

TTM4110 Simulation Lab Report

October 9, 2017

1 Report

1.1 Introduction

In this lab, we will be simulating a smart grid system. The lab is divided into two parts, part one focuses mainly on performance, while part two focuses on availability and dependability. These aspects are important to consider when building a complex system such as a smart grid. Simulation can be an excellent tool in estimating unknown parameters given a set of known or otherwise estimated parameters.

Both tasks will be solved by discrete event simulation using SIMULA with the DEMOS-library.

The objective in task one is to estimate the delay, depending on how many devices are connected, and how many servers the ELHUB has to handle the load.

The objective in task two is to estimate the downtime of the system.

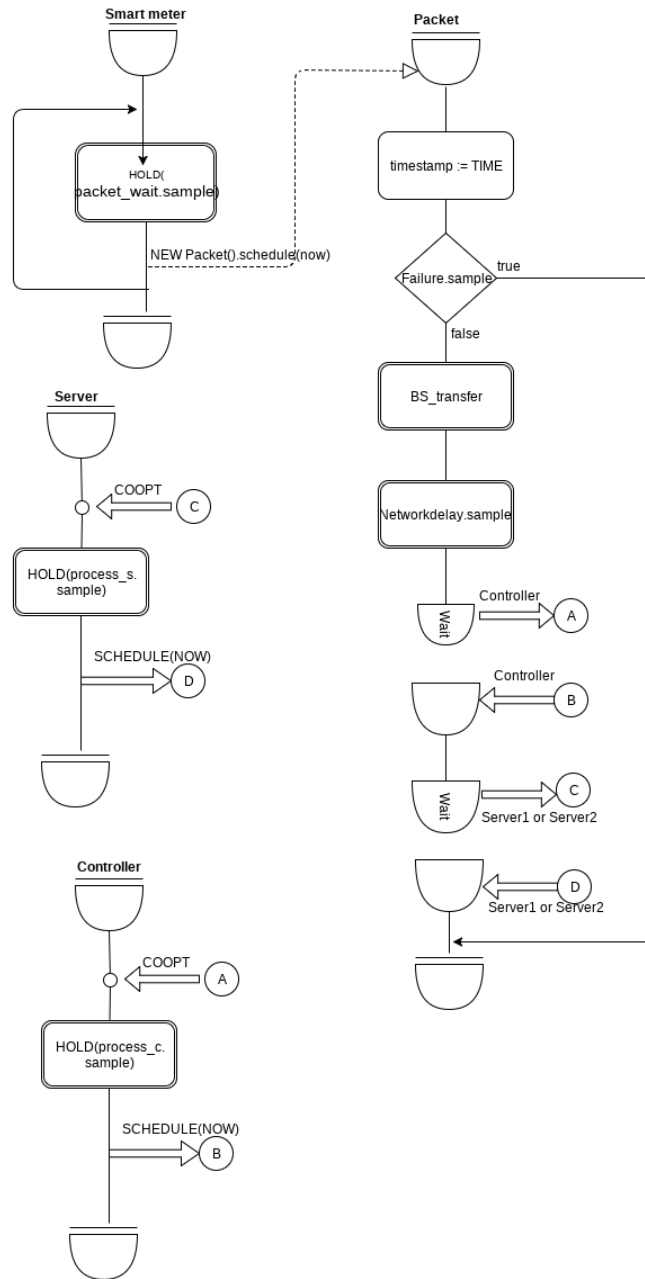
1.2 Answer to questions

1.2.1 Task 1

1) The system state is defined in the course book as the set of variables describing the system at time t . The system state thus consists of the following variables: - The individual timestamps and delay times of the packets traversing the system - The queues at the controller and servers - The probability of packet loss (between the smart meters and the base stations).

The events in this system are the transmission of packets between nodes in the network - from the smart meters to the base stations, from the base stations through the network to the controller and from the controller to one of the servers.

2)



3) See the appendix for the complete code. I have chosen not to include

the base stations or the network as entities in my models because they are constant, and do not change over time, as opposed to packets (timestamp) and the controller and servers (queues). The intensity has been chosen in an arbitrary way, using the following assumptions:

- A smart meter sends out updates two times per hour.
- Updates consists of 100kB, and each packet carries (on average) 1000 bytes of data.

This gives an intensity λ of $\frac{200kB}{1000B \cdot 60 \cdot 60} = 0.055555 \approx 0.060$ packets per second. Given that this is the result of a Poisson process, the time between packets is negatively exponentially distributed with parameter λ .

It is worth noting that all time is calculated in seconds in this implementation of the simulation..

