Final thesis

Automatic fault detection and localization in IP networks
Active probing from a single node perspective

by

Christopher Pettersson

LIU-IDA/LITH-EX-A--15/022--SE

2015-06-20
Final Thesis

Automatic fault detection and localization in IP networks
Active probing from a single node perspective

by

Christopher Pettersson

LIU-IDALITH-EX-A--15/022--SE
2015-06-20

Supervisor:        Johannes Schmidt, LiU IDA
                  Nicklas Weman, Ericsson
Examiner:          Ola Leifler, LiU IDA
Abstract

Fault management is a continuously demanded function in any kind of network management. Commonly it is carried out by a centralized entity on the network which correlates collected information into likely diagnoses of the current system states. We survey the use of active-on-demand-measurement, often called active probes, together with passive readings from the perspective of one single node. The solution is confined to the node and is isolated from the surrounding environment. The utility for this approach, to fault diagnosis, was found to depend on the environment in which the specific node was located within. Conclusively, the less environment knowledge, the more useful this solution presents. Consequently this approach to fault diagnosis offers limited opportunities in the test environment. However, greater prospects was found for this approach while located in a heterogeneous customer environment.
Acknowledgements

I would like to thank my supervisor at Ericsson, Nicklas Weman, for swiftly and positively making sure I got answers to all my questions. I also want to thank, Mats Ljungberg, Michael Lundkvist and everyone else at Ericsson who gladly provided me with information, support and encouragement throughout my thesis work.

Additionally I want to thank for all the advice and guidance provided by Johannes Schmidt and examiner Ola Leifler for precise suggestions and feedback throughout the thesis work.

Last but not least I want to thank my family and friends for their great support in every regard imaginable.

Thank you!

Christopher Pettersson
Linköping, Sweden
May 2015
# Table of Contents

Chapter 1  Introduction ........................................................................................................... 1
  1.1  Background .................................................................................................................. 1
  1.2  Motivation ................................................................................................................... 2
  1.3  Thesis purpose ............................................................................................................ 2
  1.4  Research questions ...................................................................................................... 2
  1.5  Scope .......................................................................................................................... 3
  1.6  Disposition .................................................................................................................. 3
  1.7  Notation ....................................................................................................................... 3

Chapter 2  Theoretical Background ...................................................................................... 5
  2.1  Fault detection and localization .................................................................................. 5
  2.2  Data collection and probing ....................................................................................... 6
  2.1  Similar tools ............................................................................................................... 7
  2.2  Active Probing ............................................................................................................. 7
  2.3  Bayesian decision networks ....................................................................................... 8
  2.4  Binary decision trees .................................................................................................. 8
  2.5  IP Networking ............................................................................................................. 9
    2.5.1  The IP stack ........................................................................................................... 9
    2.5.2  IP in telecommunications ..................................................................................... 10
    2.5.3  Auto Negotiation ................................................................................................ 10
  2.6  Node unit .................................................................................................................... 11
  2.7  Test environment ....................................................................................................... 11
  2.8  Customer environment ............................................................................................... 12

Chapter 3  Method .................................................................................................................. 15
  3.1  Pre study ..................................................................................................................... 15
  3.2  Prototype implementation ......................................................................................... 15
  3.3  Main implementation ................................................................................................. 19
  3.4  Evaluation ................................................................................................................... 24
    3.4.1  Prototype implementation ................................................................................... 24
    3.4.2  Main implementation .......................................................................................... 24

Chapter 4  Results ................................................................................................................. 26
  4.1  Pre study ..................................................................................................................... 26
  4.2  Prototype implementation ......................................................................................... 26
  4.1  Main implementation ................................................................................................. 27

Chapter 5  Discussion ............................................................................................................ 32
  5.1  Results ....................................................................................................................... 32
  5.2  Method ....................................................................................................................... 36
  5.3  Test environment ....................................................................................................... 37
  5.4  Customer environment ............................................................................................. 38
  5.5  Source criticism ......................................................................................................... 38
  5.6  The work in a wider context ...................................................................................... 38

Chapter 6  Conclusions ......................................................................................................... 39

References .......................................................................................................................... 41

Appendix A  Scenarios ......................................................................................................... 43
  A.1  Scenario 1 - Missing connectivity to remote client ................................................... 43
  A.2  Scenario 2 - Missing connectivity to remote client, point to fault location ............... 44
Appendix B  Produced code Prototype implementation ............................................ 45
  B.1  Connectivity engine ......................................................................................... 45
  B.2  Demo runner ..................................................................................................... 45
  B.3  Matching engine ............................................................................................... 45

Appendix C  Main implementation evaluation tests ..................................................... 46
  C.1  Step 1 – Unit test ............................................................................................... 46
  C.2  Step 2 – Verification test .................................................................................. 46
  C.3  Step 3 – Evaluation test .................................................................................... 47

Appendix D  Results main implementation ................................................................. 48
  D.1  Step 1 – Unit test ............................................................................................... 48
  D.2  Step 2 – Verification test .................................................................................. 48
  D.3  Step 3 – Evaluation test .................................................................................... 48
List of Figures

