

# Varanus

## Description and tutorial



### Authors:

Ricardo Fonseca {rfonseca@lasige.di.fc.ul.pt}

Eric Vial {evial@lasige.di.fc.ul.pt}

Nuno Neves {nfneves@fc.ul.pt}

Fernando Ramos {fvramos@ciencias.ulisboa.pt}

# Contents

1. Introduction .....	3
2. Setup and interaction with a prepared virtual network .....	5
2.1 Application dependencies .....	5
2.2 Building the applications .....	5
2.3 Running the applications .....	5
2.4 Interacting with a virtual network (introduction) .....	5
2.5 Interacting with a virtual network (normal operation) .....	6
2.6 Changing the behaviour of a virtual network (attack scenarios) .....	9
3. Setup of arbitrary virtual networks .....	14
3.1 Configuring the topology .....	14
3.2 Running the application .....	16
4. Manual setup of the SDN controller .....	18
4.1 Application dependencies .....	18
4.2 Building the application .....	18
4.3 Configuring the application .....	18
4.4 Running the application .....	18
5. Manual setup of the collectors .....	19
5.1 Application dependencies .....	19
5.2 Building the application .....	19
5.3 Configuring the application .....	19
5.4 Running the application .....	19
References .....	20

# 1. Introduction

Smart Grid Distribution System Operator (DSO) substations are typically distributed over a large geographical area and depend on a WAN to communicate amongst themselves and with the control centre (the WAN may consist of fibre optics, cellular or radio technologies). The exact distribution is left to the DSO to administrate and for our purposes here we simply assume that substations and the control centre are reachable via the Internet and each substation knows the IP addresses of neighbouring substations and of the control centre (these addresses can be obtained via DNS requests or configured statically).

Our solution applies the Software Defined Network (SDN) paradigm [1, 2] to manage the core (or WAN) network. We abstract the higher level of smart grid communication infrastructure as a set of SDN switches, each one connected as an edge of a substation, and the SDN controller residing in the control centre.

In the control centre, the SDN controller runs in a dedicated machine with Internet access and two distinct IP addresses that are known by all substations connected to the centre. Let us distinguish these two addresses as  $IP_A$  and  $IP_B$ .

In each substation there are one (or more) SDN switches which can be software-based or hardware-based. The main requisites for these switches are the capability of communicating with an SDN controller using the OpenFlow protocol (v1.4 or higher) [3] and the capability of setting up IP tunnels in the data plane. Each switch is configured to communicate with the SDN controller running in the control centre using address  $IP_A$  of the centre. The connections among the SDN switches themselves are configured as necessary using IP tunnels to link switches that are located in different substations.

In each substation there is also an auxiliary collector application running in a dedicated machine which is connected to every SDN switch operating within the substation via a common LAN. The collector is configured to communicate with the SDN controller running in the control centre using address  $IP_B$  of the centre.

We designed and implemented the SDN controller and collectors, as previously described, as well as two auxiliary applications for setting up and interacting with a virtual SDN in order to easily demonstrate the operation of a real network deployed by a DSO. All of these applications are available for download at [4] under the common name *varanus*. Figure 1 shows an overview of the interactive application in action.

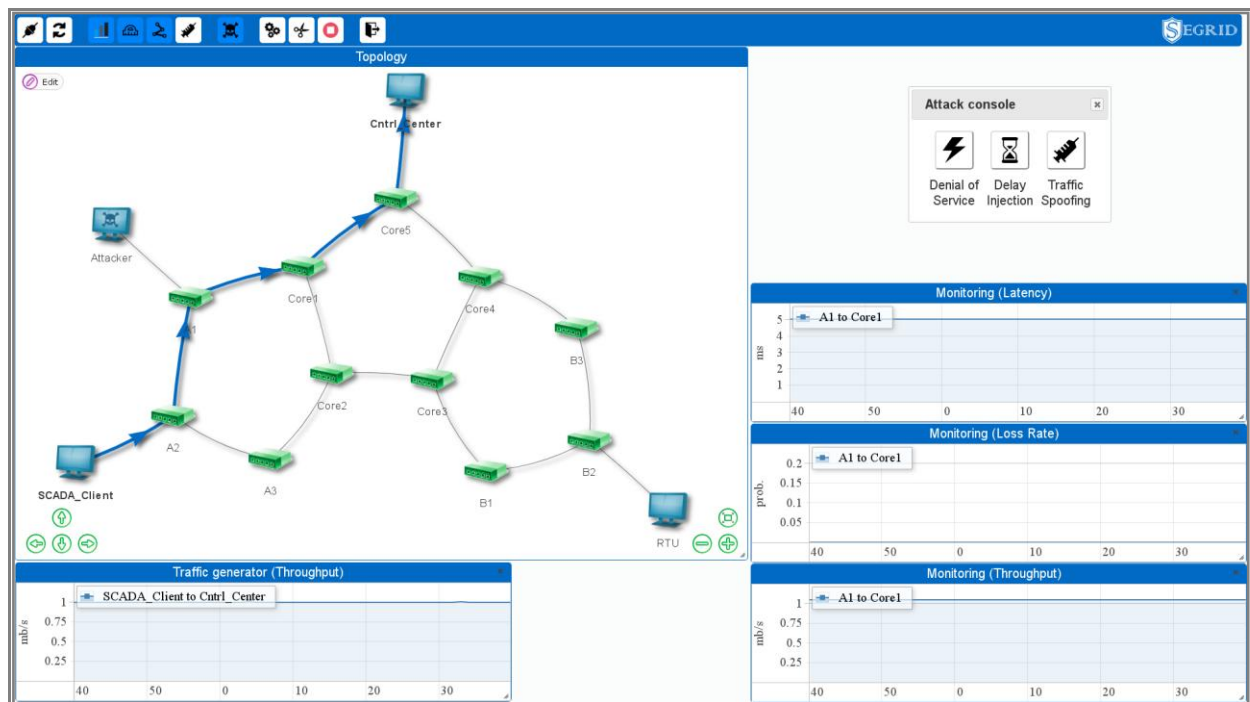
The SDN controller and collectors need to be configured appropriately by taking into account 1) the setup of the individual SDN switches in the substation edges and 2) the connection requirements of applications running inside the substations that use the SDN for communicating with other substations. Our first auxiliary application, the network manager, automatically configures the SDN controller and collectors according to the setup of the virtual network topology.

