Security for smart Electricity GRIDs

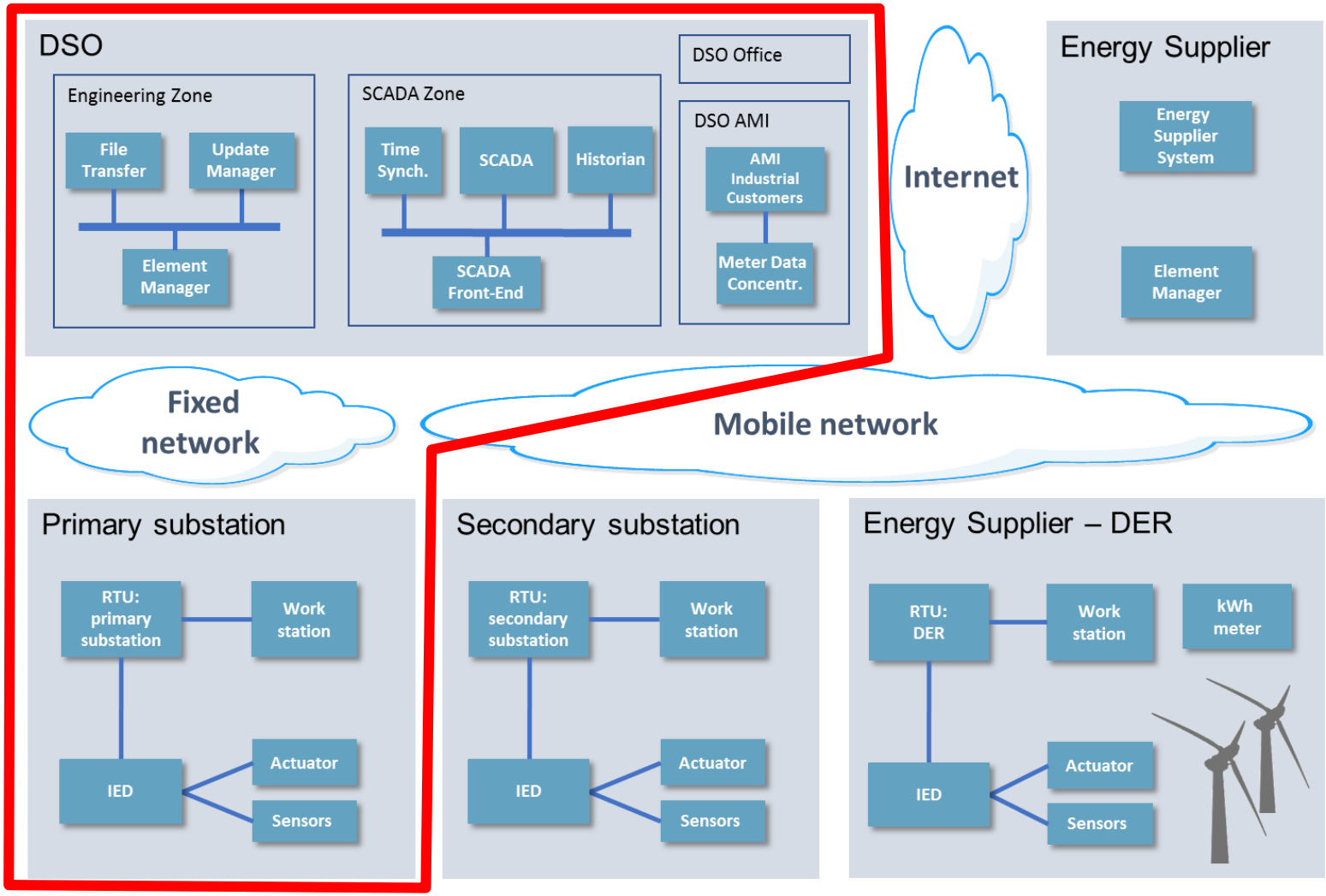
Resilient Communications

Ricardo Fonseca**, Eric Vial, Fernando Ramos, **Nuno Neves
LASIGE, Faculty of Sciences, Univ. of Lisboa
www.navigators.di.fc.ul.pt

