STOCHASTIC ACTIVITY NET MODELS FOR ADAPTIVE PROTECTION

Amit Dilip Patil

Department of Informatics and Mathematics
amit.patil@uni-passau.de

Technical Report No. MIP-2301
Chair of Computer Networking and Computer Communications
Department of Informatics and Mathematics
University of Passau
June 2023

Abstract

The technical report provides complementary materials and models for the paper "Quantifying the impact of communication delay on adaptive protection systems." The paper investigates modern protection systems in power grids that face new challenges due to distributed renewable energy sources. Protection issues introduced due to these distributed renewable energy sources, such as blinding and sympathetic tripping, are discussed. A communication network is used to solve these issues by adapting the sensitivity of the protection devices. However, the communication network is subject to strict performance requirements such as communication delay, which are investigated in the paper. A discrete event simulation-based model is developed to assess the impact of parameters such as communication delay on the adaptive protection system. The results demonstrate that using a communication network improves the performance of the protection system even when subjected to communication delays. The proposed method allows the performance analysis of protection systems. Consequently, performance thresholds can be determined for various scenarios. Furthermore, the impact of parameters such as fault impedance, protection sensitivity and renewable energy sources can be investigated using the proposed approach. The relevant models required for the above analysis are presented in this report.

1 Introduction

Modern Power Systems (PSs) are undergoing drastic changes, particularly in operation and structure. The integration of Distributed Renewable Energy Sources (DRESs) decentralizes the traditionally centralized energy source. This transition fundamentally challenges the safe operation of the system, especially its protection system. The protection system has operated under the assumption of single-source fault current feed-in from the external grid to the fault location, e.g., in distribution grid feeders. The presence of DRESs could violate this assumption leading to protection misoperation. Some protection misoperation scenarios are protection blinding, e.g., when a DRES exists between a protection device such as a Circuit Breaker (CB) and the fault location. In this scenario, if the DRES supplies fault current, the upstream current through the CB will reduce, blinding the protection device. This may cause the CB to trip late or not trip at all. Another protection misoperation is sympathetic tripping, e.g., when a CB in fault-free feeder trips due to a DRES in this feeder supplying fault current. A solution to these issues is *adaptive protection. Adaptive* protection schemes use Information and Communication Technology (ICT) to adapt the settings of protection devices. This adaptation of protection settings is achieved by sending new settings using a communication network. However, communication imposes a delay caused due to the transfer of messages. This delay could cost crucial time, especially in the context of protection, where faults could cause damage to the system. Therefore, the impact of this delay should be investigated.

This report aims to provide a complete overview of the model used in the main paper. The relevant state-of-the-art methodology and results are omitted, while the list of Stochastic Activity Net (SAN) models used in the paper are described.

2 Stochastic Activity Net models

This section provides an overview of the SAN models used in the main paper. A short description of the model accompanies each SAN model. The SAN models are implemented in the Möbius tool [2]. The models are initialized with parameters from an input binary file [1, 3].

2.1 Composed model

The overall model is composed of ten submodels. The aggregation of all submodels forms the overall model, known as the "composed" model. This composed model accumulates the shared state variables of each submodel. The composed model is shown in Figure 1. Each block named "Submodel" indicates a separate SAN model. Each block named "Rep" indicates that the connected submodel is replicated a specified number of times. The "Join" block indicates the aggregation of all submodels, specifically the aggregated shared state variables. The following subsections describe each submodel.

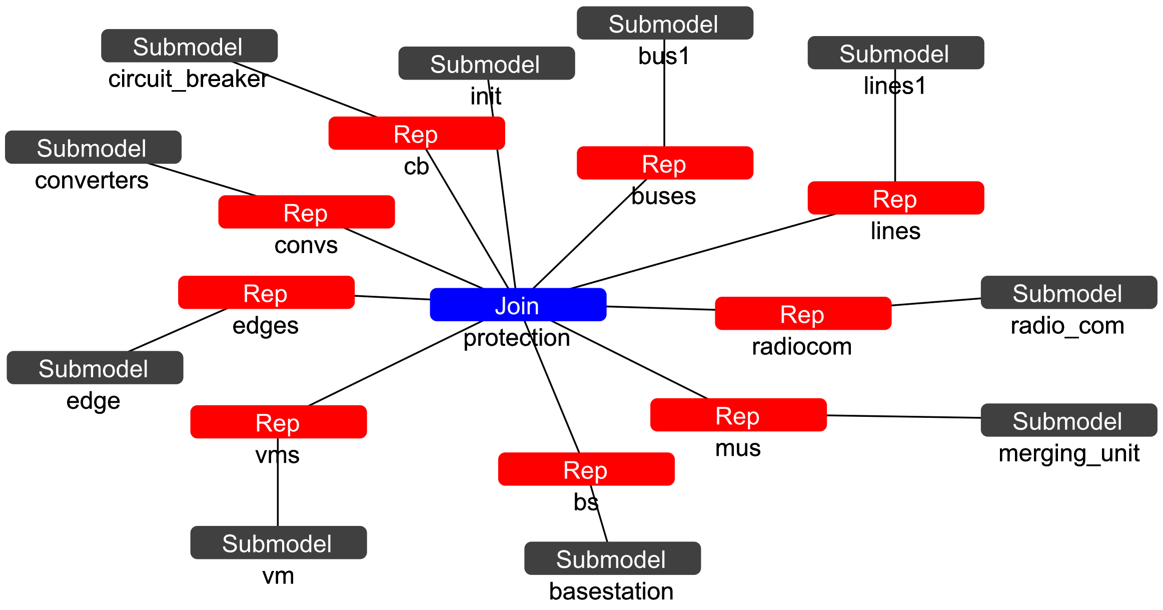

Figure 1: Composed model

2.2 Initialization model

A simulation starts with the initialization of the model. This is shown in Figure 2, where the extended places (yellow circles) are assigned with initial values. A binary file is used for each extended place to input these values. A code excerpt is shown in listing 1, where the extended place bus_OK is initialized with data from file ”bus.dat”.

