ScOSA Simulator Installation and User Guide

Alan Aguilar Jaramillo

Deutsches Zentrum für Luft- und Raumfahrt

2021

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 Wormhole Routing and Routers in Spacewire Networks 1

1.2 Task Modeling 3

1.3 Computational Node Modeling 4

CHAPTER 2: INSTALLATION GUIDE 5

CHAPTER 3: USER GUIDE 6

3.1 Running a Simulation 6

3.2 Simulation Parameters 6

3.3 Simulation Outputs 8

LIST OF TABLES

Table 1. List of Required Parameters and their Default Values 7

Table 2. Parameter Setting Format and Example 7

Table 3. List of Output Files and their Contents 9

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The ScOSA project seeks to create distributed computational systems by distributing computational tasks across a network of computers. This assignment is determined by the capabilities of the computer in question and the traffic that a particular tasks may induce on the network. These metrics are then taken into account within the VirSat ScOSA tool to generate assignments using approximated metrics for network traffic and computer utilization,

In order to validate these results, a simulator was developed such that it could realistically recreate the behavior of the distributed computational system. This simulator not only aimed to mimic the task activation patterns laid out in ScOSA, but also look to accurately simulate messages being sent across a network connected by SpaceWire.

The following sections will introduce all of the various portions of a ScOSA netowrk and how they were simulated in the ScOSA simulator.

1.1 Wormhole Routing and Routers in Spacewire Networks

Due to the limited availability of memory in space systems, SpaceWire opts to forward messages using Wormhole routing instead of traditional store-and-forward routing alternatives.

This works by assigning a pair of ports to one and other and automatically forwarding bits from one port to its assigner partner. This creates a continuous forwarding of messages across the network without relying on large buffers.

When two two input ports are competing for an output port, they are assigned in a first-come-first-served basis. The port that requests the latest has to wait for the previous message or messages to be transmitted across the desired port before it can be assigned to it.

As it is waiting, all incoming bits into that port get stored into a buffer. Each port contains a buffer size of 1 kbyte. In the event that the buffer is filled, Spacewire

routers will order the preceding node in the network to stop sending bits. This can be repeated until it reaches the sender node, which can then stop the transmission until the last port in the chain is assigned to an output port and can continue transmitting information. This phenomenon is known as **blocking**, and it is an important source of network delays and traffic within Spacewire networks.

In the simulator, the message protocol is followed as to mimic the behavior of Spacewire networks. Here, every router contains a list of ports with their respective buffers assigned to said routers.

As messages are received, the first bits contain information of which port the message needs to be forwarded through. This port information creates a `PortRequest` self-message within the router. The router will then process this request to see if the desired port is assigned or if it is available. If available, the output port will be assigned to port in which the message was received and a `ChannelRequest` will be sent across the output port towards the next node in the network. This request will ask the next node if their buffer is available to receive messages. If so, the next node in the network will return a confirmation and this will allow for the initial router to start to forward all messages contained and received in the buffer from the assigned ports.

If, however, the buffer from the next node in the path is full, then that node will not return a `ChannelRequest` until said buffer starts to be emptied out. The original router will have to wait for this confirmation and will not forward data until this is received.

If a message remains static for longer than 1 second in the buffer of a port, the parent router will discard all information in that buffer and will discard all incoming bits from that port until a new header file is received. This is as to mimic the same Watchdog Timers implemented in Spacewire, which aim to prevent the network from blocking its way into grid-lock and allow for messages to be deleted in the network if they take too long to be transmitted.

Once a router forwards the last bits of a message (which are labeled as the End-Of-Packet or EOP), the router will look for the next `PortRequest` in its queue and will repeat the process. If no requests exist, then the router will enter an idle state, waiting for new messages to be recieved.

1.2 Task Modeling

Tasks in ScOSA are representations of computational tasks to be carried out by computers (also referred to here as computational nodes). They fall under two types: periodical and dependent tasks.

Periodical tasks are executed after a fixed period of time. These tasks might generate new information that is needed to execute other tasks. These are called dependent tasks as they require the information from one or more tasks to be executed. These tasks can also generate new information that has to be sent to other dependent tasks.

If two tasks are assigned to different computational nodes and need to send information to each other, this information has to travel through the network via the Wormhole routing seen in Spacewire. If not, then the information is automatically accessed by the dependent tasks in that computational node.

In the simulator, tasks are represented as nodes in the network. All computational nodes are given one node per task available in the simulation but only activate those who are assigned to them according to the current system configuration. If a periodical task is activated, then this will attempt to execute at the beginning of every period.

Once a task is ready to be executed, be it by starting its period or because it has all of the information from its dependencies, it will notify its parent computational node informing it of its status. If the node is not busy transmitting or performing another task, it will send a command telling the task to execute. This execute command will then trigger a wait function that lasts the duration of the worst-case-execution time of that task as indicated by the user. Once this time passes,

the task will notify its parent task of its completion and ask to send information to other tasks. The parent computational node will then send that information across the network if needed.

1.3 Computational Node Modeling

In the simulator, computational units are represented as two distinct nodes in a network. The first is the Router Node (previously described in Section 1.1) in charge of handling communications with other nodes in the network. The second is a Computational Node, in charge of activating tasks and generating information to be shared with other tasks.