During execution, the SDN controller runs a TCP socket oriented service to provide up to date information about the network such as its topology, monitoring statistics about individual links and configured routes between network edges. Our second auxiliary application, the network visualiser, communicates with the SDN controller to obtain this information and displays it in a graphical form when requested. The topology displayed by the network visualiser includes only information about the data plane, i.e., the interconnections between the SDN switches and their connections to application hosts. It excludes the visualisation of the SDN controller and the collectors, and their connections to the SDN switches.

The network visualiser also communicates with the network manager to configure, as requested, Quality of Service (QoS) properties of the links connecting SDN switches, and to perform other actions such as generating traffic between hosts or directly injecting artificial traffic in the links.

In this document we first describe how to quickly setup a prepared virtual network using the network manager and interact with it using the network visualiser. Then we show how to set up arbitrary network topologies using the network manager. Finally we explain how to manually set up and run the SDN controller and collectors in the absence of the auxiliary network applications.

The tutorials in the following sections apply only to machines running a 64-bit Linux-based operating system. Any displayed command strings are intended to be typed in a terminal emulation application.



**Figure 1:** Overview of the interactive application showing e.g., an example network topology and monitoring statistics obtained from the SDN controller

## 2. Setup and interaction with a prepared virtual network

In this section we show how to set up and interact with a prepared virtual network. We explain how to build the necessary applications and how to quickly start the network manager without having to define or configure the SDN controller, collectors, SDN switches or hosts. We also explain how to start the network visualiser and how to interact with it during its execution.

### 2.1 Application dependencies

In order to build the SDN controller and collectors, *Java™ 8* and the *ant* tool [3] are required. In order to run the SDN controller and collectors, *Java™ 8* and the *libpcap* library [7] are required. In order to run the network manager, *Open vSwitch* (v2.5 or higher) [8] and *Mininet* (v2.2.2 or higher) are required. In order to run the network visualiser, the *Apache Tomcat* server (v8.5 or higher) and a web browser with enabled *JavaScript* are required.

### 2.2 Building the applications

To build all necessary applications, execute the following command from the repository top-level directory:

```
ant pack-all
```

### 2.3 Running the applications

To run the network manager with the prepared virtual network, execute the following command as root from the repository top-level directory:

```
demo/launch_network.sh
```


The network manager is launched with the prepared virtual network, automatically configuring and starting the SDN controller and collectors. To exit the network manager, type and enter "exit" in the terminal.

To run the network visualiser, execute in a separate terminal the following command from the repository top-level directory:

```
demo/launch_visualiser.sh
```

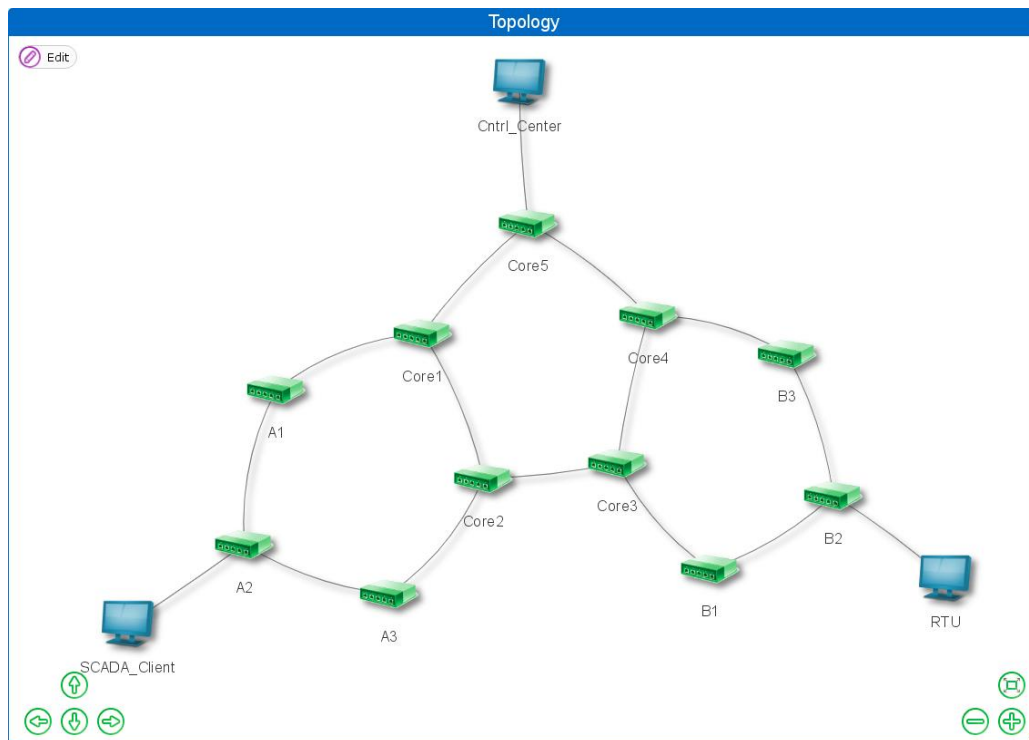
The network visualiser is launched and can be interacted with by accessing the address *http://localhost:8080/visualiser* in a web browser.

### 2.4 Interacting with a virtual network (introduction)

To start the interaction with the network visualiser in the web browser, click on the  button in the top bar in order to connect the visualiser to the SDN controller and network manager. The topology of the prepared network should appear in the "Topology" window, as depicted in Figure 2. The chosen topology was inspired by a real network topology of one of the DSOs that participated in the project.

The displayed switches operate as if they were located in the edges of distinct substations while each displayed host operates as if it were located inside the substation of its connected switch, i.e., the SCADA client supposedly runs in the substation containing switch A2 and the RTU supposedly runs in the substation containing switch B2. Note that switch Core5 operates instead as if it were located in the edge of the control centre. Also, the host "Cntrl\_Center" does not represent the SDN controller but instead a "catch-all" for


miscellaneous applications that communicate with the substations through the data plane, such as a SCADA server.



**Figure 2:** The network topology is displayed in the network visualiser

## 2.5 Interacting with a virtual network (normal operation)

In this subsection we demonstrate some features of the network visualiser under a scenario of normal operation of the network.

To display a route between any two hosts, first click on the icon of each host while holding the Ctrl key and then click on the  button in the top bar. A pop-up window should appear, as shown in Figure 3, where one can select the direction of the route. In our example we choose to display a route from the SCADA client to the control centre. The route is then displayed over the topology, as shown in Figure 4, and is automatically updated if the SDN controller reconfigures a different route.