Figure 1 Basic Bayesian decision network, T being probes and X states .......................................................... 8
Figure 2 Decision tree, T being probes and X states .............................................................................................. 9
Figure 3 Scenario 1 ................................................................................................................................................ 16
Figure 4 Structure of the software built in the prototype implementation, called Demo runner ...................... 17
Figure 5 Illustration of the sync issue .................................................................................................................. 20
Figure 6 Illustration of the auto negotiation issue .............................................................................................. 21
Figure 7 Binary decision tree from dependency matrix ......................................................................................... 22
Figure 8 Bayesian network for the thesis taken from the dependency matrix ....................................................... 23

List of Tables

Table 1 The layers described by OSI and Internet model ....................................................................................... 10
Table 2 Dependency matrix .................................................................................................................................. 22
Table 3 Result for step 1 testing ............................................................................................................................ 27
Table 4 Result for step 2 testing ............................................................................................................................. 27
Table 5 Result for step 3 testing, selected test cases ............................................................................................ 28
Table 6 Result for step 3 testing, all test cases ........................................................................................................ 28
Chapter 1

Introduction

Managing faults represents one of the certainties in any kind of network, regardless of size. Whether it is small and used for simulation in a lab or a big network providing internet access and telephone coverage to an entire country. Having a strategy to detect and locate faults as early as possible is vital to maintain the performance and stability of a network.

Over a considerable matter of time centralized software has been utilized by telecommunication systems to monitor and maintain, as well as managing faults in, the networks. However, on the nodes closest to the consumers at the radio stations much less efforts have been made. Would it not be useful to catch the faults as close to the source as possible? Can the node be empowered to actively detect and locate faults from its own perspective?

By giving the capability for the nodes themselves to perform the first line of troubleshooting, we will become more able to accurately detect and locate faults both earlier and with greater precision than with a centralized solution.

Additionally as the node itself is doing the troubleshooting, the possibility to improve can begin already in the simulation environment where the node software is being developed and tested. Previously only available to manual troubleshooters at rare occasions, the node now is authorized to reason about its connectivity, and thereby, provide all interested parties with a unique and novel perspective.

1.1 Background

In the simulation environment, just as in a real network, the node is connected to a set of other units. When developing and testing new software for the node, one does only want to be bothered by faults and issues directly related to the software under test. Historically, the amount of faults not directly related to the software under test has been high, upwards 60 %.

In comparison to the network in the customer environment, the simulation environment does not have any centralized fault management in place for detecting and localizing connectivity related faults. In the simulation environment the need for manual troubleshooting of environmental faults is vast, yet, due to time constraints it is currently not possible. As the node is also utilized in a customer environment the use of a solution for such will also be included in the thesis.

Together with experienced troubleshooters, a subset of fault causes were identified as focal points in this thesis’ work. Expressed in a compact manner, possible causes for previously mentioned questions are:

- Misconfiguration concerning link operation mode, with exploration of possible duplex mismatch situation.
- Lost connection to an explicitly configured destination vital for operation of particular features and services.

---

1 According to the informal interviews performed internally on Ericsson.
Currently the information out of a isolated single node perspective is minimal. A tool located appropriately could change this fact and provide previously inaccessible information from isolated nodes both in the test- and customer environment.

1.2 Motivation

Currently there are no practical ways to analyze the connectivity of a transportation node. It is left as a task required to be performed manually by operators using a multitude of tools to gather information on present activity in the nearest reachable area of the node.

The questions requiring answers in this thesis are: Can this process be performed in an automated fashion? Can we build a software which is capable of gathering all this information, make sense of it, and thus save time and resources for operators? Can this process gather information in a timely, reliable and structured manner?

Questions that are currently posed but cannot be answered at a reasonable cost include:

- What connections are working and which ones are not?
- Why can I not ping to this destination?
- How come I only get half the bandwidth on this particular interface?
- I am experiencing operational failure during this time period; can any inconsistencies in the connectivity be found?

1.3 Thesis purpose

The purpose of the thesis is to create a proof of concept software which is able to perform automated fault detection and localization while located on an isolated node unit.

In addition to detect and localize possible culprits, connectivity information is gathered in effort to shed some light on the connectivity-associated situation upon the node. Previously, hard-to-get information could be obtained by triggering the tool on an interval or when other detectable events occur on the node. We want to harness the information available at the node and utilize it in the most efficient manner possible to reveal the cause of faults.

This thesis aims to outline the path towards a more autonomous approach to fault management by developing a proof of concept based on the approach of Active Probing (Rish et al. 2005) which is in turn based on the experience of seasoned troubleshooters.

Basically, this work intents to put the capabilities of the troubleshooter doing manual probing, into lines of code executing on the node itself, and with potential to be used on all nodes in a network.

1.4 Research questions

- Can the Active Probing approach for fault detection and localization be applied on a single node in an IP network?
- How useful is the information from the Active Probing approach on a single node in an IP network?
- How would a potential solution be applied within a customer network?