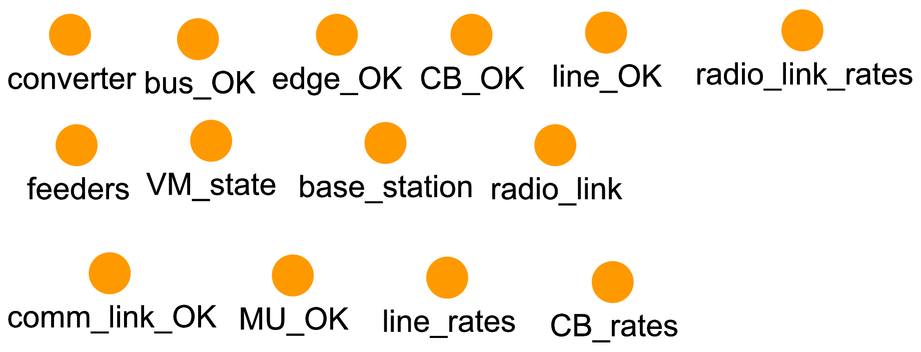

Figure 2: Initialization model

```
using namespace std;
std::ofstream ondata;
std::ifstream indata;
indata.open("C:\\Input\\bus.dat", ios::in|ios::binary);

for( i = 0; i < num_bus; i++)
{
indata>>bus_OK->Index(i)->bus_index->Mark();
indata>>bus_OK->Index(i)->disconnected->Mark();
indata>>bus_OK->Index(i)->state_ok->Mark();
indata>>bus_OK->Index(i)->fault_exists->Mark();
indata>>bus_OK->Index(i)->feeder_index->Mark();
indata>>bus_OK->Index(i)->num_line->Mark();
indata>>bus_OK->Index(i)->num_mu->Mark();
l = bus_OK->Index(i)->num_line->Mark();
m = bus_OK->Index(i)->num_mu->Mark();
for( int j = 0; j < l; j++)
        indata>>bus_OK->Index(i)->line_index->Index(j)->Mark();

for( int j = 0; j < m; j++)
        indata>>bus_OK->Index(i)->mu_index->Index(j)->Mark();
}
indata.close();
```

Listing 1: C++ code to initialize SAN model using binary files.

The main part is the *for* loop which iterates *num_bus* times. The variable *num_bus* represents the number of buses in the system. Inside the loop, information such as bus index, state flags of the bus, number of lines connected to the bus etc. is loaded. This process is performed for each extended place that requires initialization.

2.3 Bus model

After initialization, the indexing of each submodel takes place. The indexing is done to allocate one unique index to each submodel. For example, each bus will have a unique index for identification purposes. The indexing of replicas is based on the approach of Chiaradonna et al [1]. Once the initialization and indexing are complete, the Discrete Event Simulation (DES) to solve the SAN models starts. The first event in the DES is a fault (*fault_occurs*), which occurs at a bus. The fault may be cleared by activating the relevant protection device (*line_disconnects*). If the protection device fails, the fault may be cleared naturally (*fault_clears_natural*) or a maintenance crew may be sent to clear the fault manually(*maintenance_crew*). The bus model is shown in Figure 3. The properties of these events, modeled as activities, are shown in Table 1. The *line_disconnects* and *line_reconnects* are instantaneous as the time delays associated with these events are in the circuit breaker model.

The impact of communication delay on the protection system is studied by measuring the fault clearing time. The fault clearing time is measured as the time from when the fault occurs to when the fault is cleared or isolated. Therefore, these times are recorded to calculate the fault clearing times. The listing 2 shows the code to record when the fault occurs. This is done using the library function *LastActionTime*. The fault clearing time is then calculated when the fault is cleared. The time when the fault occurs is stored in extended place *time*. When the fault is cleared, the fault clearing time is calculated as shown in listing 3.

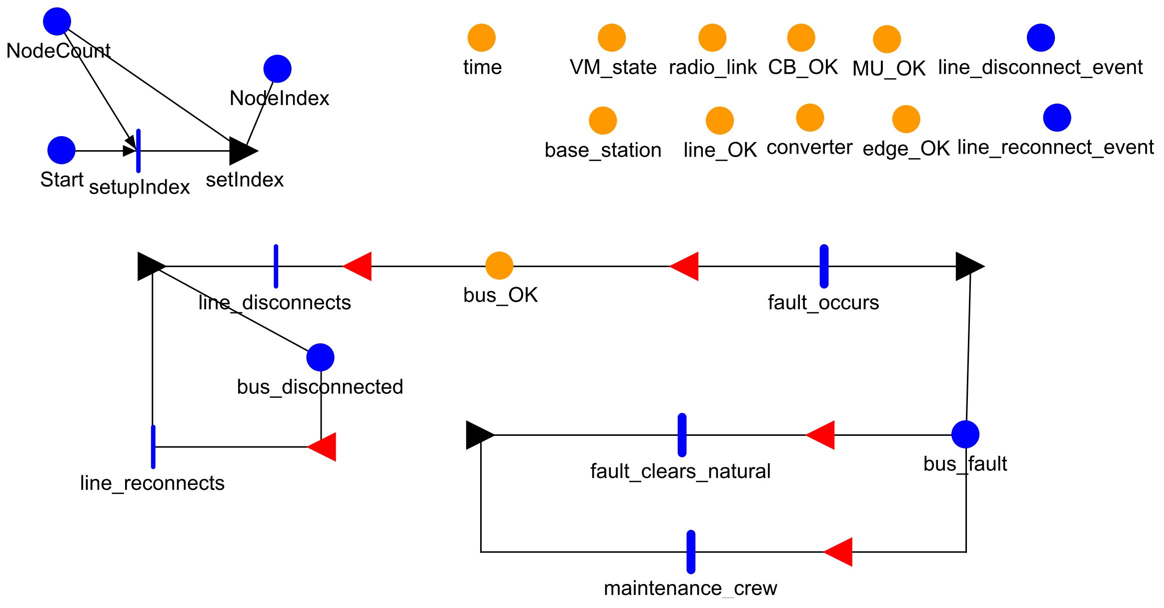

Figure 3: Bus model

Activity name	**Type**	**Time distribution**	**Parameters**
line_disconnects	Instantaneous	-	-
line_reconnects	Instantaneous	-	-
fault_occurs	Timed	Exponential	50 hours
fault_cleared_natural	Timed	Exponential	2 hours
maintenance_crew	Timed	Exponential	4 hours