**Route monitor**

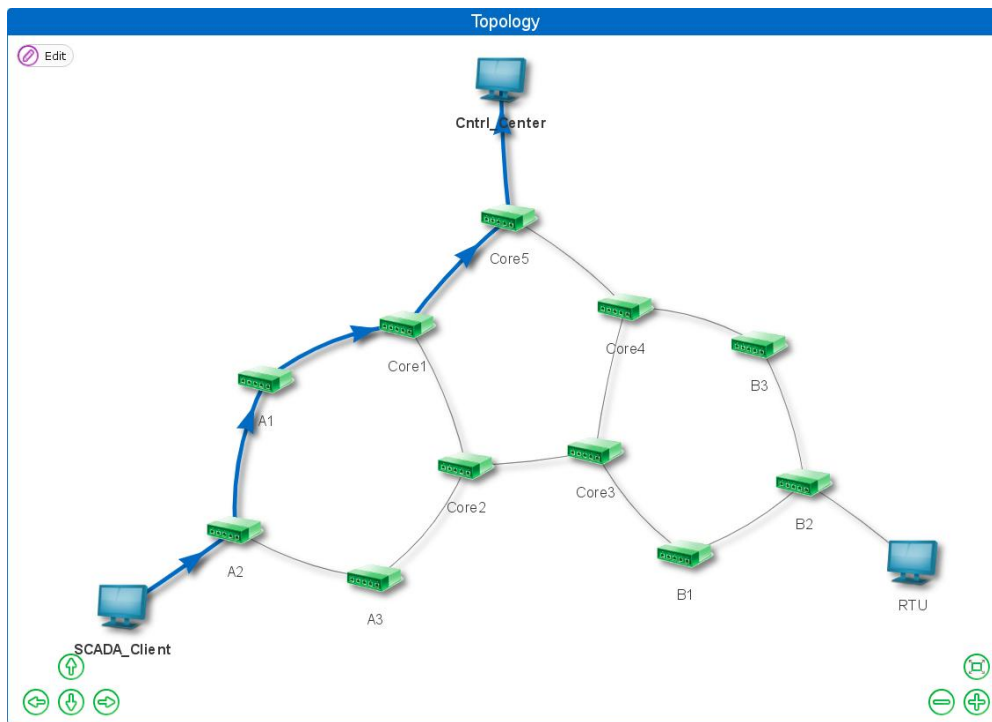
Configure route request

Route direction: SCADA\_Client to Cntrl\_


Matching rule:

Start Cancel

**Figure 3:** The route from the SCADA client to the control centre is selected for visualisation

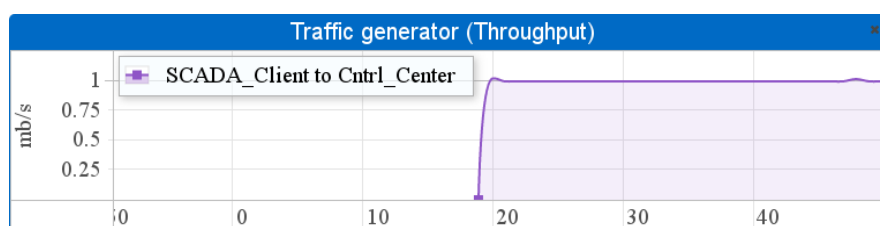


**Figure 4:** The route from the SCADA client to the control centre is displayed over the topology


To generate some traffic between any two hosts, first click on the icon of each host while holding the Ctrl key and then click on the  button in the top bar. A pop-up window should appear, as show in Figure 5, where one can select the direction of the traffic and its bandwidth. In our example we choose to generate 1 Mb/s of traffic from the SCADA client to the control centre. During the generation of traffic its reception rate is displayed over time in a graph as show in Figure 6.

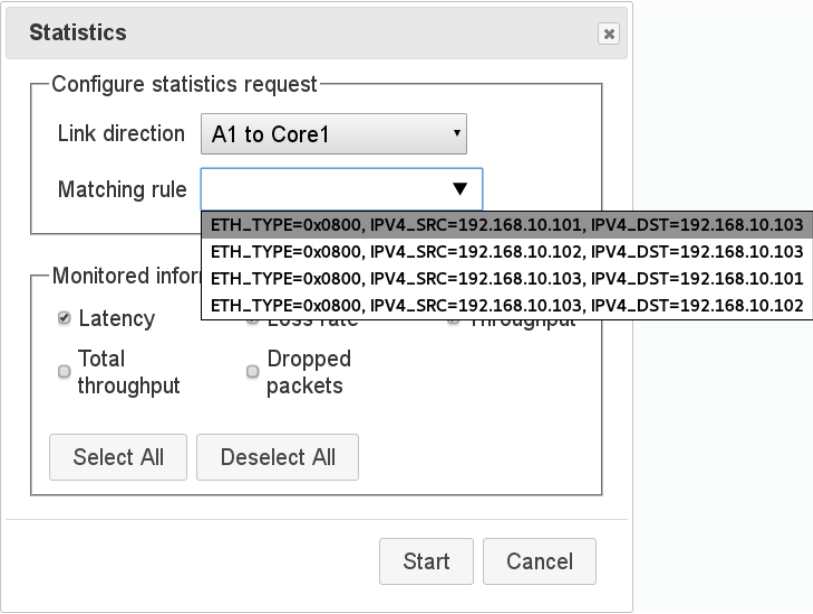
The 'Command console' window has a title bar with a close button. Below the title bar is a section titled 'Configure command'. Inside this section, there are three fields: 'Link direction' with a dropdown menu showing 'SCADA\_Client to Cntrl\_', 'Loading type' with a dropdown menu showing 'Traffic generator', and 'Bandwidth (Mbs)' with a text input field containing '1'. At the bottom of the window are two buttons: 'Start' and 'Cancel'.

**Figure 5:** Traffic generation from the SCADA client to the control centre is set up at 1 Mb/s



**Figure 6:** Reception rate of traffic from the SCADA client to the control centre is displayed over time

To display monitoring statistics about some particular traffic passing through a link connecting two switches, first click on the chosen link and then click on the  button in the top bar. A pop-up window should appear, as show in Figure 7, where one can select the link direction, the classification of the traffic being monitored, and which statistics to display. In our example we choose to display statistics for the link from switches A1 to Core1 about IP traffic from 192.168.10.101 to 192.168.10.103, which correspond to the IP addresses of the SCADA client and the control centre, respectively (to view the IP address of a particular host simply move the mouse over its icon). Each statistic is displayed over time in a distinct graph, as show in Figure 8.