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no. 607109

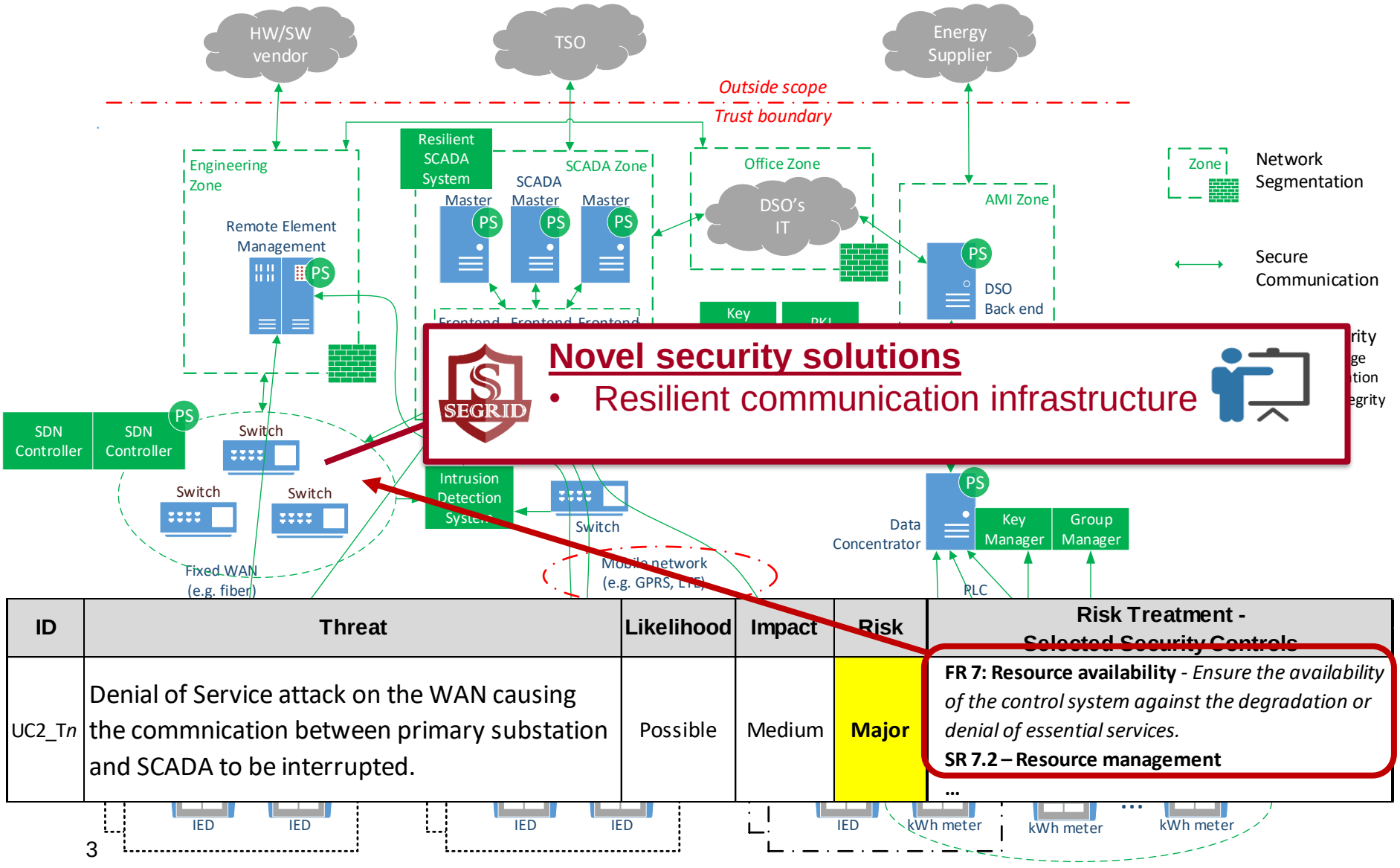
Wide Area Network of the Smart Grid

Focus within the use case 2.2



Security and Privacy Architecture DEsign

Use case 2 - Scenario 2



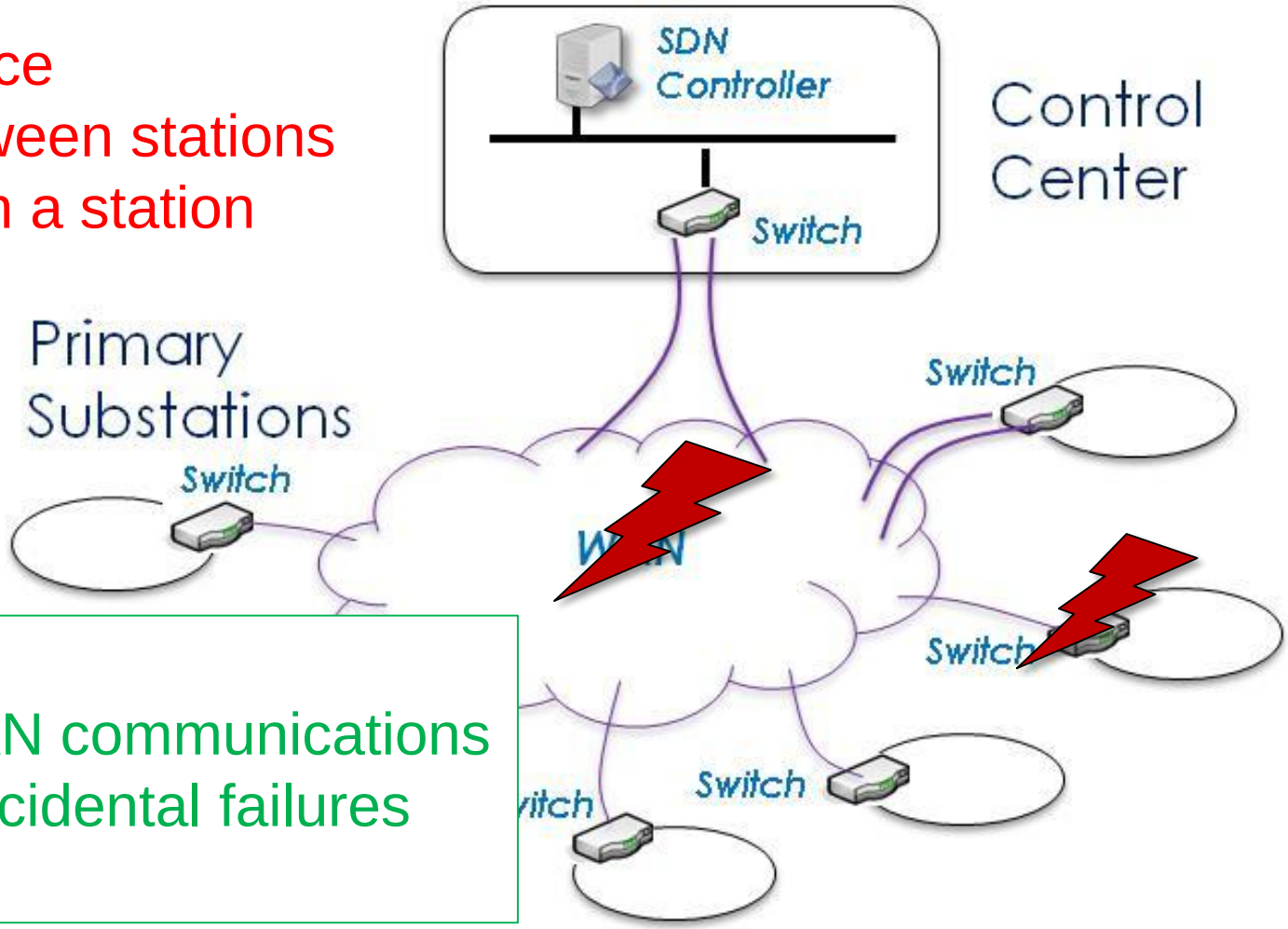
Novel security solutions

- Resilient communication infrastructure

ID	Threat	Likelihood	Impact	Risk	Risk Treatment - Selected Security Controls
UC2_T*n*	Denial of Service attack on the WAN causing the commnication between primary substation and SCADA to be interrupted.	Possible	Medium	**Major**	**FR 7: Resource availability** - *Ensure the availability of the control system against the degradation or denial of essential services.* **SR 7.2 – Resource management** ...

