CHALMERS
UNIVERSITY OF TECHNOLOGY

Data Minimization in Distributed Applications for More Privacy

Master's thesis in Algorithms, Languages and Logic

JAKOB BOMAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:NN

Data Minimization in Distributed Applications for More Privacy

JAKOB BOMAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Software Technology
Chalmers University of Technology
Gothenburg, Sweden 2016

Data Minimization in Distributed Applications for More Privacy
JAKOB BOMAN

Supervisor: Thibaud Antignac, Software Engineering
Examiner: Wolfgang Ahrendt, Software Engineering

Master's Thesis 2016:NN
Department of Computer Science and Engineering
Division of Software Technology
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2016

Data Minimization in Distributed Applications for More Privacy
JAKOB BOMAN
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The presence of connected devices in our environment is increasing. These devices form a network often called Internet of Things (or IoT for short), where everything from light-bulbs to thermostats can be controlled by an app or by another device. These services make a lot of that data available to the end user but also to malicious parties due to the devices leaking more data than intended or by bad design. This puts the end user at risk, violating its privacy and leaking sensitive data. One simple and obvious way to prevent leakages and misuses of personal data is to collect less of this data, a principle known as data minimization. However, this solution is rarely used in practice because of business models relying on personal data harvest on one hand and because of the difficulty to enforce it once it is defined what is actually needed to provide a service.

Keywords: *some keywords will be added here*

Acknowledgements

First of all I want to thank David Frisk for this outstanding LaTeX-template I used for my master thesis.
of course others will be thanked as well

Jakob Boman, Gothenburg, December 5, 2016

Contents

List of Figures

List of Tables

Listings

1

Introduction

1.1 Motivation

1.2 Aim

This thesis will investigate ways to improve privacy in a special kind of IoT (Internet of Things) devices known as Wireless Sensor Networks (WSN). WSN are networks of autonomous sensors and actuators. The goal to enhance privacy for this kind of devices will be addressed by relying on data minimization. This means the project sought to improve privacy in distributed networks by limiting the amount of personal data being processed.

To achieve this, the project sought to accomplish the following steps:

- Define Over-Collection and it's meaning with regards to this project.
- Construct an example of a Wireless Sensor Network that stops the collection when Over-Collection is occuring.

1.3 Limitations

The project will not consider faulty behaviors of a Wireless Sensor Network , meaning that the systems and algorithms will work under the assumption that all messages sent are received and all units are working as intended without malfunctions.

Only the result of the data collection will be analyzed in the sought outcome and if time complexity of the algorithm will be an issue for the project, it will not be considered as a failure should it arise. Some analysis will be done but it won't be a main focus to minifor the project.

Any model properties related to time (in the sense that they can be measured numerically) will be treated on an abstract level or be disregarded.

this section is borrowed from another MT where a quote was also included from Ben-Ari's book from 2008. I thought it was relevant for me aswell

1.4 Thesis Structure

saving for later when the thesis has shaped up

1.5 Background

This section should cover some background information to give the reader some background knowledge to what the project has been about that is required knowledge before moving forward.

1.5.1 Wireless Sensor Network (WSN)

A Wireless Sensor Network is recent improvement from the traditional sensor networks, made possible by advances in micro-electro-mechanical systems (MEMS) technology making sensor nodes that are smaller, multifunction and cheaper in comparison to previous sensors. Traditional sensors have two ways of being deployed; 1) They were positioned far away from the actual *phenomenon* (e.g. something known by sense perception) which required large sensors using complex techniques to distinguish the targets from surrounding noise. 2) Several sensors were deployed that only performed sensing and their communication topology had to be carefully engineered and they transmitted time series of the data to the central nodes which performed the communication.Wireless Sensor Networks on the other hand, is constructed by deploying a large number of sensor nodes close to the phenomenon and their position doesn't need to be engineered or predetermined.[**?**]

discuss other good things with WSNs but try to keep it relevant

1.5.2 Data Minimization

As defined by the EDPS (European Data Protection Supervisor); "The principle of data minimization means that a data controller should limit the collection of personal information to what is directly relevant and necessary to accomplish a specified purpose." [**?**]

discuss where and how the quote came to be, there's information on the link for that.