Table 1: Activities of bus model shown in Figure 3 and its properties

```
ondata.open("C:\\output\\line_fail.dat", ios::out|ios::binary|std
    ::ios_base::app);
ondata << LastActionTime << endl;
ondata.close();

double time_t;
time_t = LastActionTime;
time->Mark() = time_t;
```

Listing 2: C++ code to calculate and store fault clearing times in binary files.

```
time->Mark() = LastActionTime - time->Mark();
ondata.open("C:\\output\\line.dat", ios::out|ios::binary|std::
    ios_base::app);
ondata << time->Mark() << endl;
ondata.close();
```

Listing 3: C++ code to calculate and store fault clearing times in binary files.

2.4 Merging unit model

The fault impacts the voltage and current magnitudes at the bus. These values are measured using a merging unit. The state of a merging unit can be impacted by the communication link(*communication_link_fail*), a software failure(*mu_software_failure*), random hardware failure(*random_failure*) or loss of power supply(*bus_failure*). The model of a merging unit is shown in Figure 4. The properties of these activities are shown in Table 2. The *communication_link_fail*, *communication_link_repair*, *bus_failure* and *bus_repair* are instantaneous as the time delays associated with these events are in the radio link and bus models respectively.

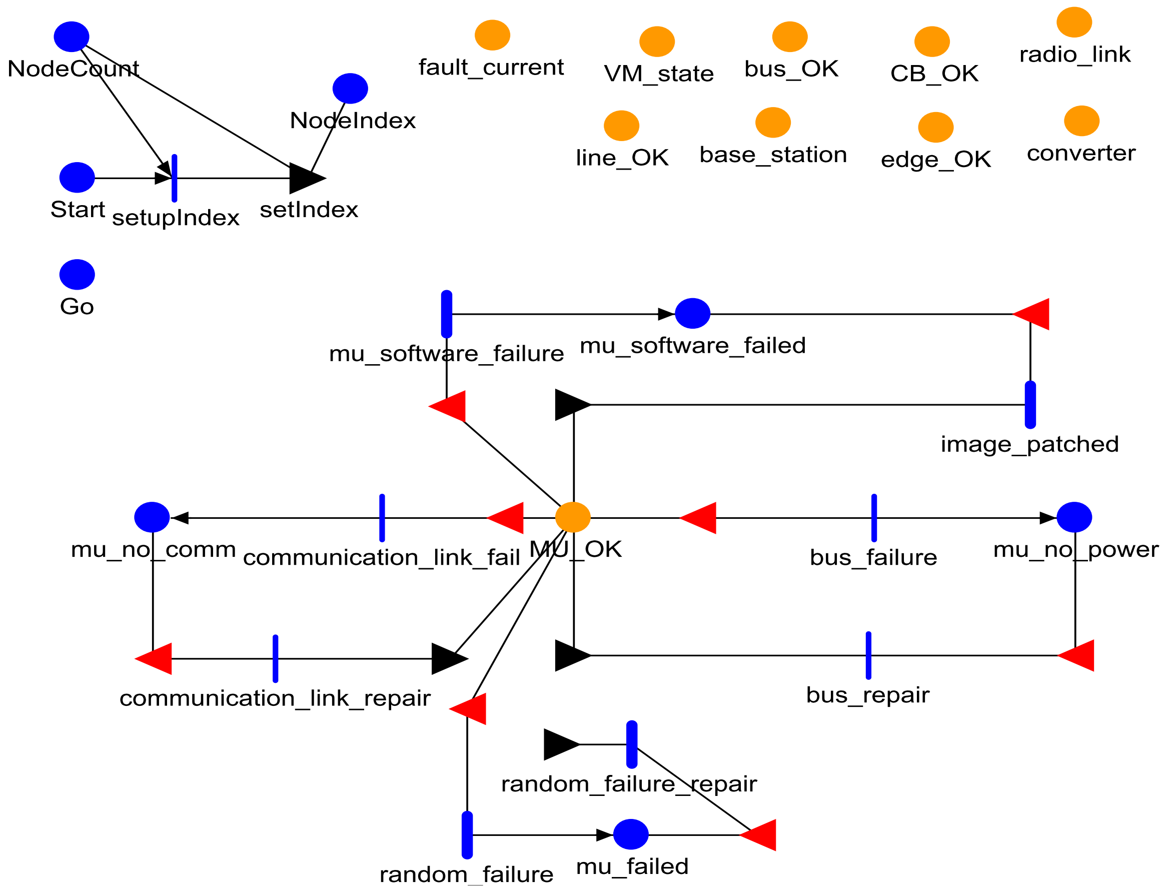

Figure 4: Merging unit model

Activity name	Type	Time distribution	Parameters
communication_link_fail	Instantaneous	-	-
communication_link_repair	Instantaneous	-	-
bus_failure	Instantaneous	-	-
bus_repair	Instantaneous	-	-
mu_software_failure	Timed	Exponential	1 428 hours
image_patched	Timed	Exponential	1 hour
random_failure	Timed	Exponential	9 000 hours
random_failure_repair	Timed	Exponential	2 hours

Table 2: Activities of merging unit model shown in Figure 4 and its properties

2.5 Radio link model

The measurements obtained from a merging unit are communicated using a communication link, which are radio links in the system considered (see original paper). The model of a radio link is shown in Figure 5. The model of the radio link is based on the work of Tesfaye et al. [4]. A key extension to their model is the addition of communication delay, modeled by the *communication_delay* activity. The state of a radio link may be impacted by the failure of a connected device *device_fail*, fading *radio_link_fading*and failure of the radio interface at the connected device *equipment_failure*, but the device remains operational. The *all_links_failed* models the case where all redundant links have failed, leading to a complete failure of the radio link. The properties of these activities are shown in Table 3. The *device_fail* and *device_repair* are instantaneous as the time delays associated with these events are in the respective device models.