Abstracting the WAN network

Attack surface
- Links between stations
- "Switch" in a station

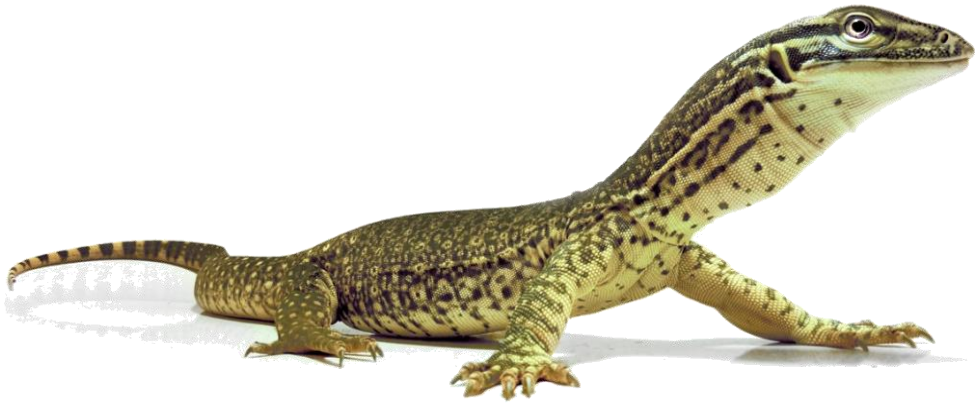


Objective

Make the WAN communications resilient to accidental failures and attacks

Varanus

Resilient WAN Communication for the Smart Grid

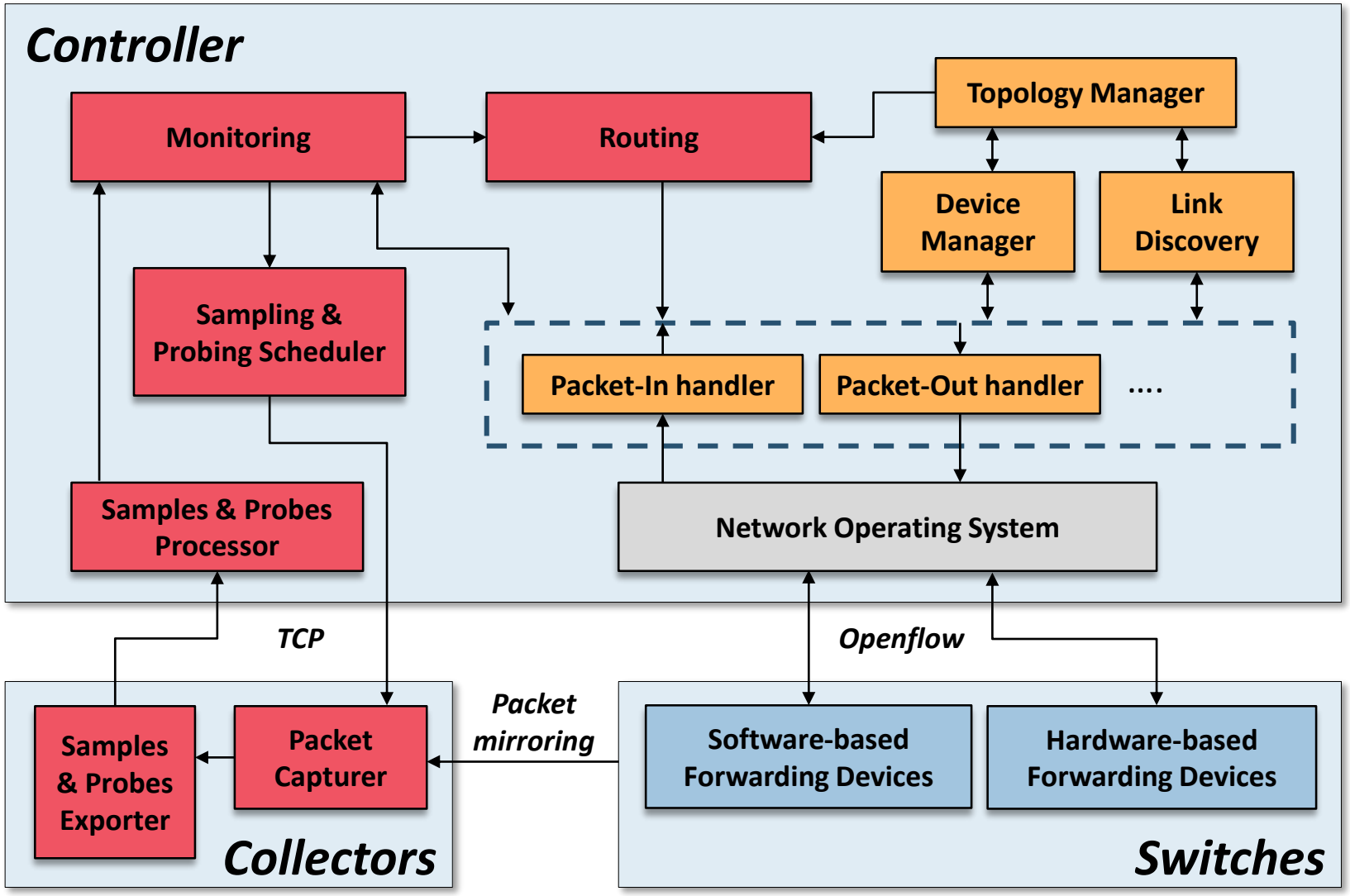


A monitor lizard (Varanus panoptus)

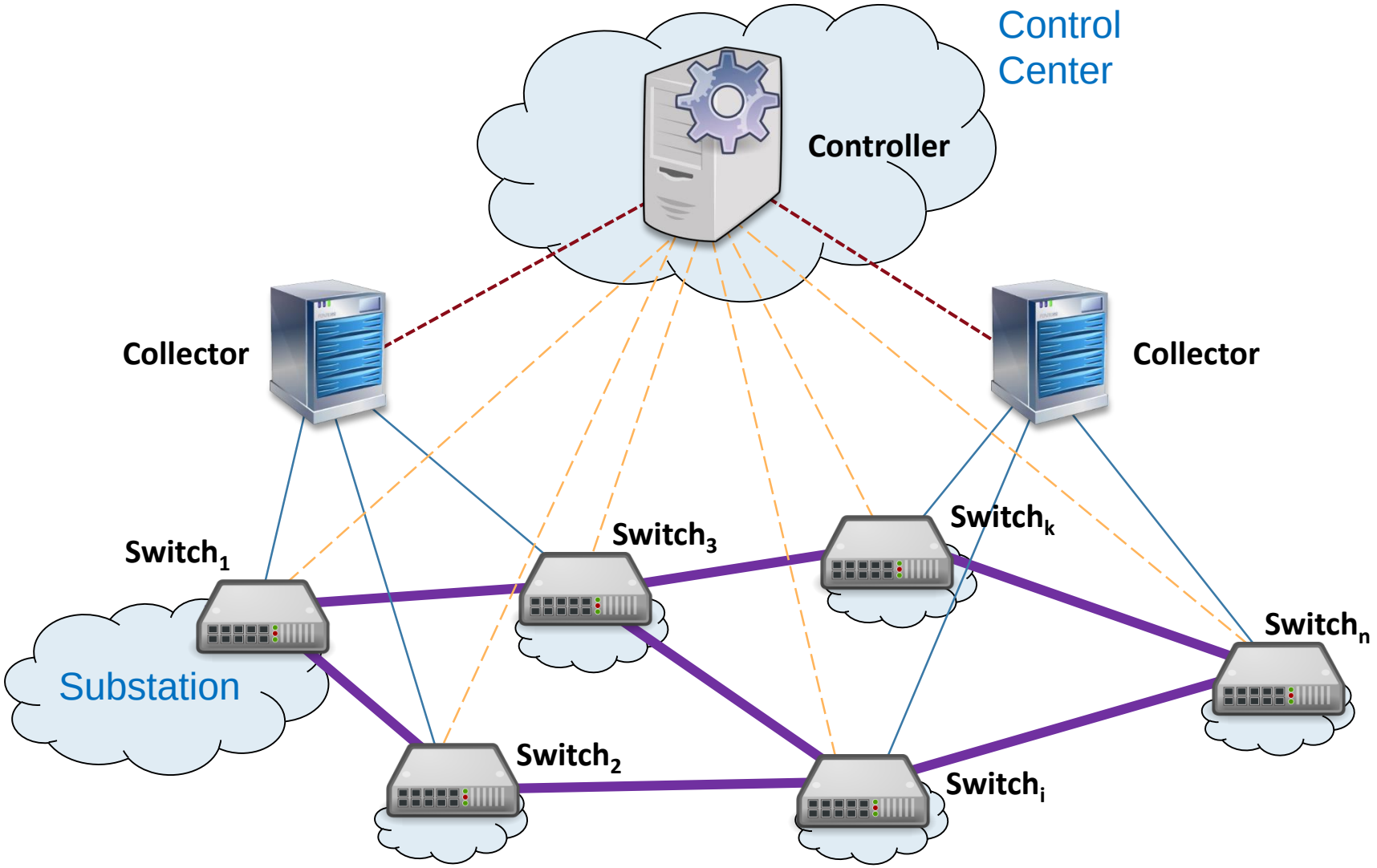
Overview of Varanus

- Varanus is a Software-Defined Network (SDN) based approach, consisting of two main components:
  - A SDN controller running specialized ***monitoring, routing and traffic engineering applications***, typically operating at the control center
  - An auxiliary ***traffic collector application*** located at multiple sites near substations (i.e., the edge communication switch)

