagetype/BufferControlMessage_m.h"
#include "../messagetype/EthernetFrame_m.h"
#include "../common/EthernetMacros.h"
```

Classes

- struct [arbiterAVBClass](#)
- class [ArbiterAVB](#)

Enumerations

- enum [avbTablePostition](#) {
 [SWITCH_MAC](#) = 0, [EGRESS_PORT_ID](#) = 1, [PRIORITY](#) = 2, [SEND_SLOPE](#) = 3,
 [IDLE_SLOPE](#) = 4 }
- enum [arbiterClassStatus](#) {
 [CLASS_IDLE](#) = 0, [CLASS_SENDING](#) = 1, [CLASS_WAITING](#) = 2, [CLASS_BLOCKED](#) = 3,
 [CLASS_NOTSHAPED](#) = 4 }

Functions

- [Define_Module](#) ([ArbiterAVB](#))

6.2.1 Enumeration Type Documentation

6.2.1.1 arbiterClassStatus

enum `arbiterClassStatus`

Enumerations for Status of Arbiter Classes

Enumerator

CLASS_IDLE	
CLASS_SENDING	
CLASS_WAITING	
CLASS_BLOCKED	
CLASS_NOTSHAPED	

6.2.1.2 avbTablePostition

enum `avbTablePostition`

Enumeration for Position in AVB configuration File

Enumerator

SWITCH_MAC	
EGRESS_PORT_ID	
PRIORITY	
SEND_SLOPE	
IDLE_SLOPE	

6.2.2 Function Documentation

6.2.2.1 Define_Module()

```
Define_Module (
    ArbiterAVB )
```

6.3 ArbiterIEEE802_1Q.cc File Reference

```
#include "ArbiterIEEE802_1Q.h"
```

6.4 ArbiterIEEE802_1Q.h File Reference

```
#include <stdio.h>
#include <fstream>
#include <string.h>
#include <omnetpp.h>
#include <map>
#include <vector>
#include "../messagetype/BufferControlMessage_m.h"
#include "../messagetype/EthernetFrame_m.h"
#include "../common/EthernetMacros.h"
```

Classes

- class [ArbiterIEEE802_1Q](#)

Functions

- [Define_Module](#) ([ArbiterIEEE802_1Q](#))

6.4.1 Function Documentation

6.4.1.1 Define_Module()

```
Define_Module (
    ArbiterIEEE802\_1Q )
```

6.5 ArbiterIEEE802_1Qbv.cc File Reference

```
#include "ArbiterIEEE802_1Qbv.h"
```

6.6 ArbiterIEEE802_1Qbv.h File Reference

```
#include <omnetpp.h>
#include <stdio.h>
#include <fstream>
#include <unordered_set>
#include <map>
#include "../messagetype/BufferControlMessage_m.h"
#include "../messagetype/EthernetFrame_m.h"
#include "../common/EthernetMacros.h"
#include "../statistics/StatisticManager.h"
```

Classes

- class [ArbiterIEEE802_1Qbv](#)

Enumerations

- enum [timeSegments](#) { [CRITICAL_SENDING](#) = 1, [CRITICAL_GUARDBAND](#) = 2, [NONCRITICAL_SENDING](#) = 3, [NONCRITICAL_GUARDBAND](#) = 4 }

Functions

- [Define_Module](#) ([ArbiterIEEE802_1Qbv](#))

6.6.1 Enumeration Type Documentation

6.6.1.1 timeSegments

enum [timeSegments](#)

Enum of TAS time segments

Enumerator

CRITICAL_SENDING	
CRITICAL_GUARDBAND	
NONCRITICAL_SENDING	
NONCRITICAL_GUARDBAND	

6.6.2 Function Documentation

6.6.2.1 Define_Module()

```
Define_Module (
    ArbiterIEEE802\_1Qbv )
```

6.7 ArbiterIEEE802_3.cc File Reference

```
#include "ArbiterIEEE802_3.h"
```

6.8 ArbiterIEEE802_3.h File Reference

```
#include <stdio.h>
#include <fstream>
#include <string.h>
#include <omnetpp.h>
#include <map>
#include <vector>
#include "../messagetype/BufferControlMessage_m.h"
#include "../messagetype/EthernetFrame_m.h"
#include "../common/EthernetMacros.h"
```

Classes

- class [ArbiterIEEE802_3](#)

Functions

- [Define_Module](#) ([ArbiterIEEE802_3](#))

6.8.1 Function Documentation

6.8.1.1 Define_Module()

```
Define_Module (
    ArbiterIEEE802\_3 )
```

6.9 BuffersModule.cc File Reference

```
#include "BuffersModule.h"
```

6.10 BuffersModule.h File Reference

```
#include <omnetpp.h>
#include "../messagetype/EthernetFrame_m.h"
#include "../messagetype/BufferControlMessage_m.h"
#include "FrameBuffer.h"
```

Classes

- class [BuffersModule](#)

Functions

- [Define_Module](#) ([BuffersModule](#))

6.10.1 Function Documentation

6.10.1.1 Define_Module()

```
Define_Module (
    BuffersModule )
```

6.11 ClassifierPriority.cc File Reference

```
#include "ClassifierPriority.h"
```

6.12 ClassifierPriority.h File Reference

```
#include <omnetpp.h>
#include "../messagetype/EthernetFrame_m.h"
```

Classes

- class [ClassifierPriority](#)

Functions

- [Define_Module](#) ([ClassifierPriority](#))

6.12.1 Function Documentation

6.12.1.1 Define_Module()

```
Define_Module (
    ClassifierPriority )
```

6.13 ConfigurationReader.cc File Reference

```
#include "../common/ConfigurationReader.h"
```

6.14 ConfigurationReader.h File Reference

```
#include <string>
#include <sstream>
#include <fstream>
#include <iostream>
#include <vector>
#include <map>
#include <omnetpp.h>
```

Classes

- class [ConfigurationReader](#)

Macros

- #define [COMMENT_SYMBOL](#) "#"
- #define [COMMENT_POSITION](#) 0

Typedefs

- typedef std::vector< std::vector< int > > [configVector](#)

6.14.1 Macro Definition Documentation

6.14.1.1 COMMENT_POSITION

```
#define COMMENT_POSITION 0
```

6.14.1.2 COMMENT_SYMBOL

```
#define COMMENT_SYMBOL ""
```

6.14.2 Typedef Documentation

6.14.2.1 configVector

```
typedef std::vector<std::vector<int> > configVector
```

6.15 ddsSampleLatencyStatistics.cc File Reference

```
#include "ddsSampleLatencyStatistics.h"  
#include "../common/ValidationHandler.h"
```

6.16 ddsSampleLatencyStatistics.h File Reference

```
#include <omnetpp.h>  
#include "helperModulesStatistics/HistogramStatistics3DMap.h"  
#include "helperModulesStatistics/VectorStatistics3DMap.h"
```

Classes

- class [SampleLatencyStatistics](#)

6.17 DelayUnit.cc File Reference

```
#include "../../../../../endnode/middleware/rtps/delayunit/DelayUnit.h"  
#include "../../../../../common/GlobalMacros.h"
```

Functions

- [Define_Module](#) (DelayUnit)

6.17.1 Function Documentation

6.17.1.1 Define_Module()

```
Define_Module (
    DelayUnit )
```

6.18 DelayUnit.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [DelayUnit](#)

6.19 E2ELatencyStatistics.cc File Reference

```
#include "E2ELatencyStatistics.h"
```

6.20 E2ELatencyStatistics.h File Reference

```
#include <omnetpp.h>
#include <string>
#include "../common/GlobalMacros.h"
#include "../messagetype/EthernetFrame_m.h"
#include "helperModulesStatistics/HistogramStatistics2DMap.h"
#include "helperModulesStatistics/VectorStatistics2DMap.h"
```

Classes

- class [E2ELatencyStatistics](#)

6.21 EgressPort.cc File Reference

```
#include "EgressPort.h"
```

6.22 EgressPort.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [EgressPort](#)

Functions

- [Define_Module](#) ([EgressPort](#))

6.22.1 Function Documentation

6.22.1.1 Define_Module()

```
Define_Module (  
    EgressPort )
```

6.23 Endpoint.cc File Reference

```
#include "../.../endnode/middleware/rtps/entities/Endpoint.h"  
#include "../.../common/ValidationHandler.h"  
#include "../.../common/RtpsMacros.h"
```

6.24 Endpoint.h File Reference

```
#include "../.../messages/RTPSEthMessage_m.h"  
#include <omnetpp.h>
```

Classes

- class [Endpoint](#)

6.25 EthernetHeaderInformation.h File Reference

Classes

- struct [ethernetHeaderInformation](#)

6.26 EthernetMacros.h File Reference

```
#include <math.h>
#include <stdlib.h>
#include "GlobalMacros.h"
```

Macros

- `#define _MIN_ETH_PAYLOAD_ (42)`
- `#define _MIN_ETH_PAYLOAD_UNTAGGED (46)`
- `#define _MAX_ETH_PAYLOAD_ (1500)`
- `#define _PAYLOAD_INTERMEDIATE_PREEMPTION_ (60)`
- `#define _FRAME_OVERHEAD_ (6 + 6 + 4 + 2 + 4)`
- `#define _PACKET_OVERHEAD_ (7 + 1)`
- `#define _PACKET_OVERHEAD_FIRST_PREEMPTION_ (7 + 1 + 6 + 6 + 4 + 2)`
- `#define _PACKET_OVERHEAD_INTERMEDIATE_PREEMPTION_ (6 + 1 + 1)`
- `#define _CRC_ (4)`
- `#define _INTERPACKET_GAP_ (12)`
- `#define _CRC_INTERPACKET_GAP_ (_INTERPACKET_GAP_ + _CRC_)`
- `#define _MIN_ETH_FRAME_SIZE_ (_MIN_ETH_PAYLOAD_ + _FRAME_OVERHEAD_)`
- `#define _MIN_ETH_PACKET_SIZE_ (_MIN_ETH_FRAME_SIZE_ + _PACKET_OVERHEAD_ + _INTERPACKET_GAP_)`
- `#define _MIN_ETH_PACKET_SIZE_PREEMPTION_ (144)`
- `#define _eth_check_for_padding_(payload) (payload >= _MIN_ETH_PAYLOAD_ ? payload: _MIN_ETH_PAYLOAD_)`
- `#define _eth_check_for_max_payload_(payload) (payload <= _MAX_ETH_PAYLOAD_ ? payload: _MAX_ETH_PAYLOAD_)`
- `#define _eth_framesize_from_payload_(payload) (payload + _FRAME_OVERHEAD_)`
- `#define _eth_packetsize_from_payload_(payload) (payload + _FRAME_OVERHEAD_ + _PACKET_OVERHEAD_ + _INTERPACKET_GAP_)`
- `#define _std_eth_colour_ (_msg_red_)`

Enumerations

- `enum _eth_egress_helper_message_type_ {
 frame_arrived = 0, frame_started_sending = 1, frame_finished_sending = 3, frame_dropped = 5,
 port_update = 10 }`

6.26.1 Macro Definition Documentation

6.26.1.1 _CRC_

```
#define _CRC_ (4)
```

6.26.1.2 `_CRC_INTERPACKET_GAP_`

```
#define _CRC_INTERPACKET_GAP_ (_INTERPACKET_GAP_ + _CRC_)
```

6.26.1.3 `_eth_check_for_max_payload_`

```
#define _eth_check_for_max_payload_(  
    payload ) (payload<=_MAX_ETH_PAYLOAD_?payload:_MAX_ETH_PAYLOAD_)
```

6.26.1.4 `_eth_check_for_padding_`

```
#define _eth_check_for_padding_(  
    payload ) (payload>=_MIN_ETH_PAYLOAD_?payload:_MIN_ETH_PAYLOAD_)
```

6.26.1.5 `_eth_framesize_from_payload_`

```
#define _eth_framesize_from_payload_(  
    payload ) (payload+ _FRAME_OVERHEAD_)
```

6.26.1.6 `_eth_packetsize_from_payload_`

```
#define _eth_packetsize_from_payload_(  
    payload ) (payload + _FRAME_OVERHEAD_ + _PACKET_OVERHEAD_ + _INTERPACKET_GAP_)
```

6.26.1.7 `_FRAME_OVERHEAD_`

```
#define _FRAME_OVERHEAD_ (6 + 6 + 4 + 2 + 4)
```

6.26.1.8 `_INTERPACKET_GAP_`

```
#define _INTERPACKET_GAP_ (12)
```

6.26.1.9 _MAX_ETH_PAYLOAD_

```
#define _MAX_ETH_PAYLOAD_ (1500)
```

6.26.1.10 _MIN_ETH_FRAME_SIZE_

```
#define _MIN_ETH_FRAME_SIZE_ (_MIN_ETH_PAYLOAD_ + _FRAME_OVERHEAD_)
```

6.26.1.11 _MIN_ETH_PACKET_SIZE_

```
#define _MIN_ETH_PACKET_SIZE_ (_MIN_ETH_FRAME_SIZE_ + _PACKET_OVERHEAD_ + _INTERPACKET_GAP_)
```

6.26.1.12 _MIN_ETH_PACKET_SIZE_PREEMPTION_

```
#define _MIN_ETH_PACKET_SIZE_PREEMPTION_ (144)
```

6.26.1.13 _MIN_ETH_PAYLOAD_

```
#define _MIN_ETH_PAYLOAD_ (42)
```

6.26.1.14 _MIN_ETH_PAYLOAD_UNTAGGED

```
#define _MIN_ETH_PAYLOAD_UNTAGGED (46)
```

6.26.1.15 _PACKET_OVERHEAD_

```
#define _PACKET_OVERHEAD_ (7 + 1)
```

6.26.1.16 _PACKET_OVERHEAD_FIRST_PREEMPTION_

```
#define _PACKET_OVERHEAD_FIRST_PREEMPTION_ (7 + 1 + 6 + 6 + 4 + 2)
```

6.26.1.17 `_PACKET_OVERHEAD_INTERMEDIATE_PREEMPTION_`