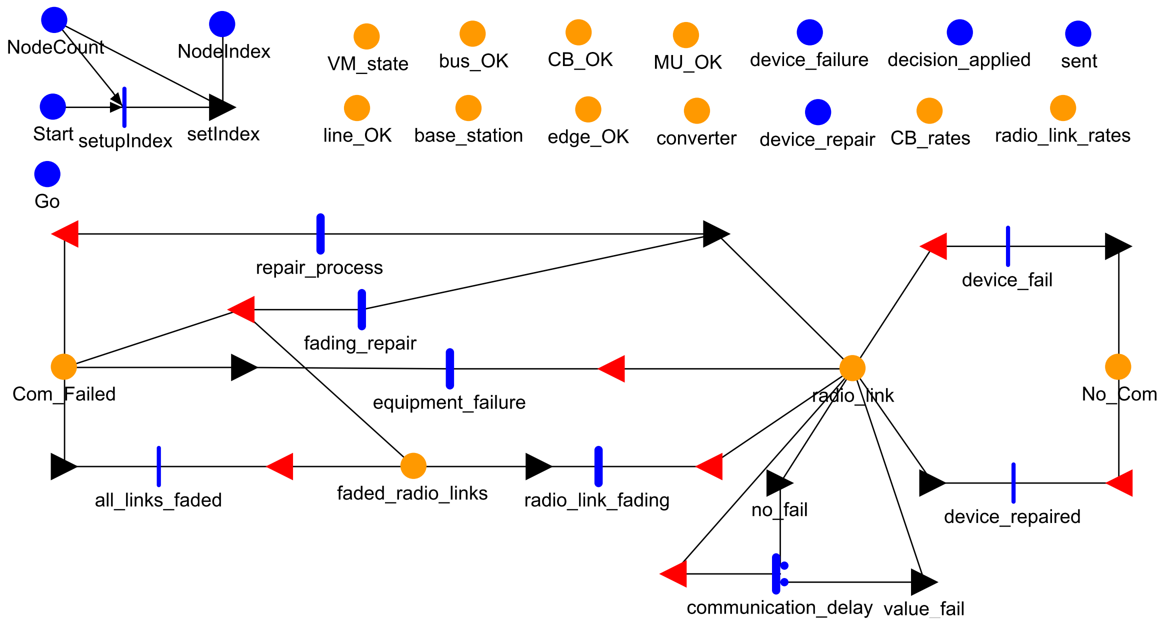

Figure 5: Radio link model

Activity name	Type	Time distribution	Parameters
device_fail	Instantaneous	-	-
device_repaired	Instantaneous	-	-
radio_link_fading	Timed	Exponential	64.8 hours
all_links_failed	Instantaneous	-	-
equipment_failure	Timed	Exponential	10 000 hours
fading_repair	Timed	Exponential	100 milliseconds
repair_process	Timed	Exponential	4 hours
communication_delay	Timed	Exponential	Input dependent

Table 3: Activities of radio link model shown in Figure 5 and its properties

2.6 Base station model

The radio links interface the field devices with the base station. The base station communicates with the edge server using a physical communication link. The model of a base station is shown in Figure 6. The base station may fail (*failure*) or be overloaded by traffic (*overload*). Correspondingly, the base station may be repaired (*repair*) or be relieved (*relief*). The properties of these activities are shown in Table 4.

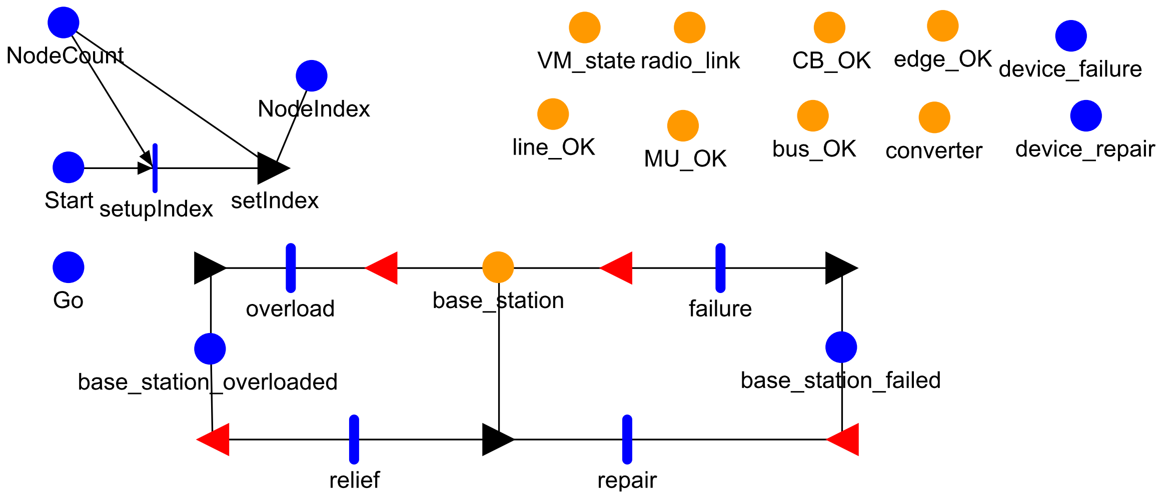

Figure 6: Base station model

Activity name	Type	Time distribution	Parameters
overload	Timed	Exponential	1 200 hours
relief	Timed	Exponential	5 hours
failure	Timed	Exponential	5 000 hours
repair	Timed	Exponential	10 hours

Table 4: Activities of base station model shown in Figure 6 and its properties

2.7 Edge server model

The edge server hosts the VMs required for the protection functions. The model of an edge server is shown in Figure 7. The edge server is connected to the base station via physical communication links, which may fail (*communication_link_fail*) and be repaired (*communication_link_repair*). The edge server may lose power (*power_loss*), leading to a no-power state. The edge server may be overloaded due to omission failures (*omission_timing_fail*) and consequently relieved. The edge server may also suffer a crash failure, which could be predicted (*edge_predicted_crash_fail*) or unpredicted (*edge_unpredicted_crash_fail*). There are relevant mechanisms to continue operation in these scenarios: VM migration (*migration_delay*) for predicted failures and deploy a snapshot (*snapshot_deploy_delay*), if available, for unpredicted failures. A backup edge server is used in both cases until a repair action is completed. The properties of these activities are shown in Table 5. The *power_loss* and *power_restore* are instantaneous as the time delays associated with these events are in the respective device models.