As tasks are completed in their assigned nodes, these notify their parent Computational Node of their completion and of the list of tasks that need to be informed of said completion. Each computational node contains a list of which tasks are assigned to which nodes in the network. The computational nodes use their lists to know if they need to route a message to another node in the network or if they themselves are assigned that dependent tasks and can skip forwarding the information.

If they are assigned to a dependent task that needs to be notified, they will immediately forward that information to the task and will not lose any time transferring that data. If, however, the target dependent task is assigned to another node, then the parent computational node of the completed task will generate a message of a size indicated by the task that was just completed, and forward it to its router node to start forwarding across the network. Only once the transmission of the messages is completed will the computational node check if any of its tasks are ready to be triggered and will trigger them in the order in which they became available to be triggered.

CHAPTER 2

INSTALLATION GUIDE

The following procedure has only been tested for Windows 10. However, it should apply to any system that can support OMNET++.

1. Download and Install OMNET++ in your system (see OMNET++ Installation Guide https://doc.omnetpp.org/omnetpp/InstallGuide.pdf).

2. Open OMNET++ and set the Workspace directory to the desired folder.

3. Download ScOSA Simulator from Git repository (see https://gitlab.dlr.de/virsat/research/scosa-simulator.git)

4. Place the ScOSA simulation files in the directory chosen as ONMET++'s Workspace directory.

5. Validate the install by opening `scosa-simulator/simulations/omnet.ini` file in OMNET++ and pressing the green arrow in the menu bar to run the simulation.

CHAPTER 3

USER GUIDE

3.1 Running a Simulation

The simulator is designed to read JSON files outputted by VirSat ScOSA and recreate the network as described in said files. VirSat ScOSA allows for the user to output all possible configurations to a user-defined directory. To recreate these networks from said files, the user must follow the next procedure:

- Export network configuration JSON files from VirSat ScOSA.
- Insure that the files follow the naming scheme `config<CONFIGURATION NUMBER>.json`.
- Place folder containing the exported JSON files into the `scosa-simulator/input` directory.
- Set simulation parameters to match scenario (see Section 3.2).
- Run network simulation in OMNET++.

3.2 Simulation Parameters

OMNET++ allows for the user to set parameters for the simulation being performed as well as setting parameters for the nodes within the simulated network. The ScCOSA simulator takes advantage of this by using simulation parameters as the main way of interfacing the user with the simulation being performed.

3.2.1 Parameter List and Default Values

To set up a simulation, the user must specify the following list of parameters to the ScOSA simulator:

Parameter Name	Use	Default Value
`networkName`	Indicates the name of the directory containing desired JSON files	"lab_example"
`configNum`	Indicates the initial configuration number	0
`numConfigs`	Counts the total number of configurations to be loaded into the simulation	16
`simTimeOut`	Time to be simulated (in seconds)	10

Table 1. List of Required Parameters and their Default Values

3.2.2 Editing Parameters

All simulation parameters have to be edited in the `omnetpp.ini` file contained within the `simulations` directory in the Simulator files. To edit a particular parameter, the user must follow the following format:

`ScOSA.<PARAMETER NAME> = <PARAMETER VALUE>`
I.e. `ScOSA.networkName = "aton_example"`

Table 2. Parameter Setting Format and Example

If a parameter is not changed in the `omnetpp.ini` file, then the parameter will assume its default value.

NOTE: the user must not delete the line in the `omnetpp.ini` file that establishes the name of the network as "ScOSA", as this is what triggers the network initializer functions.

3.2.3 Choosing a Network

As previously mentioned, all parameters listed in Section 3.2.1 must be edited when choosing a new network to be simulated. Not only must the name of the network must be the same as the directory containing the JSON files with the desired network

descriptions, but the `numConfigs` value must be set to the total number of JSON files contained within the aforementioned folder. The `configNum` can be set to any number as long as it is smaller than the value set to `numConfigs`. Any discrepancy in this information may lead to an error when initializing the simulation.

The `simTimeOut` value can be set to any desired value and is unrestricted. It is recommended to be longer than the longest period of all periodic tasks being simulated in the network, as to capture the behavior of the network accurately.

3.3 Simulation Outputs

Once a simulation times out according to the `simTimeOut` parameter, the simulator will then compile the performance metrics of the simulation and output them onto four csv files. These files are saved in the `scosa-simulator/output` directory in a folder of the same name as the network being simulated.

3.3.1 Output File List

The list and description of the files generated can be seen on Table 3.

File Name	Content Description	Values
`delays`	Records time between a header file and end of packet reception.	SenderNodeId SenderName DestinatioNodeId DestinationName PathLength Configuration MessageSize SentTime ReceptionTime TransmissionDuration TaskId TaskName
`port_assignment`	Records duration of port assignments within a router node.	SenderNodeId SenderNodeName DestinatioNodeId DestinationNodeName StarTime EndTime Duration
`task_trigger`	Records time of task activation and duration.	NodeId NodeName TaskId TaskName StartTime EndTime
`transmission`	Records time in which a particular link was being used to transmit data.	SenderNodeId SenderNodeName DestinatioNodeId DestinationNodeName StarTime EndTime Duration TaskId TaskName

Table 3. List of Output Files and their Contents