Software modules



Main Components

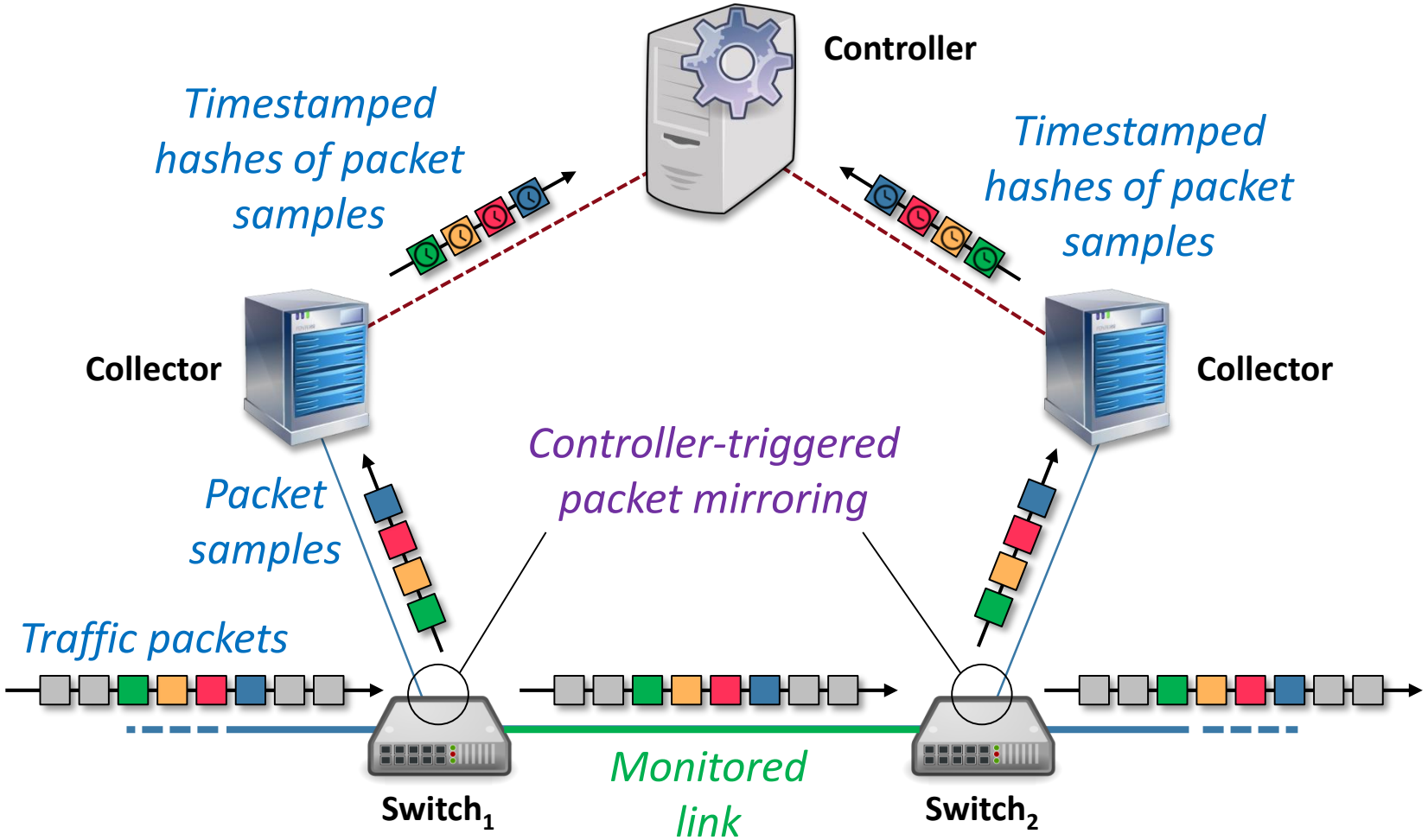


Operation Summary

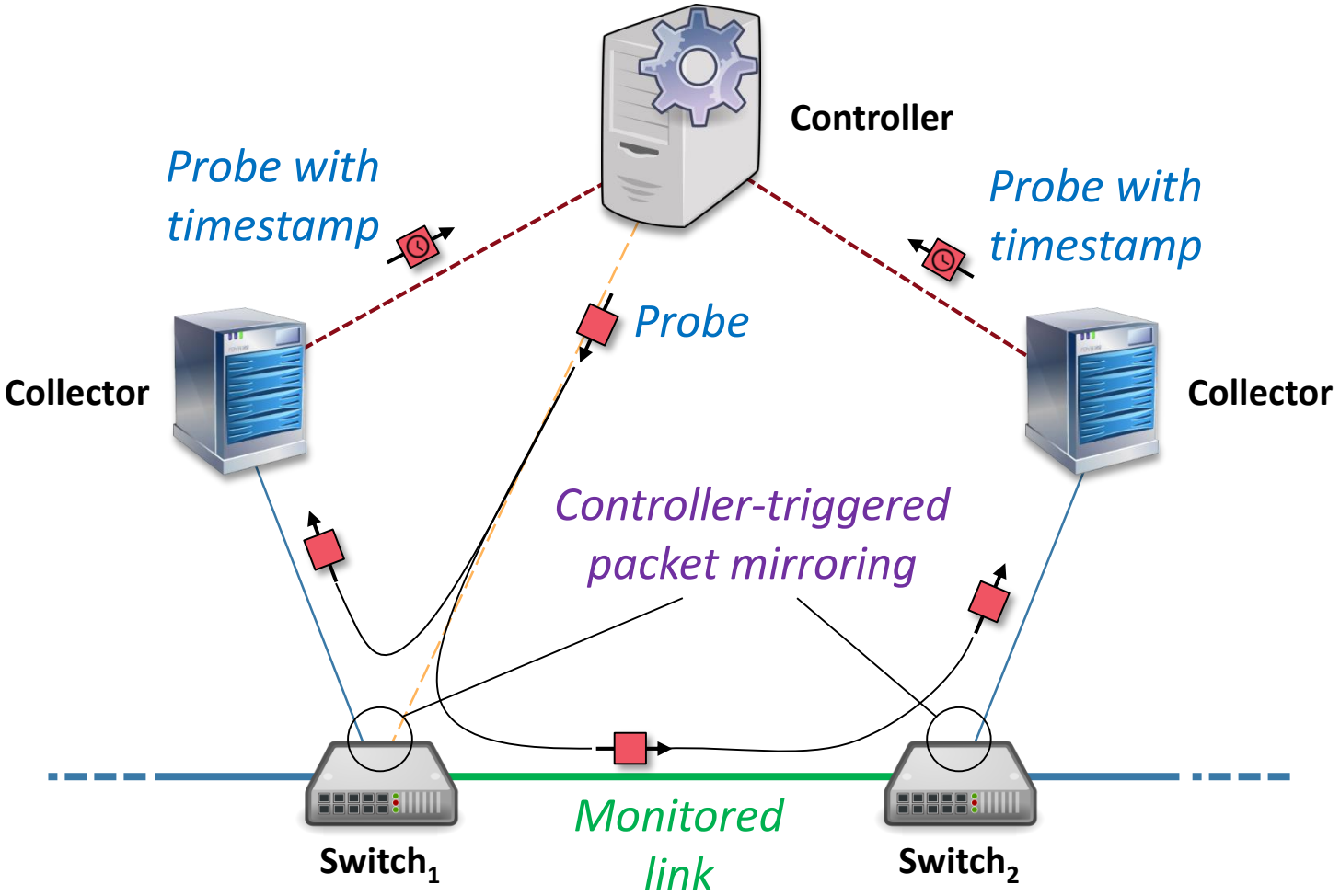
- Building an updated view of the network
  - at random times, the controller makes the switches sample the traffic from specific links to the collectors
  - collectors timestamp and aggregate sampled traffic, and send the data to the controller (in the absence of traffic, special-purpose probes are used instead)
  - the controller calculates metrics about the links from the samples or probes (e.g., latency, loss rate, throughput)

- Configuring the network
  - the controller adjusts routes according to monitored metrics by re-installing forwarding rules in the switches

Traffic sampling



Monitoring with a Probe



Latency calculation

1. The controller requests a sampling/probing operation on a particular link by setting up the two endpoint switches

2. The controller waits until the operation finishes and stores any received timestamped samples/probes from the collector(s) for the two switches

3. Upon receiving two identical samples/probes from both switches, the controller calculates the time difference between the timestamps

4. Calculate average latency and other statistics

Requires: clock synchronization among the collectors

Other Calculations

After the arrival of the hash samples, ...

- Loss rate calculation

the controller counts the number of samples/probes from the source collection (**T**), then finds the number of samples/probes that are in common at both source and destination collections (**C**)

$$\textit{loss rate} = 1 - C/T$$

- Throughput calculation

the controller obtains the duration of the sampling operation (**D**), then finds the total size (*in bits*) of samples that are present in both source and destination collections (**S**)

$$\textit{throughput} = S/D \qquad [\textit{bits/s}]$$

Route Selection

1. The controller keeps track of the network topology in a graph structure & stores the latest metrics for each link

2. Each link is assigned a "weight" value that takes into account the current value of the metrics (e.g., latency and loss rate) --- higher values for latency or loss rate metrics result in higher "weight" values

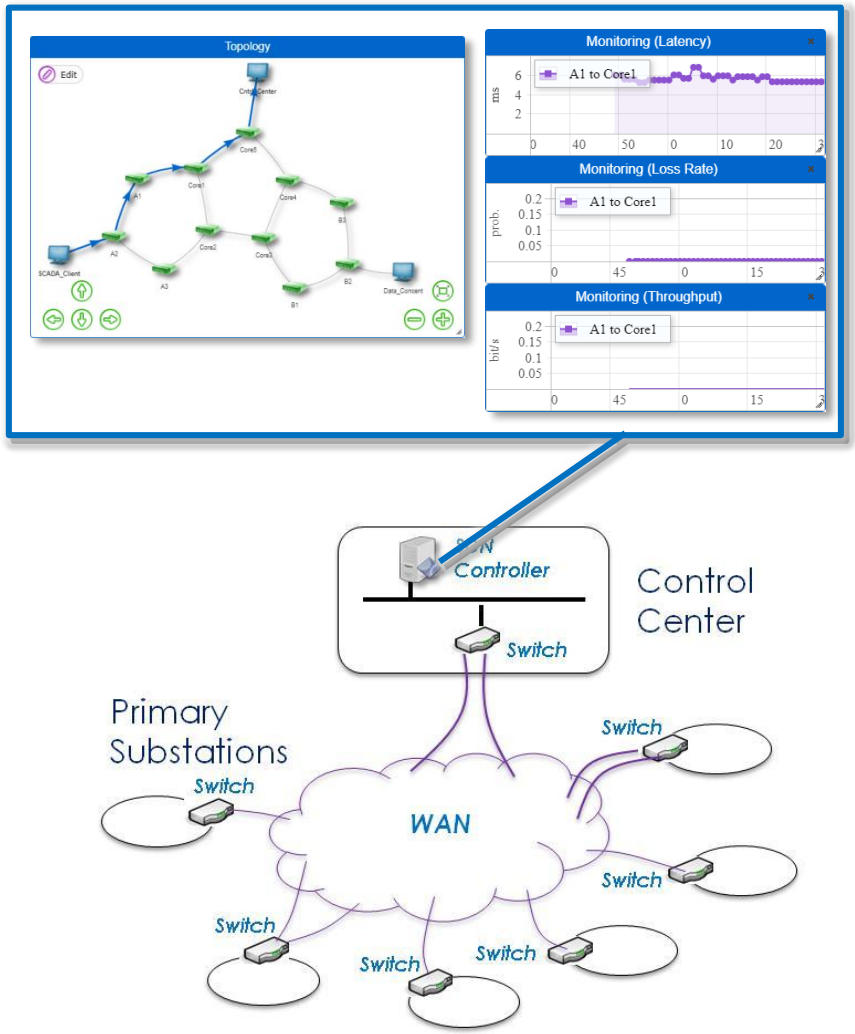
3. Periodically executes a "weighted-shortest-path" algorithm on the graph structure to find the best routes between selected switches (e.g., those corresponding to switches connected to substations)