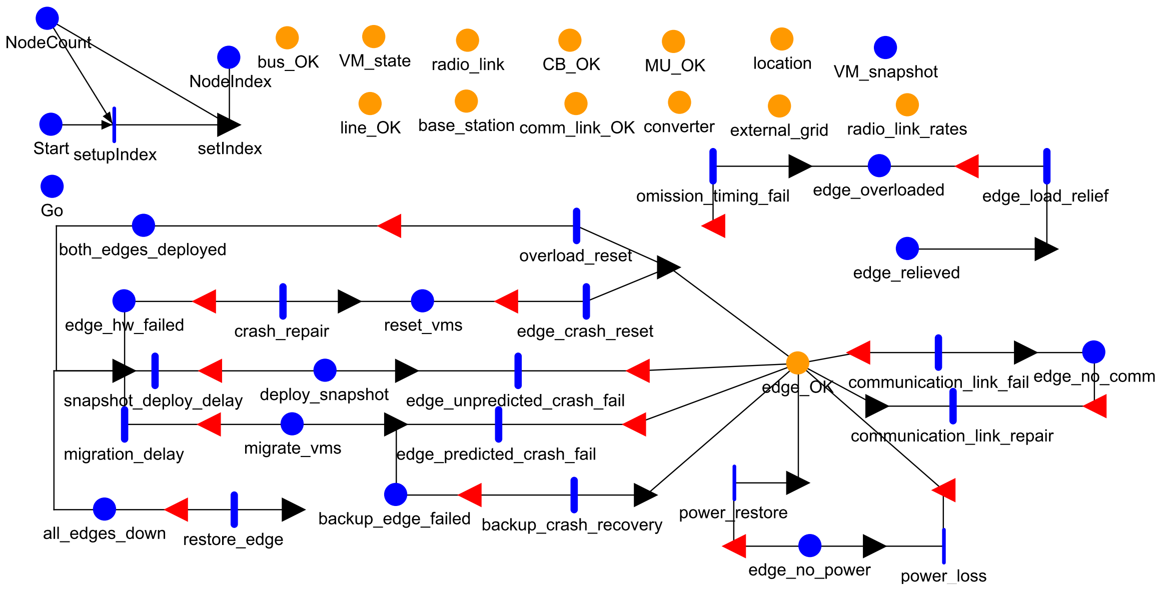

Figure 7: Edge server model

Activity name	Type	Time distribution	Parameters
power_loss	Instantaneous	-	-
power_restore	Instantaneous	-	-
communication_link_fail	Timed	Exponential	9 000 hours
communication_link_repair	Timed	Exponential	6 hours
omission_timing_fail	Timed	Exponential	1200 hours
edge_load_relief	Timed	Exponential	5 hours
edge_predicted_crash_fail	Timed	Exponential	4 000 hours
edge_unpredicted_crash_fail	Timed	Exponential	8 333 hours
backup_crash_recovery	Timed	Exponential	4 hours
restore_edge	Timed	Exponential	2 hours
migration_delay	Timed	Exponential	15 minutes
snapshot_deploy_delay	Timed	Exponential	15 minutes
crash_repair	Timed	Exponential	4 hours
edge_crash_reset	Timed	Exponential	15 minutes
overload_reset	Timed	Exponential	15 minutes

Table 5: Activities of edge server model shown in Figure 7 and its properties

2.8 Virtual machine model

The edge server hosts the Virtual Machines (VMs) required for protection functions. These VMs are modeled using the SAN in Figure 8. The VM may stop functioning due to a hardware failure (*edge_hw_failure*) if the edge server fails or if the VM software crashes (*vm_crash_failure*). If the VM crashes, a snapshot could be used to maintain service (*use_snapshot*). The snapshot can only be used if available (a token in place VM_snapshot). The VM receives measurements from the merging units, based on which it detects and locates the fault (*detect_locate*). The new protection settings are sent if the fault is detected and located (*power_flow_decision*). The output gate after this activity sets the flags in the relevant extended places to trigger events in other submodels, e.g., communication delay for new settings. When the next set of measurements is obtained, the VM checks if the fault is cleared (*cleared_check*). The properties of these activities are shown in Table 6. The *edge_hw_failure* and *edge_hw_repair* are instantaneous as the time delays associated with these events are in the respective device models.