1.5 Scope

This thesis will put emphasis upon the process of how to automate the detection and localization of IP connectivity related faults. No attention will be spent on how to localize multiple simulation faults.
Due to accessibility to hardware we will not be able to evaluate the thesis software practically in a customer environment. Instead a test environment will be utilized. Issues concerning software bugs will not be handled. Neither will issues that are exclusive to the simulation environment.

The intended use of the software tool is to replace the manual component in troubleshooting connectivity faults. Only existing tools and information, at the disposal of an experienced troubleshooter, will be considered for input to the thesis software. The software will not attempt to correct localized faults, the objective is to research and report information with significant impact to the normal operation.

1.6 Disposition

The rest of the report is divided into 5 chapters; 2 Theoretical Background, 3 Method, 4 Results, 5 Discussion and 6 Conclusions.

Chapter 2, Theoretical Background, brings forward the field of research as well as giving the reader knowledge about concepts used through in the thesis.

Chapter 3, Method, describes the work performed in order to answer to the research questions.

Chapter 4, Results, describes the findings from the evaluation performed upon the software created.

Chapter 5, Discussion; looks back at the results and, with the aid of both the background and research questions, discusses the findings.

Chapter 6, Conclusions, presenting answers to the research questions founded in the knowledge gathered through the thesis as a whole.

1.7 Notation

In order to aid the reader in grasping important concepts used through the thesis, these main concepts are further explained below.

**Fault detection** is the process of discovering if there is at least one faulty component in a system (Rish et al. 2005). When performing fault detection in a network, interest lies in whether a fault exists or not, rather than in locating it.

**Fault localization** is the process of analyzing a collection of observed or gathered fault indications, with the intent to find an explanation for this collected knowledge. In literature this is also referred to as fault isolation, root cause analysis or event correlation (Steinder & Sethi 2004).

The **node or node unit** is an IP router that is running custom software developed by Ericsson.

A **probe** is a method for obtaining information about the system and its surrounding environment. In this thesis as well as in the article about Active probing (Rish et al. 2005) a probe may return either a 0 for success or 1 for fail.

**State** is a way to describe the characteristics of the system at a certain time. In this thesis a state is normally a definition of a collection of probe results representing a hypothesis regarding the connectivity of a destination/interface combination.

**Simulation environment**, also referred to as **Test environment** is the environment Ericsson uses to verify the stability of the software. A multitude of tools help build this infrastructure, it is running on authentic physical hardware.

**Customer environment**, this term refers to an environment serving normal end users. This environment can be rather large and is generally also heterogenic.

**ComCli** is the shell used to communicate with the Ericsson developed routing software, this is a resource used to get configuration information for the node unit.

**Connectivity related faults** is a fault directly, or closely, related to the function of IP network connections. A ping response is directly related, while the functional state of a feature like clock sync is closely related. That is because the feature will cease to perform its intended function in the event of a lost connection to the synchronization server.
An **Interface** in this thesis is usually a network interface having an IP address and used for IP communication.
Chapter 2

Theoretical Background

This chapter provides an insight into the theory regarded as the basis for the work carried out in this thesis.

2.1 Fault detection and localization

The need to handle problems in IP networks has been around since the early days of networking. As the networks have evolved and grown, so has the research area for how to manage them in the best possible way. One primary part of the general field of network management, is fault management.

There is a strong bond between the areas of fault detection, and fault localization; mainly due to the fact that both require similar knowledge. Knowledge regarding the symptoms can be associated with the fault, and vice versa.

Steinder and Sethi (2004) define fault detection, fault localization and testing to be the process of fault diagnosis; where testing is the process of verifying whether the result of the fault localization is true to fact, or not.

When comparing the work involved with performance of fault detection and fault localization, localization has been suggested harder; as for example in the case where active probing is being used by Rish et al. (2005). This point of view can be more easily understood by considering the system under scrutiny only having two states, where a fault is either present or not. Then, only one symptom has to be found to detect a faulty state, rather than a full or qualifying set of symptoms that is needed in order to locate a specific fault.

Fault detection is built around the principle of looking for information that deviates in a specific way from what would be expected. The more precise information that exists about the fault and how it is manifested in the system, the more precise fault detection and localization can be achieved.

Through times many approaches have been constructed to provide automated detection and localization of faults, the most relevant ones follow below.

A rule-based approach assumes that precise information regarding what information to look for, is present. It relies on rules to detect faults, but can also be adapted to localize faults. The information in a rule based system are generally not updated, attempts to remedy that characteristic have been made in the construction of fuzzy rule based approaches (Thottan & Ji 2003). The drawbacks of a rule based approach is that it is generally to slow for real time fault detection and the amount of knowledge required to make it effective can be a problem.

Finite state machines depend on sequences of events, and can in addition to detecting faults also aid in the process to localize them. The information generally has to be derived from extensive sets of training data for the state machine to be able to detect patterns leading to a fault. One of the main drawbacks of a finite state machine solution for fault detection lies in its assumption that all alarms are true. Another drawback is the restriction of the length of the state chains, this approach does not scale well, especially not in a dynamic environment (Thottan & Ji 2003).
Pattern matching is an approach that just describes the deviations from a normal behavior (Feather et al. 1993). The normal behavior may be learned and continuously updated to require minimal external information needed. Drawbacks are that it may take time to build up a perception of what is normal. By adding the capabilities to create signatures of anomalous events this approach can be extended to localize faults. This is realized by Feather et. al (1993) by using a fault feature vector.