```
#define _PACKET_OVERHEAD_INTERMEDIATE_PREEMPTION_ (6 + 1 + 1)
```

6.26.1.18 `_PAYLOAD_INTERMEDIATE_PREEMPTION_`

```
#define _PAYLOAD_INTERMEDIATE_PREEMPTION_ (60)
```

6.26.1.19 `_std_eth_colour_`

```
#define _std_eth_colour_ (_msg_red_)
```

6.26.2 Enumeration Type Documentation

6.26.2.1 `_eth_egress_helper_message_type_`

```
enum _eth_egress_helper_message_type_
```

Enumerator

frame_arrived	
frame_started_sending	
frame_finished_sending	
frame_dropped	
port_update	

6.27 EthMsgCreator.cc File Reference

```
#include "../msgcreators/EthMsgCreator.h"
#include "../../common/GlobalMacros.h"
#include "../../common/EthernetMacros.h"
#include "../../messagetype/EthernetFrame_m.h"
```

6.28 EthMsgCreator.h File Reference

```
#include <omnetpp.h>
```


Classes

- class [EthMsgCreator](#)

Functions

- [Define_Module](#) ([EthMsgCreator](#))

6.28.1 Function Documentation

6.28.1.1 Define_Module()

```
Define_Module (
    EthMsgCreator )
```

6.29 ForwardingLogic.cc File Reference

```
#include "ForwardingLogic.h"
```

6.30 ForwardingLogic.h File Reference

```
#include <stdio.h>
#include <fstream>
#include <string.h>
#include <omnetpp.h>
#include <map>
#include <vector>
#include "../common/ConfigurationReader.h"
#include "../common/GlobalMacros.h"
#include "../messagetype/EthernetFrame_m.h"
```

Classes

- class [ForwardingLogic](#)

Enumerations

- enum [macTableEntriesPositon](#) { [SWITCH_MAC](#) = 0, [STREAM_ID](#) = 1, [NEXT_HOP_MAC](#) = 2 }
- enum [simulationOutputType](#) { [INIT_OUTPUT](#) = 0, [MAC_TABLE_CONFIGURATION_INFO](#) = 1 }
- enum [errorType](#) { [INCORRECT_MESSAGE_TYPE](#) = 0, [UNKNOWN_STREAM_ID](#) = 1, [INVALID_NUMBER_MAC_TABLE_ENT](#) = 2, [INVALID_PORT_ID](#) = 3 }

Functions

- [Define_Module](#) ([ForwardingLogic](#))

6.30.1 Enumeration Type Documentation

6.30.1.1 `errorType`

enum `errorType`

Options for throw an error for error handling

INCORRECT_MESSAGE_TYPE: cMessage cannot cast into an Ethernet frame UNKNOWN_STREAM_ID: Entered stream ID in .config-file is unequal to streamID in Ethernet frame (no matching entry in .config-file) INVALID_NUMBER_MAC_TABLE_ENTRIES: the number of entries in MAC Table and therefore in .config-file is unequal 3 INVALID_PORT_ID: No matching portID to connected next module MAC (hop) is found.

Enumerator

INCORRECT_MESSAGE_TYPE	
UNKNOWN_STREAM_ID	
INVALID_NUMBER_MAC_TABLE_ENTRIES	
INVALID_PORT_ID	

6.30.1.2 `macTableEntriesPositon`

enum `macTableEntriesPositon`

Each entry in the configVector macTable is at a certain position.

First position the MAC address of a specific switch is entered Second position the global streamID of an Ethernet frame is stored. Third position the MAC address of the next connected module (hop) to which the Ethernet frame will be forwarded is stored.

Enumerator

SWITCH_MAC	
STREAM_ID	
NEXT_HOP_MAC	

6.30.1.3 `simulationOutputType`

enum `simulationOutputType`

Options for printing outputs during the simulation

INIT_OUT: prints characteristics about the switch MAC_TABLE_CONFIGURATION_INFO: only set if no entry in MAC table is set for a specific switch

Enumerator

INIT_OUTPUT	
MAC_TABLE_CONFIGURATION_INFO	

6.30.2 Function Documentation

6.30.2.1 Define_Module()

```
Define_Module (
    ForwardingLogic )
```

6.31 FrameBuffer.cc File Reference

```
#include "FrameBuffer.h"
```

6.32 FrameBuffer.h File Reference

```
#include <omnetpp.h>
#include <iostream>
#include "../messagetype/EthernetFrame_m.h"
```

Classes

- class [FrameBuffer](#)

Enumerations

- enum [bufferUpdate](#) { [ADDED_FRAME](#) = 0, [REMOVED_FRAME](#) = 1 }

6.32.1 Enumeration Type Documentation

6.32.1.1 bufferUpdate

```
enum bufferUpdate
```

Enum for updating the [FrameBuffer](#). ADDED_FRAME indicates that a new frame is stored into the buffer. REMOVED_FRAME indicates that a frame is deleted.

Enumerator

ADDED_FRAME	
REMOVED_FRAME	

6.33 FramesLostStatistics.cc File Reference

```
#include "../statistics/FramesLostStatistics.h"
```

6.34 FramesLostStatistics.h File Reference

```
#include <omnetpp.h>
#include "helperModulesStatistics/HistogramStatistics2DMap.h"
#include "helperModulesStatistics/VectorStatistics2DMap.h"
```

Classes

- class [FramesLostStatistics](#)

6.35 GlobalMacros.h File Reference

```
#include <omnetpp.h>
#include <unordered_set>
```

Classes

- struct [Frer_ID](#)
- struct [lostFrameCounter](#)

Macros

- `#define _min_(x, y) (y<x?y:x)`
- `#define _max_(x, y) (x>y?x:y)`
- `#define _msg_red_ 0`
- `#define _msg_green_ 1`
- `#define _msg_blue_ 2`
- `#define _msg_white_ 3`
- `#define _msg_yellow_ 4`
- `#define _msg_cyan_ 5`
- `#define _msg_magenta_ 6`
- `#define _msg_black_ 7`
- `#define _CAN2CAN_LOOKUP_ 0`
- `#define _CAN2ETH_LOOKUP_ 0`
- `#define _ETH2CAN_LOOKUP_ 0`
- `#define _ETH2ETH_LOOKUP_ 0`

Typedefs

- typedef std::vector< std::vector< std::string > > [csvVector](#)

Functions

- template<typename T >
T [truncate_min_max](#) (T input, T min, T max)
- omnetpp::simtime_t [simtime_modulo](#) (omnetpp::simtime_t s1, omnetpp::simtime_t s2)

6.35.1 Macro Definition Documentation

6.35.1.1 `_CAN2CAN_LOOKUP_`

```
#define _CAN2CAN_LOOKUP_ 0
```

6.35.1.2 `_CAN2ETH_LOOKUP_`

```
#define _CAN2ETH_LOOKUP_ 0
```

6.35.1.3 `_ETH2CAN_LOOKUP_`

```
#define _ETH2CAN_LOOKUP_ 0
```

6.35.1.4 `_ETH2ETH_LOOKUP_`

```
#define _ETH2ETH_LOOKUP_ 0
```

6.35.1.5 `_max_`

```
#define _max_(  
    x,  
    y ) (x>y?x:y)
```

6.35.1.6 `_min_`

```
#define _min_(  
    x,  
    y ) (y<x?y:x)
```

6.35.1.7 `_msg_black_`

```
#define _msg_black_ 7
```

6.35.1.8 `_msg_blue_`

```
#define _msg_blue_ 2
```

6.35.1.9 `_msg_cyan_`

```
#define _msg_cyan_ 5
```

6.35.1.10 `_msg_green_`

```
#define _msg_green_ 1
```

6.35.1.11 `_msg_magenta_`

```
#define _msg_magenta_ 6
```

6.35.1.12 `_msg_red_`

```
#define _msg_red_ 0
```

6.35.1.13 `_msg_white_`

```
#define _msg_white_ 3
```

6.35.1.14 `_msg_yellow_`

```
#define _msg_yellow_ 4
```

6.35.2 Typedef Documentation

6.35.2.1 `csvVector`

```
typedef std::vector<std::vector<std::string> > csvVector
```

6.35.3 Function Documentation

6.35.3.1 `simtime_modulo()`

```
omnetpp::simtime_t simtime_modulo (  
    omnetpp::simtime_t s1,  
    omnetpp::simtime_t s2 ) [inline]
```

`s1 % s2`; what is the remainder of dividing `s1` by `s2`

6.35.3.2 `truncate_min_max()`

```
template<typename T >  
T truncate_min_max (  
    T input,  
    T min,  
    T max ) [inline]
```

6.36 HistogramStatistics2DMap.cc File Reference

```
#include "HistogramStatistics2DMap.h"
```

6.37 HistogramStatistics2DMap.h File Reference

```
#include <omnetpp.h>
#include "../common/GlobalMacros.h"
```

Classes

- class [HistogramStatistics2DMap](#)

6.38 HistogramStatistics3DMap.cc File Reference

```
#include "HistogramStatistics3DMap.h"
```

6.39 HistogramStatistics3DMap.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [HistogramStatistics3DMap](#)

6.40 HistoryEntry.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [ReaderHistory](#)
- class [HistorySampleEntry](#)
- class [SampleFragment](#)

6.41 IngressPort.cc File Reference

```
#include "IngressPort.h"
```


6.42 IngressPort.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "../common/EthernetMacros.h"
#include "../messagetype/EthernetFrame_m.h"
```

Classes

- class [IngressPort](#)

Functions

- [Define_Module](#) ([IngressPort](#))

6.42.1 Function Documentation

6.42.1.1 Define_Module()

```
Define_Module (
    IngressPort )
```

6.43 JoinReaders.cc File Reference

```
#include "../../../endnode/middleware/rtps/adapters/JoinReaders.h"
```

Functions

- [Define_Module](#) ([JoinReaders](#))

6.43.1 Function Documentation

6.43.1.1 Define_Module()

```
Define_Module (
    JoinReaders )
```

6.44 JoinReaders.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [JoinReaders](#)

6.45 LinkTransmissionStatistics.cc File Reference

```
#include "LinkTransmissionStatistics.h"
```

6.46 LinkTransmissionStatistics.h File Reference

```
#include <omnetpp.h>
#include <map>
#include "helperModulesStatistics/VectorStatistics3DMap.h"
```

Classes

- class [LinkTransmissionStatistics](#)

6.47 Participant.cc File Reference

```
#include "../endnode/middleware/rtps/participant/Participant.h"
#include "../endnode/middleware/rtps/messages/RegisterEntity_m.h"
#include "../endnode/middleware/rtps/messages/RTPSEthMessage_m.h"
#include "../common/ValidationHandler.h"
#include "string"
#include "sstream"
```

6.48 Participant.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [Participant](#)

Functions

- [Define_Module](#) ([Participant](#))

6.48.1 Function Documentation

6.48.1.1 Define_Module()

```
Define_Module (
    Participant )
```

6.49 Reader.cc File Reference

```
#include "../../../../../endnode/middleware/rtps/entities/Reader.h"
#include <string.h>
#include <math.h>
#include "../messages/RTPSEthMessage_m.h"
#include "../../../../../common/EthernetMacros.h"
#include "../../../../../messagetype/EthernetFrame_m.h"
#include "../../../../../endnode/middleware/rtps/messages/RegisterEntity_m.h"
#include "../../../../../endnode/middleware/rtps/messages/Sample_m.h"
```

6.50 Reader.h File Reference

```
#include "../../../../../endnode/middleware/rtps/entities/Endpoint.h"
#include "../../../../../endnode/middleware/rtps/entities/HistoryEntry.h"
#include "../../../../../common/ValidationHandler.h"
#include <omnetpp.h>
```

Classes

- class [Reader](#)

Functions

- [Define_Module](#) ([Reader](#))

6.50.1 Function Documentation

6.50.1.1 Define_Module()

```
Define_Module (
    Reader )
```

6.51 README_RTPS.md File Reference

6.52 Rtps.cc File Reference

```
#include "../.../endnode/middleware/rtps/Rtps.h"
```

Functions

- [Define_Module](#) ([Rtps](#))

6.52.1 Function Documentation

6.52.1.1 Define_Module()

```
Define_Module (
    Rtps )
```

6.53 Rtps.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [Rtps](#)

6.54 RtpsMacros.h File Reference

```
#include <math.h>
#include <stdlib.h>
#include "GlobalMacros.h"
#include "EthernetMacros.h"
```

Macros

- #define `_MIN_RTPS_PAYLOAD_` (0)
- #define `_MAX_RTPS_PAYLOAD_` (4294967296)
- #define `_UDP_HEADER_SIZE_` (8)
Sizes UDP IP Header.
- #define `_IP_HEADER_SIZE_` (20)
- #define `_RTPS_HEADER_SIZE_` (20)
Size of RTPS header.
- #define `_RTPS_INFO_DST_SIZE_` (16)
Sizes of relevant RTPS submessages (static part)
- #define `_RTPS_INFO_TS_SIZE_` (12)
- #define `_RTPS_HEARTBEATFRAG_SIZE` (28)
- #define `_RTPS_DATAFRAG_STATIC_SIZE_` (36)
- #define `_RTPS_NACKFRAG_STATIC_SIZE_` (32)
- #define `_RTPS_MAX_MSG_SIZE_` (`_MAX_ETH_PAYLOAD_ - _IP_HEADER_SIZE_ - _UDP_HEADER_SIZE_`)
Maximum RTPS sizes.
- #define `_RTPS_MAX_FRAG_SIZE_` (`_RTPS_MAX_MSG_SIZE_ - _RTPS_HEADER_SIZE_ - _RTPS_INFO_DST_SIZE_ - _RTPS_INFO_TS_SIZE_ - _RTPS_DATAFRAG_STATIC_SIZE_`)
- #define `_rtps_check_for_max_payload_`(payload) (payload<=`_MAX_RTPS_PAYLOAD_`?payload:`_MAX_RTPS_PAYLOAD_`)
RTPS specific methods.
- #define `_rtps_add_udp_ip_overhead_`(rtps_msg_size) (rtps_msg_size + `_IP_HEADER_SIZE_` + `_UDP_HEADER_SIZE_`)

Enumerations

- enum `rtpsEntityKind` {
`unknown` = 0, `participant` = 1, `writerWithKey` = 2, `writerNoKey` = 3,
`readerNoKey` = 4, `readerWithKey` = 7 }
RTPS entity kinds.