This covers two important aspects of data minimization, the first being that data should only be kept for as long as it is useful for an application and the second being that they should only collect "relevant" data. The latter is more interesting to the project, since the project's aim is to solve part of this problem.

2
Theory

This chapter provides an introduction into the theoretical elements used throughout the course of the project.

2.1 Formal Verification

The act of formal verification means to make use of mathematical techniques to make sure that a design upholds a defined functional correctness.[**?**] This means, that if we assume we have the following: a model of a design, a description of the environment where the design is supposed to operate in and some properties we wish the design to uphold. With this information, one may want to construct some input sequences, that are in the allowed in domain of the environment, that would violate the properties stated. A common practice for finding such patterns today are random simulations or directed tests. Formal verification allows for an extended approach to this, as it allows both to search for input sequences that violates the properties but also allows to mathematically prove that the stated properties holds when no input sequences exist.

2.2 Model Checking

A traditional approach to verifying concurrent systems is based on using extensive testing and simulation to find and eliminate unwanted occurrences from the system, but this way can easily miss crucial errors when the system that's being tested has a large number of possible states[**?**]. An alternative technique that was developed in the 1980's by Clarke et al. is called *temporal logic model checking* or "Model Checking".
Model Checking is an automated technique to verify finite state concurrent systems, by letting a tool verify that a model holds for certain properties. The process of applying Model Checking to a design is separated into several tasks; *modeling*, *specification* and *verification*.

> **Modeling:** First task is to translate a design into a format which is accepted by a model checking tool. This is either a compilation task or a task in abstracting certain aspects of the design to eliminate irrelevant or unimportant details, due to limitations on time an memory.

Specification: Second task is to state which properties the design is supposed to have. This is usually done using in a logical formalism, commonly in temporal logic, which can express assertions on a system evolving over time.

Verification: The final step is allowing the tool to verify the specification on the model. This will either be a positive result, meaning the model satisfies the properties, or a negative result where the properties aren't. A negative result can also be that the model's state space is too large to fit into a computer, which will require the model to be further abstracted to be verified.

2.2.1 Model Checking Workflow

show structure of model checking workflow

show a structure of a model checking workflow

2.2.2 State Space Explosion

mention the problem with state space explosion

will explain the problems with having a to precise model

2.3 Promela & SPIN

The model checking tool used for this project is called Simple Promela Interpreter (SPIN) and the language it accepts is called Promela, which is an acronym for Process Meta Language.

describe usages of SPIN

2.3.1 Operational Semantics of Promela

explain why this section is relevant

Definitions 7.1-7.5 defined in Spin reference manual (p.155-157) [**?**]

Definition 1. (Variable)
A *variable* is a tuple (*name*, *scope*, *domain*, *inival*, *curval*) where
name is an *identifier* that is unique within the given *scope*,
scope is either *global* or local to a specific *process*,
domain is a finite set of *integers*,
inival, the initial value, is an *integer* from the given *domain*, and
curval, the current value, is also an *integer* from the given *domain*.

Definition 2. (Message)
A *message* is an ordered set of *variables* (Def 1).

Definition 3. (Message Channel)
A *message channel* is a tuple (*ch_id*, *nslots*, *contents*) where
ch_id is a positive *integer* that uniquely identifies the channel,
nslots is an *integer*, and

contents is an ordered set of *messages*(Def 2) with maximum cardinality *nslots*.

Definition 4. (Process)
A *process* is a tuple (*pid*, *lvars*, *lstates*, *initial*, *curstate*, *trans*) where
pid is a positive *integer* that uniquely identifies the process,
lvars is a finite set of local *variables* (Def 1), each with a *scope*
lstates is a finite set of *integers*,
intial and *curstate* are elements of set *lstates*, and
trans is a finite set of *transitions*(Def 5) on lstates.

