TTM4110 Simulation Lab Report

October 8, 2017

1 Report

In this lab, we will be simulating a smartgrid-system. Task one focuses on the performance, while task two focuses on the dependability and availability of the system.

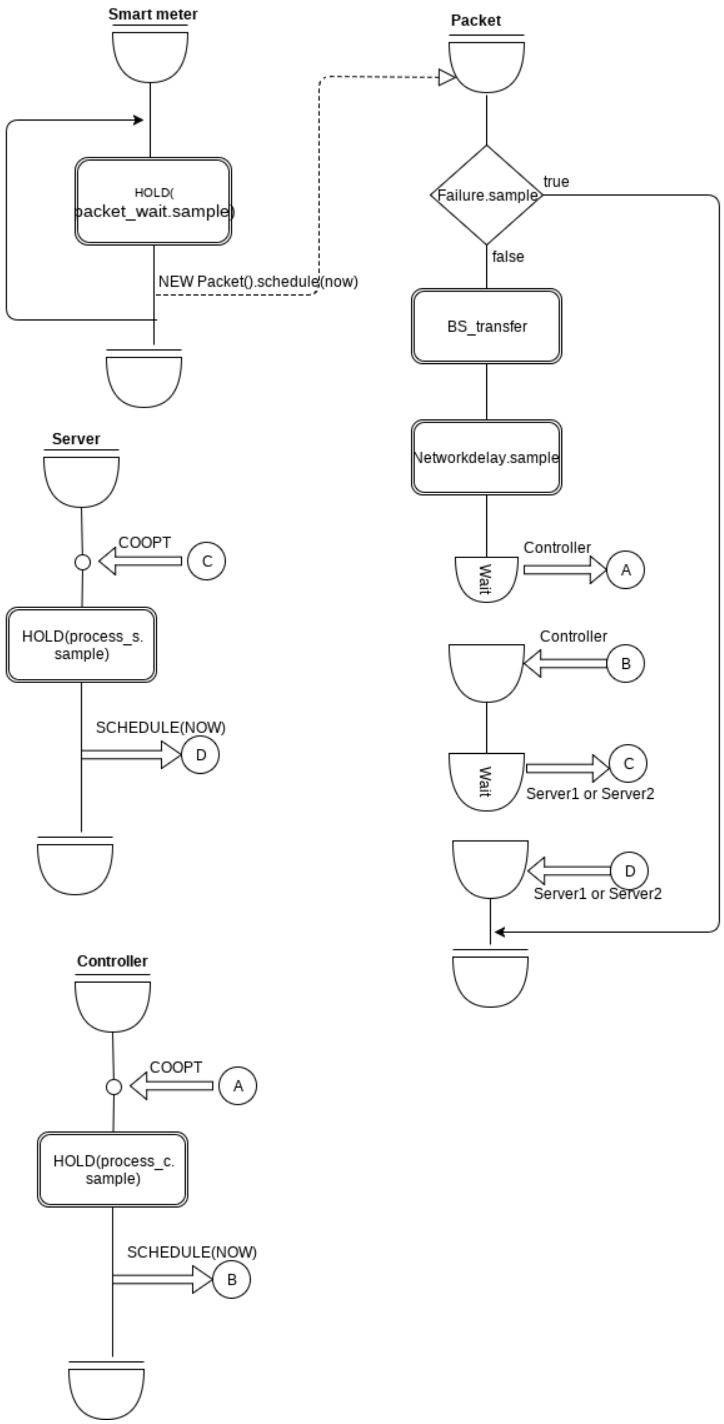
1.1 Introduction

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)


Smart meter
HOLD(
packet_wait.sample)
NEW Packet().schedule(now)
Packet
Failure.sample
true
false
BS_transfer
Networkdelay.sample
Wait
Controller
A
Controller
B
Wait
C
Server1 or Server2
D
Server1 or Server2
Server
COOPT
C
HOLD(process_s.
sample)
SCHEDULE(NOW)
D
Controller
COOPT
A
HOLD(process_c.
sample)
SCHEDULE(NOW)
B

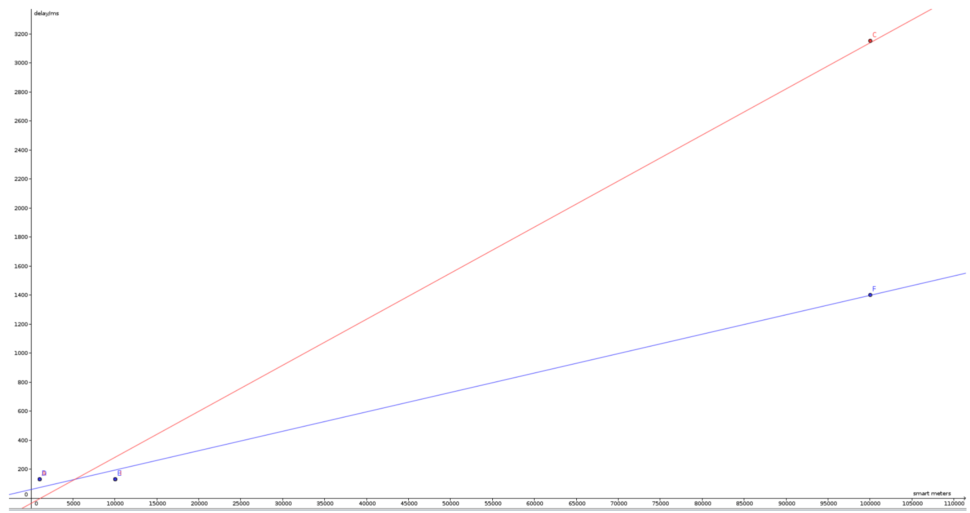
3) See the appendix for the complete code. I have chosen not to include the base stations or the network as entities in my models because they are constant, and do not change over time, as opposed to packets (timestamp) and the controller and servers (queues). The intensity has been chosen in an arbitrary way, using the following assumptions:
1: A smart meter sends out updates two times per hour.
2: Updates consists of 100kB, and each packet carries (on average) 1000 bytes of data.
This gives an intensity λ of $200kB/(1000B \cdot 60 \cdot 60) = 0.055555 \approx 0.060$ 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 the hundreds of thousands, the delay time increases at a slower pace with more servers than with fewer.
Summarized:

```
N_s = 2:

N_sm = 1000:
TITLE OBS/   AVERAGE/EST.ST.DV/ STD.ERR./   MINIMUM/   MAXIMUM
delay 94932     0.1321.991&-0026.462&-005       0.110       0.350

N_sm = 10000:
TITLE OBS/   AVERAGE/EST.ST.DV/ STD.ERR./   MINIMUM/   MAXIMUM
delay 94929     0.1332.000&-0026.490&-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&-0026.461&-005       0.110       0.350

N_sm = 10000:
TITLE OBS/   AVERAGE/EST.ST.DV/ STD.ERR./   MINIMUM/   MAXIMUM
delay 94929     0.1321.990&-0026.460&-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
```