Simulating with one smart meter, 50 base stations and two servers gives the following result.

```

*****
*
*                               R E P O R T                               *
*
*****

D I S T R I B U T I O N S
*****

TITLE      /  (RE)SET/  OBS/TYPE  /      A/      B/      SEED
MNO delay  0.000  94933  NEGEXP    50.000      33427485
Packet wait 0.000 100001 NEGEXP    6.000&-002  22276755
Process C   0.000  94933  NEGEXP   10000.000  46847980
Process S   0.000  94933  NEGEXP    500.000  43859043
BS failure  0.000 100000 DRAW     5.000&-002  64042082

C O U N T S
*****

TITLE      /  (RE)SET/  OBS
over 200ms 0.000  1124
Received   0.000  94933

T A L L I E S
*****

TITLE      /  (RE)SET/  OBS/  AVERAGE/EST.ST.DV/  STD.ERR./  MINIMUM/  MAXIMUM
delay      0.000  94933  0.1321.992&-0026.465&-005  0.110  0.350

B I N S
*****

TITLE      /  (RE)SET/  OBS/INIT/ MAX/ NOW/ AV. FREE/ AV. WAIT/QMAX
finished   0.000 100000  0*****  0 50073.5021.668&+006  1

W A I T   Q U E U E S
*****

TITLE      /  (RE)SET/  OBS/  QMAX/  QNOW/  Q  AVERAGE/ZEROS/  AV. WAIT
Server queue 0.000  94933  2    2    2.000  0    35.133
Server queue* 0.000  94933  1    0    0.000  94933  0.000

ControllerQ  0.000  94933  1    1    1.000  1    17.568
ControllerQ * 0.000  94933  1    0  3.071&-011  94932  5.395&-010

```

4) It appears that when we increase the numbers of smart meters, the delay increases. This makes sense, as more smart meters means more packets, meaning longer queue-times. The delay time seems to be about equal when looking at small numbers of smart meters, however when reaching larger numbers, the delay time increases at a slower pace with more servers than

with fewer. This is reasonable, as queues will fill up faster when fewer servers are handling packets.

Note that it appears that the packet loss increase, but this is probably not true. What is happening is that packets dropped are recorded earlier, while more succesful packets are stuck in queues. This is simply a side effect of having a set number of packets, and recording lost packets the instant they are lost.

Summarized:

$N_s = 2$:

$N_{sm} = 1000$:

| TITLE | OBS/ | AVERAGE/EST | ST.DV/ | STD.ERR./ | MINIMUM/ | MAXIMUM |
|-------|-------|-----------------|------------|-----------|----------|---------|
| delay | 94932 | 0.1321.991&−002 | 6.462&−005 | | 0.110 | 0.350 |

$N_{sm} = 10000$:

| TITLE | OBS/ | AVERAGE/EST | ST.DV/ | STD.ERR./ | MINIMUM/ | MAXIMUM |
|-------|-------|-----------------|------------|-----------|----------|---------|
| delay | 94929 | 0.1332.000&−002 | 6.490&−005 | | 0.110 | 0.349 |

$N_{sm} = 100000$:

| TITLE | OBS/ | AVERAGE/EST | ST.DV/ | STD.ERR./ | MINIMUM/ | MAXIMUM |
|-------|-------|-------------|------------------|-----------|----------|---------|
| delay | 77136 | 31.538 | 18.2076.555&−002 | | 0.112 | 63.272 |

$N_s = 4$:

$N_{sm} = 1000$:

| TITLE | OBS/ | AVERAGE/EST | ST.DV/ | STD.ERR./ | MINIMUM/ | MAXIMUM |
|-------|-------|-----------------|------------|-----------|----------|---------|
| delay | 94932 | 0.1321.991&−002 | 6.461&−005 | | 0.110 | 0.350 |