6.54.1 Macro Definition Documentation

6.54.1.1 `_IP_HEADER_SIZE_`

```
#define _IP_HEADER_SIZE_ (20)
```

6.54.1.2 `_MAX_RTPS_PAYLOAD_`

```
#define _MAX_RTPS_PAYLOAD_ (4294967296)
```

6.54.1.3 `_MIN_RTPS_PAYLOAD_`

```
#define _MIN_RTPS_PAYLOAD_ (0)
```

CONSTANT definitions Minimum and maximum RTPS payload

6.54.1.4 `_rtps_add_udp_ip_overhead_`

```
#define _rtps_add_udp_ip_overhead_(  
    rtps_msg_size ) (rtps_msg_size + _IP_HEADER_SIZE_ + _UDP_HEADER_SIZE_)
```

6.54.1.5 `_rtps_check_for_max_payload_`

```
#define _rtps_check_for_max_payload_(  
    payload ) (payload<=_MAX_RTPS_PAYLOAD_?payload:_MAX_RTPS_PAYLOAD_)
```

RTPS specific methods.

6.54.1.6 `_RTPS_DATAFRAG_STATIC_SIZE_`

```
#define _RTPS_DATAFRAG_STATIC_SIZE_ (36)
```

6.54.1.7 `_RTPS_HEADER_SIZE_`

```
#define _RTPS_HEADER_SIZE_ (20)
```

Size of RTPS header.

6.54.1.8 `_RTPS_HEARTBEATFRAG_SIZE`

```
#define _RTPS_HEARTBEATFRAG_SIZE (28)
```

6.54.1.9 `_RTPS_INFO_DST_SIZE_`

```
#define _RTPS_INFO_DST_SIZE_ (16)
```

Sizes of relevant RTPS submessages (static part)

6.54.1.10 _RTPS_INFO_TS_SIZE_

```
#define _RTPS_INFO_TS_SIZE_ (12)
```

6.54.1.11 _RTPS_MAX_FRAG_SIZE_

```
#define _RTPS_MAX_FRAG_SIZE_ (_RTPS_MAX_MSG_SIZE_ - _RTPS_HEADER_SIZE_ - _RTPS_INFO_DST_SIZE_  
- _RTPS_INFO_TS_SIZE_ - _RTPS_DATAFRAG_STATIC_SIZE_)
```

6.54.1.12 _RTPS_MAX_MSG_SIZE_

```
#define _RTPS_MAX_MSG_SIZE_ (_MAX_ETH_PAYLOAD_ - _IP_HEADER_SIZE_ - _UDP_HEADER_SIZE_)
```

Maximum RTPS sizes.

6.54.1.13 _RTPS_NACKFRAG_STATIC_SIZE_

```
#define _RTPS_NACKFRAG_STATIC_SIZE_ (32)
```

6.54.1.14 _UDP_HEADER_SIZE_

```
#define _UDP_HEADER_SIZE_ (8)
```

Sizes UDP IP Header.

6.54.2 Enumeration Type Documentation

6.54.2.1 rtpsEntityKind

```
enum rtpsEntityKind
```

RTPS entity kinds.

Enumerator

unknown	
participant	
writerNoKey	
writerWithKey	
readerNoKey	
readerWithKey	

6.55 RtpsToEthernetAdapter.cc File Reference

```
#include "../.../endnode/middleware/rtps/adapters/RtpsToEthernetAdapter.h"
#include "../.../messages/RTPSEthMessage_m.h"
#include "../.../common/EthernetMacros.h"
#include "../.../common/ValidationHandler.h"
#include "../.../common/RtpsMacros.h"
```

Functions

- [Define_Module](#) ([RtpsToEthernetAdapter](#))

6.55.1 Function Documentation

6.55.1.1 Define_Module()

```
Define_Module (
    RtpsToEthernetAdapter )
```

6.56 RtpsToEthernetAdapter.h File Reference

```
#include "../.../common/ValidationHandler.h"
#include "../.../endnode/middleware/rtps/messages/RTPSEthMessage_m.h"
#include <omnetpp.h>
```

Classes

- class [RtpsToEthernetAdapter](#)

Macros

- `#define _adapter_src_verbose_ if (false)`

6.56.1 Macro Definition Documentation

6.56.1.1 `_adapter_src_verbose_`

```
#define _adapter_src_verbose_ if (false)
```

6.57 SampleMsgCreator.cc File Reference

```
#include "../msgcreators/SampleMsgCreator.h"  
#include "../../../common/GlobalMacros.h"  
#include "../../../common/RtpsMacros.h"  
#include "../../../common/ValidationHandler.h"  
#include "../../../endnode/middleware/rtps/messages/Sample_m.h"
```

6.58 SampleMsgCreator.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [SampleMsgCreator](#)

Functions

- [Define_Module](#) ([SampleMsgCreator](#))

6.58.1 Function Documentation

6.58.1.1 `Define_Module()`

```
Define_Module (  
    SampleMsgCreator )
```

6.59 SampleSink.cc File Reference

```
#include "SampleSink.h"
```

Functions

- [Define_Module](#) ([SampleSink](#))

6.59.1 Function Documentation

6.59.1.1 Define_Module()

```
Define_Module (
    SampleSink )
```

6.60 SampleSink.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <map>
#include "../messagetype/EthernetFrame_m.h"
#include "../common/ValidationHandler.h"
#include "../endnode/middleware/rtps/messages/Sample_m.h"
#include "../statistics/StatisticManager.h"
#include <omnetpp.h>
```

Classes

- class [SampleSink](#)

Macros

- #define [_eth_sampleSink_verbose_](#) if(true)

6.60.1 Macro Definition Documentation

6.60.1.1 _eth_sampleSink_verbose_

```
#define _eth_sampleSink_verbose_ if(true)
```

6.61 Shaper.cc File Reference

```
#include "../endnode/middleware/rtps/shaper/Shaper.h"
#include "../common/ValidationHandler.h"
```

Functions

- [Define_Module](#) ([Shaper](#))

6.61.1 Function Documentation

6.61.1.1 Define_Module()

```
Define_Module (
    Shaper )
```

6.62 Shaper.h File Reference

```
#include "../.../endnode/middleware/rtps/messages/RTPSEthMessage_m.h"
#include <omnetpp.h>
```

Classes

- class [Shaper](#)

6.63 Sink.cc File Reference

```
#include "Sink.h"
#include <sstream>
```

6.64 Sink.h File Reference

```
#include <omnetpp.h>
#include <stdio.h>
#include <string.h>
#include <sstream>
#include <map>
#include "../.../common/ValidationHandler.h"
#include "../.../messagetype/EthernetFrame_m.h"
#include "../.../statistics/StatisticManager.h"
```

Classes

- class [Sink](#)

Functions

- [Define_Module](#) ([Sink](#))

6.64.1 Function Documentation

6.64.1.1 [Define_Module](#)()

```
Define_Module (
    Sink )
```

6.65 [StatisticManager.cc](#) File Reference

```
#include "StatisticManager.h"
```

6.66 [StatisticManager.h](#) File Reference

```
#include <map>
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "FramesLostStatistics.h"
#include "ddsSampleLatencyStatistics.h"
#include "E2ELatencyStatistics.h"
#include "LinkTransmissionStatistics.h"
#include "TimeAwareShaperSegmentStatistic.h"
```

Classes

- class [StatisticManager](#)

6.67 [TimeAwareShaperSegmentStatistic.cc](#) File Reference

```
#include "TimeAwareShaperSegmentStatistic.h"
```

6.68 TimeAwareShaperSegmentStatistic.h File Reference

```
#include <omnetpp.h>
#include <map>
#include "helperModulesStatistics/VectorStatistics2DMap.h"
```

Classes

- class [TimeAwareShaperSegmentStatistic](#)

Enumerations

- enum [tasTimeSegments](#) { [CRITICAL_SENDING_SEGMENT](#) = 2, [CRITICAL_GUARDBAND_SEGMENT](#) = 1, [NONCRITICAL_SENDING_SEGMENT](#) = -2, [NONCRITICAL_GUARDBAND_SEGMENT](#) = -1 }

6.68.1 Enumeration Type Documentation

6.68.1.1 tasTimeSegments

enum [tasTimeSegments](#)

Enum of TAS time segments

Enumerator

CRITICAL_SENDING_SEGMENT	
CRITICAL_GUARDBAND_SEGMENT	
NONCRITICAL_SENDING_SEGMENT	
NONCRITICAL_GUARDBAND_SEGMENT	

6.69 TrafficSource.cc File Reference

```
#include "../trafficsrcs/TrafficSource.h"
#include "../../../common/GlobalMacros.h"
#include "../../../common/ValidationHandler.h"
```

6.70 TrafficSource.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [TrafficSource](#)

Macros

- `#define _burst_src_verbose_ if (true)`

Functions

- [Define_Module](#) ([TrafficSource](#))

6.70.1 Macro Definition Documentation

6.70.1.1 [_burst_src_verbose_](#)

```
#define \_burst\_src\_verbose\_ if (true)
```

6.70.2 Function Documentation

6.70.2.1 [Define_Module\(\)](#)

```
Define_Module (  
    TrafficSource )
```

6.71 UdpIpStack.cc File Reference

```
#include "../endnode/udpipstack/UdpIpStack.h"  
#include "../common/ValidationHandler.h"  
#include "../endnode/middleware/rtps/messages/RTPSEthMessage_m.h"  
#include "../messagetype/EthernetFrame_m.h"  
#include <string.h>  
#include <sstream>
```

6.72 UdpIpStack.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [UdpIpStack](#)

Functions

- [Define_Module](#) ([UdpIpStack](#))

6.72.1 Function Documentation

6.72.1.1 Define_Module()

```
Define_Module (
    UdpIpStack )
```

6.73 ValidationHandler.h File Reference

```
#include <omnetpp.h>
```

Classes

- class [ValidationHandler](#)

6.74 VectorStatistics2DMap.cc File Reference

```
#include "VectorStatistics2DMap.h"
```

6.75 VectorStatistics2DMap.h File Reference

```
#include <omnetpp.h>
#include <tuple>
#include "../common/GlobalMacros.h"
```

Classes

- class [VectorStatistics2DMap](#)

6.76 VectorStatistics3DMap.cc File Reference

```
#include <sstream>
#include "VectorStatistics3DMap.h"
```

6.77 VectorStatistics3DMap.h File Reference

```
#include <omnetpp.h>
#include <tuple>
#include <map>
```

Classes

- class [VectorStatistics3DMap](#)

6.78 Writer.cc File Reference

```
#include "../endnode/middleware/rtps/entities/Writer.h"
#include <math.h>
#include "../messages/RTPSEthMessage_m.h"
#include "../common/ValidationHandler.h"
#include "../endnode/middleware/rtps/messages/RegisterEntity_m.h"
#include "../endnode/middleware/rtps/messages/Sample_m.h"
#include "../common/RtpsMacros.h"
```

6.79 Writer.h File Reference

```
#include "../endnode/middleware/rtps/entities/Endpoint.h"
#include "../endnode/middleware/rtps/entities/HistoryEntry.h"
#include "../endnode/middleware/rtps/messages/RTPSEthMessage_m.h"
#include <omnetpp.h>
```

Classes

- class [Writer](#)

Functions

- [Define_Module](#) ([Writer](#))

6.79.1 Function Documentation

6.79.1.1 Define_Module()