**Statistics**

Configure statistics request

Link direction: A1 to Core1

Matching rule:   
 ETH\_TYPE=0x0800, IPV4\_SRC=192.168.10.101, IPV4\_DST=192.168.10.103  
 ETH\_TYPE=0x0800, IPV4\_SRC=192.168.10.102, IPV4\_DST=192.168.10.103  
 ETH\_TYPE=0x0800, IPV4\_SRC=192.168.10.103, IPV4\_DST=192.168.10.101  
 ETH\_TYPE=0x0800, IPV4\_SRC=192.168.10.103, IPV4\_DST=192.168.10.102

Monitored info

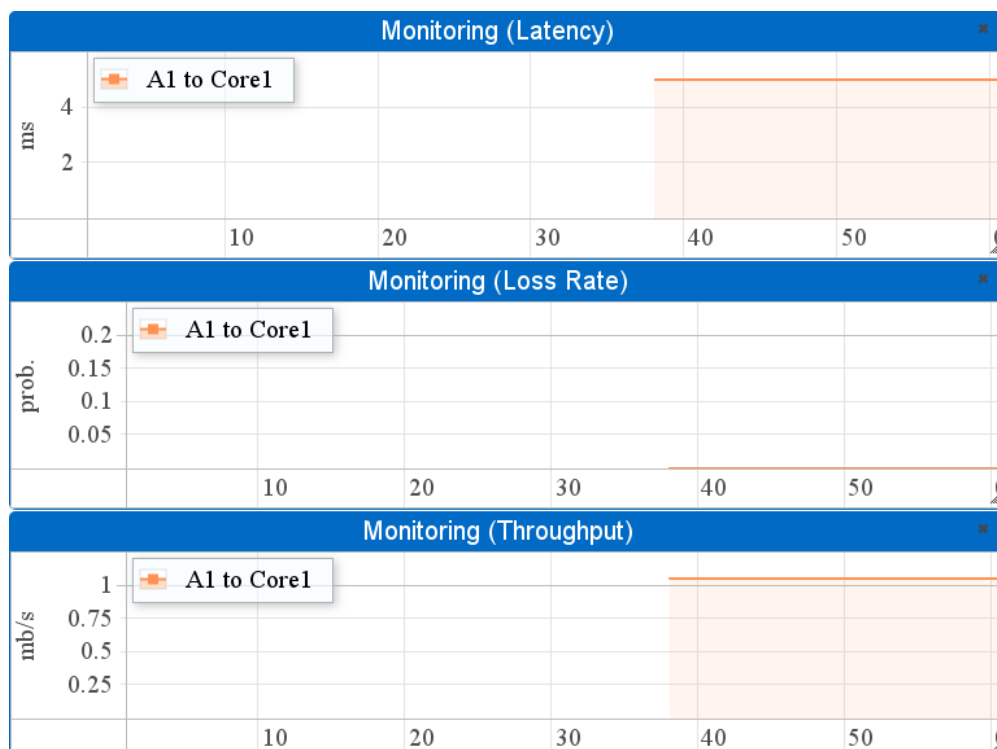
☒ Latency ☐ Loss rate ☐ Throughput

☐ Total throughput ☐ Dropped packets

Select All Deselect All


Start Cancel

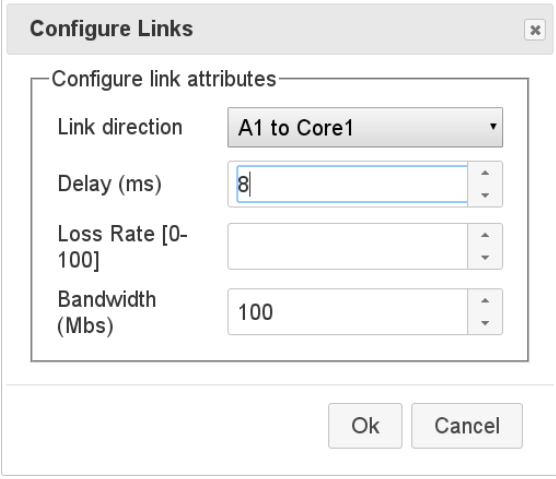
**Figure 7:** Monitoring statistics for the link from switches A1 to Core1, about IP traffic from 192.168.10.101 to 192.168.10.103



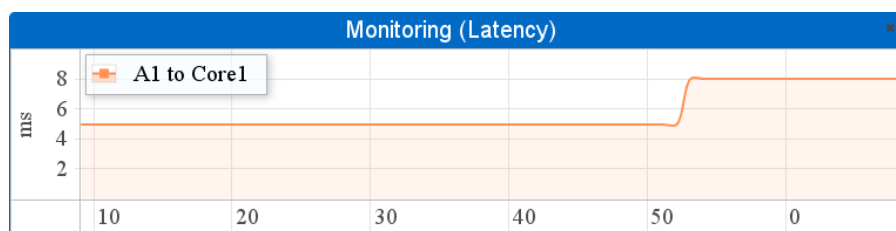
**Figure 8:** Monitoring statistics are displayed over time in individual graphs



To configure QoS properties of specific links, first click on the chosen link and then click on the  button in the top bar. A pop-up window should appear, as shown in Figure 9, where one can select the link direction and adjust its delay, loss rate or bandwidth properties. In our example, we change the delay property from a previous value of 5 ms to 8 ms. After the delay property is reconfigured one should see an increase in the latency graph, as shown in Figure 10. To end the demonstration under normal operation, reconfigure the aforementioned delay property back to 5 ms.




**Figure 9:** The delay property of the link from switches A1 to Core1 is reconfigured from 5 ms to 8 ms



**Figure 10:** An increase in the latency graph for the link from switches A1 to Core1 is observed after the link delay is updated to 8 ms