Definition 5. (Transition)
A *transition* in process P is defined by a tuple (*tr_id*, *source*, *target*, *cond*, *effect*, *prty*, *rv*) where
tr_id is a non-negative *integer*,
source and *target* are elements from *P.lstates* (i.e. *integers*),
cond is a boolean condition from the global *system state*(Def 6),
effect is a function that modifies the global *system state*(Def 6),
prty and *rv* are *integers*.

Definition 6. (System State)
A global *system state* is a tuple of the form (*gvars*, *procs*, *chans*, *exclusive*, *handshake*, *timeout*, *else*, *stutter*) where
gvars is a finite set of *variables* (Def 1) with *global* scope,
procs is a finite set of *processes* (Def 4),
chans is a finite set of *message channels* (Def 3),
exclusive and *handshake* are *integers*,
timeout, *else* and *stutter* are *booleans*.

2.4 Related Work

2.4.1 Smart City

some background into their approach[**?**]
some comparison to this project

2.4.2 Privacy Enhancing Technologies (PET)

compare this to your work

3
Modeling & Specification

3.1 Definitions

3.1.1 Basic WSN

A basic Wireless Sensor Network was defined as a starting point for the models. It consisted of a set of collection nodes (referred to as "nodes"), a central server (referred to as "the server") and finally an environment (the observed source). An illustration of this can be seen in Figure 3.1. This example became the initial working example for the project and helped shape the first models.

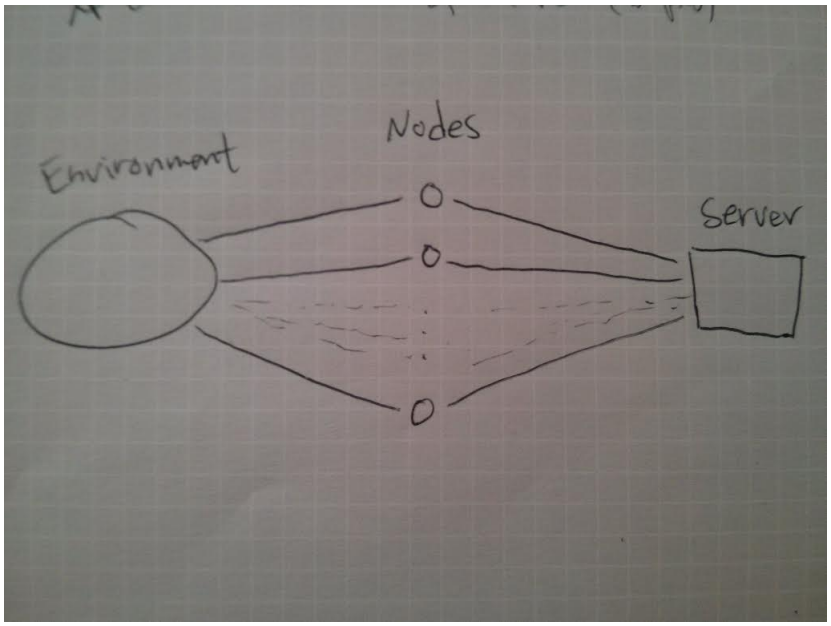

Figure 3.1: An illustration of a Wireless Sensor Network

In this setup, the environment is considered an entity (same as a node or a server). This simplification was made so the environment would be easier to develop.
We will henceforth refer to entities in the system as **actors** in the system.

textflow in report might be weird now

3.1.2 Actors

To describe the interaction between two actors in the system, behavior models were used (e.g. Figure 3.4). Where the name of each actor is shown in the boxes at the

top. The message channel used between them is shown as the arrows, where the arrow-head points to the actor receiving the message and the contents of the message is referenced above it. Finally the ordering of the messages are in a ascending order from the top, meaning the first message sent is shown furthest to the top of the figure.