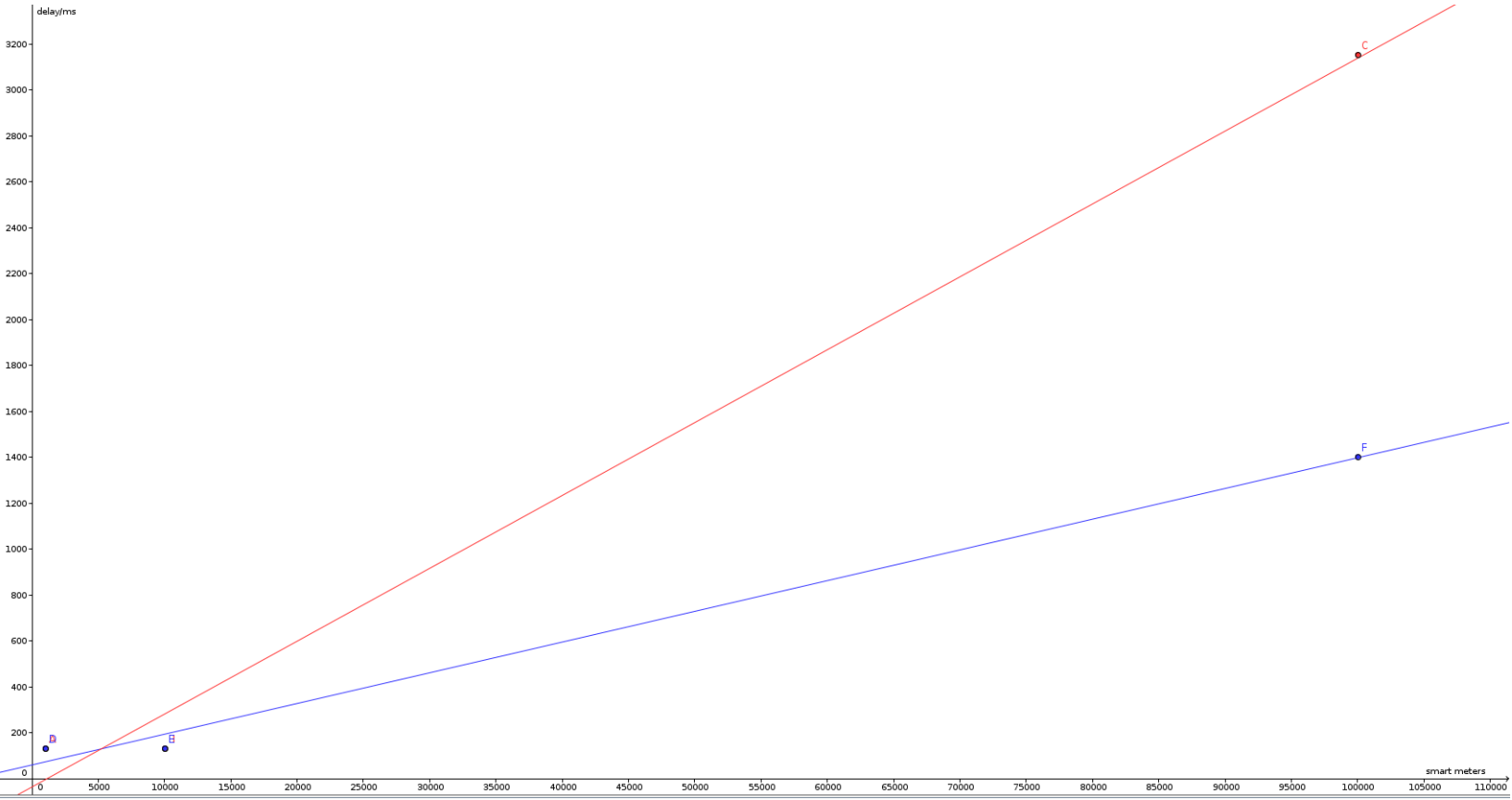
$N_{sm} = 10000$:

| TITLE | OBS/ | AVERAGE/EST | ST.DV/ | STD.ERR./ | MINIMUM/ | MAXIMUM |
|-------|-------|-----------------|------------|-----------|----------|---------|
| delay | 94929 | 0.1321.990&−002 | 6.460&−005 | | 0.110 | 0.348 |

$N_{sm} = 100000$:

| TITLE | OBS/ | AVERAGE/EST | ST.DV/ | STD.ERR./ | MINIMUM/ | MAXIMUM |
|-------|-------|-------------|-----------------|-----------|----------|---------|
| delay | 86986 | 14.037 | 8.0822.740&−002 | | 0.112 | 28.215 |

Put into a graph, red being for $N_s = 2$, blue being $N_s = 4$.