## 2.6 Changing the behaviour of a virtual network (attack scenarios)

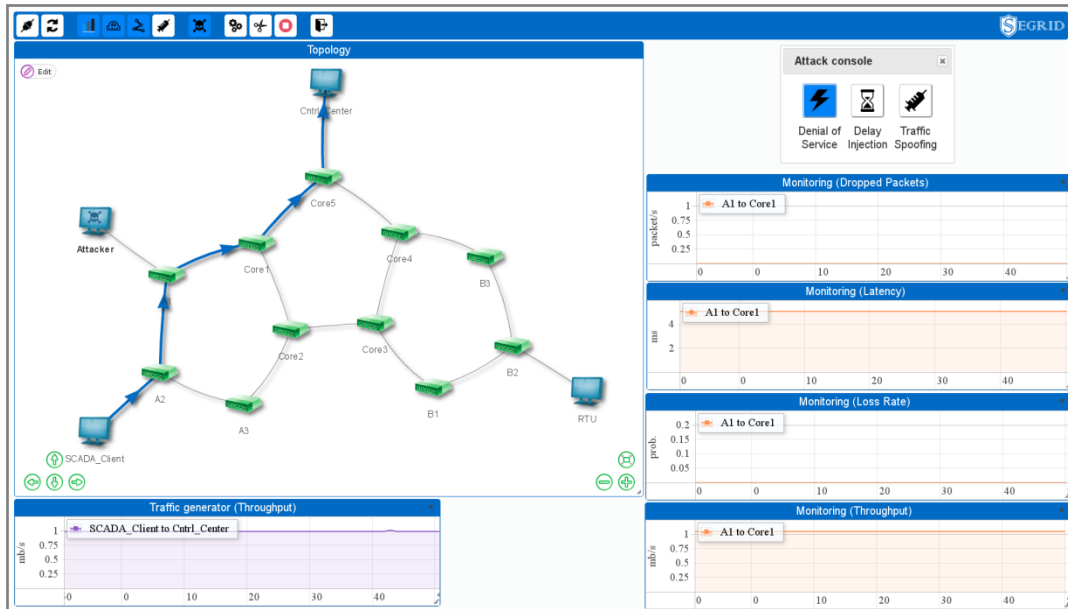
In this subsection we demonstrate three pre-defined attack scenarios that can be easily set up in the network visualiser. To begin, open the attack console by clicking on the  button in the top bar. A pop-up window should appear and a previously configured attacker host joins the network and connects to switch A1 (in reality this host is always connected to the switch but is only displayed by the network visualiser while the attack console is open).

The first attack is a Denial of Service (DoS) attempt against a specific switch, which consists in an attack host generating a large volume of unauthenticated traffic against the adjacent switch with the intent of interfering with its packet forwarding capability and forcing it to drop legitimate traffic due to resource exhaustion.

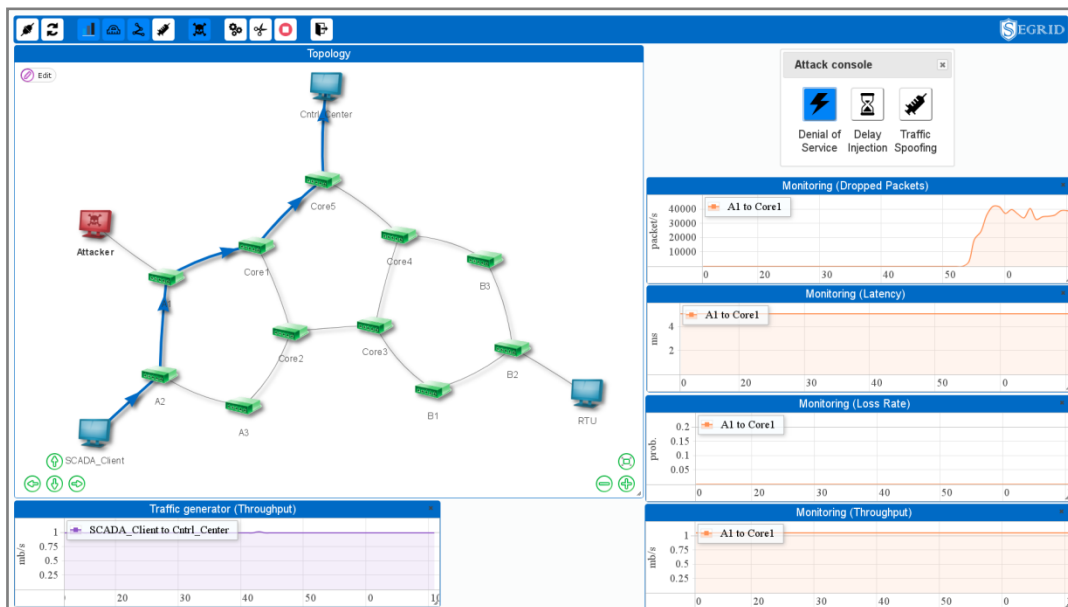
To start the attack, first click on the chosen host to be attacking and then click on the left button in the attack console. In our example, as shown in Figure 11, we choose the attacker host which directs the attack against switch A1, and in order to demonstrate the effects of the attack we also generate traffic from the SCADA client to the control centre (whose reception rate graph is displayed in the bottom left corner of the browser window). We also select for display over time some monitoring statistics about traffic passing

through the link from switches A1 to Core1, such as dropped packets (by switch A1) and link latency, loss rate and throughput (whose graphs are displayed in the right side of the browser window).

While the attack is in progress, one should see an increase in the dropped packets graph while all other graphs remain stable, as show in Figure 12. This happens because switch A1 simply drops all unauthenticated traffic, which does not overload the switch.



**Figure 11:** The DoS attack from the attacker host against switch A1 is selected while we observe the reception rate graph for traffic from the SCADA client to the control centre, and the dropped packets (by switch A1), latency, loss rate and throughput graphs for the link from switches A1 to Core1

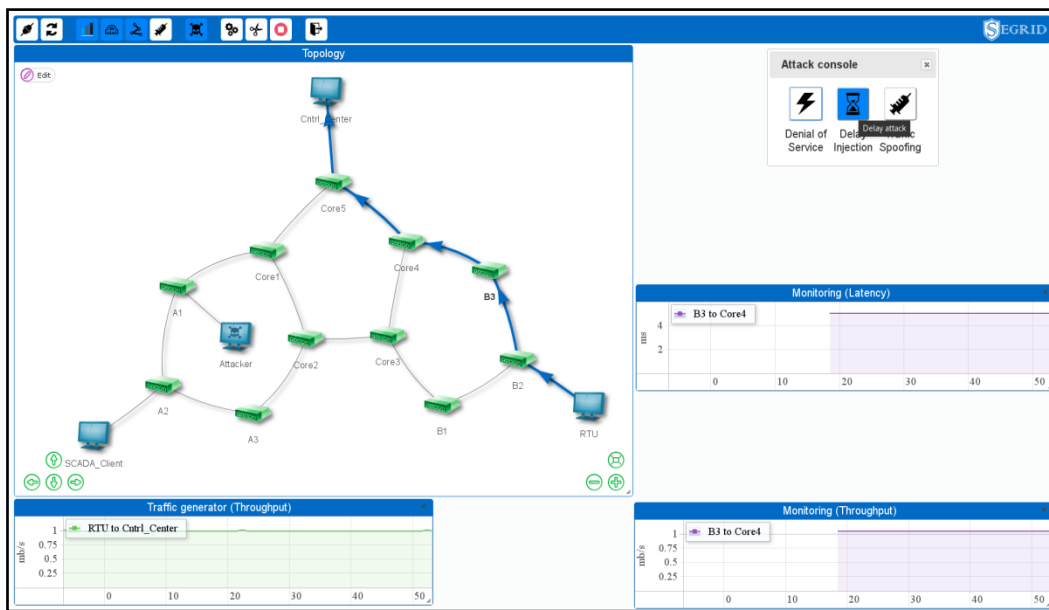


**Figure 12:** An increase in the dropped packets graph is observed during the DoS attack while the other graphs remain stable