3.1.3 Decisions

A decision, or a decision procedure is an algorithm that terminates with a yes or no answer given a decision problem.[**?**] *more text regarding decision processes will be added here*

Definition 7. (Decision Process)
A process is called a decision process for T if it is sound and complete with respect to every formula of T.

Definition 7 also requires some more definitions, but this is an important one so added it for now.

3.1.4 Over-Collection

will explain the meaning over-collection for this project

Definition 8. (Collecting)
A process P collects a data point d in a state s if after leaving the state then $d \in \{P_{c'} \setminus P_c\}$.

Definition 9. (To Function)
define what it means for a process to function

Definition 10. (Over-Collection)
Over-collection is the state when a process collects more data than it requires to function.

Formal Definition: Let a process P be able to collect data entries and to evaluate boolean expressions.
$P_{eval} : D \rightarrow \textbf{Bool}$
Let a service $S(x, y, ...)$ be a boolean expression depending on variables $x, y, ...$
We say the process P dedicated to the service S, noted $< P, S >$, over-collects data if and only if P collects any data concerning one of the variables appearing in S after S has been evaluated to be true.

3.2 Modeling

As a starting point for defining the models, first different architectural choices were considered. This was done to help define different cases of Wireless Sensor Network that could use decisions.

3.2.1 Variations

The different variations considered were:

- Centralized or Decentralized decision making
- Conjunctive or Disjunctive decision analysis
- Centralized or Distributed communication

3.2.1.1 Decision Making

The first choice reflected how much the sensor nodes would analyze the data. Since nodes can have a processing unit, they could potentially analyze the collected data and make a decision on their own.

3.2.1.2 Decision Analysis

The second choice reflected how the decision were processed, if the data from a single data point could trigger a decision or if the decision considered data from multiple entries in it's evaluation.

3.2.1.3 Communication

The final variation reflected how the network communicated, it was considered centralized if all communication were sent through a central unit, such as a server, or if nodes were allowed to communicate independently to each other.

3.2.2 Initialization

An initial model was made for one variation, a model with a centralized decision making, disjunctive decision analysis and centralized communication. This was made as a starting point for other variations and also help define the properties sought of the network.
Furthermore, the project defined the individual components of the Wireless Sensor Network to three actors: *Node*, *Server* and *Environment*.

3.2.3 Server Actor

The server is an actor receiving messages from nodes and storing it for later usage. A server's behavior will vary depending on the structure of the system. If the decision is taken centrally the server will be the one checking for over-collection, otherwise it will be a node. Also if the communication is managed through the server, if the nodes doesn't communicate with each other, the server will act as a repeater for the decision.
In Figure 3.2 is the behavior for a system where server makes the decision and nodes doesn't communicate with each other. First, the node sends some data, the server checks for over-collection and replies accordingly. The response will either be a "stop" signaling that over-collection has occurred and the node should stop collecting or it tells it that it can continue collecting.