```
Define_Module (
    Writer )
```


Index

- [_CAN2CAN_LOOKUP_](#)
 - [GlobalMacros.h, 187](#)
- [_CAN2ETH_LOOKUP_](#)
 - [GlobalMacros.h, 187](#)
- [_CRC_](#)
 - [EthernetMacros.h, 179](#)
- [_CRC_INTERPACKET_GAP_](#)
 - [EthernetMacros.h, 179](#)
- [_ETH2CAN_LOOKUP_](#)
 - [GlobalMacros.h, 187](#)
- [_ETH2ETH_LOOKUP_](#)
 - [GlobalMacros.h, 187](#)
- [_FRAME_OVERHEAD_](#)
 - [EthernetMacros.h, 180](#)
- [_INTERPACKET_GAP_](#)
 - [EthernetMacros.h, 180](#)
- [_IP_HEADER_SIZE_](#)
 - [RtpsMacros.h, 195](#)
- [_MAX_ETH_PAYLOAD_](#)
 - [EthernetMacros.h, 180](#)
- [_MAX_RTPS_PAYLOAD_](#)
 - [RtpsMacros.h, 195](#)
- [_MIN_ETH_FRAME_SIZE_](#)
 - [EthernetMacros.h, 181](#)
- [_MIN_ETH_PACKET_SIZE_](#)
 - [EthernetMacros.h, 181](#)
- [_MIN_ETH_PACKET_SIZE_PREEMPTION_](#)
 - [EthernetMacros.h, 181](#)
- [_MIN_ETH_PAYLOAD_](#)
 - [EthernetMacros.h, 181](#)
- [_MIN_ETH_PAYLOAD_UNTAGGED](#)
 - [EthernetMacros.h, 181](#)
- [_MIN_RTPS_PAYLOAD_](#)
 - [RtpsMacros.h, 195](#)
- [_PACKET_OVERHEAD_](#)
 - [EthernetMacros.h, 181](#)
- [_PACKET_OVERHEAD_FIRST_PREEMPTION_](#)
 - [EthernetMacros.h, 181](#)
- [_PACKET_OVERHEAD_INTERMEDIATE_PREEMPTION_](#)
 - [EthernetMacros.h, 181](#)
- [_PAYLOAD_INTERMEDIATE_PREEMPTION_](#)
 - [EthernetMacros.h, 182](#)
- [_RTPS_DATAFRAG_STATIC_SIZE_](#)
 - [RtpsMacros.h, 196](#)
- [_RTPS_HEADER_SIZE_](#)
 - [RtpsMacros.h, 196](#)
- [_RTPS_HEARTBEATFRAG_SIZE](#)
 - [RtpsMacros.h, 196](#)
- [_RTPS_INFO_DST_SIZE_](#)
 - [RtpsMacros.h, 196](#)
- [_RTPS_INFO_TS_SIZE_](#)
 - [RtpsMacros.h, 196](#)
- [_RTPS_MAX_FRAG_SIZE_](#)
 - [RtpsMacros.h, 197](#)
- [_RTPS_MAX_MSG_SIZE_](#)
 - [RtpsMacros.h, 197](#)
- [_RTPS_NACKFRAG_STATIC_SIZE_](#)
 - [RtpsMacros.h, 197](#)
- [_UDP_HEADER_SIZE_](#)
 - [RtpsMacros.h, 197](#)
- [_adapter_src_verbose_](#)
 - [RtpsToEthernetAdapter.h, 198](#)
- [_burst_src_verbose_](#)
 - [TrafficSource.h, 204](#)
- [_eth_check_for_max_payload_](#)
 - [EthernetMacros.h, 180](#)
- [_eth_check_for_padding_](#)
 - [EthernetMacros.h, 180](#)
- [_eth_egress_helper_message_type_](#)
 - [EthernetMacros.h, 182](#)
- [_eth_framesize_from_payload_](#)
 - [EthernetMacros.h, 180](#)
- [_eth_packetize_from_payload_](#)
 - [EthernetMacros.h, 180](#)
- [_eth_sampleSink_verbose_](#)
 - [SampleSink.h, 200](#)
- [_max_](#)
 - [GlobalMacros.h, 187](#)
- [_min_](#)
 - [GlobalMacros.h, 187](#)
- [_msg_black_](#)
 - [GlobalMacros.h, 188](#)
- [_msg_blue_](#)
 - [GlobalMacros.h, 188](#)
- [_msg_cyan_](#)
 - [GlobalMacros.h, 188](#)
- [_msg_green_](#)
 - [GlobalMacros.h, 188](#)
- [_msg_magenta_](#)
 - [GlobalMacros.h, 188](#)
- [_msg_red_](#)
 - [GlobalMacros.h, 188](#)
- [_msg_white_](#)
 - [GlobalMacros.h, 188](#)
- [_msg_yellow_](#)
 - [GlobalMacros.h, 189](#)
- [_rtps_add_udp_ip_overhead_](#)
 - [RtpsMacros.h, 196](#)

- `_rtps_check_for_max_payload_`
 - `RtpsMacros.h`, 196
- `_std_eth_colour_`
 - `EthernetMacros.h`, 182
- `~ArbiterAVB`
 - `ArbiterAVB`, 10
- `~ArbiterIEEE802_1Q`
 - `ArbiterIEEE802_1Q`, 24
- `~ArbiterIEEE802_1Qbv`
 - `ArbiterIEEE802_1Qbv`, 32
- `~ArbiterIEEE802_3`
 - `ArbiterIEEE802_3`, 48
- `~BuffersModule`
 - `BuffersModule`, 55
- `~ClassifierPriority`
 - `ClassifierPriority`, 63
- `~ConfigurationReader`
 - `ConfigurationReader`, 65
- `~DelayUnit`
 - `DelayUnit`, 68
- `~E2ELatencyStatistics`
 - `E2ELatencyStatistics`, 69
- `~EgressPort`
 - `EgressPort`, 71
- `~Endpoint`
 - `Endpoint`, 72
- `~EthMsgCreator`
 - `EthMsgCreator`, 75
- `~ForwardingLogic`
 - `ForwardingLogic`, 78
- `~FrameBuffer`
 - `FrameBuffer`, 86
- `~FramesLostStatistics`
 - `FramesLostStatistics`, 91
- `~HistogramStatistics2DMap`
 - `HistogramStatistics2DMap`, 94
- `~HistogramStatistics3DMap`
 - `HistogramStatistics3DMap`, 97
- `~HistorySampleEntry`
 - `HistorySampleEntry`, 102
- `~IngressPort`
 - `IngressPort`, 106
- `~JoinReaders`
 - `JoinReaders`, 109
- `~LinkTransmissionStatistics`
 - `LinkTransmissionStatistics`, 111
- `~Participant`
 - `Participant`, 114
- `~Reader`
 - `Reader`, 117
- `~ReaderHistory`
 - `ReaderHistory`, 124
- `~Rtps`
 - `Rtps`, 126
- `~RtpsToEthernetAdapter`
 - `RtpsToEthernetAdapter`, 127
- `~SampleFragment`
 - `SampleFragment`, 129
- `~SampleLatencyStatistics`
 - `SampleLatencyStatistics`, 131
- `~SampleMsgCreator`
 - `SampleMsgCreator`, 134
- `~Shaper`
 - `Shaper`, 137
- `~StatisticManager`
 - `StatisticManager`, 142
- `~TimeAwareShaperSegmentStatistic`
 - `TimeAwareShaperSegmentStatistic`, 146
- `~TrafficSource`
 - `TrafficSource`, 149
- `~UdpIpStack`
 - `UdpIpStack`, 152
- `~VectorStatistics2DMap`
 - `VectorStatistics2DMap`, 155
- `~VectorStatistics3DMap`
 - `VectorStatistics3DMap`, 159
- `~Writer`
 - `Writer`, 164
- `accumulateInteger`
 - `VectorStatistics2DMap`, 155
 - `VectorStatistics3DMap`, 159
- `acked`
 - `SampleFragment`, 130
- `activationPeriod`
 - `LinkTransmissionStatistics`, 112
- `addE2ELatency`
 - `E2ELatencyStatistics`, 70
- `ADDED_FRAME`
 - `FrameBuffer.h`, 186
- `addInteger`
 - `HistogramStatistics3DMap`, 97
 - `VectorStatistics2DMap`, 155
 - `VectorStatistics3DMap`, 159
- `addIntegerResult`
 - `HistogramStatistics2DMap`, 94
- `addLatency`
 - `HistogramStatistics2DMap`, 94
 - `HistogramStatistics3DMap`, 98
 - `VectorStatistics2DMap`, 156
 - `VectorStatistics3DMap`, 160
- `addLostFrame`
 - `FramesLostStatistics`, 91
- `addLostFrameTrace`
 - `FramesLostStatistics`, 91
- `addMessage`
 - `SampleLatencyStatistics`, 132
- `addNextFragmentToSendList`
 - `Writer`, 164
- `ArbiterAVB`, 9
 - `~ArbiterAVB`, 10
 - `ArbiterAVB`, 10
 - `arbiterIsBusy`, 19
 - `arbitrateBufferStatusList`, 11
 - `avbTableVector`, 19
 - `calculateClassInformationCredits`, 11
 - `calculateCredits`, 11

- calculateCreditValues, 11
- calculatedRechargeTime, 13
- calculateTransmissionDelay, 13
- calculateTransmissionSpeed, 13
- channelTransmissionRate, 19
- checkAndCastBufferControlMessage, 13
- checkAndCastEthernetFrame, 14
- classInformations, 19
- collectCreditsStatistics, 14
- config, 19
- configFile, 19
- controlArbitrationBufferStatusList, 14
- creditValueStatisticStreamA, 20
- creditValueStatisticStreamB, 20
- currentBufferStatusList, 20
- dealWithControlMessage, 15
- dealWithEthernetFrame, 15
- dealWithExternalMessage, 15
- dealWithInternalMessage, 16
- debugStatus, 16
- debugStatusInitialization, 16
- egressPortIndex, 20
- ethernetFrameWaitForSend, 20
- handleMessage, 16
- initialize, 16
- initializeAVBCClassInformation, 16
- loadAVBCClassInformationFromFile, 17
- macAddress, 20
- numberPriorities, 20
- parentChannelOut, 20
- parentGateOut, 21
- parentModul, 21
- prepareBufferStatusList, 17
- prepareClassInformations, 17
- sendBufferAuthorizedBuffer, 17
- sendBufferRequest, 18
- sendFrameOut, 18
- setRechargeTime, 18
- settriggerCreditRecharged, 18
- settriggerDelayEnd, 18
- triggerSelf, 21
- updateClassInformationsStatus, 18
- ArbiterAVB.cc, 169
- ArbiterAVB.h, 169
 - arbiterClassStatus, 170
 - avbTablePostition, 170
 - CLASS_BLOCKED, 170
 - CLASS_IDLE, 170
 - CLASS_NOTSHAPED, 170
 - CLASS_SENDING, 170
 - CLASS_WAITING, 170
 - Define_Module, 170
 - EGRESS_PORT_ID, 170
 - IDLE_SLOPE, 170
 - PRIORITY, 170
 - SEND_SLOPE, 170
 - SWITCH_MAC, 170
- arbiterAVBCClass, 21
- classCreditValue, 21
- classFrameWaiting, 22
- classIsShaped, 22
- classStatus, 22
- idleSlope, 22
- lastArbitrationTimeStamp, 22
- sendSlope, 22
- timerIsScheduled, 22
- arbiterClassStatus
 - ArbiterAVB.h, 170
- ArbiterIEEE802_1Q, 23
 - ~ArbiterIEEE802_1Q, 24
 - ArbiterIEEE802_1Q, 24
 - arbiterIsBusy, 28
 - arbitrateBufferStatusList, 24
 - calculateTransmissionDelay, 24
 - calculateTransmissionSpeed, 25
 - channelTransmissionRate, 28
 - controlArbitrationBufferStatusList, 25
 - currentBufferStatusList, 28
 - dealWithControlMessage, 25
 - dealWithEthernetFrame, 25
 - dealWithExternalMessage, 26
 - dealWithInternalMessage, 26
 - ethernetFrameWaitForSend, 28
 - handleMessage, 26
 - initialize, 26
 - numberPriorities, 28
 - parentChannelOut, 28
 - parentGateOut, 28
 - parentModul, 29
 - prepareBufferStatusList, 26
 - sendBufferAuthorizedBuffer, 27
 - sendBufferRequest, 27
 - sendFrameOut, 27
 - settriggerDelayEnd, 27
 - triggerSelf, 29
- ArbiterIEEE802_1Q.cc, 170
- ArbiterIEEE802_1Q.h, 171
 - Define_Module, 171
- ArbiterIEEE802_1Qbv, 29
 - ~ArbiterIEEE802_1Qbv, 32
 - ArbiterIEEE802_1Qbv, 31
 - arbiterIsBusy, 43
 - arbitrateBufferStatusList, 32
 - calculateGuardBandDurationFromPayloadSize, 32
 - calculateTimeModulo, 34
 - calculateTransmissionDelay, 34
 - controlArbitrationBufferStatusList, 35
 - convertDoubleIntoTime, 35
 - criticalGuardBandSegmentDuration, 43
 - criticalIntervalDuration, 43
 - criticalPriorityClasses, 43
 - criticalSendingSegmentDuration, 43
 - currentBufferStatusList, 44
 - currentState, 44
 - dealWithControlMessage, 35
 - dealWithEthernetFrame, 36

- dealWithExternalMessage, 36
- dealWithInternalMessage, 36
- dealWithSegmentTimerMessage, 37
- dealWithTriggerSelf, 37
- ethernetFrameWaitForTransmission, 44
- frameStoredInBuffer, 44
- getCriticalClasses, 37
- getCriticalIntervalDuration, 37
- getInitialStateAndCalculateTriggerTimer, 38
- getNextState, 38
- getPeriodDuration, 38
- getPortSpeed, 39
- getSegmentDurationFromCurrentState, 39
- getTimeOffset, 39
- handleMessage, 40
- initialize, 40
- initializeTimerMessage, 40
- isCritical, 44
- maximumCriticalPayloadSize, 44
- maximumNoncriticalPayloadSize, 45
- nextState, 45
- noncriticalGuardBandSegmentDuration, 45
- noncriticalIntervalDuration, 45
- noncriticalSendingSegmentDuration, 45
- numberSupportedPriorities, 45
- periodDuration, 46
- portIndex, 46
- prepareBufferStatusList, 40