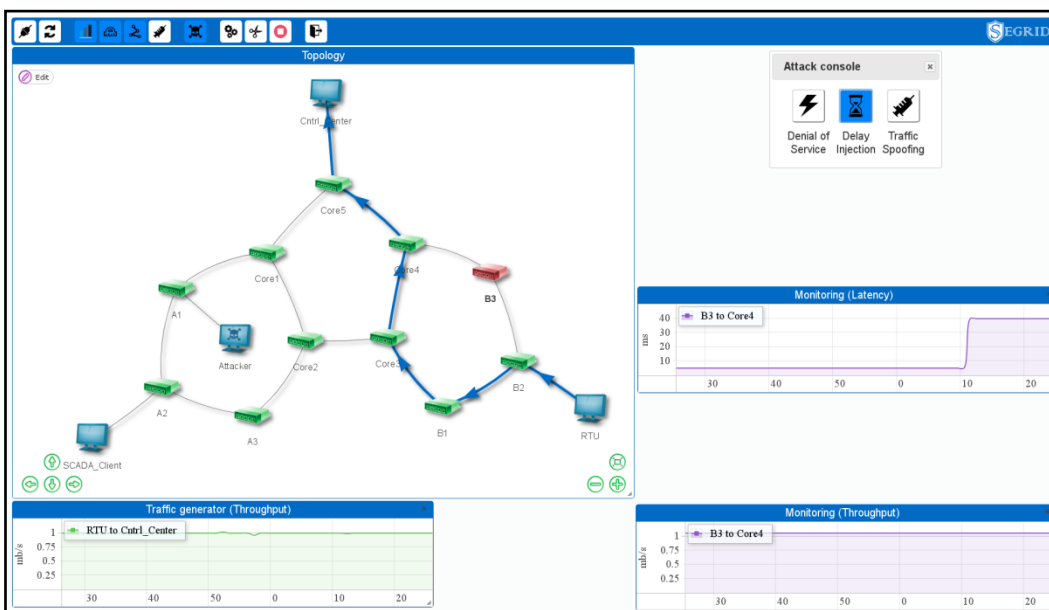
The second attack consists in making a particular switch malicious (supposedly after being compromised), where it will start delaying all exiting traffic by a certain amount of time. The intent of the attack is to

degrade the quality of service of traffic flowing through the switch, eventually violating the properties of some smart grid application.

To start the attack, first click on the chosen switch to be attacked and then click on the middle button in the attack console. In our example, as show in Figure 13, we choose switch B3 to be attacked, and in order to demonstrate the effects of the attack we also generate traffic from the RTU to the control centre (whose reception rate graph is displayed in the bottom left corner of the browser window). In addition, we select for display over time the route from the RTU to the control centre and the latency for the link from switches B3 to Core4 (whose graph is displayed in the right side of the window). While the attack is in progress, one should see an increase in the latency graph and a route update to avoid the compromised switch, as show in Figure 14.



**Figure 13:** The delay injection attack is against switch B3, while we observe the latency graph for the link from switches B3 to Core4

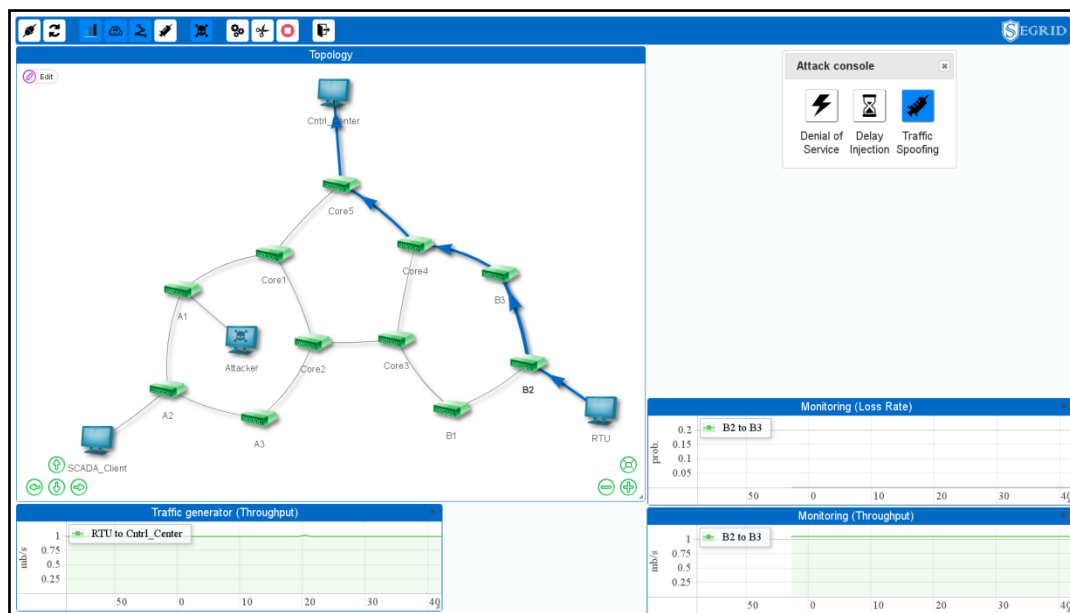


**Figure 14:** An increase in the latency graph for the link from switches B3 to B4 is observed and the route from the RTU to the control centre is updated in order to avoid the compromised switch