Statistical analysis is an approach presented by Thottan & Ji (2003) as an alternative not requiring continuous retraining. It works by correlating abrupt changes in a number of parameters and thereby detecting anomalies.

As we have seen above the most common way to localize faults is through different kinds of correlation, event correlation in the case of (Rish et al. 2005). And the majority of the previously mentioned approaches use it in some way or another.

2.2 Data collection and probing

The basis for all kind of detection and localization is data, which is acquired in mainly two ways, passive and active (Tang et al. 2005). Passive data collection are mainly concerned with collecting data that other sources make available in different ways. It can be configurations, logs, alarms and all kind of data that is available to fetch somewhere. In a network setting Muller et al. (Muller et al. 2011) shows that a basic Linux system allows for a fairly comprehensive passive data collection.

Active data collection is data that does not explicitly exist but is retrievable by using tools such as ping (Muuss 1983) and traceroute (Jacobson 1987). In order to allow the retrieval of active data, specific questions have to be posed, for example: “Do I get response from a destination IP?”. This is called probing and regardless of outcome, the probe result aid in limiting the number of possible causes.

Research in the area propose mainly two ways of doing probing, pre-planned and active probing (Rish et al. 2005), also referred to as offline and online probing (Yu et al. 2010) respectively. Active probing (Rish et al. 2005) is a continuation of the research performed on pre-planned probing. Where not only static sets of probes are used, but where also the result of earlier probes are used to choose the next set of probes to get extra value when considering the context in which probes are used.

One of the primary considerations in a solution that includes any kind of active data collection, is resource usage. This is one of the main reasons that research is being conducted towards active probing and away from preplanned probing, to decrease the amount of network traffic caused by the probes themselves. This, without compromising the value in the data provided by the probes. It becomes even more important in a setting where the active data collection is used to monitor a network and not just in cases of fault localization. As extensive probing may be directly counterproductive to the original intent of improving the stability and performance of the network (Mohamed & Basir 2010).

Due to the applicability and need within the field, methods to combine passive and active data collection approaches into more optimized hybrid systems in efforts to mitigate the drawbacks of the individual methods by themselves have been developed (Tang et al. 2005). Active Integrated fault Reasoning (AIR) (Tang et al. 2005) make use of a the passive data for reasoning and if that is not considered enough to explain the cause for the fault, probes are selected and executed to provide only the missing data for the cause to be explained.

Another approach utilizes a combination of pre-planned and active probing to reach a similar goal (Yu et al. 2010). By using active probing to narrow down the scope of possible causes to a fault and then, upon reaching a predefined threshold, switch to preplanned probing. In such a case computational work as well as time can be saved in the process of finding the cause of a fault. Depending on the implementation of event correlation from probe results to localized fault, the motivation for this kind of hybrid approaches may vary; which is the case with the use of a decision tree, where the computational overhead is minimal.
2.1 Similar tools

There are a multitude of tools aiming to aid in the process of localizing a fault. One type of tools approach the problem by learning the normal operation of the node. To that type belongs Cohen et al. (2005) and Huy et al. (2008). Cohen et al. (2005) uses an algorithm to create signatures that highlight the special characteristics of every captured system state. Cohen et al. (2005) also argue for the alleged ineffectiveness of raw values.

Huy et al. (2008) with the tool FlowMon on the contrary shows that as long as the variables, in this case counters, are chosen with care, they provide good points of reference for analysis of different state of the system. Both of these rely on the aspect of time, this is something that put great emphasis of continuity of the system where they are deployed.

The information that is collectable on an IP routing node has been examined by Muller et al. (2011). A tool is also presented along with the locations of where most information related to network is stored on a generic Linux machine.

The idea to utilize the capabilities of probes to detect, and possibly localize faults in a network has been shown and is explained by Rish et al. (2005), which also is one of the research groups that consistently work to improve the active probing as a monitoring and fault localization technique. From the perspective of using probes in an intentional and qualitative manner that minimize the negative performance effects caused by them on the network, this is valuable and relevant research. The main thing to note about the approach taken by Rish et al. (2005) involves an evaluation step where the value of information probes may generate, is of integral importance to the efficiently of the approach. This demand for previous knowledge is something that the approaches by Cohen et al. (2005) and Huy et al. (2008) do not require.

An approach for detection of faults on an application level is presented by Kandula et al. (2009) which also implements the suggested solution in the software NetMedic. The approach used does not rely extensively on knowledge, instead it focuses on the exploration of and the dependency between components. Not totally unlike the approach by Cohen et al. (2005) and Huy et al. (2008).

Yet another approach that is quite different from the rest is NetPrint by Aggarwal et al. (2009). It basically does what Muller et al. (2011) proposes and sends that information to a central server that indexes that information based on the binary parameter, if it is a working or non-working configuration. This way it is learning centrally and is able to match similar network compositions in order to suggest configuration changes. The focus of this solution is put on; home routers, and home networks.