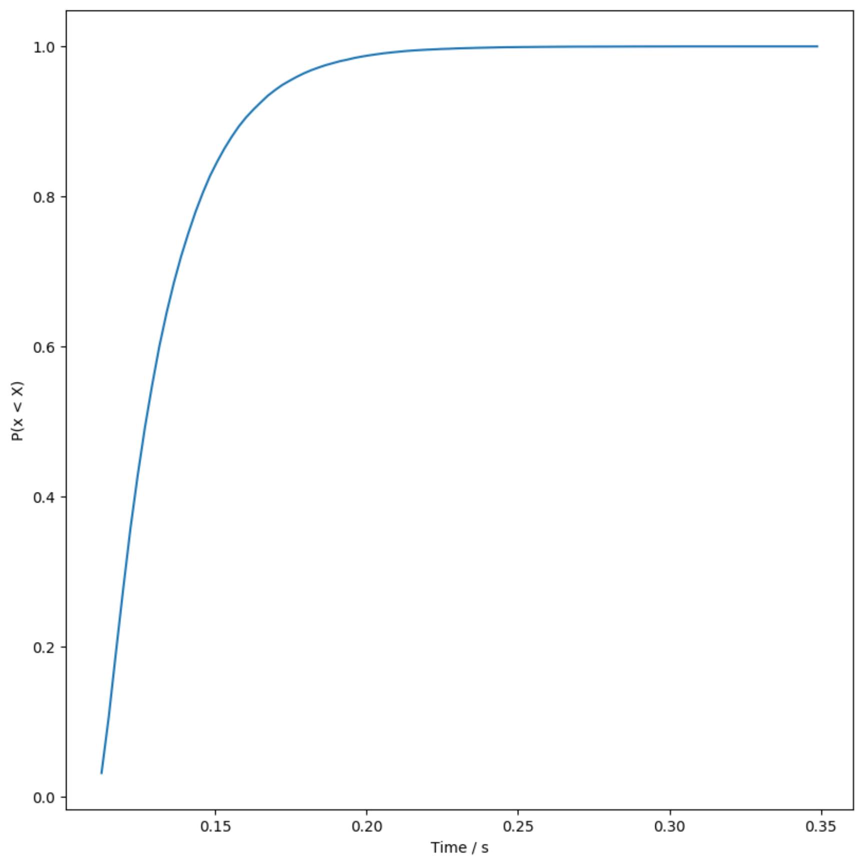
Put into a graph, red being for $N_s = 2$, blue being $N_s = 4$.
The program version with a logging function is attached in the appendix.
5)
If a packet is lost, it is not taken into account when calculating the delay,



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: $N_s = 2, N_s m = 10000$.

$$94929\, packets\ sent,\ 5071\, packets\ lost\ => \ P(packet\, lost)\ = 0.0534188709457$$

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


P(x < X)
1.0
0.8
0.6
0.4
0.2
0.0
0.15
0.20
0.25
0.30
0.35
Time / s

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) ¡activity diagram¿
See the appendix for the code.
3) Result of a run of the simulation:
4) 4) In my model I have chosen to not include switchover time, as I was told that it was not necessarry. As such, I will be assuming that the switchover time is 0.
Using the analytic expression we get:

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

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

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

Inserting the numbers given into the last equation yields:

$$U_{hub} = 0.0049760 \approx 0.498\% \; downtime$$

In my simulation the ELHUB had 0.490% downtime, which is minusculely lower than the analytic expression gives. It appears that when the simulation time approaches infinity, the simulated downtime will approach the analytic downtime.

1.3 Summary

Appendix

1.3.1 Code from task 1-3

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

```
Total downtime:
4906.2646404195
As a percentage:0.49063%

                       CLOCK TIME = 1.000&+006
**********************************************************************
*                                                                    *
*                             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
serverfail w        0.000  29860 NEGEXP   1.000&-002               33427485
controllerfa        0.000   9957 NEGEXP   1.000&-002               22276755
restart s           0.000  29857 NEGEXP        1.000               46847980
restart c           0.000   9956 NEGEXP        2.000               43859043
passtoact           0.000      0 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   9960     0.493     0.4934.943&-0033.145&-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  39650    3    0    3      1.4891.518&-003    2


                   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  19920     1     1      1.000     1     50.200
```

```
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

```
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("hehe").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;
```