- printInitializationConfiguration, 41
- segmentTriggerTimerMessage, 46
- sendBufferAuthorizedBuffer, 41
- sendBufferRequest, 41
- sendFrameOut, 41
- setEthPacketSizeFromPayload, 42
- setTriggerDelayEnd, 42
- setUpVectorCriticalPriorities, 42
- switchMAC, 46
- timeOffset, 46
- triggerSelf, 46
- triggerTimer, 47
- ArbiterIEEE802_1Qbv.cc, 171
- ArbiterIEEE802_1Qbv.h, 171
 - CRITICAL_GUARDBAND, 172
 - CRITICAL_SENDING, 172
 - Define_Module, 172
 - NONCRITICAL_GUARDBAND, 172
 - NONCRITICAL_SENDING, 172
 - timeSegments, 172
- ArbiterIEEE802_3, 47
 - ~ArbiterIEEE802_3, 48
 - ArbiterIEEE802_3, 48
 - arbiterIsBusy, 52
 - arbitrateBufferStatusList, 48
 - calculateTransmissionDelay, 48
 - calculateTransmissionSpeed, 49
 - channelTransmissionRate, 52
 - controlArbitrationBufferStatusList, 49
 - currentBufferStatusList, 52
 - dealWithControlMessage, 49
 - dealWithEthernetFrame, 50
 - dealWithExternalMessage, 50
 - dealWithInternalMessage, 50
 - debugStatusInitialization, 50
 - ethernetFrameWaitForSend, 52
 - handleMessage, 50
 - initialize, 51
 - numberPriorities, 52
 - parentChannelOut, 53
 - parentGateOut, 53
 - parentModul, 53
 - prepareBufferStatusList, 51
 - rememberLastWinner, 53
 - sendBufferAuthorizedBuffer, 51
 - sendBufferRequest, 51
 - sendFrameOut, 51
 - settriggerDelayEnd, 52
 - triggerSelf, 53
- ArbiterIEEE802_3.cc, 172
- ArbiterIEEE802_3.h, 173
 - Define_Module, 173
- arbiterIsBusy
 - ArbiterAVB, 19
 - ArbiterIEEE802_1Q, 28
 - ArbiterIEEE802_1Qbv, 43
 - ArbiterIEEE802_3, 52
- arbitrateBufferStatusList
 - ArbiterAVB, 11
 - ArbiterIEEE802_1Q, 24
 - ArbiterIEEE802_1Qbv, 32
 - ArbiterIEEE802_3, 48
- avbTablePostition
 - ArbiterAVB.h, 170
- avbTableVector
 - ArbiterAVB, 19
 - ConfigurationReader, 67
- bestEffort
 - Reader, 121
 - Writer, 165
- bufferControlLevelRequest
 - BuffersModule, 55
- bufferSize
 - FrameBuffer, 88
- BuffersModule, 53
 - ~BuffersModule, 55
 - bufferControlLevelRequest, 55
 - BuffersModule, 55
 - castMessageIntoBufferControlMessage, 55
 - castMessageIntoEthernetFrame, 55
 - dealWithBufferControl, 56
 - dealWithEthernetFrame, 56
 - dealWithExternalMessage, 57
 - deleteBuffers, 57
 - deleteFramesInBuffers, 57
 - finish, 57
 - frameBuffers, 62
 - getBufferControlInstruction, 57

- getFrameBufferInformation, 58
- getMaximumBufferSize, 58
- getNumberSupportedPriorities, 58
- handleBufferControlInstruction, 59
- handleMessage, 59
- initialize, 59
- isFristArrivingFrameWithPriority, 59
- isMessageControlMessage, 60
- maximumBufferSize, 62
- numberSupportedPriorities, 62
- sendBufferControlMessage, 60
- sendEthernetFrameFromBuffer, 61
- setInitinalUserInputs, 61
- setUpBufferInformation, 61
- setUpFrameBuffers, 61
- storeEthernetFrameInBuffer, 62
- triggerUpdateMessageNewPriorityArrival, 62
- BuffersModule.cc, 173
- BuffersModule.h, 173
 - Define_Module, 174
- bufferUpdate
 - FrameBuffer.h, 185
- cache
 - ReaderHistory, 125
- calculateClassInformationCredits
 - ArbiterAVB, 11
- calculateCredits
 - ArbiterAVB, 11
- calculateCreditValues
 - ArbiterAVB, 11
- calculatedRechargeTime
 - ArbiterAVB, 13
- calculateGuardBandDurationFromPayloadSize
 - ArbiterIEEE802_1Qbv, 32
- calculateRtpsMsgSize
 - Endpoint, 72
- calculateTimeModulo
 - ArbiterIEEE802_1Qbv, 34
- calculateTransmissionDelay
 - ArbiterAVB, 13
 - ArbiterIEEE802_1Q, 24
 - ArbiterIEEE802_1Qbv, 34
 - ArbiterIEEE802_3, 48
- calculateTransmissionSpeed
 - ArbiterAVB, 13
 - ArbiterIEEE802_1Q, 25
 - ArbiterIEEE802_3, 49
- castMessageIntoBufferControlMessage
 - BuffersModule, 55
- castMessageIntoEthernetFrame
 - BuffersModule, 55
- channelTransmissionRate
 - ArbiterAVB, 19
 - ArbiterIEEE802_1Q, 28
 - ArbiterIEEE802_3, 52
- checkAndCastBufferControlMessage
 - ArbiterAVB, 13
- checkAndCastEthernetFrame
 - ArbiterAVB, 14
 - IngressPort, 106
- checkCompletion
 - Reader, 117
- checkDestinationsMAC
 - Sink, 140
- checkForMissingFrame
 - Sink, 140
- checkLifespan
 - Reader, 118
- checkLifespanOfSamples
 - Writer, 164
- CLASS_BLOCKED
 - ArbiterAVB.h, 170
- CLASS_IDLE
 - ArbiterAVB.h, 170
- CLASS_NOTSHAPED
 - ArbiterAVB.h, 170
- CLASS_SENDING
 - ArbiterAVB.h, 170
- CLASS_WAITING
 - ArbiterAVB.h, 170
- classCreditValue
 - arbiterAVBClass, 21
- classFrameWaiting
 - arbiterAVBClass, 22
- ClassifierPriority, 63
 - ~ClassifierPriority, 63
 - ClassifierPriority, 63
 - framePriority, 64
 - handleMessage, 64
 - initialize, 64
 - numberPriorities, 64
 - parentModul, 64
- ClassifierPriority.cc, 174
- ClassifierPriority.h, 174
 - Define_Module, 174
- classInformations
 - ArbiterAVB, 19
- classIsShaped
 - arbiterAVBClass, 22
- classStatus
 - arbiterAVBClass, 22
- cleanHistoryCache
 - Reader, 118
- collectCreditsStatistics
 - ArbiterAVB, 14
- COMMENT_POSITION
 - ConfigurationReader.h, 175
- COMMENT_SYMBOL
 - ConfigurationReader.h, 175
- complete
 - HistorySampleEntry, 102
- config
 - ArbiterAVB, 19
- configFile
 - ArbiterAVB, 19
- ConfigurationReader, 65

- ~ConfigurationReader, 65
- avbTableVector, 67
- ConfigurationReader, 65
- fileLine, 67
- getAVBTableEntriesFromInputFile, 66
- getInstance, 66
- getMACTableEntriesFromInputFile, 66
- inputFile, 67
- instance, 67
- isCommentLine, 66
- macTableVector, 67
- openConfigurationFile, 66
- readOutInputFile, 66
- splitFileLineAndGetEntry, 66
- ConfigurationReader.cc, 175
- ConfigurationReader.h, 175
 - COMMENT_POSITION, 175
 - COMMENT_SYMBOL, 175
 - configVector, 176
- configVector
 - ConfigurationReader.h, 176
- controlArbitrationBufferStatusList
 - ArbiterAVB, 14
 - ArbiterIEEE802_1Q, 25
 - ArbiterIEEE802_1Qbv, 35
 - ArbiterIEEE802_3, 49
- convertDoubleIntoTime
 - ArbiterIEEE802_1Qbv, 35
- createRtpsMsgFromFragment
 - Writer, 164
- creditValueStatisticStreamA
 - ArbiterAVB, 20
- creditValueStatisticStreamB
 - ArbiterAVB, 20
- CRITICAL_GUARDBAND
 - ArbiterIEEE802_1Qbv.h, 172
- CRITICAL_GUARDBAND_SEGMENT
 - TimeAwareShaperSegmentStatistic.h, 203
- CRITICAL_SENDING
 - ArbiterIEEE802_1Qbv.h, 172
- CRITICAL_SENDING_SEGMENT
 - TimeAwareShaperSegmentStatistic.h, 203
- criticalGuardBandSegmentDuration
 - ArbiterIEEE802_1Qbv, 43
- criticalGuardbandSegmentStarts
 - TimeAwareShaperSegmentStatistic, 146
- criticalIntervalDuration
 - ArbiterIEEE802_1Qbv, 43
- criticalPriorityClasses
 - ArbiterIEEE802_1Qbv, 43
- criticalSendingSegmentDuration
 - ArbiterIEEE802_1Qbv, 43
- criticalSendingSegmentStarts
 - TimeAwareShaperSegmentStatistic, 147
- csvVector
 - GlobalMacros.h, 189
- currentBufferStatusList
 - ArbiterAVB, 20
- ArbiterIEEE802_1Q, 28
- ArbiterIEEE802_1Qbv, 44
- ArbiterIEEE802_3, 52
- currentSampleSequenceNumber
 - Writer, 165
- currentState
 - ArbiterIEEE802_1Qbv, 44
- ddsSampleLatencyStatistics.cc, 176
- ddsSampleLatencyStatistics.h, 176
- deadline
 - Reader, 121
- deadlineViolation
 - HistorySampleEntry, 102
- dealWithBufferControl
 - BuffersModule, 56
- dealWithControlMessage
 - ArbiterAVB, 15
 - ArbiterIEEE802_1Q, 25
 - ArbiterIEEE802_1Qbv, 35
 - ArbiterIEEE802_3, 49
- dealWithEthernetFrame
 - ArbiterAVB, 15
 - ArbiterIEEE802_1Q, 25
 - ArbiterIEEE802_1Qbv, 36
 - ArbiterIEEE802_3, 50
 - BuffersModule, 56
 - IngressPort, 106
- dealWithExternalMessage
 - ArbiterAVB, 15
 - ArbiterIEEE802_1Q, 26
 - ArbiterIEEE802_1Qbv, 36
 - ArbiterIEEE802_3, 50
 - BuffersModule, 57
 - IngressPort, 107
- dealWithInternalMessage
 - ArbiterAVB, 16
 - ArbiterIEEE802_1Q, 26
 - ArbiterIEEE802_1Qbv, 36
 - ArbiterIEEE802_3, 50
 - IngressPort, 107
- dealWithMulticast
 - ForwardingLogic, 78
- dealWithSegmentTimerMessage
 - ArbiterIEEE802_1Qbv, 37
- dealWithTriggerSelf
 - ArbiterIEEE802_1Qbv, 37
- dealWithUnicast
 - ForwardingLogic, 78
- debugStatus
 - ArbiterAVB, 16
- debugStatusInitialization
 - ArbiterAVB, 16
 - ArbiterIEEE802_3, 50
- Define_Module
 - ArbiterAVB.h, 170
 - ArbiterIEEE802_1Q.h, 171
 - ArbiterIEEE802_1Qbv.h, 172
 - ArbiterIEEE802_3.h, 173

- BuffersModule.h, 174
- ClassifierPriority.h, 174
- DelayUnit.cc, 176
- EgressPort.h, 178
- EthMsgCreator.h, 183
- ForwardingLogic.h, 185
- IngressPort.h, 191
- JoinReaders.cc, 191
- Participant.h, 193
- Reader.h, 193
- Rtps.cc, 194
- RtpsToEthernetAdapter.cc, 198
- SampleMsgCreator.h, 199
- SampleSink.cc, 200
- Shaper.cc, 201
- Sink.h, 202
- TrafficSource.h, 204
- UdplpStack.h, 205
- Writer.h, 207
- DelayUnit, 67
 - ~DelayUnit, 68
 - DelayUnit, 68
 - handleMessage, 68
 - initialize, 69
- DelayUnit.cc, 176
 - Define_Module, 176
- DelayUnit.h, 177
- deleteBuffers
 - BuffersModule, 57
- deleteFramesInBuffer
 - FrameBuffer, 86
- deleteFramesInBuffers
 - BuffersModule, 57
- destinationEntityId
 - Writer, 166
- destinationMac
 - Writer, 166
- droppedBytesInBuffer
 - FrameBuffer, 89
- droppedBytesVector
 - FrameBuffer, 89
- droppedFramesInBuffer
 - FrameBuffer, 89
- droppedFramesVector
 - FrameBuffer, 89
- E2ELatencyStatistics, 69
 - ~E2ELatencyStatistics, 69
 - addE2ELatency, 70
 - E2ELatencyStatistics, 69
 - finish, 70
 - histogramE2Estimates, 70
 - vectorE2Estimates, 70
- E2ELatencyStatistics.cc, 177
- E2ELatencyStatistics.h, 177
- EGRESS_PORT_ID
 - ArbiterAVB.h, 170
- EgressPort, 71
 - ~EgressPort, 71
- EgressPort, 71
- EgressPort.cc, 177
- EgressPort.h, 177
 - Define_Module, 178
- egressPortIndex
 - ArbiterAVB, 20
- empty
 - HistogramStatistics3DMap, 100
 - VectorStatistics3DMap, 161
- enableFineGrainedLatencyStats
 - EthMsgCreator, 76
- Endpoint, 71
 - ~Endpoint, 72
 - calculateRtpsMsgSize, 72
 - Endpoint, 72
 - entityId, 73
 - entityKind, 73
 - getNextUniqueRtpsMsgId, 73
 - rtpsMsgSequenceId, 73
- Endpoint.cc, 178
- Endpoint.h, 178
- entityId
 - Endpoint, 73
 - Reader, 121
 - Writer, 166