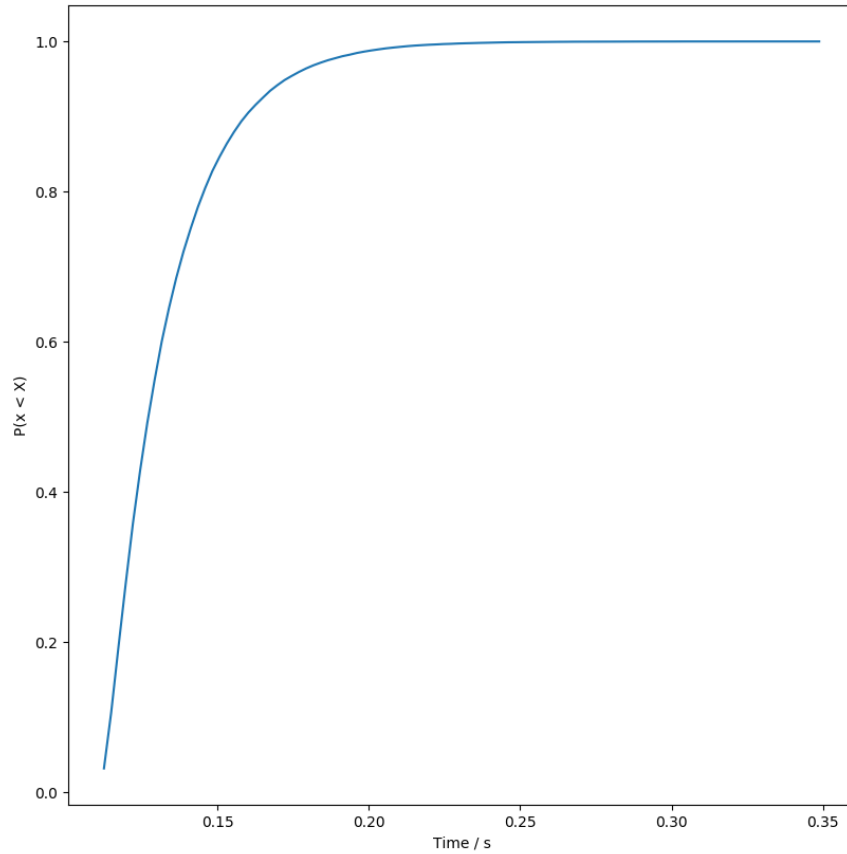
The program version with a logging function is attached in the appendix.

5)

If a packet is lost, it is not taken into account when calculating the delay, nor is it taken into account when plotting the CDF for the delay as the delay of a dropped packet is undefined. Parameters set for the generation of this data: $N_s = 2$, $N_s m = 10000$.

94929 packets sent, 5071 packets lost $\rightarrow P(\text{packet lost}) = 0.0534188709457$

CDF for end-to-end packet delay:



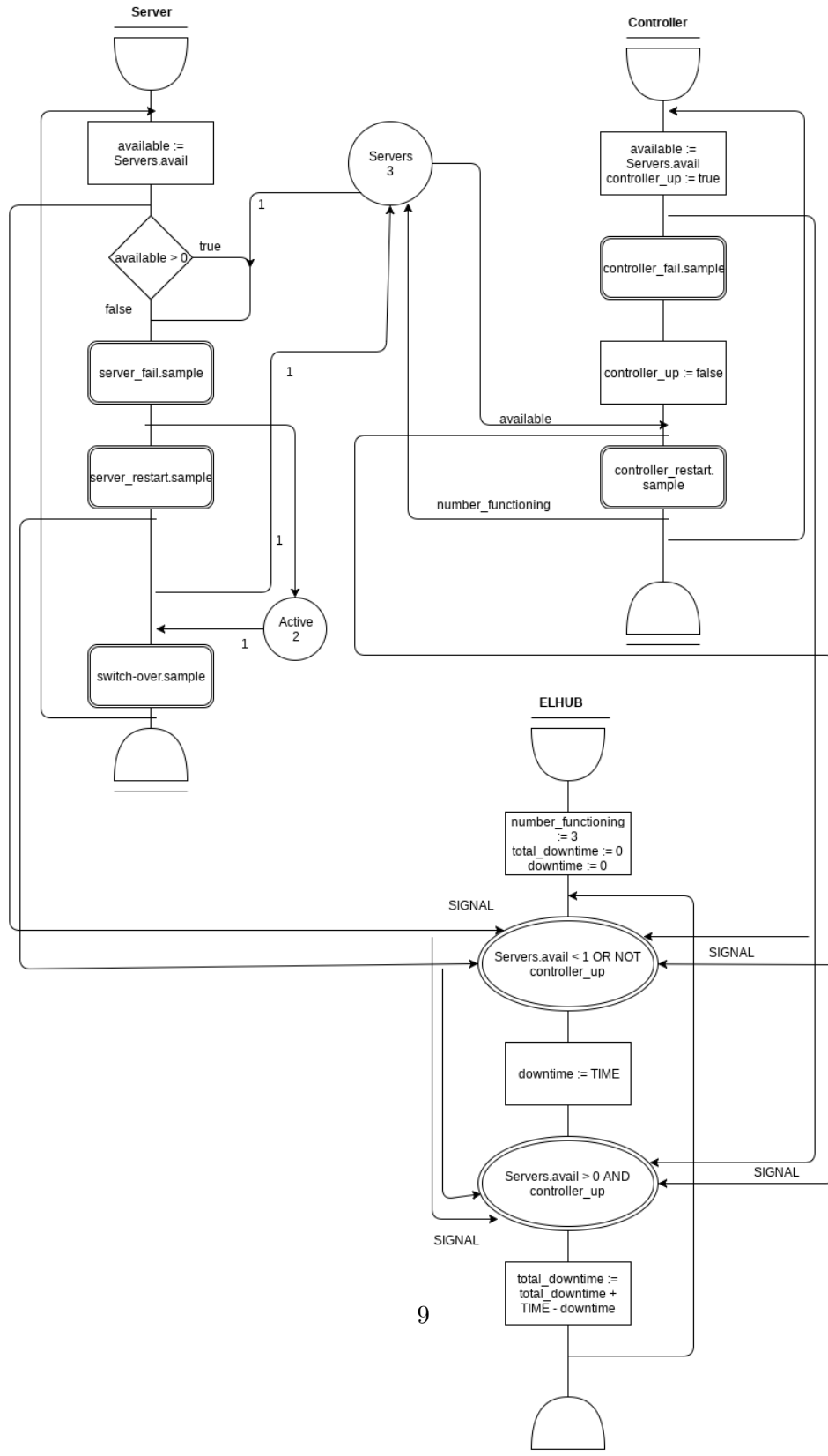
The source for the data and the python-script used to generate the plot is in the appendix