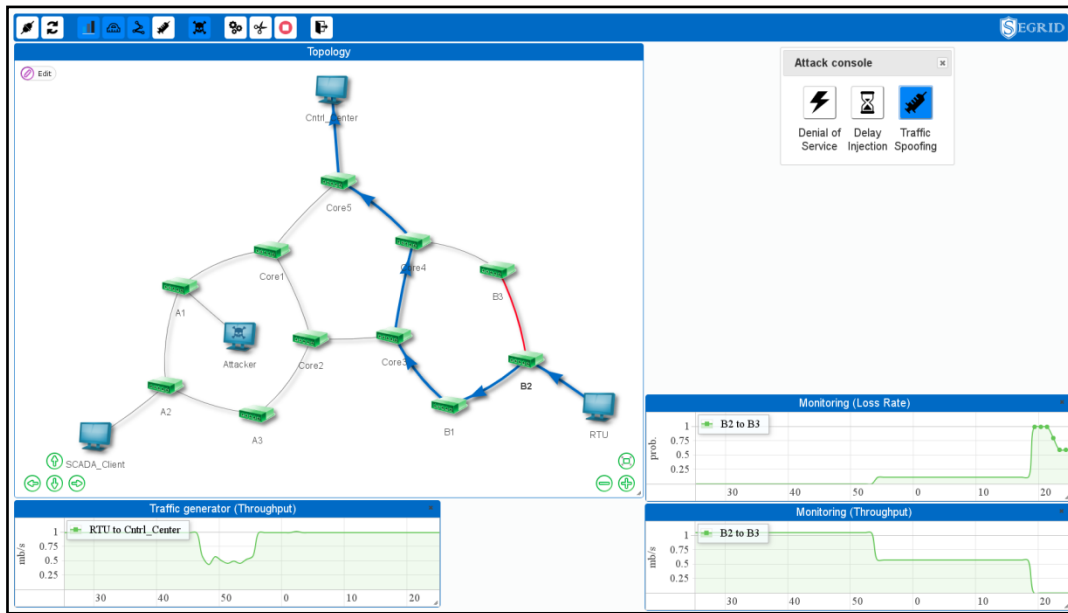
The third attack is a traffic spoofing attack against a specific link connecting two switches. It consists in intercepting the traffic flowing through the link (as if, e.g., a network router was compromised) and changing some bits in the application payloads of the packets, leaving the packet headers untouched with the intent of forcing the receiver application at the destination host to drop the invalid packets (due to modified payloads) but remain invisible to traditional monitoring techniques which only take into account the packet headers.

To start the attack, first click while holding the Ctrl key on both the chosen link to be attacked and its source switch, and then click on the right button in the attack console. In our example, as show in Figure 15, we choose the link from switches B2 to B3 to be attacked, and in order to demonstrate the effects of the attack we also generate traffic from the RTU to the control centre (whose reception rate graph is displayed in the bottom left corner of the browser window). We also select for display over time the route from the RTU to the control centre and the loss rate for the attacked link (whose graph is displayed in the right side of the window).

While the attack is in progress, one should see an increase in the loss rate graph and a route update to avoid the compromised link, as show in Figure 16.



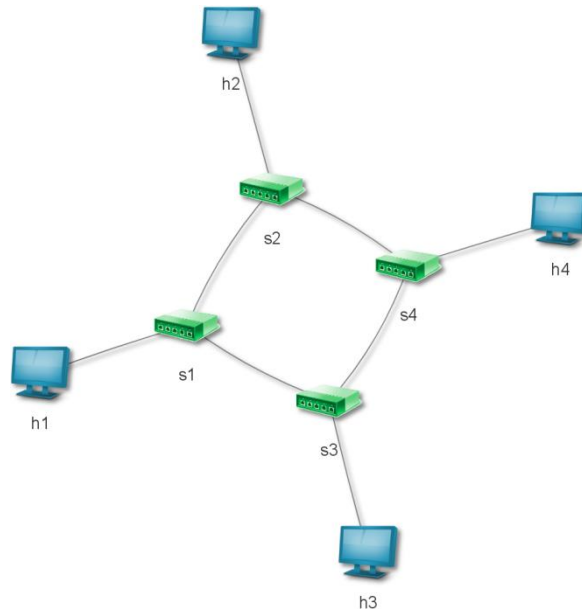
**Figure 15:** The traffic spoofing attack against the link from switches B2 to B3 is selected, while we observe the loss rate graph for the same link



**Figure 16:** An increase in the loss rate graph for the link from switches B2 to B3 is observed and the route from the RTU to the control centre is updated in order to avoid the compromised link

### 3. Setup of arbitrary virtual networks

With the network manager application it is possible to set up a virtual network with an arbitrary topology. We explain how to do so with an example whose topology viewed with the network visualiser is depicted in Figure 17. The dependencies and building instructions are the same as the previous section with a prepared network.



**Figure 17:** Example of a network topology set up with the network manager

#### 3.1 Configuring the topology

From the repository top-level directory create the file *network-manager/config/example.py* and add the following content:

```
from config.util.topo import VaranusTopo
from varanuspy.functions import resetqos, setqos, ssrclinks, switches
from varanuspy.rcli import start_rcli
from varanuspy.utils import resolve

def pre_start_config( mr, extra_args, local_varanus_home ):
    topo = VaranusTopo( mr, extra_args, local_varanus_home=local_varanus_home )

    # Controllers
    topo.add_local_sdncontroller( ip=resolve( 'localhost' ), port=6653 )

    # Collectors
    topo.add_local_collector( 'c1', 1 )
    topo.add_local_collector( 'c2', 2 )
    topo.add_local_collector( 'c3', 3 )
    topo.add_local_collector( 'c4', 4 )

    # Switches
    topo.add_local_ovs_switch( 's1', 1 )
    topo.add_local_ovs_switch( 's2', 2 )
    topo.add_local_ovs_switch( 's3', 3 )
    topo.add_local_ovs_switch( 's4', 4 )

    # Hosts
    topo.add_local_vir_host( 'h1' )
    topo.add_local_vir_host( 'h2' )
```

```

topo.add_local_vir_host( 'h3' )
topo.add_local_vir_host( 'h4' )
topo.add_host_connection( 'h1', 'h3' )
topo.add_host_connection( 'h3', 'h1' )
topo.add_host_connection( 'h2', 'h4' )
topo.add_host_connection( 'h4', 'h2' )

# Links
topo.add_vir_collector_link( 'c1', 0, 's1', 0 )
topo.add_vir_collector_link( 'c2', 0, 's2', 0 )
topo.add_vir_collector_link( 'c3', 0, 's3', 0 )
topo.add_vir_collector_link( 'c4', 0, 's4', 0 )
topo.add_vir_switch_link( 's1', 12, 's2', 12 )
topo.add_vir_switch_link( 's1', 13, 's3', 13 )
topo.add_vir_switch_link( 's2', 24, 's4', 24 )
topo.add_vir_switch_link( 's3', 34, 's4', 34 )
topo.add_vir_host_link( 'h1', 11, '192.168.10.101/24', 's1', 11 )
topo.add_vir_host_link( 'h2', 22, '192.168.10.102/24', 's2', 22 )
topo.add_vir_host_link( 'h3', 33, '192.168.10.103/24', 's3', 33 )
topo.add_vir_host_link( 'h4', 44, '192.168.10.104/24', 's4', 44 )

return topo.conclude()

def post_start_config( mr, extra_args, local_varanus_home ):
    net = mr.net

    # Setup base QoS in all switch links
    band = '100000000' # bit/s
    netem = 'delay 5ms'
    resetqos()
    for sw in switches( net ):
        for L in ssrclinks( mr.net, sw ):
            setqos( mr.net, L.nsrc, L.ndst, band, netem )

    # Start the collectors
    mr.get_host( 'c1' ).netnode.start()
    mr.get_host( 'c2' ).netnode.start()
    mr.get_host( 'c3' ).netnode.start()
    mr.get_host( 'c4' ).netnode.start()

    # Auto-start the remote CLI
    start_rcli( 32770, net )