- entityKind
 - Endpoint, 73
- errorHandling
 - ForwardingLogic, 79
- errorType
 - ForwardingLogic.h, 184
- ethernetFrame
 - ForwardingLogic, 84
- ethernetFrameFinishedSending
 - LinkTransmissionStatistics, 111
- ethernetFrameStartedSending
 - LinkTransmissionStatistics, 111
- ethernetFrameWaitForSend
 - ArbiterAVB, 20
 - ArbiterIEEE802_1Q, 28
 - ArbiterIEEE802_3, 52
- ethernetFrameWaitForTransmission
 - ArbiterIEEE802_1Qbv, 44
- ethernetHeaderInformation, 73
 - payloadLength, 74
 - priority, 74
 - ready, 74
- EthernetHeaderInformation.h, 178
- EthernetMacros.h, 179
 - _CRC_, 179
 - _CRC_INTERPACKET_GAP_, 179
 - _FRAME_OVERHEAD_, 180
 - _INTERPACKET_GAP_, 180
 - _MAX_ETH_PAYLOAD_, 180
 - _MIN_ETH_FRAME_SIZE_, 181
 - _MIN_ETH_PACKET_SIZE_, 181
 - _MIN_ETH_PACKET_SIZE_PREEMPTION_, 181
 - _MIN_ETH_PAYLOAD_, 181

- _MIN_ETH_PAYLOAD_UNTAGGED, 181
 - _PACKET_OVERHEAD_, 181
 - _PACKET_OVERHEAD_FIRST_PREEMPTION_, 181
 - _PACKET_OVERHEAD_INTERMEDIATE_PREEMPTION_, 181
 - _PAYLOAD_INTERMEDIATE_PREEMPTION_, 182
 - _eth_check_for_max_payload_, 180
 - _eth_check_for_padding_, 180
 - _eth_egress_helper_message_type_, 182
 - _eth_framesize_from_payload_, 180
 - _eth_packetsize_from_payload_, 180
 - _std_eth_colour_, 182
 - frame_arrived, 182
 - frame_dropped, 182
 - frame_finished_sending, 182
 - frame_started_sending, 182
 - port_update, 182
- EthMsgCreator, 74
 - ~EthMsgCreator, 75
 - enableFineGrainedLatencyStats, 76
 - EthMsgCreator, 75
 - finish, 75
 - getNextUniqueFrameId, 75
 - handleMessage, 76
 - initialize, 76
 - uniqueFrameId, 76
- EthMsgCreator.cc, 182
- EthMsgCreator.h, 182
 - Define_Module, 183
- fileLine
 - ConfigurationReader, 67
- finish
 - BuffersModule, 57
 - E2ELatencyStatistics, 70
 - EthMsgCreator, 75
 - FramesLostStatistics, 91
 - HistogramStatistics2DMap, 94
 - HistogramStatistics3DMap, 98
 - JoinReaders, 109
 - LinkTransmissionStatistics, 112
 - Participant, 114
 - Reader, 118
 - RtpsToEthernetAdapter, 127
 - SampleLatencyStatistics, 132
 - SampleMsgCreator, 134
 - Shaper, 137
 - Sink, 140
 - TimeAwareShaperSegmentStatistic, 147
 - VectorStatistics2DMap, 156
 - VectorStatistics3DMap, 160
 - Writer, 164
- finishAll
 - StatisticManager, 143
- forwardingByStreamID
 - ForwardingLogic, 79
- ForwardingLogic, 77
 - ~ForwardingLogic, 78
 - dealWithMulticast, 78
 - dealWithUnicast, 78
 - errorHandling, 79
 - ethernetFrame, 84
 - forwardingByStreamID, 79
 - ForwardingLogic, 78
 - getMACTable, 80
 - getOnlyEntriesForThisSwitch, 80
 - getPortIDFromNextHOP, 80
 - handleMessage, 81
 - initialize, 81
 - isConnectedModuleEndNode, 81
 - isHopMACAddressEqualEndNodeMAC, 82
 - isHopMACAddressEqualSwitchMAC, 82
 - isValidNumberOfEntries, 83
 - macTable, 84
 - macTableFile, 84
 - parentSwitch, 84
 - portIDInvalid, 83
 - printSimulationOutputs, 83
 - streamID, 84
 - switchIndex, 84
 - switchMAC, 84
- ForwardingLogic.cc, 183
- ForwardingLogic.h, 183
 - Define_Module, 185
 - errorType, 184
 - INCORRECT_MESSAGE_TYPE, 184
 - INIT_OUTPUT, 185
 - INVALID_NUMBER_MAC_TABLE_ENTRIES, 184
 - INVALID_PORT_ID, 184
 - MAC_TABLE_CONFIGURATION_INFO, 185
 - macTableEntriesPosition, 184
 - NEXT_HOP_MAC, 184
 - simulationOutputType, 184
 - STREAM_ID, 184
 - SWITCH_MAC, 184
 - UNKNOWN_STREAM_ID, 184
- fragmentSequenceNumber
 - SampleFragment, 130
- fragmentSize
 - SampleFragment, 130
 - Writer, 166
- frame_arrived
 - EthernetMacros.h, 182
- frame_dropped
 - EthernetMacros.h, 182
- frame_finished_sending
 - EthernetMacros.h, 182
- frame_started_sending
 - EthernetMacros.h, 182
- FrameBuffer, 85
 - ~FrameBuffer, 86
 - bufferSize, 88
 - deleteFramesInBuffer, 86
 - droppedBytesInBuffer, 89
 - droppedBytesVector, 89

- droppedFramesInBuffer, 89
- droppedFramesVector, 89
- FrameBuffer, 86
- frameBuffer, 89
- frameNumberInBuffer, 89
- frameNumberVector, 89
- framePriority, 89
- freeSpaceInBuffer, 86
- getFrameFillLevel, 87
- getFramePriority, 87
- getNextFrameLength, 87
- popNextFrame, 87
- pushFrameToBack, 88
- storedBytesInBuffer, 90
- storedBytesVector, 90
- updateFreeBufferSize, 88
- frameBuffer
 - FrameBuffer, 89
- FrameBuffer.cc, 185
- FrameBuffer.h, 185
 - ADDED_FRAME, 186
 - bufferUpdate, 185
 - REMOVED_FRAME, 186
- frameBuffers
 - BuffersModule, 62
- frameLatencyInjectionToReception
 - Reader, 121
- frameLatencyInjectionToReceptionVec
 - Reader, 121
- frameNumberInBuffer
 - FrameBuffer, 89
- frameNumberVector
 - FrameBuffer, 89
- framePriority
 - ClassifierPriority, 64
 - FrameBuffer, 89
- frameSequenceld
 - RtpsToEthernetAdapter, 128
- FramesLostStatistics, 90
 - ~FramesLostStatistics, 91
 - addLostFrame, 91
 - addLostFrameTrace, 91
 - finish, 91
 - FramesLostStatistics, 90
 - histogramLostFramesPerStream, 92
 - vectorLostFramesPerStream, 92
 - vectorLostFramesTracePerStream, 92
- FramesLostStatistics.cc, 186
- FramesLostStatistics.h, 186
- framesLostStats
 - StatisticManager, 144
- frameStoredInBuffer
 - ArbiterIEEE802_1Qbv, 44
- freeSpaceInBuffer
 - FrameBuffer, 86
- Frer_ID, 92
 - port_ids, 92
- generationTime
 - HistorySampleEntry, 102
- generatorEnable
 - TrafficSource, 150
- getAVBTableEntriesFromInputFile
 - ConfigurationReader, 66
- getBufferControlInstruction
 - BuffersModule, 57
- getCache
 - Reader, 118
- getCriticalClasses
 - ArbiterIEEE802_1Qbv, 37
- getCriticalIntervalDuration
 - ArbiterIEEE802_1Qbv, 37
- getDelayStatistics
 - StatisticManager, 143
- getEntry
 - HistogramStatistics3DMap, 99
 - VectorStatistics2DMap, 156
 - VectorStatistics3DMap, 161
- getFrameBufferInformation
 - BuffersModule, 58
- getFrameFillLevel
 - FrameBuffer, 87
- getFramePriority
 - FrameBuffer, 87
- getFramesLostStatistics
 - StatisticManager, 143
- getInitialStateAndCalculateTriggerTimer
 - ArbiterIEEE802_1Qbv, 38
- getInstance
 - ConfigurationReader, 66
 - StatisticManager, 143
- getLinkTransmissionStatistics
 - StatisticManager, 143
- getMACTable
 - ForwardingLogic, 80
- getMACTableEntriesFromInputFile
 - ConfigurationReader, 66
- getMaximumBufferSize
 - BuffersModule, 58
- getNextFrameLength
 - FrameBuffer, 87
- getNextFrameSequenceld
 - RtpsToEthernetAdapter, 128
- getNextSampleSequenceNumberId
 - SampleMsgCreator, 134
- getNextState
 - ArbiterIEEE802_1Qbv, 38
- getNextUniqueFrameId
 - EthMsgCreator, 75
- getNextUniqueRtpsMsgId
 - Endpoint, 73
- getNumberSupportedPriorities
 - BuffersModule, 58
- getOnlyEntriesForThisSwitch
 - ForwardingLogic, 80
- getPeriodDuration
 - ArbiterIEEE802_1Qbv, 38

- getPortIDFromNextHOP
 - ForwardingLogic, 80
- getPortSpeed
 - ArbiterIEEE802_1Qbv, 39
- getSampleLatencyStatistics
 - StatisticManager, 144
- getSegmentDurationFromCurrentState
 - ArbiterIEEE802_1Qbv, 39
- getTimeAwareShaperStatistics
 - StatisticManager, 144
- getTimeOffset
 - ArbiterIEEE802_1Qbv, 39
- GlobalMacros.h, 186
 - _CAN2CAN_LOOKUP_, 187
 - _CAN2ETH_LOOKUP_, 187
 - _ETH2CAN_LOOKUP_, 187
 - _ETH2ETH_LOOKUP_, 187
 - _max_, 187
 - _min_, 187
 - _msg_black_, 188
 - _msg_blue_, 188
 - _msg_cyan_, 188
 - _msg_green_, 188
 - _msg_magenta_, 188
 - _msg_red_, 188
 - _msg_white_, 188
 - _msg_yellow_, 189
 - csvVector, 189
 - simtime_modulo, 189
 - truncate_min_max, 189
- globalStreamId
 - HistorySampleEntry, 102
 - TrafficSource, 151
 - Writer, 166
- handleBufferControlInstruction
 - BuffersModule, 59
- handleMessage
 - ArbiterAVB, 16
 - ArbiterIEEE802_1Q, 26
 - ArbiterIEEE802_1Qbv, 40
 - ArbiterIEEE802_3, 50
 - BuffersModule, 59
 - ClassifierPriority, 64
 - DelayUnit, 68
 - EthMsgCreator, 76
 - ForwardingLogic, 81
 - IngressPort, 107
 - JoinReaders, 110
 - Participant, 114
 - Reader, 119
 - RtpsToEthernetAdapter, 128
 - SampleMsgCreator, 134
 - SampleSink, 136
 - Shaper, 137
 - Sink, 140
 - TrafficSource, 149
 - UdplpStack, 152
 - Writer, 164
- hasFinishedAll
 - StatisticManager, 144
- histogramE2Estimates
 - E2ELatencyStatistics, 70
- histogramLostFramesPerStream
 - FramesLostStatistics, 92
- HistogramStatistics2DMap, 92
 - ~HistogramStatistics2DMap, 94
 - addIntegerResult, 94
 - addLatency, 94
 - finish, 94
 - HistogramStatistics2DMap, 93
 - modulePrefix, 95
 - prefix, 95
 - storedMeasurements, 95
 - submodulePrefix, 95
 - valuePrefix, 95
- HistogramStatistics2DMap.cc, 189
- HistogramStatistics2DMap.h, 190
- HistogramStatistics3DMap, 96
 - ~HistogramStatistics3DMap, 97
 - addInteger, 97
 - addLatency, 98
 - empty, 100
 - finish, 98
 - getEntry, 99
 - HistogramStatistics3DMap, 97
 - modulePrefix, 100
 - prefix, 100
 - storedMeasurements, 100
 - submodulePrefix, 100
 - valuePrefix, 100
- HistogramStatistics3DMap.cc, 190
- HistogramStatistics3DMap.h, 190
- historyCache
 - Writer, 166
- HistoryEntry.h, 190
- HistorySampleEntry, 101
 - ~HistorySampleEntry, 102
 - complete, 102
 - deadlineViolation, 102
 - generationTime, 102
 - globalStreamId, 102
 - HistorySampleEntry, 102
 - historySizeExceeded, 103
 - lifespanExpired, 103
 - numberFragments, 103
 - readerEntityId, 103
 - readerParticipantEntityId, 103
 - rtpsMsgLatencies, 103
 - sampleFragmentArray, 104
 - sampleSequenceNumber, 104
 - sampleSize, 104
 - subscriberReceiveTime, 104
 - writerArrivalTime, 104
 - writerEntityId, 104
 - writerFirstInjectionTime, 104
 - writerParticipantEntityId, 105

- historySizeExceeded
 - HistorySampleEntry, [103](#)
- historySizeQoS
 - Reader, [121](#)
 - Writer, [167](#)
- IDLE_SLOPE
 - ArbiterAVB.h, [170](#)
- idleSlope
 - arbiterAVBClass, [22](#)
- INCORRECT_MESSAGE_TYPE
 - ForwardingLogic.h, [184](#)
- IngressPort, [105](#)
 - ~IngressPort, [106](#)
 - checkAndCastEthernetFrame, [106](#)
 - dealWithEthernetFrame, [106](#)
 - dealWithExternalMessage, [107](#)