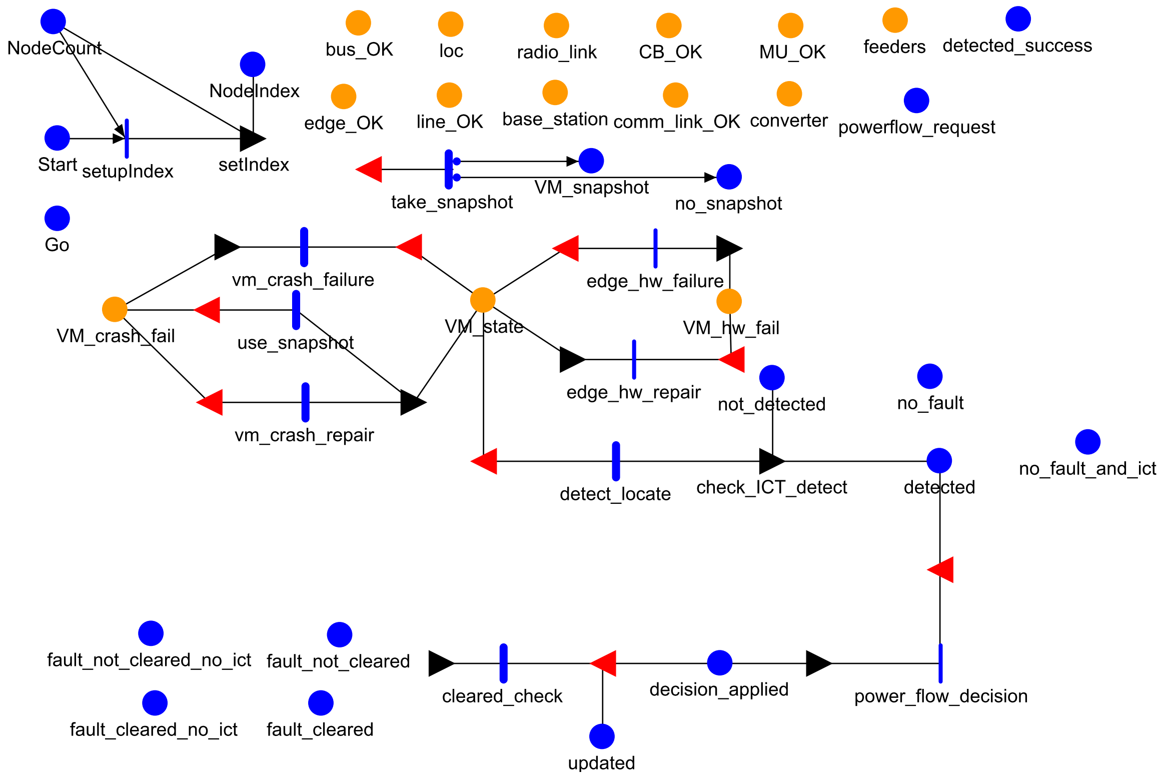

Figure 8: Virtual machine model

Activity name	Type	Time distribution	Parameters
edge_hw_failure	Instantaneous	-	-
edge_hw_repair	Instantaneous	-	-
take_snapshot	Timed	Exponential	15 minutes
vm_crash_repair	Timed	Exponential	12 hours
vm_crash_failure	Timed	Exponential	1 923 hours
use_snapshot	Timed	Exponential	10 minutes
detect_locate	Timed	Exponential	50 milliseconds
power_flow_decision	Instantaneous	-	-
cleared_check	Timed	Exponential	100 milliseconds

Table 6: Activities of virtual machine model shown in Figure 8 and its properties

2.9 Circuit breaker model

The protection device is modeled as a circuit breaker. The circuit breaker receives new settings using a radio link. The circuit breaker model is shown in Figure 9. The circuit breaker may fail (*cb_failure*) and consequently repaired (*cb_repair*). The circuit breaker may also lose connectivity if the radio link fails (*communication_link_fail*). The connectivity is restored when the radio links are repaired (*communication_link_repair*). The state of the circuit breaker may change from closed to open if the current through the device is high enough to trigger the device. The opening of a circuit breaker is modeled by (*cb_opening*), and the closing is modeled by (*cb_closing*). The properties of these activities are shown in Table 7. The *communication_link_fail* and *communication_link_repair* are instantaneous as the time delays associated with these events are in the respective device models. As in the paper, fault clearing times are measured, and the simulated times of events are stored. This model stores the simulated time the circuit breaker opens and closes. This code is shown in listing 4.

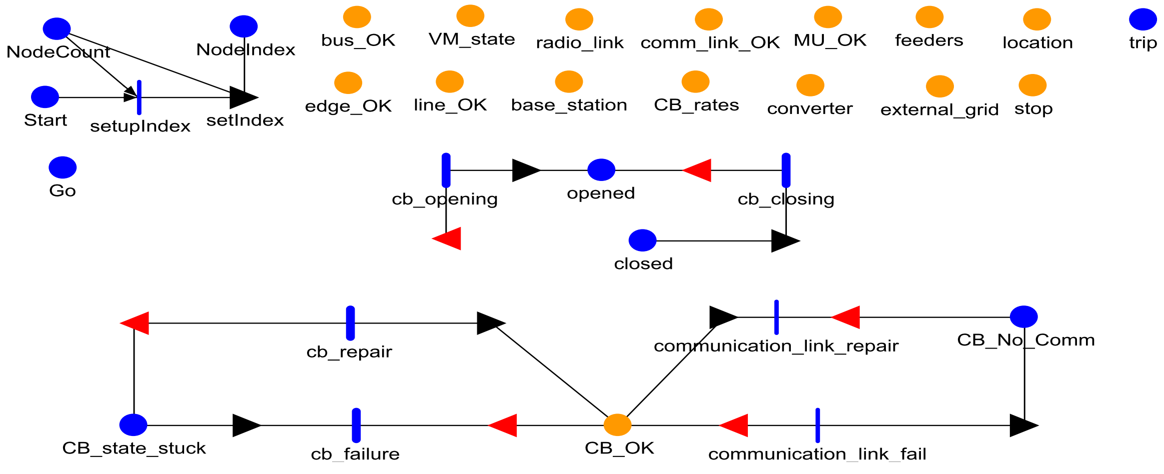

Figure 9: Circuit breaker model

```
using namespace std;
std::ofstream ondata;
ondata.open("C:\\output\\cb_open.dat", ios::out|ios::binary|std::
    ios_base::app);
ondata << LastActionTime << endl;
ondata.close();
```

Listing 4: C++ code to store simulated time of cb_opening activity in binary files.