2.2 Active Probing

Active Probing has been mentioned multiple times in previous sections. Due to the fit of the approach to the thesis the details are covered in greater detail than the alternative approaches.

The solutions of Active probing are researched by Rish et al. (2004), (2005) and several more publications by the same research team. Active probing, as described, uses Bayesian inference in two ways. First, deciding upon what probes to send next, given the result of the already sent ones. And secondly, finding out what faults are most likely, given the result of already sent probes.

In their article from 2005, Rish et al. (2005) also explore the capabilities of their approach for detecting multiple simultaneous faults. This by using a dynamic Bayesian network that also adds the dimension of time into the inference.

Active probing is primarily concerned with detecting and localizing faults in networks. But is also shown to be capable of providing reasoning regarding services affected by connectivity faults.

Refer to the article (2005) for further information regarding the algorithm and other details about the active probing approach. Brodie et al. (2008) also holds a patent for the Active Probing approach for fault detection and localization in networks.
2.3 Bayesian decision networks

Bayesian decision networks are built around the same theories as any Bayesian network\(^2\). Given the initial relational probabilities between the nodes are known beforehand, we are able to recalculate the posterior probabilities as new knowledge regarding the values of nodes is acquired. What we get is a network where we can simply read out the conditional probabilities at each node adjusted for the knowledge we have acquired up until that point. The result we are looking for in most cases is the maximum conditional probability for a node given the knowledge we have provided the model with (Rish et al. 2005). By iteratively choosing the most probable node until an instance-specific threshold has been reached, each node can be evaluated separately and momentarily. Due to this kind of situation analysis, hence out of a momentary perspective, we can ensure that the most optimal path has been taken.

![Figure 1 Basic Bayesian decision network, T being probes and X states.](image)

The advantage of this approach is that all knowledge gathered is taken into account at every step of the process.

The drawback of the inference process is that it is computationally demanding and does not scale very well as the amount of nodes in the network grows. Literature has shown that exact Bayesian inference is NP-hard (Cooper 1990). Also the construction of the initial network requires extensive knowledge and tuning to operate efficiently. An easier way, would be to make sure historical data exists; to derive prior probability distributions from; and thereby decreasing the amount of assumptions having to be made.

2.4 Binary decision trees

A binary decision tree is a binary tree that contains a question in each node. The answer to the question decides onto which path to continue down the tree. The process continues until a leaf has been reached. The leaf node of the tree does not contain questions, it describes a result. A result that is true only when all the questions on the path from the root node to the particular leaf have been answered in a

particular way. A binary decision tree can also be called a flowchart, describing the flow to a result based on questions (Beygelzimer et al. 2005).

The advantage of binary decision trees is that they are easy to follow and implement. They do not require high computational performance for diagnostics, and scale in the same way a binary search tree would in regards to time complexity when performing search operations.

The downside of the decision tree approach in general is the problem of maintaining and updating. Also, they offer no possibility to ask the questions in a different order depending on what results that is more or less likely. This means that the execution time is fairly constant for a decision tree (Beygelzimer et al. 2005). In some regards, this can be seen as preplanned probing, despite the interactive operation.

![Decision tree](https://en.wikipedia.org/wiki/Binary_search_tree)

Figure 2 Decision tree, T being probes and X states.

### 2.5 IP Networking

To better understand the faults that are used as focus for this thesis, a short explanation of concepts relating to IP networks in general, follows below.

#### 2.5.1 The IP stack

Internet, as we commonly call it in our everyday life, is essentially a collection of standardized conventions called protocols on how computers are allowed to communicate with each other. The structure of these protocols are defined in the Internet protocol suite. The structure is normally described in layers, where each protocol is generally associated with one particular layer. There are multiple models describing these layers. Two of the primary ones are the Internet model (Force 1989) consisting of 4 layers and the Open Systems Interconnection (OSI) model (International Organization for Standardization 2000) that uses 7 layers.

---

3 https://en.wikipedia.org/wiki/Binary_search_tree
The layers of concern to us are: the link, internet, transport, and application layer from the Internet Model. The Internet layer uses the Internet Protocol (IP) to define names of entities and to facilitate the routing of packets, as IP is a packed oriented protocol. Above the internet layer is the transportation layer that traditionally uses either TCP which is a connection oriented protocol or UDP which is a connectionless protocol. Based on the requirement on the transmission TCP, UDP or other protocols may be the most suitable option. On top of the transportation is the application layer where more usage adapted protocols are defined like HTTP, FTP and DHCP.

### 2.5.2 IP in telecommunications

Today an increasing amount of communication is transported over networks that uses the internet protocol for routing. This has led to users and subsequently operators finding the need to be able to handle IP traffic in the mobile networks. This has resulted in operators using IP on top of the hardware to get the most out of current and future infrastructure. A routing unit is placed on site to handle the IP communication and this unit is communicating with the internet using the internet protocol. The placement of this routing unit is at the site of the antennas that receive mobile calls over GPRS/3G/LTE which means that they are generally at locations with very limited physical access to the hardware as a result of wireless coverage being the primary factor when choosing the location. The responsibility of this unit is to terminate traffic from all hardware on the site and, using the internet protocol, transmit everything as IP traffic onwards in the operator network\(^4\).