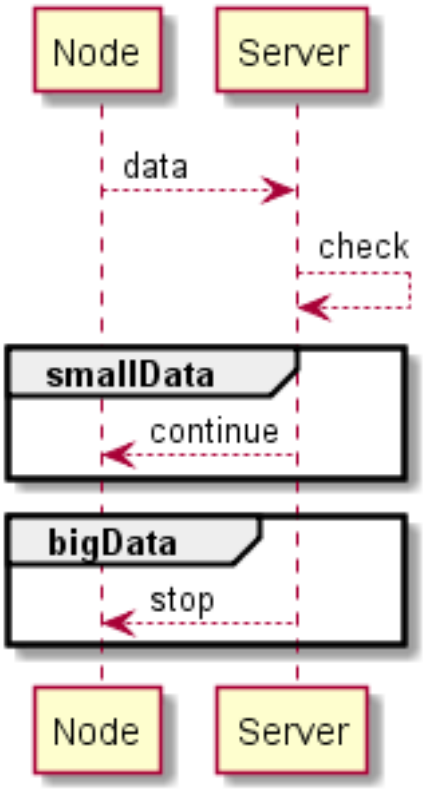

Figure 3.2: Behavior Model between Server and the Node

In addition to the behavior model, an Finite State Automaton(FSA) were designed for the server (Figure 3.3). The initial state being `Idle_a`, which the server will stay in until some `data` is received. "Data" being either `bigData` or `smallData`; `data` $\in$ {`smallData`, `bigData`}. The data is received in `Idle_a` and `Idle_s` and then checked in `Answ.` and `Hold`, meaning only the outgoing transitions from `Idle_a` and `Idle_s` is incoming data, the other are calculated internally. This also means the server will loop indefinetly between `Answ` and `Idle_a` as long as only smallData is received. When `bigData` is received the server will enter the `Idle_s-Hold` loop instead, which denotes the states where the server is requesting the nodes to stop collecting more data.

3.2.4 Environment Actor

The process for the environment actor had two steps:

1. Generate random data
2. Serve random data to a requesting node

As mentioned before, the first step is not intuitive for an environment since the observed source isn't randomly varying, but for modeling purposes this is a simplification made to reduce the complexity of the model. In Figure 3.4 the behavior between a node and the the environment is described.
The corresponding FSA for the environment is seen in Figure **??**. The environment will stay in the initial state `W` (short for "Waiting"), until a node `meter` it. Then the

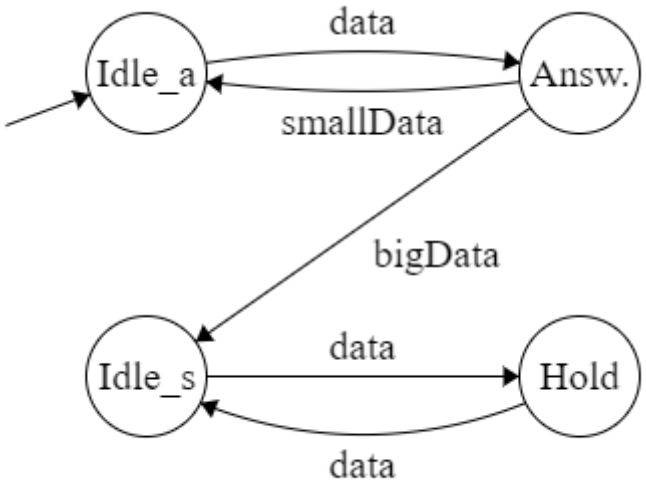

Figure 3.3: Finite State Automata for the Server Actor

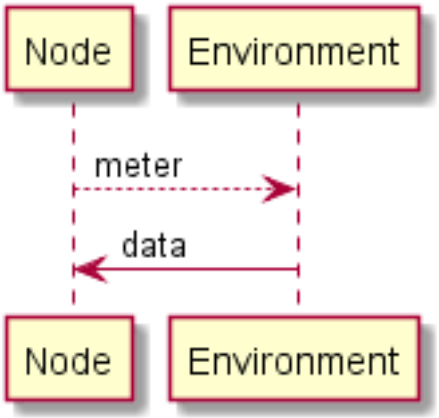

Figure 3.4: Behavior Model for the Environment

data is "generated" in `G` (short for "generate data") and served back to the node.

3.2.5 Node Actor

As seen in the behavior model for the node actor (Figure **??**), it captures the majority of a typical scenario for the entire system. That is intuitive since the node communicates with both of the other actors of the system and is a intermediatepart of the system. The scenario is when a node collects data, that doesn't cause a system-change, and forwards it to the server.

is this the right word?

The alternative behavior for system is described in Figure **??** instead. There the data collected causes the server to make the decision that the node should stop collecting.

This behavior can be described in a FSA, as seen in Figure **??**.The node meters data from the environment and passes it forward to the system. There it waits (noted by the state `Wait`) for a response before returning to the `Idle` state.

argue why I've kept the states to a minimum here?