1.2.2 Task 2

1) The system states are now: - One or more Server(s) up, controller up -
 No servers up, controller up - One or more server(s) up, controller down -
 No servers up, controller down

The events being a server going up/down and the controller going up/down.
 The ELHUB is unavailable when the controller is down, or when no servers are up, or both.

2)



See the appendix for the code.

The implementation uses the resource-class to keep track of the servers, both available and passive. A server has two states; passive and active. I have made the following assumptions in this task: - When a server restarts, it is fixed, and after a restart it is instantly passive. - A server cannot fail while in passive mode

I think these are reasonable assumptions - servers may fail for no "reason" at all, but it is much more likely to fail when stressed. Therefore I assume the probability that an idle server failing is trivial. A server going into passive mode after a restart, and then into active mode if needed, and after a switch-over time is also a fair assumption.

3) Result of a run of the simulation with `sim_time` set to 10 million:

```

Total downtime:
    49517.1583969386
As a percentage:0.49517%

CLOCK TIME = 1.000&+007
*****
*
*
*
*
*
*****

D I S T R I B U T I O N S
*****

TITLE      / (RE)SET/  OBS/TYPE  /      A/      B/      SEED
serverfail w  0.000 199858 NEGEXP  1.000&-002      33427485
controllerfa  0.000 99413 NEGEXP  1.000&-002      22276755
restart s     0.000 199856 NEGEXP      1.000      46847980
restart c     0.000 99412 NEGEXP      2.000      43859043
passtoact    0.000 199856 NEGEXP      5.000      64042082

T A L L I E S
*****

TITLE      / (RE)SET/  OBS/  AVERAGE/EST.ST.DV/ STD.ERR./  MINIMUM/  MAXIMUM/
Downtime    0.000 99426    0.498    0.4991.582&-0032.429&-006    6.593

R E S O U R C E S
*****

TITLE      / (RE)SET/  OBS/ LIM/ MIN/ NOW/  % USAGE/ AV. WAIT/QMAX
AvailServers 0.000 298251  3  0  3    1.1581.689&-003    2
Active       0.000 199856  2  0  0    99.991    49.043    1

C O N D I T I O N   Q U E U E S
*****

TITLE      / (RE)SET/  OBS/ QMAX/ QNOW/ Q AVERAGE/ZEROS/  AV. WAIT
Hub failure 0.000 198852  1  1  1    1.000    2    50.288