### 2.5.3 Auto Negotiation

Auto negotiation is a way for two ends of an Ethernet link to agree upon the same duplex mode and data rate for communication. It resides in the physical layer of the OSI model (International Organization for Standardization 2000) and is required for 1000BASE-T gigabit Ethernet and compatible, but not mandatory, for 10BASE-T, normal Ethernet. Full duplex means that the signals are modeled so that there is one data stream in each direction, resulting in greater throughput for the link. Half duplex means that the data can only go one way at a time, require the two sides to take turns in transmitting data.

A fault that is related to the Auto negotiation standard is duplex mismatch. A duplex mismatch may occur when only one, or none, of the two ends of a link support Auto negotiation. The fallback setting for the Auto negotiation standard is to use half duplex if the negotiation turn up empty.

Problems occur when one end of the link assumes full duplex and the other end does not. Resulting in a situation where one end is sending traffic without waiting for the other. This creates collisions on the link when the traffic intensity increases; yet, it is barely noticeable when there is low amounts of traffic on the link, due to at the same time lower amounts of collisions. This makes duplex mismatch a

---

\(^4\) Internal documentation
troublesome fault to locate as ping probes generally will show no indication of an issue (Shalunov & Carlson 2005).

There are two primary ways to identify a duplex mismatch, one is to check the amount of reported cyclic redundancy check (CRC) errors. A CFC-error indicates packets that have not arrived intact and may indicate the occurrence of duplex mismatch since the amount of collisions dramatically increase together with the bandwidth utilization (Shalunov & Carlson 2005). The other way is to look at the amount of Runt frames, which are packets that are smaller than the minimum allowed packet size for the Ethernet standard (IEEE 802.3 (IEEE 802.3 Ethernet working group 2015)). The reasons for this being an indicator follow the same deduction as the CFC-errors, more collisions resulting in more damaged frames.

### 2.6 Node unit

The node unit is a router/switch that offers IP transportation network connectivity to the radio base station. It uses primarily the IP protocol and may perform advanced tasks such as shaping traffic and supporting advances Quality of Service (QoS) features. The node operates on layer 1, 2 and 3 of the OSI stack. The node configuration is performed via a custom shells called COMCli and EMCli that can be accessed via SSH protocol (Network Working Group et al. 2006). An object model is used to construct relations between different configured entities containing parameters and references to other objects.

In the process of manual troubleshooting on the node several tools are used. Among these are ping, traceroute, arping, tcpdump etc.

There are two features in the software that may be of use in order to perform continuous measurements of connectivity, namely “Ping measurement” and TWAMP (Network Working Group et al. 2008). They are both limited in the functionality they provide and can only monitor one connection at a time.

### 2.7 Test environment

The test environment at Ericsson consist of a collection of tools that through an elaborate process secures the quality and stability of the produced code. The tests are written in Java and use the internally developed framework Java Common Auto Tester (JCAT) in conjunction with Maven (The Apache Software Foundation 2015) and TestNG (Beust 2014).

There are hundreds of tests that are run in a great number of suits continuously to verify new code. The full chain is contained within this system, from compilation and installment of software under test to performance measurement with generated traffic. A short summary of the different components will follow.

Maven is a tool to handle the building, reporting, and documentation of software in an uniform and centralized way. Employing a project object model (POM), is well suited to facilitate the task required in a big software development organization.

TestNG is an open source project that facilitates unit-, integration- and functional testing for Java software. It is inspired by JUnit (Saff et al. 2014) but is substantially more data driven. It utilizes a multitude of annotations to be able to provide flexibility. This flexibility is partly achieved through using XML files to define the interactions and executions of different classes and groups of tests, which in turn are defined in annotated test cases written in Java.

Java Common Auto Tester (JCAT) is a internally developed framework that facilitates the automated testing of software and products. It aids in the execution of all kinds of tests, from unit- to network testing in a practical and economic fashion.

---

5 Internal documentation
2.8 Customer environment

The customer environment is in this case the telecommunication network. Its primary task is to provide stable digital coverage over large areas. The task of the node unit is to route and forward IP traffic. A customer network can contain thousands of node units and is managed by a centralized system, called Operations Support System (OSS) 6.

No fault management is performed on the node unit, except the transmitting of predefined alarms when certain hardcoded conditions are met.

The different tasks facilitated by OSS are: Configuration-, Performance-, Accounting-, Security-, and Fault management. When focusing on the perspectives of fault detection and localization, Fault Management Expert (FMX) software is of most interest. FMX is a rule based system that is hooked into OSS to improve network surveillance as well as aiding the operators. Also, by translating human knowledge into rules; FMX is capable of correlating big amounts of data, gathered in purpose of localizing faults, and thereby help gaining a better picture of the managed network 7.

FMX is built around graphically configured rules and make use of the information gathered within the network, from both OSS and other connected units. Further, FMX does root cause analysis, service impact analysis, and geographical- and functional visualizations. It also attempts to remedy faults on the node, given node connectivity.