4. For each new found "best" route, the controller updates forwarding rules in the switches along the path (and removes any previous rules from past routes)

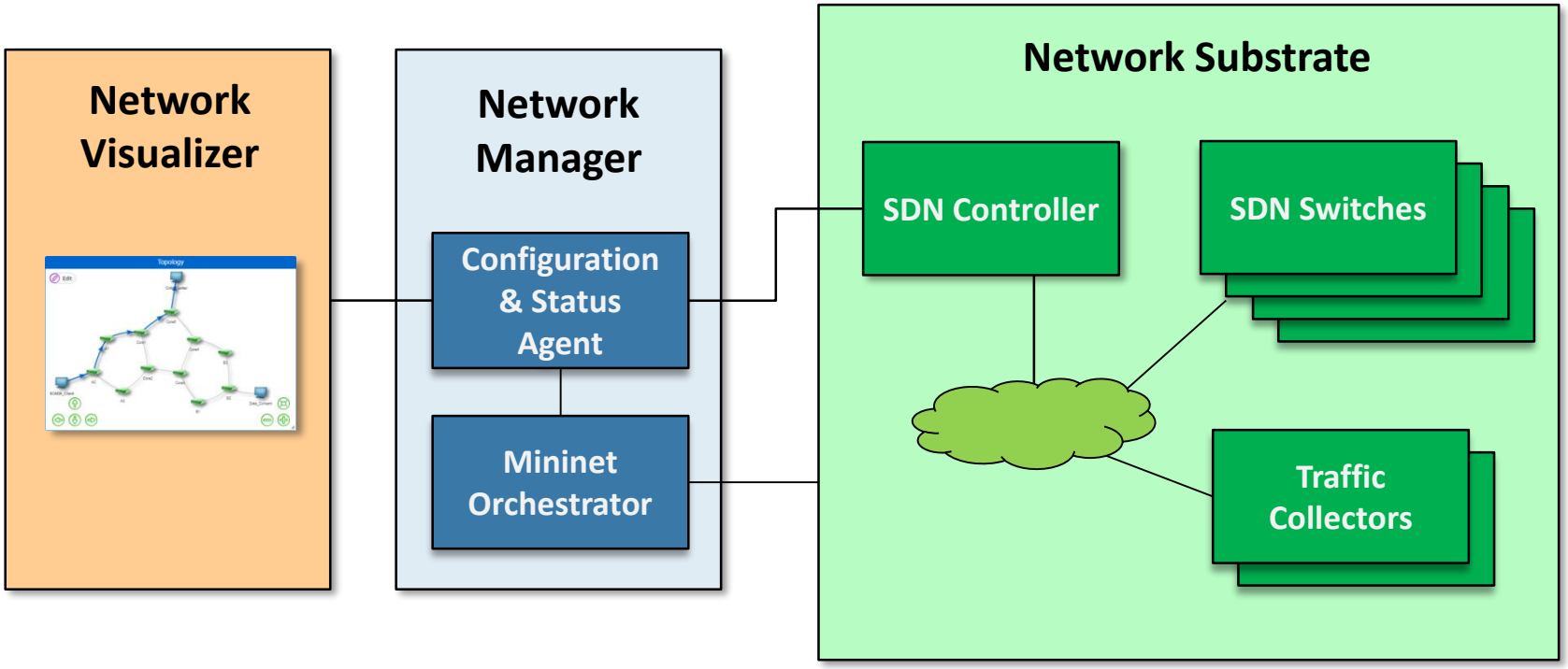
Demonstration

Demo objective

- Provide an overall view of the communications infrastructure, including the topology
- Display configured routes and statistics being monitored from the network interconnections
- Show resilience of the infrastructure operation under various attack scenarios



Experimental Environment



Network

- Supports **arbitrary network topologies** using Mininet, where
  - hosts and switches run in independent containers (lightweight virtual machines) and communicate using IP based protocols
  - industry-grade software switches are interconnected using virtual ethernet interfaces (hardware switches also supported)
- Individual network links have configurable properties such as traffic delay, packet loss and available bandwidth that supports realistic emulation of physical conditions

The topology is specified and then implemented by creating the necessary hosts, switches and links between them

```
import os.path

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


def pre_start_config( mr, extra_args ):
    """ Configure a MininetRunner object before Mininet starts.
        - mr         : a MininetRunner object
        - extra_args: extra arguments passed by the command line
    """
    varanus_home = os.path.join( '/home', 'eric', 'varanus' )

    topo = VaranusTopo( mr, extra_args )


    # Controllers
    topo.add_local_sdncontroller( varanus_home, \
                                  sudo='eric', \
                                  ip='127.0.0.1', port=6653 )


    # Switches
    rb = topo.get_ring_builder( 1 )
    node_builder = topo.get_ring_local_switch_builder()
    rb.add_node( node_builder, 'Core1' )
    rb.add_node( node_builder, 'A1' )
    rb.add_node( node_builder, 'A2' )
    rb.add_node( node_builder, 'A3' )
    rb.add_node( node_builder, 'Core2' )
    rb.add_node( node_builder, 'Core3' )
    rb.add_node( node_builder, 'B1' )
    rb.add_node( node_builder, 'B2' )
    rb.add_node( node_builder, 'B3' )
    rb.add_node( node_builder, 'Core4' )
    rb.add_node( node_builder, 'Core5' )
    topo.add_existing_ovs_switch_ring( rb.build() )

    topo.add_ring_vir_link( 1, 'Core1', 1, 'Core2' )
    topo.add_ring_vir_link( 1, 'Core3', 1, 'Core4' )


    # Hosts
    topo.add_ring_local_vir_host( 'SCADA_Client', 1, '192.168.10.101/24', 1, 'A2' )
    topo.add_ring_local_vir_host( 'Data_Concent', 1, '192.168.10.102/24', 1, 'B2' )
    topo.add_ring_local_vir_host( 'Cntrl_Center', 1, '192.168.10.103/24', 1, 'Core5' )
    topo.add_ring_local_vir_host( 'Attacker', 1, '192.168.10.104/24', 1, 'A1', hidden=True )


    # Collectors
    topo.add_ring_local_vir_collector( 'c1', varanus_home, 1, 1 )


    return topo.conclude()
```