```

4) I have solved this task using a simulation with and a simulation without switch-over time. For the one without switch-over time we get the following: Using the analytic expression we get:

$$U_s = \frac{\lambda_{fs}^3}{(\lambda_{fs} + \mu_{rs})^3}$$

$$U_c = \frac{\lambda_{fc}}{(\lambda_{fc} + \mu_{rc})}$$

$$U_c \approx 0.004975124$$

$$U_{hub} = U_s + U_c - U_s \cdot U_c$$

Inserting the numbers given into the last equation yields:

$$U_{hub} \approx 0.0049760 = 0.4976\% \text{ downtime}$$

Using the analytic expression for U_s when the switch-over time is non-zero gives:

$$U_s \approx \frac{\lambda_{fs}^3 (\lambda_{fs} (4\mu_{rs}^2 + 9\mu_{rs}\mu_{sw} + 3\mu_{sw}^3))}{3\mu_{rs}^2 (\mu_{rs} + \mu_{sw}) (\lambda_{fs} (8\mu_{rs} + 3\mu_{sw}) + \mu_{rs} (2\mu_{rs} + \mu_{sw}))}$$

$$U_s \approx 0.000005563$$

$$U_{hub} = U_s + U_c - U_s \cdot U_c$$

Inserting U_s and U_c into the equation for U_{hub} yields:

$$U_{hub} = 0.004980659939888693 \approx 0.004981 = 0.4981\%$$

In my simulation without switch-over time the ELHUB had 0.491% downtime, which is minusculely lower than the analytic expression gives. In the simulation with switch-over time, I achieved 0.4975% downtime. It appears that when the simulation time approaches infinity, the simulated downtime will approach the analytically correct downtime. This would mean that this simulation is a correct solution.

1.3 Summary

In this lab, we have looked at the performance and dependability of a smart grid. We have simulated the delay of packets traversing the system, and the uptime of parts of the system.

Appendix

1.3.1 Code from task 1-3

```

BEGIN
EXTERNAL CLASS demos = "../demos/demos.atr";
demos BEGIN

REAL Tw, Tn, Tc, Ts, p_r, intensity;
INTEGER num_BS, sim_n, num_s, num_meters, i, j, meters_per_bs, packet_n;
REF(Bin) finished_packets;
REF(RDist) MNO_delay, process_c, process_s, packet_wait;
REF(BDist) BS_failure;
REF(Tally) emp_delay;
REF(WaitQ) serverq, controllerq;
REF(Count) over, packets_recv;

ENTITY CLASS SM;
BEGIN
INTEGER i;

LOOP:
hold(packet_wait.sample);
NEW Packet(edit("pakke",i)).schedule(now);
i := i + 1;
packet_n := packet_n + 1;
REPEAT;
END;

ENTITY CLASS Packet;
BEGIN
LONG REAL ts;
ts := time;
if BS_failure.sample then BEGIN
finished_packets.give(1);
!outInt(packet_n, 8);
!outText(",");
!outInt(0,1);
!outimage;
END
ELSE BEGIN

```

```

hold(Tw);
hold(MNO_delay.sample);

controllerq.wait;
serverq.wait;

emp_delay.update(time - ts);

if (time - ts) > 0.200 then over.update(1);
finished_packets.give(1);
packets_recv.update(1);

! Uncomment to get a trace of packets;
! outInt(packet_n, 8);
! outText(",");
! outfix(time - ts, 7, 12);
! outimage;
END;
END;

ENTITY CLASS Controller(waitq_);
REF(WaitQ) waitq_;
BEGIN
REF(Packet) pakka;

LOOP:
pakka :- waitq_.coopt;
hold(process_c.sample);
pakka.schedule(now);
REPEAT;
END;

ENTITY CLASS Server(waitq_);
REF(WaitQ) waitq_;
BEGIN

REF(Packet) pakka;

LOOP:
pakka :- waitq_.coopt;

```

```

hold(process_s.sample);
pakka.schedule(now);
REPEAT;
END;

! Variable numbers;
Tw := 110 / 1000;
Tn := 20 / 1000;
Tc := 0.1 / 1000;
Ts := 2 / 1000;
p_r := 0.95;
num_BS := 50;
num_s := 2;
sim_n := 100000;
num_meters := 10000;
packet_n := 0;
meters_per_bs := num_meters / num_BS;
intensity := 0.06;

! Distributions;
MNO_delay := NEW NegExp("MNO delay", 1/Tn);
packet_wait := NEW NegExp("Packet wait", intensity);
process_c := NEW NegExp("Process C", 1/Tc);
process_s := NEW NegExp("Process S", 1/Ts);
BS_failure := NEW Draw("BS failure", 1 - p_r);

! Variable classes;
emp_delay := NEW Tally("delay");

finished_packets := NEW Bin("finished", 0);
serverq := NEW WaitQ("Server queue");
controllerq := NEW WaitQ("ControllerQ");

! Counts ;
over := NEW Count("over 200ms");
packets_recv := NEW Count("Received");

! Instantiate classes my dudes;
NEW Controller("C", controllerq).schedule(0.0);