---

6 http://en.wikipedia.org/wiki/Operations_support_system
7 According to the informal interviews performed internally on Ericsson.
Chapter 3

Method

3.1 Pre study

Since the majority of the practical knowledge is maintained by the Ericsson personnel in their daily work with related topics; gathering information by informal interviews was considered to be important. A group of Ericsson employees were chosen, due to their firsthand experience with the systems, essentially connected with the thesis. Their association with the system was ranging from system developer, to customer focused troubleshooters. As the knowledge for this particular scenario in this environment was quite scattered, no one possessed a coherent picture. My task was to acquire enough knowledge to obtain that picture myself.

As the previous work in the area was massive, a lot of approaches were discarded in accordance with the known constraints of the targeted system.

Summaries by Steinder & Sethi and Thottan & Ji (2004; 2003), provided a foundation for grasping previous work performed on this area.

A collection of implementation approaches were considered; two of the most feasible were the Active probing approach by Rish et al. (2005), and the decision tree approach, found in Beygelzimer et al. (2005). Both approaches, fully capable of using probes for the goal of detecting and localizing faults.

3.2 Prototype implementation

The prototype implementation was set out to confirm, or dismiss, the hypothesis that it is possible to create a tool that can perform automated detection and localization on the node itself. Since no previous tool exists, neither had any attempts been found succeeding, in executing automated troubleshooting on the node; this first prototype was to unveil the outlook for continued work towards such a tool.

Due to the constraints discovered in, and enforced by, the environment; all progress was valuable and needed in order to build a tool that would collect and process previously underutilized information. A tool, that would also be able to use other existing tools available on the node, to gather more information based on need.

To guide the work on the prototype implementation, a scenario was constructed. This scenario 1 can be found in its entirety in appendix A.1. In addition, the scenario also gave the requirements, setting, and evaluation framework, prior to the prototype implementation. Adding a rigidity to the work and remove unnecessary ambiguity.
The actions, defined in the scenario, should not be regarded as all the steps towards answering the greater thesis questions. The intent was merely to give enough insight, into obstacles, and possible solutions; for the main implementation to proceed in a smoother, more predictable fashion.

One of the obstacles was how to acquire the information required, in order to enable the localization of the faults. This included information from two primary sources. The first kind included the result from ping (Muuss 1983) probe, as well as data from the Arp cache\(^8\), and the interface status. This information was gathered within the system, where the software was running. A second source of information, also desired, was information from the configuration, managing the setup of the node. This class of data was to be gathered by interfacing with the ComCli; used to both view, and modify the configuration of the Ericsson software on the node.

In order to provide the required setting for Scenario 1, the simulation environment was used. This was the same environment that was used for the evaluation of the main implementation. The simulation environment was thereby forming the development-, test-, and validation environment; for both prototype, and the final software. Hence, grasping the context set by the simulation environment, where paramount to understanding the environment, within which the work for this thesis took place. Therefore, it was carefully investigated.

The origin of the scenario was based in the requirement to use active troubleshooting measures in the process of detecting, and localizing, connectivity related faults in the simulation environment. The approach taken in Scenario 1, was that the piece of code should prove itself capable of detecting the fault, created in accordance with the scenario. The tool shall, in this instance, work according to the same procedures, followed by an experienced troubleshooter that is familiar with the node unit and the type of environment.

The scenario concerns the troubleshooting of the connectivity, to a destination, for which the destination IP address was known. The environment was configured, and traffic was sent in the network, to simulate an active network situation. The first action, performed by the software, was to check the connectivity of the known destinations. Information available by parsing the ArpCache\(^8\) and interfacing with the transportation software running on the unit via the ComCli\(^9\). This compiled a list of all known destination IP addresses. The list was then used by the software to survey the connectivity to these destinations. After the initial execution of the software, one connection was severed, so that a probe would fail; in this case a ping probe. Then the software was run once more; the list would still be the same, and the software would at this point report a connection related fault, since the ping probe failed.

The continuation of the realization that the ping probe had failed was to survey the way towards the destination to conclude whether the error was located at the destination, or on the path leading to it. For the prototype implementation this was done with traceroute (Jacobson 1987).

The result expected from the prototype implementation was a report, on what changes had been observed in relation to the connectivity of the system.

---

\(^8\) http://man.cx/arp

\(^9\) Internal documentation
At this stage, the objective was to make sure that all components, required for a tool running on the unit, was available. By constructing a software that could fulfill the requirements of the scenario, the foundation for the future development could be guaranteed.

Due to the foundational nature of the prototype implementation; the decision, not to put any greater emphasis upon the actual implementation algorithm at this point, was made. Creating a well thought out implementation was left for the main implementation. At a time when the existence of all required resources was verified. The purpose of the prototype implementation was therefor to verify that all required resources was working in the environment on the node relevant for this thesis. For example; prove the programmatic access to node configuration information, and demonstrate the use of ping (Muuss 1983) to survey the connectivity to relevant destinations.