 - dealWithInternalMessage, [107](#)
 - handleMessage, [107](#)
 - IngressPort, [106](#)
 - ingressPortID, [108](#)
 - initialize, [107](#)
 - linkUtilizationStatistics, [108](#)
 - macAddress, [108](#)
 - numBytes, [108](#)
 - sampleTimePeriod, [108](#)
 - triggerSelf, [108](#)
- IngressPort.cc, [190](#)
- IngressPort.h, [191](#)
 - Define_Module, [191](#)
- ingressPortID
 - IngressPort, [108](#)
- INIT_OUTPUT
 - ForwardingLogic.h, [185](#)
- initialize
 - ArbiterAVB, [16](#)
 - ArbiterIEEE802_1Q, [26](#)
 - ArbiterIEEE802_1Qbv, [40](#)
 - ArbiterIEEE802_3, [51](#)
 - BuffersModule, [59](#)
 - ClassifierPriority, [64](#)
 - DelayUnit, [69](#)
 - EthMsgCreator, [76](#)
 - ForwardingLogic, [81](#)
 - IngressPort, [107](#)
 - JoinReaders, [110](#)
 - Participant, [114](#)
 - Reader, [119](#)
 - Rtps, [126](#)
 - RtpsToEthernetAdapter, [128](#)
 - SampleMsgCreator, [135](#)
 - SampleSink, [136](#)
 - Shaper, [138](#)
 - Sink, [140](#)
 - TrafficSource, [150](#)
 - UdplpStack, [153](#)
 - Writer, [165](#)
- initializeAVBClassInformation
 - ArbiterAVB, [16](#)
- initializeTimerMessage
 - ArbiterIEEE802_1Qbv, [40](#)
- inputFile
 - ConfigurationReader, [67](#)
- insertSampleInCache
 - Reader, [119](#)
- instance
 - ConfigurationReader, [67](#)
- interBurstMsg
 - TrafficSource, [151](#)
- intraBurstMsg
 - TrafficSource, [151](#)
- INVALID_NUMBER_MAC_TABLE_ENTRIES
 - ForwardingLogic.h, [184](#)
- INVALID_PORT_ID
 - ForwardingLogic.h, [184](#)
- isCommentLine
 - ConfigurationReader, [66](#)
- isConnectedModuleEndNode
 - ForwardingLogic, [81](#)
- isCritical
 - ArbiterIEEE802_1Qbv, [44](#)
- isFristArrivingFrameWithPriority
 - BuffersModule, [59](#)
- isHopMACAddressEqualEndNodeMAC
 - ForwardingLogic, [82](#)
- isHopMACAddressEqualSwitchMAC
 - ForwardingLogic, [82](#)
- isMessageControlMessage
 - BuffersModule, [60](#)
- isValidNumberOfEntries
 - ForwardingLogic, [83](#)
- JoinReaders, [109](#)
 - ~JoinReaders, [109](#)
 - finish, [109](#)
 - handleMessage, [110](#)
 - initialize, [110](#)
 - JoinReaders, [109](#)
- JoinReaders.cc, [191](#)
 - Define_Module, [191](#)
- JoinReaders.h, [192](#)
- lastArbitrationTimeStamp
 - arbiterAVBClass, [22](#)
- lifespan
 - Reader, [122](#)
 - Writer, [167](#)
- lifespanExpired
 - HistorySampleEntry, [103](#)
- linkTransmission
 - StatisticManager, [145](#)
- LinkTransmissionStatistics, [110](#)
 - ~LinkTransmissionStatistics, [111](#)
 - activationPeriod, [112](#)
 - ethernetFrameFinishedSending, [111](#)
 - ethernetFrameStartedSending, [111](#)
 - finish, [112](#)
 - LinkTransmissionStatistics, [110](#)

- LinkTransmissionStatistics.cc, [192](#)
- LinkTransmissionStatistics.h, [192](#)
- linkUtilizationStatistics
 - IngressPort, [108](#)
- loadAVBClassInformationFromFile
 - ArbiterAVB, [17](#)
- lostFrameCounter, [112](#)
 - lostFrames, [112](#)
 - newestFrameID, [112](#)
- lostFrameCounters
 - Sink, [141](#)
- lostFrames
 - lostFrameCounter, [112](#)
- MAC_TABLE_CONFIGURATION_INFO
 - ForwardingLogic.h, [185](#)
- macAddress
 - ArbiterAVB, [20](#)
 - IngressPort, [108](#)
 - Sink, [141](#)
- macTable
 - ForwardingLogic, [84](#)
- macTableEntriesPositon
 - ForwardingLogic.h, [184](#)
- macTableFile
 - ForwardingLogic, [84](#)
- macTableVector
 - ConfigurationReader, [67](#)
- maximumBufferSize
 - BuffersModule, [62](#)
- maximumCriticalPayloadSize
 - ArbiterIEEE802_1Qbv, [44](#)
- maximumNoncriticalPayloadSize
 - ArbiterIEEE802_1Qbv, [45](#)
- maxNbrStreams
 - ReaderHistory, [125](#)
- modulePrefix
 - HistogramStatistics2DMap, [95](#)
 - HistogramStatistics3DMap, [100](#)
 - VectorStatistics2DMap, [157](#)
 - VectorStatistics3DMap, [161](#)
- myMac
 - SampleSink, [136](#)
- nbrReaders
 - Participant, [114](#)
- nbrStreams
 - ReaderHistory, [125](#)
- nbrWriters
 - Participant, [115](#)
- newestFrameID
 - lostFrameCounter, [112](#)
- NEXT_HOP_MAC
 - ForwardingLogic.h, [184](#)
- nextMsgReady
 - Shaper, [138](#)
- nextState
 - ArbiterIEEE802_1Qbv, [45](#)
- NONCRITICAL_GUARDBAND
 - ArbiterIEEE802_1Qbv.h, [172](#)
 - NONCRITICAL_GUARDBAND_SEGMENT
 - TimeAwareShaperSegmentStatistic.h, [203](#)
 - NONCRITICAL_SENDING
 - ArbiterIEEE802_1Qbv.h, [172](#)
 - NONCRITICAL_SENDING_SEGMENT
 - TimeAwareShaperSegmentStatistic.h, [203](#)
 - noncriticalGuardBandSegmentDuration
 - ArbiterIEEE802_1Qbv, [45](#)
 - noncriticalGuardbandSegmentStarts
 - TimeAwareShaperSegmentStatistic, [147](#)
 - noncriticalIntervalDuration
 - ArbiterIEEE802_1Qbv, [45](#)
 - noncriticalSendingSegmentDuration
 - ArbiterIEEE802_1Qbv, [45](#)
 - noncriticalSendingSegmentStarts
 - TimeAwareShaperSegmentStatistic, [147](#)
 - numberFragments
 - HistorySampleEntry, [103](#)
 - numberPriorities
 - ArbiterAVB, [20](#)
 - ArbiterIEEE802_1Q, [28](#)
 - ArbiterIEEE802_3, [52](#)
 - ClassifierPriority, [64](#)
 - numberSupportedPriorities
 - ArbiterIEEE802_1Qbv, [45](#)
 - BuffersModule, [62](#)
 - numBytes
 - IngressPort, [108](#)
- openConfigurationFile
 - ConfigurationReader, [66](#)
- parentChannelOut
 - ArbiterAVB, [20](#)
 - ArbiterIEEE802_1Q, [28](#)
 - ArbiterIEEE802_3, [53](#)
- parentGateOut
 - ArbiterAVB, [21](#)
 - ArbiterIEEE802_1Q, [28](#)
 - ArbiterIEEE802_3, [53](#)
- parentModul
 - ArbiterAVB, [21](#)
 - ArbiterIEEE802_1Q, [29](#)
 - ArbiterIEEE802_3, [53](#)
 - ClassifierPriority, [64](#)
- parentSwitch
 - ForwardingLogic, [84](#)
- Participant, [113](#)
 - ~Participant, [114](#)
 - finish, [114](#)
 - handleMessage, [114](#)
 - initialize, [114](#)
 - nbrReaders, [114](#)
 - nbrWriters, [115](#)
 - Participant, [113](#)
 - participantId, [115](#)
 - readerIds, [115](#)
 - writerIds, [115](#)

- participant
 - RtpsMacros.h, 197
- Participant.cc, 192
- Participant.h, 192
 - Define_Module, 193
- participantId
 - Participant, 115
- pathDelayStats
 - StatisticManager, 145
- payloadLength
 - ethernetHeaderInformation, 74
- pendingBursts
 - TrafficSource, 151
- pendingIntraBurstSize
 - TrafficSource, 151
- periodDuration
 - ArbiterIEEE802_1Qbv, 46
- popNextFrame
 - FrameBuffer, 87
- port_ids
 - Frer_ID, 92
- port_update
 - EthernetMacros.h, 182
- portIDInvalid
 - ForwardingLogic, 83
- portIndex
 - ArbiterIEEE802_1Qbv, 46
- prefix
 - HistogramStatistics2DMap, 95
 - HistogramStatistics3DMap, 100
 - VectorStatistics2DMap, 157
 - VectorStatistics3DMap, 161
- prepareBufferStatusList
 - ArbiterAVB, 17
 - ArbiterIEEE802_1Q, 26
 - ArbiterIEEE802_1Qbv, 40
 - ArbiterIEEE802_3, 51
- prepareClassInformations
 - ArbiterAVB, 17
- printInitializationConfiguration
 - ArbiterIEEE802_1Qbv, 41
- printSimulationOutputs
 - ForwardingLogic, 83
- PRIORITY
 - ArbiterAVB.h, 170
- priority
 - ethernetHeaderInformation, 74
- processEthernetFrameAtDestination
 - Sink, 140
- pushFrameToBack
 - FrameBuffer, 88
- Reader, 115
 - ~Reader, 117
 - bestEffort, 121
 - checkCompletion, 117
 - checkLifespan, 118
 - cleanHistoryCache, 118
 - deadline, 121
 - entityId, 121
 - finish, 118
 - frameLatencyInjectionToReception, 121
 - frameLatencyInjectionToReceptionVec, 121
 - getCache, 118
 - handleMessage, 119
 - historySizeQoS, 121
 - initialize, 119
 - insertSampleInCache, 119
 - lifespan, 122
 - Reader, 117
 - readerHistoryCache, 122
 - registerEvent, 122
 - registerReader, 119
 - relevantNewSample, 119
 - SampleLatencySinceGenerationStat, 122
 - SampleLatencySinceGenerationStatVec, 122
 - SampleLatencySinceSampleInjectionStat, 122
 - SampleLatencySinceSampleInjectionStatVec, 123
 - SampleLatencySinceWriterArrivalStat, 123
 - SampleLatencySinceWriterArrivalStatVec, 123
 - sendSampleToSink, 120
 - sourceMac, 123
 - streamEthernetPriority, 123
 - transportPriority, 123
 - updateCache, 120
- Reader.cc, 193
- Reader.h, 193
 - Define_Module, 193
- readerEntityId
 - HistorySampleEntry, 103
- ReaderHistory, 124
 - ~ReaderHistory, 124
 - cache, 125
 - maxNbrStreams, 125
 - nbrStreams, 125
 - ReaderHistory, 124
- readerHistoryCache
 - Reader, 122
- readerIds
 - Participant, 115
- readerNoKey
 - RtpsMacros.h, 197
- readerParticipantEntityId
 - HistorySampleEntry, 103
- readerWithKey
 - RtpsMacros.h, 197
- README_RTPS.md, 194
- readOutInputFile
 - ConfigurationReader, 66
- ready
 - ethernetHeaderInformation, 74
- received
 - SampleFragment, 130
- registerEvent
 - Reader, 122
 - Writer, 167
- registerReader

- Reader, 119
- registerWriter
 - Writer, 165
- relevantNewSample
 - Reader, 119
- rememberLastWinner
 - ArbiterIEEE802_3, 53
- REMOVED_FRAME
 - FrameBuffer.h, 186
- report
 - SampleLatencyStatistics, 132
- Rtps, 125
 - ~Rtps, 126
 - initialize, 126
 - Rtps, 126
- Rtps.cc, 194
 - Define_Module, 194
- Rtps.h, 194
- rtpsEntityKind
 - RtpsMacros.h, 197
- RtpsMacros.h, 194
 - _IP_HEADER_SIZE_, 195
 - _MAX_RTPS_PAYLOAD_, 195
 - _MIN_RTPS_PAYLOAD_, 195
 - _RTPS_DATAFRAG_STATIC_SIZE_, 196
 - _RTPS_HEADER_SIZE_, 196
 - _RTPS_HEARTBEATFRAG_SIZE_, 196
 - _RTPS_INFO_DST_SIZE_, 196
 - _RTPS_INFO_TS_SIZE_, 196
 - _RTPS_MAX_FRAG_SIZE_, 197
 - _RTPS_MAX_MSG_SIZE_, 197
 - _RTPS_NACKFRAG_STATIC_SIZE_, 197
 - _UDP_HEADER_SIZE_, 197
 - _rtps_add_udp_ip_overhead_, 196
 - _rtps_check_for_max_payload_, 196
- participant, 197
- readerNoKey, 197
- readerWithKey, 197
- rtpsEntityKind, 197
- unknown, 197
- writerNoKey, 197
- writerWithKey, 197
- rtpsMsgLatencies
 - HistorySampleEntry, 103
- rtpsMsgQueue
 - Shaper, 138
- rtpsMsgSequenceld
 - Endpoint, 73
- RtpsToEthernetAdapter, 126
 - ~RtpsToEthernetAdapter, 127
 - finish, 127
 - frameSequenceld, 128
 - getNextFrameSequenceld, 128
 - handleMessage, 128
 - initialize, 128
 - RtpsToEthernetAdapter, 127
 - sendRtpsPacket, 128