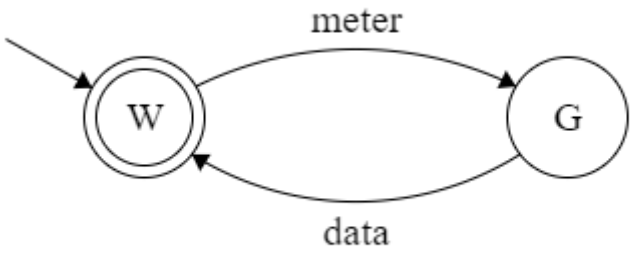

Figure 3.5: FSA for the Environment Actor

3.3 Specification

This section presents the properties used to verify the system. The process of defining the properties were an iterative approach and several versions were considered, this section only covers the final properties were the result of this process.

3.3.1 Properties

The properties defined on the network were formulated using *Linear Time Logic* (LTL). This decision came from the fact that LTL were native to SPIN and the models were abstracted to only focus on the relevant parts to the project, LTL could provide a simple and direct specification to that problem.

3.3.1.1 Correctness

The primary property sought of the system was that it was working as intended. This was formulated as a safety correctness property, to ensure that when decision had been taken, the system respond to the by changing its' behavior.

Definition 11. Safety Correctness
When the stop-decision is taken, the system should stop collecting.

LTL: $\Box(O \rightarrow (\Diamond D))$

Where **O** and **D** corresponds to the state where the stop-decision is taken and the states where collection is stopped respectively. This captures the sought system change; whenever the system reaches the state O, eventually it will reach state D. An immediate change is not required, therefore the timingis relaxed by the eventually-operator.

is this the proper word?

3.3.1.2 Liveness

The second property was intuitive for the system since the initial models were constructed in such a way that when data is sent to the server, the first thing the server does it analyze it and respond accordingly depending on what data was sent.

mention it was verified by design?

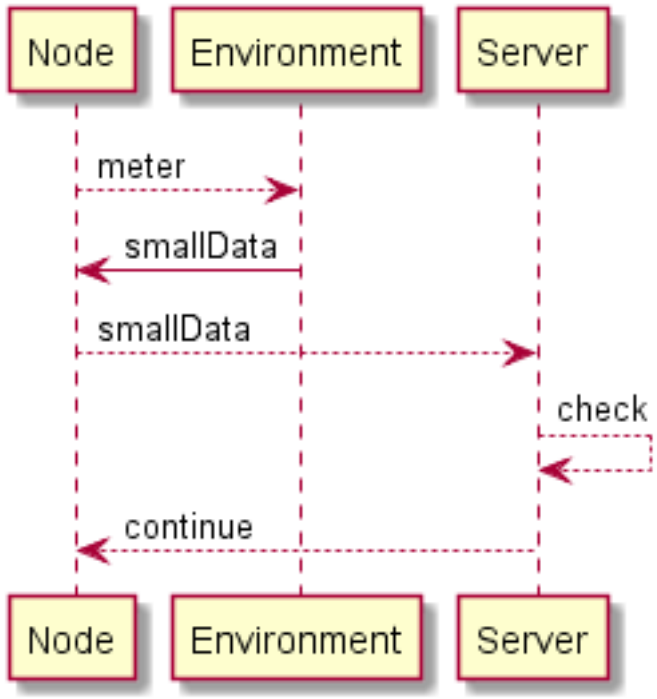

Figure 3.6: Behavior Model for a Node

Definition 12. Liveness (sending)
Eventually a node sends it's data to the server.

LTL: ◊ Node_Send

Where Node_Send denotes the state where the node sends the data to the server.

Definition 13. Liveness (replying)
If a node sends data to the server, eventually the server replies to the node.

LTL: □(Node_Send → ◊ Server_Reply)

server_reply doesnt exist atm.

Where Node_Send means the same as previously and Server_Reply denotes the state where the server responds to the node.