```

The configuration file is a python file which is loaded by the network manager to set up the virtual SDN. It defines two functions which are called at different moments in the execution of the application. The function *pre\_start\_config* is executed before any component is started. It defines the topology of the network containing the SDN switches and hosts (depicted in Figure 17) and, since the network manager starts the SDN controller and collectors itself, it also defines the controller and the collectors and their respective links to the switches. The function *post\_start\_config* is executed after the SDN controller, switches and hosts are started (due to technical reasons the collectors must be explicitly started here). It can be used to configure Quality of Service (QoS) properties of the links between the switches, such as bandwidth or packet delay.

### 3.2 Running the application

To run the network manager with the configured example network, execute the following command as root from the repository top-level directory:

```
network-manager/run_netmanager.py --config example --arp --autocfg
```

The name of the configuration file (without the *.py* suffix) is passed in the command line. The *--arp* option automatically creates ARP table entries between all configured hosts. The *--autocfg* automatically

configures the SDN controller and collectors according to the configured network topology. To exit the network manager, type and enter "exit" in the terminal.

To view and interact with the example topology, follow the instructions for the network visualiser in the previous section with a prepared network.



## 4. Manual setup of the SDN controller

This section explains how to manually set up and run the SDN controller in the absence of the auxiliary network applications.

### 4.1 Application dependencies

In order to build the SDN controller, *Java™ 8* and the *ant* tool [5] are required. In order to run the SDN controller, *Java™ 8* is required.

### 4.2 Building the application

To build the SDN controller, execute the following command from the repository top-level directory:

```
ant pack-sdncontroller
```

### 4.3 Configuring the application

From the repository top-level directory open the file *sdncontroller/config/2-manualcfg-sdncontroller.properties* and add the following properties:

```
net.varanus.sdncontroller.monitoring.MonitoringModule.collectorhandler_localPort={pn}

net.varanus.sdncontroller.flowdiscovery.FlowDiscoveryModule.staticFlowedConnections=\
[ \
"{SID1} [{p1}]>>{SID2} [{p2}] | v14[eth_type=0x0800, ipv4_src={IP1}, ipv4_dst={IP2}]", \
"{SID2} [{p2}]>>{SID1} [{p1}] | v14[eth_type=0x0800, ipv4_src={IP2}, ipv4_dst={IP1}]" \
]
```

The first property defines the local TCP port number {pn} used by the collectors to connect to the SDN controller.

The second property defines connections between SDN switches with specific traffic classifications. The traffic classification is flexible and allows for fine control of specific parameters in OSI [6] layers 2, 3 and 4. The configured examples define a two-way channel between two SDN switches with IP traffic between two specific hosts. {SID1} and {SID2} are the numerical OpenFlow identifiers for the switches themselves, and {p1} and {p2} are the numerical OpenFlow identifiers for the switch ports. {IP1} and {IP2} are IP addresses of specific hosts located at different substations that use the correspondent SDN switches as gateways for an IP channel between both hosts. For example, {IP1} could represent the IP address of a SCADA server and {IP2} the IP address of an RTU that sends events to the server.

### 4.4 Running the application

To run the SDN controller, execute the following command from the repository top-level directory:

```
sdncontroller/run-sdncontroller.sh
```

## 5. Manual setup of the collectors

This section explains how to manually set up and run each collector in the absence of the auxiliary network applications.

### 5.1 Application dependencies

In order to build a collector, *Java™ 8* and the *ant* tool [3] are required. In order to run a collector, *Java™ 8* and the *libpcap* library [7] are required.

### 5.2 Building the application

To build a collector, execute the following command from the repository top-level directory:

```
ant pack-collector
```

### 5.3 Configuring the application

From the repository top-level directory open the file *collector/config/2-manualcfg-collector.properties* and add the following properties:

```
controllerAddress={IP}:{pn}

switchIfaceMapping=\
{\
  "{C1}" : {\
    "{SID1}" : "{INT11}",\
    "{SID2}" : "{INT12}"\
  },\
  "{C2}" : {\
    "{SID3}" : "{INT21}"\
  }\
}
```

The first property defines the remote IP address {IP} and TCP port number {pn} used to connect to the SDN controller.

The second property defines associations between collectors and sets of SDN switches that are locally connected to each collector machine. The configured examples define an association between a collector and two switches, and an association between another collector and a single switch. {C1} and {C2} are the numerical identifiers for the collectors. {SID1}, {SID2} and {SID3} are the numerical OpenFlow identifiers for the SDN switches. {INT11}, {INT12} and {INT21} are the names of the network interfaces on the collector machines that receive traffic from the connected switches.

### 5.4 Running the application

To run a collector, execute the following command as root from the repository top-level directory:

```
collector/run-collector.sh -id {CID}
```

Each collector must have a unique numerical identifier. {CID} is the identifier of the collector being run.

## References

- [1] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. Rothenberg, S. Azodolmolky, S. Uhlig, "Software-Defined Networking: A comprehensive survey", in Proceedings of the IEEE, vol. 103, no. 1, 2015.
- [2] F. Hu, Q. Hao, K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation", IEEE Communications Surveys & Tutorials, vol. 16, no. 4, 2014.
- [3] OpenFlow specification, available at <https://www.opennetworking.org/software-defined-standards/specifications/>
- [4] Source code for *varanus*, a set of applications providing resilient communications in the Smart Grid, available at <https://git.lasige.di.fc.ul.pt/navigators/varanus>
- [5] Ant build tool, available at <https://ant.apache.org/>
- [6] ISO/IEC 7498-1:1994, available at <https://www.iso.org/standard/20269.html>
- [7] LibPCAP library for network traffic capture, available at <https://www.tcpdump.org/>
- [8] Open vSwitch, a production quality, multilayer open virtual switch, available at <http://openvswitch.org/>
- [9] Mininet network emulator, available at <http://mininet.org/>
- [10] Apache Tomcat server, available at <https://tomcat.apache.org/>