Activity name	Type	Time distribution	Parameters
communication_link_fail	Instantaneous	-	-
communication_link_repair	Instantaneous	-	-
cb_repair	Timed	Exponential	2 hours
cb_failure	Timed	Exponential	3 636 hours
cb_opening	Timed	Investigated in paper	-
cb_closing	Timed	Investigated in paper	-

Table 7: Activities of circuit breaker model shown in Figure 9 and its properties

2.10 Lines model

Figure 10 shows the model of a power system line. A line in a power system remains connected until a protection device, a circuit breaker in this model, remains closed. The line is disconnected if the circuit breaker opens (*cb_opened*). When the line reconnects ((*cb_closed*), the line is reconnected. This model includes a possible extension of the model from the paper. In this version, a fault could occur at a line as well. The fault modeling in the SAN model is similar to the fault modeling at a bus. A fault may occur; in this case, a low-impedance fault would cause the line to disconnect quickly. In contrast, a high-impedance fault may be cleared naturally or by a maintenance crew. The properties of these activities are shown in Table 8. As the line fault scenario was not analyzed in the main paper, these parameters are not set. The *cb_opened* and *cb_closed* are instantaneous as the time delays associated with these events are in the respective device models.

Activity name	Type	Time distribution	Parameters
cb_opened	Instantaneous	-	-
cb_closed	Instantaneous	-	-
fault_occurs	Timed	Exponential	x hours
hif_cleared_natural	Timed	Exponential	x hours
maintenance_crew	Timed	Exponential	x hours

Table 8: Activities of line model shown in Figure 10 and its properties

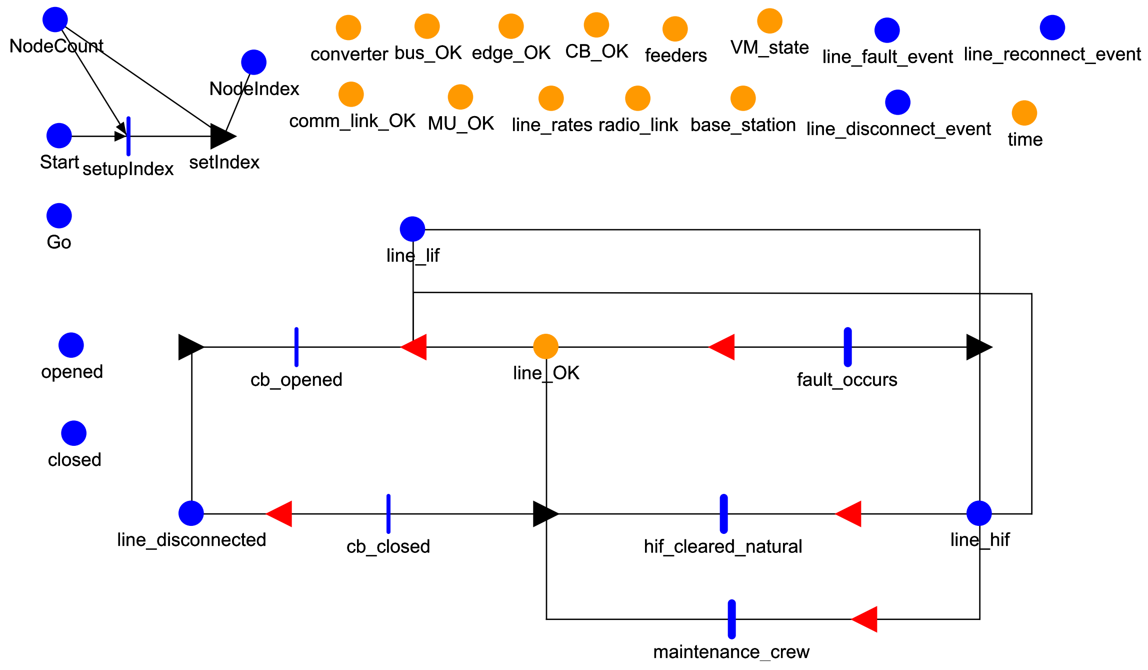

Figure 10: Line model

2.11 Converters model

Activity name	Type	Time distribution	Parameters
communication_link_fail	Instantaneous	-	-
communication_link_repair	Instantaneous	-	-
bus_failure	Instantaneous	-	-
bus_repair	Instantaneous	-	-
cb_opened	Instantaneous	-	-
cb_closed	Instantaneous	-	-
inject_fault_current	Instantaneous	-	-
communication_gateway_fail	Timed	Exponential	5 000 hours
communication_gateway_repair	Timed	Exponential	2 hours
PLC_fail	Timed	Exponential	7 000 hours
PLC_repair	Timed	Exponential	2 hours
DSP_fail	Timed	Exponential	5 000 hours
DSP_repair	Timed	Exponential	2 hours

Table 9: Activities of converters model shown in Figure 11 and its properties

The fault current is provided by the DRESs that are converter coupled into the system. Figure 11 shows the SAN model for a converter. The converter is capable of communication. However, the converter's role is primarily to provide the fault current. Once a fault occurs, the converter detects the fault due to the voltage change and