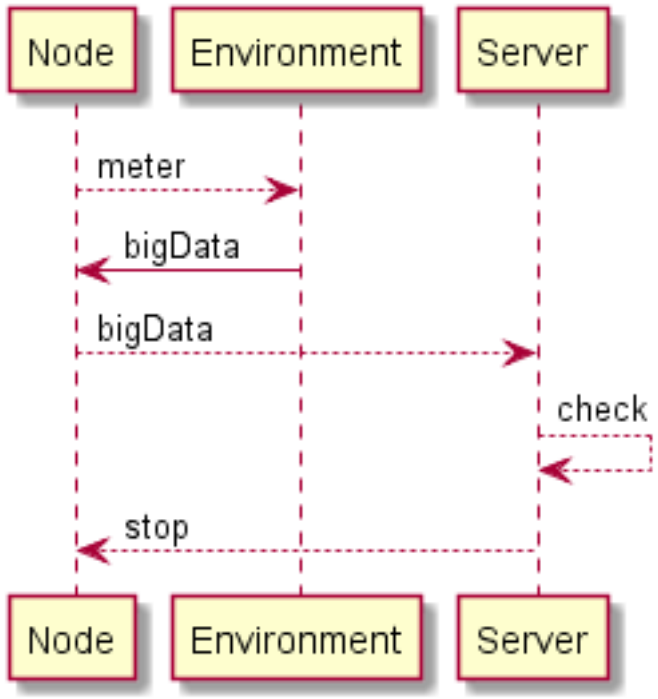

Figure 3.7: Behavior Model for a Node over-collecting

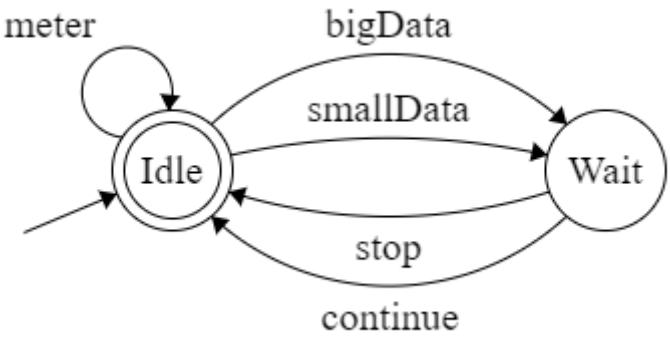

Figure 3.8: FSA for the Node Actor

4
Design

introduction-text: I seek to use SPIN/Promela for my models and first I need to justify why I did so and compare it to other tools...

Analysis of UPPAAL vs. TLC for verifying the WS-AT Protocol. [**?**]

Survey regarding the NuSVM "symbolic" model checker.[**?**]

Tool	**SPIN**	**UPPAAL**	**NuSVM**
specification language	promela	timed automata network with shared variables	...
necessary user's background	programming	programming	...
expressiveness of spec. language	...	restricted, communicating state machines, C-like (but finite) data structures, inductive approach	...
model checker characteristics	...	verifies the full specification language (with time)	...
modeling / verification speed	...	slower modeling, faster verification	...
verification of time/cost features	...	straightforward modeling and state-of-the-art verification support	...
parameterized reasoning	...	...	...

Table 4.1: Comparison between the model checkers SPIN, UPPAAL and NuSVM. [**?**]

4.1 Modeling in Promela

The system consists of message channels between three different classes of procedures **Node**, **Server** and **Environment**. The node allows a dynamic number of instances to run at start-up and it's set by a predefined macro named `NUM_NODES`. As an abstraction, the project considered the data sent in the network as a set of two possibilities. Either the sent data causes a system change, the system realizes it should stop collecting to prevent over-collection, or it doesn't and it continues as before. This were noted as `bigData` and `smallData`, where `bigData` causes the system change and `smallData` doesn't.

The system procedures communicate using shared communication channels, `envChan` for the communication between the `Nodes` and `Environment` and `servChan` for the communication between the `Server` and the `Nodes`.