Example topology configuration file

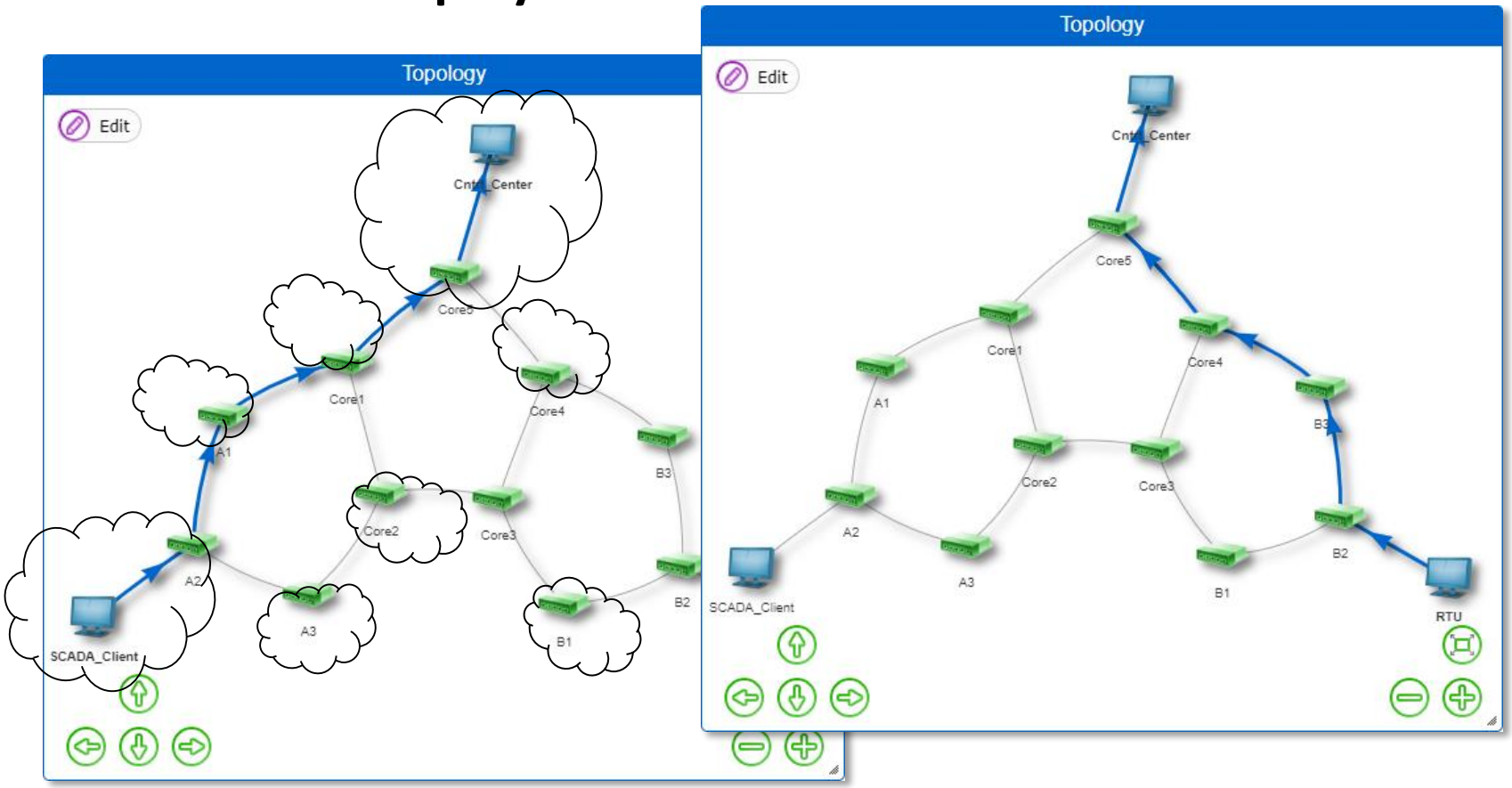
Network visualizer

- Displays the network topology with named components and graph-like representations of hosts/switches and links
- Visualization of data provided by the controller
  - Traffic routes between selected hosts
  - Monitoring statistics about specific links (latency, loss rate, throughput)
- Traffic generation between selected hosts
- Configuration of link properties such as traffic delay, packet loss and available bandwidth

Demonstration
under normal operation

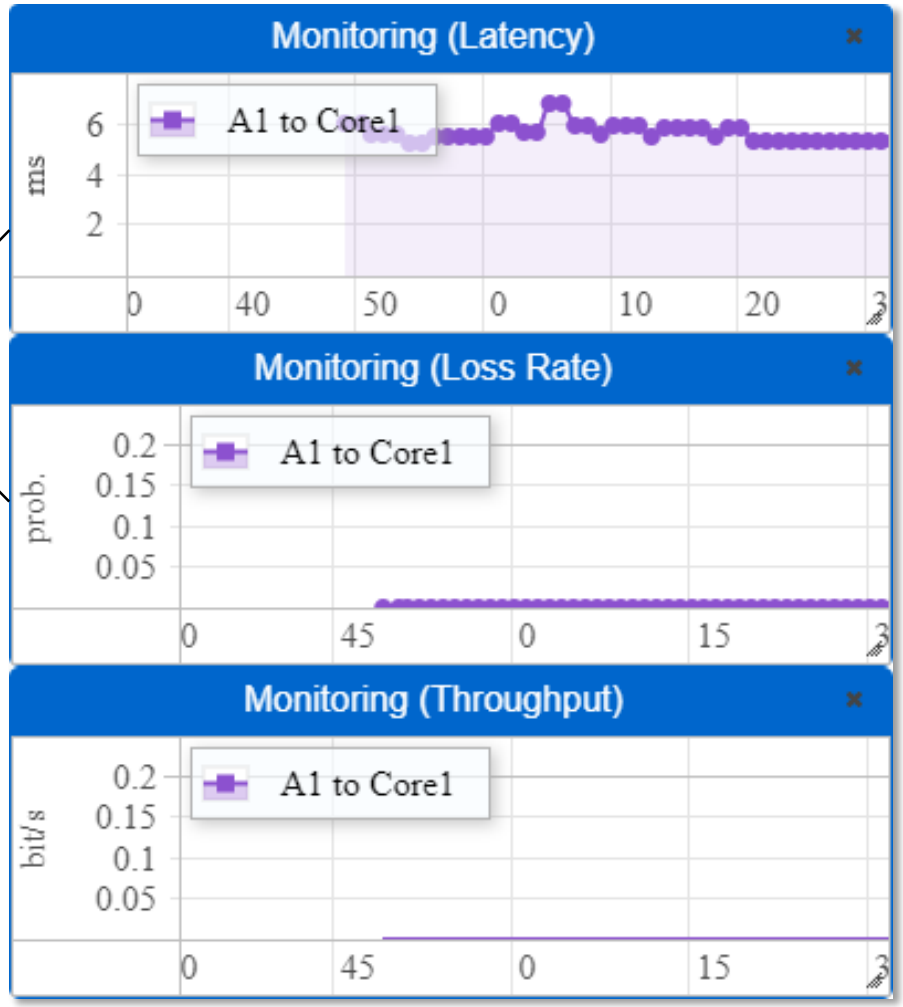
Route visualization

Multiple-ring topology inspired by one DSOs WAN deployment



Link monitoring (no traffic)

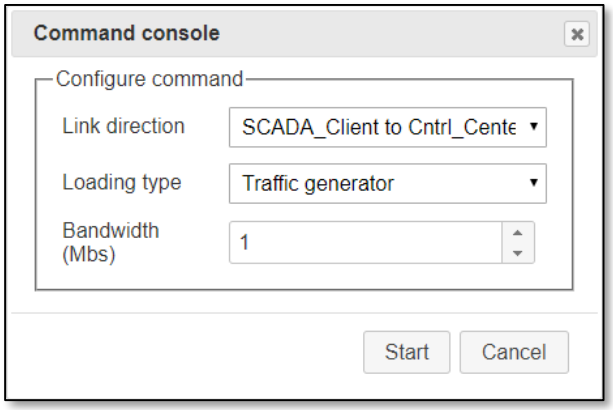
Graphs show dots when metrics are calculated using probes