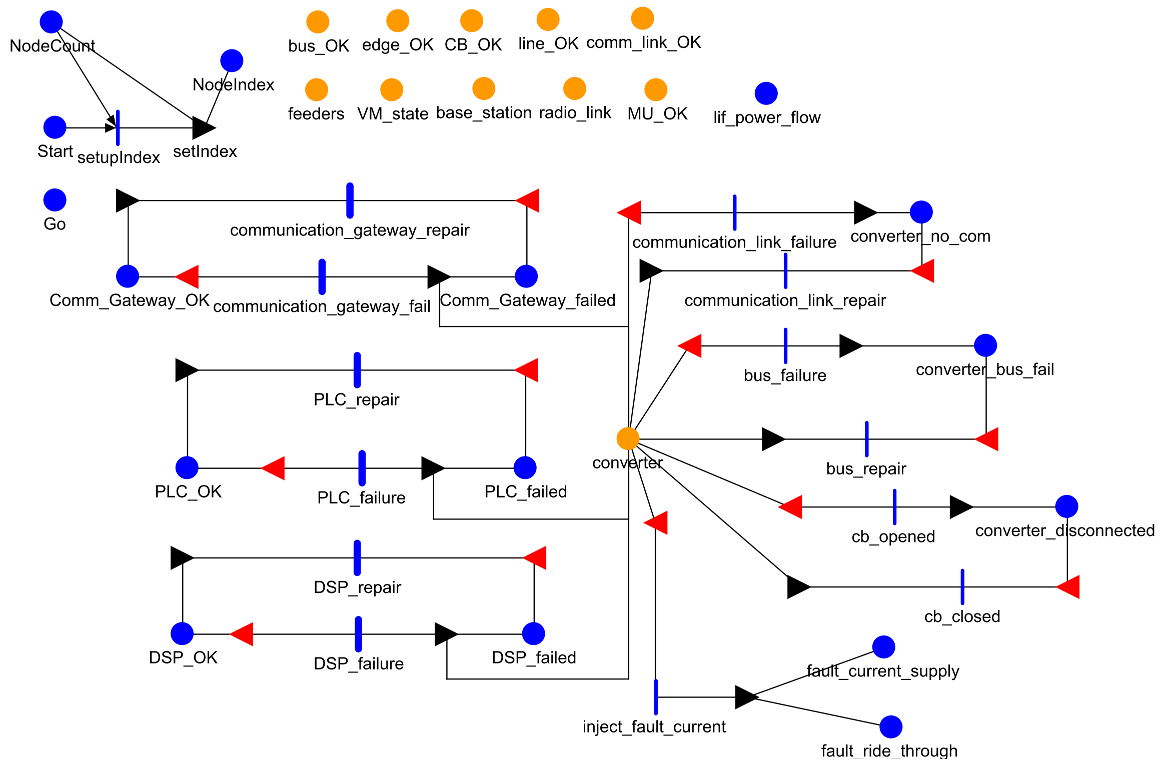

Figure 11: Converters model

supplies fault current (*inject_fault_current*). The fault current flow is calculated in a Python program outside SANs and updated in SANs. However, external events may impact the state of the converter. The converter may go offline if the connected bus fails (*bus_failure*), if the upstream circuit breaker disconnects (*cb_opened*), or if a relevant communication component fails. The properties of these activities are shown in Table 9. Except (*inject_fault_current*), the instantaneous activities depend on events in the respective device models.

```


print('Python function')
fault_location = fault_bus_index    #Store fault location

def powerflow(conv_state):

  net = nw.case33bw()   #Initialize 33 bus system
  net.ext_grid["s_sc_min_mva"] = 25
  net.ext_grid["s_sc_max_mva"] = 25
  net.ext_grid["rx_min"] = 0.2
  net.ext_grid["rx_max"] = 0.35
  net.ext_grid["r0x0_max"] = 0.4
  net.ext_grid["x0x_max"] = 1.0

  net.line["endtemp_degree"] = 20
  net.line["r0_ohm_per_km"] = 0.244
  net.line["x0_ohm_per_km"] = 0.336
  net.line["c0_nf_per_km"] = 2000

  # Place converter-coupled generators in grid; b is the bus index
    while p is the short circuit capacity of the converter-coupled
    generator in mva
  for b, p in [(10, 10), (20, 10), (23, 10), (28, 10)]:
    pp.create_sgen(net, b, sn_mva=p, k=1.3)

  #Update the state of each generator based on state variables
   from SANs
  for i in range(len(conv_state)):
    if( conv_state[i] == 1):
        net.sgen.loc[[i], ['in_service']] = True
    else:
      net.sgen.loc[[i], ['in_service']] = False

  # Perform short circuit calculation
  sc.calc_sc(net, fault="1ph", topology="radial", case="max", ith=
   True, ip=True, branch_results=True, return_all_currents=True,
   r_fault_ohm=r, x_fault_ohm=x)

  #Extract fault current flow for a fault at a specific bus and
   store in a file, which is fed back to SANs
  first = net.res_line_sc.loc[fault_location]
  output_df = pd.DataFrame({'current': round(first['ikss_ka'], 3)
   *1000})
  output_df.to_csv('D:\\output\\current.dat', index=False, header=
   False)


def myfunction():
  #Read converter state from a file created during SAN simulation
  file = open('D:\\output\\converter.dat', 'r+')
  lst = []
  for line in file:
    lst += [line.split()]
  converter = [float(x[0]) for x in lst]
  file.seek(0)
  file.truncate()
  conv_state = [int(item) for item in converter]

  powerflow(converter)
  return('powerflow complete')


print('State: ', myfunction())
```

Listing 5: Python code to calculate and store fault current flow.

Listing 5 shows the Python code required for short circuit calculation. Once a fault occurs, the SAN model calls this Python code, which returns the current flow for a fault with a certain impedance at a specific bus. The Python code writes the results to an external file, from which the SANs read the current values. Those current values are allocated to line and circuit breakers, based on which the protection devices may or may not operate.

3 Concluding remarks

This report provides complementary information and models to the paper "Quantifying the impact of communication delay on adaptive protection" . The report details the SAN models used for the analysis in the main paper. The details about the model, structure, time distribution assumptions and parameters are included. A brief description of each model is included to describe the modeling of the system and its relevance in the *adaptive* protection scenario.

References

[1] S. Chiaradonna, F. Di Giandomenico, and P. Lollini, "Definition, implementation and application of a model-based framework for analyzing interdependencies in electric power systems," *International Journal of Critical Infrastructure Protection*, vol. 4, no. 1, pp. 24–40, 2011.

[2] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. Doyle, W. Sanders, and P. Webster, "The mobius modeling tool," in *Proceedings 9th International Workshop on Petri Nets and Performance Models*, pp. 241–250, 2001.

[3] W. H. Sanders and J. F. Meyer, "Stochastic activity networks: formal definitions and concepts," in *School organized by the European Educational Forum*, pp. 315–343, Springer, 2000.

[4] T. A. Zerihun and B. E. Helvik, "Dependability of smart distribution grid protection using 5g," in *2019 3rd International Conference on Smart Grid and Smart Cities (ICSGSC)*, pp. 51–59, IEEE, 2019.

List of Figures

List of Tables