```

```

for i:=1 step 1 until num_s do
NEW Server(edit("Server",i), serverq).schedule(0.0);

for i:=1 step 1 until num_meters do
NEW SM(edit("SM",i)).schedule(0.0);

finished_packets.take(sim_n);

END;
END;

```

1.3.2 Python code used to generate CDF-graph

Note: the file results.csv has to be in the same folder as the script, in order to run.

```

#!/usr/bin/python2.7
import numpy as np
import matplotlib.pyplot as plt

with open('results.csv','r') as f:
    lines = [x.split(',') for x in f.readlines()]

data = []
for line in lines:
    tmp = [x.strip() for x in line]
    data.append(tmp)
raw = []
for line in data:
    if line[-1] != 0 and line[-1] != '0':
        raw.append(float(line[-1]))

print "[*] Estimating probability of packetloss.."

lost = 0
for k in data:
    if k[1]=='0': lost+=1

print "{} packets sent, {} packets lost => P(packet lost) = {}".format(len

```



```

arr = np.array(raw)

h, X1 = np.histogram(arr, bins=100, normed=True)
dx = X1[1] - X1[0]
F1 = np.cumsum(h) * dx

plt.plot(X1[1:], F1)
plt.xlabel("Time / s")
plt.ylabel("P(x < X)")

plt.show()

```

1.3.3 Code from task 2-2 and 2-3

With switch-over time:

```

BEGIN
EXTERNAL CLASS DEMOS = "/home/simen/simula/demos/demos.atr";
DEMOS BEGIN

REAL tot_downtime, server_fail_intensity, controller_fail_intensity, restart_s, restart_c, p;
INTEGER num_server;
REF(RDIST) server_fail_wait, controller_fail_wait, restart_s, restart_c, p;
REF(RES) available_servers, active_;
BOOLEAN controller_up;
REF(CONDQ) hub_failure;
REF(TALLY) downtime_;

ENTITY CLASS SERVER(init_);
BOOLEAN init_;
BEGIN
INTEGER available;

if not init_ then goto passive;
active_.acquire(1);

active_:
LOOP:
available := available_servers.avail;

```

```

    if available > 0 THEN
    BEGIN
    HOLD(server_fail_wait.sample);
    available_servers.acquire(1);
    hub_failure.signal;

    num_server := num_server - 1;
    active_.release(1);

    HOLD(restart_s.sample);

    if (controller_up) THEN
    BEGIN
    available_servers.release(1);
    hub_failure.signal;
    END;
    num_server := num_server + 1;
    goto passive;
    END
    ELSE
    BEGIN

    HOLD(server_fail_wait.sample);
    num_server := num_server - 1;
    active_.release(1);

    HOLD(restart_s.sample);
    num_server := num_server + 1;
    goto passive;
    END;
    REPEAT;

    passive:
    active_.acquire(1);
    HOLD(pass_to_act.sample);
    goto active;
    END;

    ENTITY CLASS CONTROLLER;

```

```

BEGIN
INTEGER available;

LOOP:
controller_up := true;

HOLD(controller_fail_wait.sample);
controller_up := false;

available := available_servers.avail;
available_servers.acquire(available);
hub_failure.signal;

HOLD(restart_c.sample);

available_servers.release(num_server);
hub_failure.signal;
REPEAT;
END;

! ELHUB-class. Keeps track of downtime using a conditional queue;
ENTITY CLASS HUB;
BEGIN
INTEGER available;
LONG REAL downtime;

! The Hub is down when 0 servers are available, or the controller is down;
LOOP:
hub_failure.waituntil(available_servers.avail < 1 or not controller_up);
downtime := time;
hub_failure.waituntil(available_servers.avail > 0 and controller_up);
downtime := time - downtime;
tot_downtime := tot_downtime + downtime;
downtime_.update(downtime);
REPEAT;
END;

! System variables;
server_fail_intensity := 0.01;
controller_fail_intensity := 0.01;