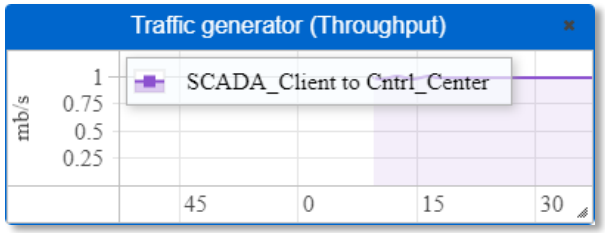


Monitoring indicates zero throughput when there is no traffic in the link

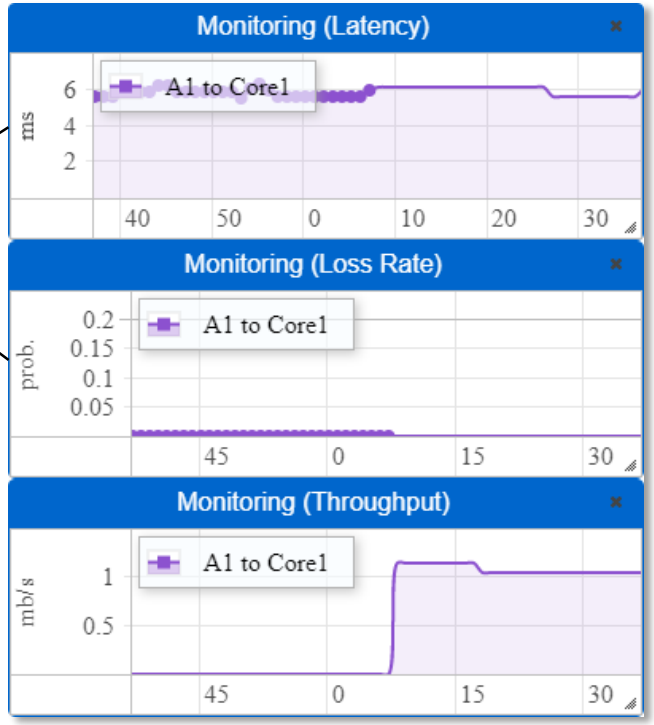
Link monitoring (with traffic)

Traffic is generated from client to server



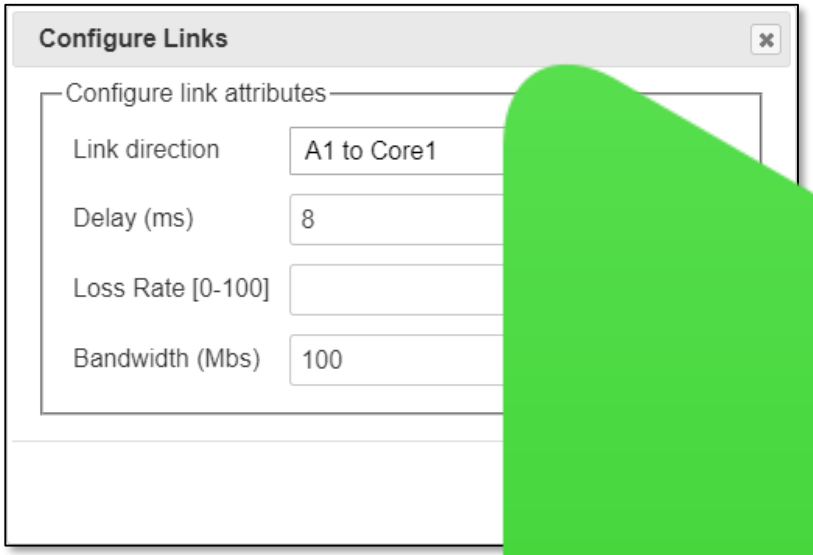


Graphs show a straight line when metrics are calculated using traffic



Monitoring indicates current throughput when traffic traverses the link

Link monitoring (latency change)

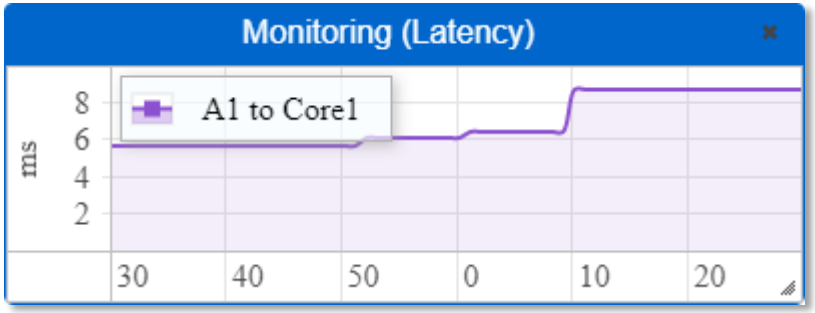


The link's point-to-point delay is adjusted

demonstration

es

Monitoring indicates a higher latency when the link's delay increases



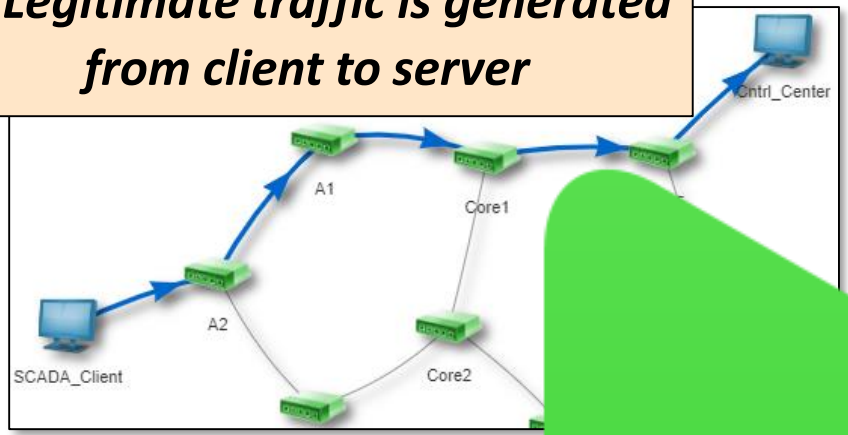
Demonstration
under various attack scenarios

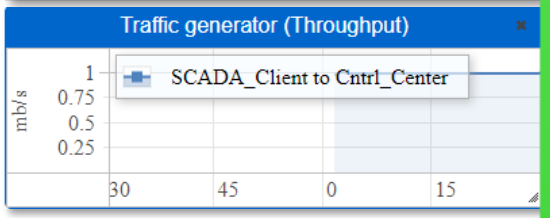
Scenarios under consideration

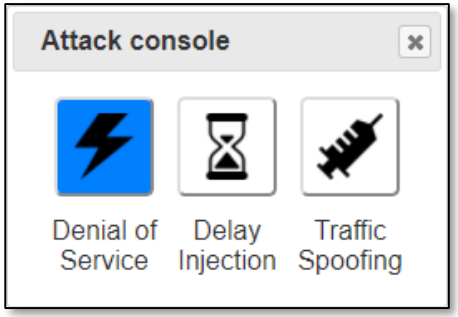
- Based on the risk analysis performed within SEGRID, we identified several relevant attack scenarios
- The 3 scenarios that will be demonstrated
  - a denial of service attack on the WAN
    (the attacker physically enters a primary station, connects his machine, and generates a large amount of traffic to be sent through the WAN to cause a DoS)
  - delay induced on the edge switch of a primary station
    (the attacker takes control of an edge switch of a primary station, and delays all packets being transmitted locally, including the ones being forwarded)
  - traffic flowing through a link is maliciously altered
    (the attacker takes control of a link between two primary stations, and maliciously modifies the packets as they go through)

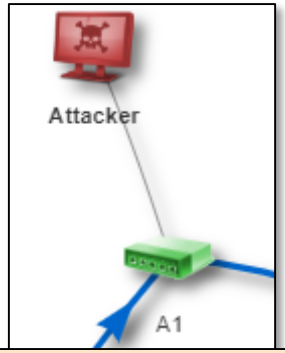
1: DoS attempt

1. ***Legitimate traffic is generated from client to server***



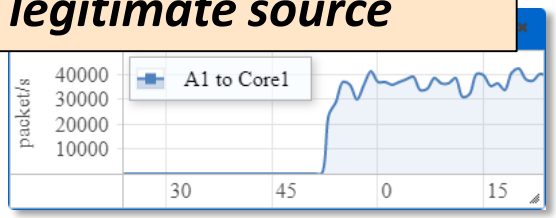


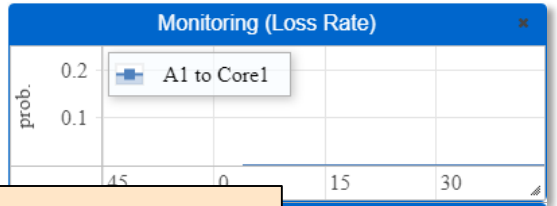




2. ***An attacker connects to a nd injects a large amount affic into the network***