4.1.1 Environment

The environment is an abstraction made to simplify the work. It's considered to be a shared resource between the nodes where each node can individually meter the environment and then communicate it to the server. To achieve this the environment is constructed as an atomic statement so when a node puts up a request on the channel it's instantly handled before any other statement is executed. To handle the randomness between the outcomes (so both types of the data can be metered) an `if`-statement without guards is used.

argue translation?

Listing 4.1: Environment code

```
1  active proctype Env() {
2
3    Idle:
4      if
5      :: atomic {
6        envChan ? meter ->
7          if
8          :: envChan ! bigData;
9          :: envChan ! smallData;
10         fi;
11         goto Idle;
12     }
13     fi;
14 }
```

4.1.2 Server

The server consists of two primary states, the first being the initial state, noted below as `Answering`, where data is assumed to be collected and the second state, noted as `Stopping` where the system starts requesting that the nodes stop collecting to prevent over-collection. The states beginning with `"Idle_"` are just looping to check if a node is sending data.

Listing 4.2: Server code

```
1  active proctype Server() {
2
3  chan active_chan;
4  int i=0;
5
6  Idle_Answering:
7          if
8          :: nempty(servChan[i]) ->
9              active_chan = servChan[i];
10             goto Answering;
11         :: empty(servChan[i]) ->
12             i=(i+1)%NUM_NODES;
13             goto Idle_Answering;
14         fi;
15
16 Idle_Stopping:
17         if
18         :: nempty(servChan[i]) ->
19             active_chan = servChan[i];
20             goto Stopping;
21         :: empty(servChan[i]) ->
22             i=(i+1)%NUM_NODES;
23             goto Idle_Stopping;
24         fi;
25
26 Answering:
27         if
28         :: active_chan ? smallData ->
29             active_chan ! continue;
30             goto Idle_Answering;
31         :: active_chan ? bigData ->
32             active_chan ! stop;
33             goto Idle_Stopping;
34         fi;
35
36 Stopping:
37         if
38         :: active_chan ? smallData ->
39             active_chan ! stop;
40             goto Idle_Stopping;
41         :: active_chan ? bigData ->
42             active_chan ! stop;
43             goto Idle_Stopping;
44         fi;
45 }
```

4.1.3 Node

The node is initialized with a channel to communicate to the server with. It starts by attempting to meter the environment, then communicates it to the server and proceeds into `Waiting` to wait for an answer.

Listing 4.3: Server code

```
1  proctype Node(chan out) {
2  Idle:
3          envChan ! meter;
4          if
5          :: envChan ? bigData ->
6             out ! bigData;
7             goto Waiting;
8          :: envChan ? smallData ->
9             out ! smallData;
10            goto Waiting;
11         fi;
12 Waiting:
13         atomic {
14           if
15           :: out ? continue -> goto Idle;
16           :: out ? stop ->
17           fi;
18         }
19 DoneColl: // node will shutdown here.
20 }
```

4.1.4 Translation

this section will generally discuss the translation and the new errors that could occur, but are handled by design.
A node ...

4.2 Verification

Discuss the results from the verification, present modifications done to fix any errors that might occur (perhaps show a interesting case of this).

5
Implementation

Discuss different approaches to verify the implementation and argue for the one I decided on.

5.1 Code Generation

Decide on a tool, discuss it

5.2 Satisfaction

Explain how I used my approach to verify the implementation

5.3 Analysis

Analyze the result of the generation and discuss limitations on the current models. E.g. redundancy from the generation or weaknesses in terms of security

6
Discussion

Discuss different choices made and why they were made.

- discuss why I used formal verification & model checking instead of traditional approaches
- discuss why I didn't build all models from the start
- discuss why I made simplifications to the initial models
- discuss why I chosed to use SPIN/Promela as a tool

7
Conclusion

Conclude the results of the report, did it go as expected? What progress did you make and what didn't you achieve that you had hoped? Did you reach the aim stated and did you keep yourself in the scope & limitations?

8

Ethics

this section will discuss ethical aspects and what ethical impacts it can have.