```

```

restart_s_mean := 1;
restart_c_mean := 2;
pass_to_act_mean := 5;
controller_up := true;
num_server := 3;
tot_downtime := 0;

! DEMOS-objects and distributions;
downtime_ := NEW TALLY("Downtime");
hub_failure := NEW CONDQ("Hub failure");

available_servers := NEW RES("AvailServers", num_server);
active_ := NEW RES("Active", 2);

server_fail_wait := NEW NegExp("serverfail wait", server_fail_intensity);
controller_fail_wait := NEW NegExp("controllerfail wait", controller_fail_intensity);
restart_s := NEW NegExp("restart s", restart_s_mean);
restart_c := NEW NegExp("restart c", restart_c_mean);
pass_to_act := NEW NegExp("passtoact", pass_to_act_mean);

! Initialise objects;
NEW HUB("Hub").schedule(0.0);
NEW CONTROLLER("C1").schedule(0.0);
NEW SERVER("S1", true).schedule(0.0);
NEW SERVER("S2", true).schedule(0.0);
NEW SERVER("S3", false).schedule(0.0);

! Arbitrarily chosen number for keeping simulation going;
sim_time := 10000000;
HOLD(sim_time);

! Output recorded downtime;

outText("Total downtime:");
outimage;
outFix(tot_downtime, 10, 20);
outimage;
outText("As a percentage:");
tot_downtime := tot_downtime / sim_time * 100;

```

```

outFix(tot_downtime , 5, 7);
outText("%");
outimage;

```

```

END;
END;

```

Without switch-over time:

```

BEGIN
EXTERNAL CLASS DEMOS = "../demos/demos.atr";
DEMOS BEGIN

REAL tot_downtime , tmp , server_fail_intensity , controller_fail_intensity ,
INTEGER num_server ;
REF(RDIST) server_fail_wait , controller_fail_wait , restart_s , restart_c , p
REF(RES) available_servers ;
BOOLEAN controller_up ;
REF(CONDQ) hub_failure ;
REF(TALLY) downtime_ ;

ENTITY CLASS SERVER;
BEGIN
INTEGER available ;

LOOP:
available := available_servers.avail ;

if available > 0 THEN
BEGIN

HOLD(server_fail_wait.sample);
available_servers.acquire(1);
hub_failure.signal ;

num_server := num_server - 1 ;

HOLD(restart_s.sample);

```

```

    if (controller_up) THEN
    BEGIN
        available_servers.release(1);
        hub_failure.signal;
    END;

    num_server := num_server + 1;

    END
    ELSE
    BEGIN

    HOLD(server_fail_wait.sample);
    num_server := num_server - 1;

    HOLD(restart_s.sample);
    num_server := num_server + 1;

    END;
    REPEAT;
    END;

    ENTITY CLASS CONTROLLER;
    BEGIN
    INTEGER available;

    LOOP:
    controller_up := true;

    HOLD(controller_fail_wait.sample);
    controller_up := false;

    available := available_servers.avail;
    available_servers.acquire(available);
    hub_failure.signal;

    HOLD(restart_c.sample);

    available_servers.release(num_server);
    hub_failure.signal;

```

```

REPEAT;
END;

ENTITY CLASS HUB;
BEGIN
INTEGER available;
LONG REAL downtime;

! The Hub is down when 0 servers are available, or the controller is down;
LOOP:
hub_failure.waituntil(available_servers.avail < 1 or not controller_up);
downtime := time;
hub_failure.waituntil(available_servers.avail > 0 and controller_up);
downtime := time - downtime;
tot_downtime := tot_downtime + downtime;
downtime_.update(downtime);
REPEAT;
END;

! Variables for the system;
server_fail_intensity := 0.01;
controller_fail_intensity := 0.01;
restart_s_mean := 1;
restart_c_mean := 2;
pass_to_act_mean := 5;
controller_up := true;
num_server := 3;
tot_downtime := 0;
! DEMOS-objects and distributions;
downtime_ := NEW TALLY("Downtime");
hub_failure := NEW CONDQ("Hub failure");
available_servers := NEW RES("AvailServers", num_server);
server_fail_wait := NEW NegExp("serverfail wait", server_fail_intensity);
controller_fail_wait := NEW NegExp("controllerfail wait", controller_fail_i
restart_s := NEW NegExp("restart s", restart_s_mean);
restart_c := NEW NegExp("restart c", restart_c_mean);
pass_to_act := NEW NegExp("passtoact", pass_to_act_mean);

NEW HUB("Hub").schedule(0.0);
NEW CONTROLLER("C1").schedule(0.0);

```

```

NEW SERVER(" S1 ").schedule (0.0);
NEW SERVER(" S2 ").schedule (0.0);
NEW SERVER(" S3 ").schedule (0.0);

sim_time := 1000000;
HOLD(sim_time);

outText(" Total  downtime:");
outimage;
outFix(tot_downtime , 10, 15);
outimage;
outText(" As a  percentage:");
tot_downtime := tot_downtime / sim_time * 100;
outFix(tot_downtime , 5, 7);
outText("%");
outimage;

END;
END;

```