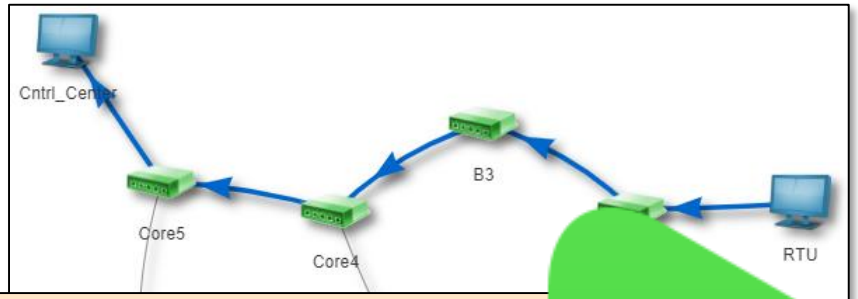
3. ***The switch drops the injected traffic since it is not from a legitimate source***



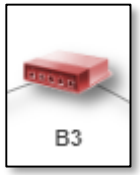


4. ***Monitoring indicates that legitimate traffic is unaffected by the attack***

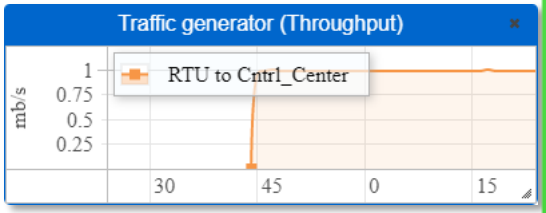
2: Traffic delay injection



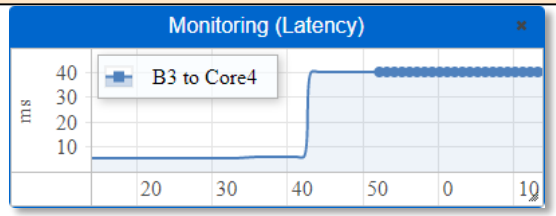
1. ***Legitimate traffic is ge***
from the RTU to the s



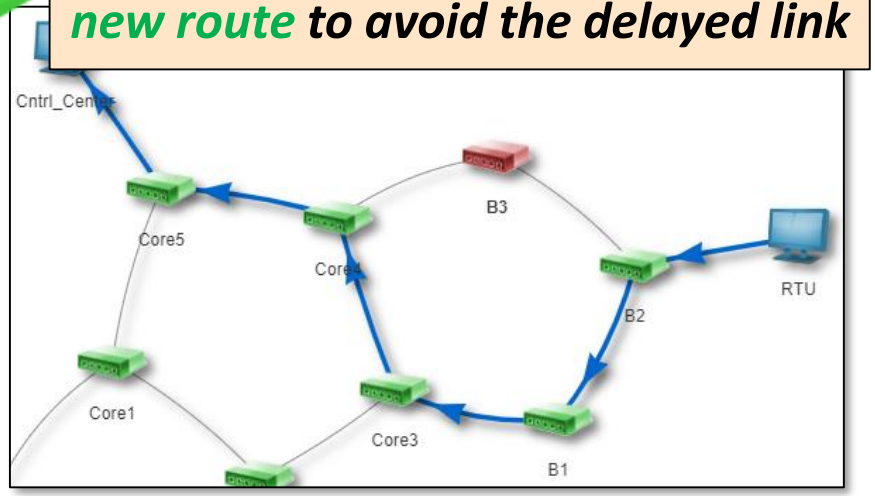
2. ***An attacker compromises a switch and delays all traffic***



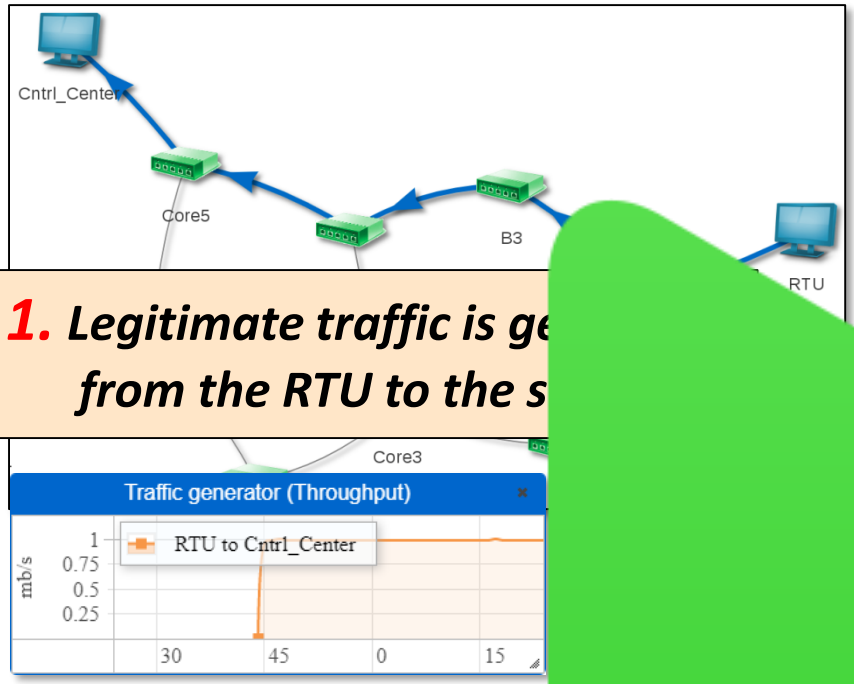
3. ***Monitoring indi an increase of the link latency***



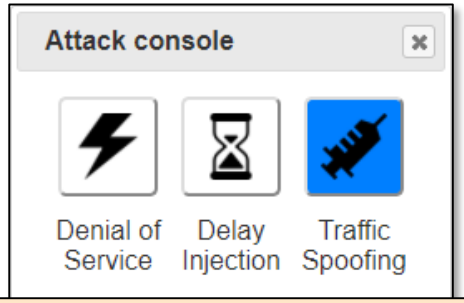
4. ***The controller recalculates a new route to avoid the delayed link***

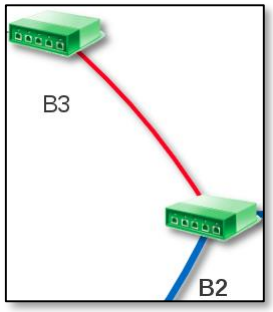


3: Traffic spoofing



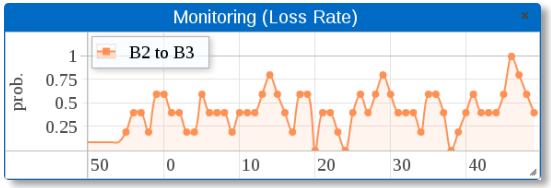
1. ***Legitimate traffic is ge***
from the RTU to the s



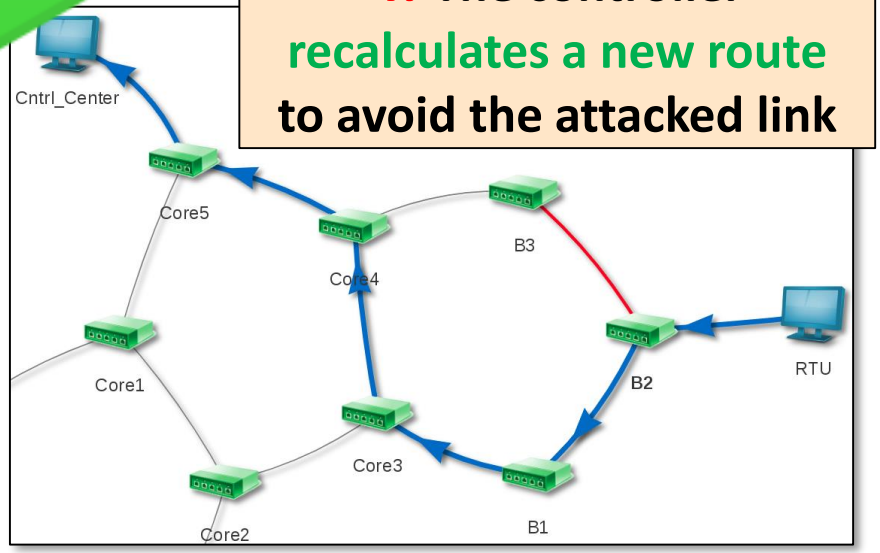


2. ***An attacker intercepts a link and***
lthily modifies some traffic

3. **Monitoring neverth**
indicates an increase of t link loss rate



4. **The controller recalculates a new route to avoid the attacked link**



Thank you! Any questions?

This was:

Resilient Communications

Web: **http://segrid.eu**

http://www.navigators.di.fc.ul.pt