- RtpsToEthernetAdapter.cc, 198
 - Define_Module, 198
- RtpsToEthernetAdapter.h, 198
 - _adapter_src_verbose_, 198
- SampleFragment, 129
 - ~SampleFragment, 129
 - acked, 130
 - fragmentSequenceNumber, 130
 - fragmentSize, 130
 - received, 130
 - SampleFragment, 129
 - sampleReference, 130
 - send, 130
 - sendTime, 131
- sampleFragmentArray
 - HistorySampleEntry, 104
- SampleLatencySinceGenerationStat
 - Reader, 122
- SampleLatencySinceGenerationStatVec
 - Reader, 122
- SampleLatencySinceSampleInjectionStat
 - Reader, 122
- SampleLatencySinceSampleInjectionStatVec
 - Reader, 123
- SampleLatencySinceWriterArrivalStat
 - Reader, 123
- SampleLatencySinceWriterArrivalStatVec
 - Reader, 123
- SampleLatencyStatistics, 131
 - ~SampleLatencyStatistics, 131
 - addMessage, 132
 - finish, 132
 - report, 132
 - SampleLatencyStatistics, 131
 - sampleLatencyStatisticsHistogram, 133
 - sampleLatencyStatisticsVector, 133
- sampleLatencyStatisticsHistogram
 - SampleLatencyStatistics, 133
- sampleLatencyStatisticsVector
 - SampleLatencyStatistics, 133
- sampleLatencyStats
 - StatisticManager, 145
- SampleMsgCreator, 133
 - ~SampleMsgCreator, 134
 - finish, 134
 - getNextSampleSequenceNumberId, 134
 - handleMessage, 134
 - initialize, 135
 - SampleMsgCreator, 134
 - sampleSequenceNumberId, 135
- SampleMsgCreator.cc, 199
- SampleMsgCreator.h, 199
 - Define_Module, 199
- sampleReference
 - SampleFragment, 130
- sampleSequenceNumber
 - HistorySampleEntry, 104
- sampleSequenceNumberId
 - SampleMsgCreator, 135

- SampleSink, 135
 - handleMessage, 136
 - initialize, 136
 - myMac, 136
- SampleSink.cc, 199
 - Define_Module, 200
- SampleSink.h, 200
 - _eth_sampleSink_verbose_, 200
- sampleSize
 - HistorySampleEntry, 104
- sampleTimePeriod
 - IngressPort, 108
- scheduleNextBurst
 - TrafficSource, 150
- scheduleNextIntraBurst
 - TrafficSource, 150
- segmentTriggerTimerMessage
 - ArbiterIEEE802_1Qbv, 46
- send
 - SampleFragment, 130
- send_from_queue
 - Shaper, 138
- send_queue
 - Writer, 167
- SEND_SLOPE
 - ArbiterAVB.h, 170
- sendBufferAuthorizedBuffer
 - ArbiterAVB, 17
 - ArbiterIEEE802_1Q, 27
 - ArbiterIEEE802_1Qbv, 41
 - ArbiterIEEE802_3, 51
- sendBufferControlMessage
 - BuffersModule, 60
- sendBufferRequest
 - ArbiterAVB, 18
 - ArbiterIEEE802_1Q, 27
 - ArbiterIEEE802_1Qbv, 41
 - ArbiterIEEE802_3, 51
- sendEthernetFrameFromBuffer
 - BuffersModule, 61
- sendEvent
 - Writer, 167
- sendFrameOut
 - ArbiterAVB, 18
 - ArbiterIEEE802_1Q, 27
 - ArbiterIEEE802_1Qbv, 41
 - ArbiterIEEE802_3, 51
- sendMessage
 - Writer, 165
- sendPacket
 - TrafficSource, 150
- sendRtpsPacket
 - RtpsToEthernetAdapter, 128
- sendSampleToSink
 - Reader, 120
- sendSlope
 - arbiterAVBClass, 22
- sendTime
 - SampleFragment, 131
- setEthPacketSizeFromPayload
 - ArbiterIEEE802_1Qbv, 42
- setInitialUserInputs
 - BuffersModule, 61
- setRechargeTime
 - ArbiterAVB, 18
- settriggerCreditRecharged
 - ArbiterAVB, 18
- setTriggerDelayEnd
 - ArbiterIEEE802_1Qbv, 42
- settriggerDelayEnd
 - ArbiterAVB, 18
 - ArbiterIEEE802_1Q, 27
 - ArbiterIEEE802_3, 52
- setUpBufferInformation
 - BuffersModule, 61
- setUpFrameBuffers
 - BuffersModule, 61
- setUpVectorCriticalPriorities
 - ArbiterIEEE802_1Qbv, 42
- Shaper, 136
 - ~Shaper, 137
 - finish, 137
 - handleMessage, 137
 - initialize, 138
 - nextMsgReady, 138
 - rtpsMsgQueue, 138
 - send_from_queue, 138
 - Shaper, 137
 - shaperEnabled, 138
 - shapingDuration, 138
- Shaper.cc, 200
 - Define_Module, 201
- Shaper.h, 201
- shaperEnabled
 - Shaper, 138
- shapingDuration
 - Shaper, 138
 - Writer, 167
- simtime_modulo
 - GlobalMacros.h, 189
- simulationOutputType
 - ForwardingLogic.h, 184
- Sink, 139
 - checkDestinationsMAC, 140
 - checkForMissingFrame, 140
 - finish, 140
 - handleMessage, 140
 - initialize, 140
 - lostFrameCounters, 141
 - macAddress, 141
 - processEthernetFrameAtDestination, 140
 - trackMissingDataMap, 141
- Sink.cc, 201
- Sink.h, 201
 - Define_Module, 202
- sourceMac

- Reader, 123
- Writer, 168
- splitFileLineAndGetEntry
 - ConfigurationReader, 66
- StatisticManager, 141
 - ~StatisticManager, 142
 - finishAll, 143
 - framesLostStats, 144
 - getDelayStatistics, 143
 - getFramesLostStatistics, 143
 - getInstance, 143
 - getLinkTransmissionStatistics, 143
 - getSampleLatencyStatistics, 144
 - getTimeAwareShaperStatistics, 144
 - hasFinishedAll, 144
 - linkTransmission, 145
 - pathDelayStats, 145
 - sampleLatencyStats, 145
 - StatisticManager, 142
 - tsnTASStatistics, 145
- StatisticManager.cc, 202
- StatisticManager.h, 202
- storedBytesInBuffer
 - FrameBuffer, 90
- storedBytesVector
 - FrameBuffer, 90
- storedMeasurements
 - HistogramStatistics2DMap, 95
 - HistogramStatistics3DMap, 100
 - VectorStatistics2DMap, 157
 - VectorStatistics3DMap, 161
- storeEthernetFrameInBuffer
 - BuffersModule, 62
- STREAM_ID
 - ForwardingLogic.h, 184
- streamEthernetPriority
 - Reader, 123
 - Writer, 168
- streamID
 - ForwardingLogic, 84
- submodulePrefix
 - HistogramStatistics2DMap, 95
 - HistogramStatistics3DMap, 100
 - VectorStatistics2DMap, 157
 - VectorStatistics3DMap, 162
- subscriberReceiveTime
 - HistorySampleEntry, 104
- SWITCH_MAC
 - ArbiterAVB.h, 170
 - ForwardingLogic.h, 184
- switchIndex
 - ForwardingLogic, 84
- switchMAC
 - ArbiterIEEE802_1Qbv, 46
 - ForwardingLogic, 84
- tasPortTimeSegments
 - TimeAwareShaperSegmentStatistic, 148
- tasTimeSegments
 - TimeAwareShaperSegmentStatistic.h, 203
- ThrowError
 - ValidationHandler, 153
- TimeAwareShaperSegmentStatistic, 145
 - ~TimeAwareShaperSegmentStatistic, 146
 - criticalGuardbandSegmentStarts, 146
 - criticalSendingSegmentStarts, 147
 - finish, 147
 - noncriticalGuardbandSegmentStarts, 147
 - noncriticalSendingSegmentStarts, 147
 - tasPortTimeSegments, 148
 - TimeAwareShaperSegmentStatistic, 146
- TimeAwareShaperSegmentStatistic.cc, 202
- TimeAwareShaperSegmentStatistic.h, 203
 - CRITICAL_GUARDBAND_SEGMENT, 203
 - CRITICAL_SENDING_SEGMENT, 203
 - NONCRITICAL_GUARDBAND_SEGMENT, 203
 - NONCRITICAL_SENDING_SEGMENT, 203
 - tasTimeSegments, 203
- timeOffset
 - ArbiterIEEE802_1Qbv, 46
- timerIsScheduled
 - arbiterAVBClass, 22
- timeSegments
 - ArbiterIEEE802_1Qbv.h, 172
- trackMissingDataMap
 - Sink, 141
- TrafficSource, 148
 - ~TrafficSource, 149
 - generatorEnable, 150
 - globalStreamId, 151
 - handleMessage, 149
 - initialize, 150
 - interBurstMsg, 151
 - intraBurstMsg, 151
 - pendingBursts, 151
 - pendingIntraBurstSize, 151
 - scheduleNextBurst, 150
 - scheduleNextIntraBurst, 150
 - sendPacket, 150
 - TrafficSource, 149
- TrafficSource.cc, 203
- TrafficSource.h, 203
 - _burst_src_verbose_, 204
 - Define_Module, 204
- transportPriority
 - Reader, 123
 - Writer, 168
- triggerSelf
 - ArbiterAVB, 21
 - ArbiterIEEE802_1Q, 29
 - ArbiterIEEE802_1Qbv, 46
 - ArbiterIEEE802_3, 53
 - IngressPort, 108
- triggerTimer
 - ArbiterIEEE802_1Qbv, 47
- triggerUpdateMessageNewPriorityArrival
 - BuffersModule, 62

- truncate_min_max
 - GlobalMacros.h, 189
- tsnTASStatistics
 - StatisticManager, 145
- UdplpStack, 152
 - ~UdplpStack, 152
 - handleMessage, 152
 - initialize, 153
 - UdplpStack, 152
- UdplpStack.cc, 204
- UdplpStack.h, 204
 - Define_Module, 205
- uniqueFrameld
 - EthMsgCreator, 76
- unknown
 - RtpsMacros.h, 197
- UNKNOWN_STREAM_ID
 - ForwardingLogic.h, 184
- updateCache
 - Reader, 120
- updateClassInformationsStatus
 - ArbiterAVB, 18
- updateFreeBufferSize
 - FrameBuffer, 88
- validateInputBounds
 - ValidationHandler, 153
- ValidationHandler, 153
 - ThrowError, 153
 - validateInputBounds, 153
- ValidationHandler.h, 205
- valuePrefix
 - HistogramStatistics2DMap, 95
 - HistogramStatistics3DMap, 100
 - VectorStatistics2DMap, 157
 - VectorStatistics3DMap, 162
- vectorE2EStatistics
 - E2ELatencyStatistics, 70
- vectorLostFramesPerStream
 - FramesLostStatistics, 92
- vectorLostFramesTracePerStream
 - FramesLostStatistics, 92
- VectorStatistics2DMap, 154
 - ~VectorStatistics2DMap, 155
 - accumulateInteger, 155
 - addInteger, 155
 - addLatency, 156
 - finish, 156
 - getEntry, 156
 - modulePrefix, 157
 - prefix, 157
 - storedMeasurements, 157
 - submodulePrefix, 157
 - valuePrefix, 157
 - VectorStatistics2DMap, 154
- VectorStatistics2DMap.cc, 205
- VectorStatistics2DMap.h, 205
- VectorStatistics3DMap, 157
 - ~VectorStatistics3DMap, 159
 - accumulateInteger, 159
 - addInteger, 159
 - addLatency, 160
 - empty, 161
 - finish, 160
 - getEntry, 161
 - modulePrefix, 161
 - prefix, 161
 - storedMeasurements, 161
 - submodulePrefix, 162
 - valuePrefix, 162
 - VectorStatistics3DMap, 158
- VectorStatistics3DMap.cc, 206
- VectorStatistics3DMap.h, 206
- Writer, 162
 - ~Writer, 164
 - addNextFragmentToSendList, 164
 - bestEffort, 165
 - checkLifespanOfSamples, 164
 - createRtpsMsgFromFragment, 164
 - currentSampleSequenceNumber, 165
 - destinationEntityId, 166
 - destinationMac, 166
 - entityId, 166
 - finish, 164
 - fragmentSize, 166
 - globalStreamId, 166
 - handleMessage, 164
 - historyCache, 166
 - historySizeQoS, 167
 - initialize, 165
 - lifespan, 167
 - registerEvent, 167
 - registerWriter, 165
 - send_queue, 167
 - sendEvent, 167
 - sendMessage, 165
 - shapingDuration, 167
 - sourceMac, 168
 - streamEthernetPriority, 168
 - transportPriority, 168
 - Writer, 163
- Writer.cc, 206
- Writer.h, 206
 - Define_Module, 207
- writerArrivalTime
 - HistorySampleEntry, 104
- writerEntityId
 - HistorySampleEntry, 104
- writerFirstInjectionTime
 - HistorySampleEntry, 104
- writerIds
 - Participant, 115
- writerNoKey
 - RtpsMacros.h, 197
- writerParticipantEntityId
 - HistorySampleEntry, 105

writerWithKey
RtpsMacros.h, [197](#)