The chosen language was Python, version 2.7 (Python Software Foundation 2014). The reasons behind this was that python is a very portable language, which was also true for a substantial amount of the available libraries. Resulting in no additional software, except the python 2.7 standard library, was required for the software to run. This was an advantage for use within the desired systems.

A modular solution architecture was chosen in order to facilitate easier reuse of code in the future main implementation. An example was the construction of probes as independent methods. They aid in the reuse of code in future implementations.

As we were focusing on connectivity, the destinations and the interfaces had an integral importance in the actions, performed by the software. The connectivity of a destination could be considered an isolated, and independent unit, in the troubleshooting being performed. The interfaces, on the other hand, may be used by multiple destination IP addresses. The approach used in this respect was to bundle the interface, along with the destination IP address, and process them as one unit. The implementation has the option to optimize in this respect, if desired, but it was not done within the scope of the prototype implementation.

The probe functionality has been encapsulated in methods of their own. Due to the nature of active probing, the order, in which the probes were executed, varied from one run of the software, to the next. The best way to make sure that the execution of the individual probes was not affected by the order of which they were executed, was to make sure they all conformed into a universal interface. Since we earlier concluded that an IP destination address, and an interface, could be considered as one unit in the regards of probing, they would appear to be the natural arguments that each probe should be required to accept. This resulted in that every probe should take an IP address and an interface name as arguments.

For the return values, a similar convention had to be defined. The chosen approach was that a probe should return either a 0, if successful, or 1, if unsuccessful. These return values were inspired by the way most command line tools report exit status and how probe results are represented in the article by Rish et al. (Rish et al. 2005). For example, ping return a 0 if everything went fine, and a 1, if destination did not respond to the ping. This was the principle chosen for the probes as well.
The probes are designed in such a way that a binary result is satisfying for evaluation of the system state. This also makes the process faster, as the amount of possible outcomes will limit the possible “state space”.

The probes were basic functions that provide the software access to the tools, commonly used during a manual troubleshooting of the faults looked for. A summary of the probes and their most important properties can be found below.

**Ping**-probe, survey the connectivity to the destination IP address. It uses the normal terminal and the exit code to convey the result returned.

**ifStatus**-probe return the status of the given interface, using the information from the underlying system.

**ifConfigured**-probe, returns, whether the interface was configured or not, using the information from the underlying system.

**hasNTP**-probe, confirms, whether the destination in question were used for synchronization purposes, or not. This was valuable knowledge, in order to be able to locate which services were affected by a potential connectivity fault; affecting the destination, or interface, for which it originated.

**hasAutoNeg**-probe, return, whether the interface has auto negotiation configured, or not.

**collisionOk**-probe, look at the amount of collisions and CRC-errors, on the given interface, in relation to the total amount of packets. Returning, as the name suggests, if the interface may be communicating with a link interface, that is not employing proper auto negotiation standards.

In addition to probes, some information gathering functions was also used. The **ArpCache** is one of the commands that was used to gather information on which destinations, and which interfaces, to be surveyed for connectivity related faults. And **Ifconfig** was used to gather information about the status and connectivity of interfaces. As all destinations were associated with an interface, this information was important in order to be able to localize a fault concerning the interface.

The probes could be divided into two groups; passive diagnosing probes, and active probes. This partitioning idea resides in the work on Active Integrated fault Reasoning (AIR) technique by Tang et al. (Tang et al. 2005). The passive diagnosis probes use the knowledge already existing on the unit. For example, information regarding which destinations were configured for certain services like sync; and which destinations have been communicated with recently. The probe “hasNTP” performed passive diagnosis by utilizing information already existing on the unit. Active probes were actions employed in order to acquire information concerning the connectivity. Ping was the most obvious of these examples. In order to know the result of the ping, the active probe had to be executed. And the result was new information not previously present on the node unit.

The other functions required, was constructed as separate modules, or engines. Each engine interacted with other modules in a predefined way to perform the desired task.

The demo runner, was the main one, that utilized the others in order to perform the designated task of surveying the connectivity of the node. Arriving in a result that would show if any of the supposedly working destinations did not work, according to belief. Over time this result could also illustrate the characteristics of the connectivity situation on the node, and highlight points of particular interest.

To conclude the prototype implementation, and allow the findings to guide the main implementation, an evaluation was carried out at the end of the prototype implementation. These findings were presented in the Prototype results 4.2.

---

10 http://man.cx/arp
3.3 Main implementation

The main implementation outlined the process of creating a software, taking advantage of the prototype implementation knowledge; merge it with the fault specific knowledge, acquired up until this point, into an implementation that was capable of using probes, to return a likely verdict of the current state of each connection upon the node.

Through the prototype implementation it has been proven, that an implementation, that fulfills the desired software operation, was possible in the environment on the node. This, together with the literature study, made it convincing that an adaptive probing approach (Natu & Sethi 2007) would be a relevant path to walk.

One such approach was Active probing, as described by Rish et al. (2005), another, was decision trees (Beygelzimer et al. 2005) that takes advantage of the possibilities given by active and mindful utilization of probes, with the aim of both detecting and localizing faults; which was one of the cornerstones for this thesis.

One of the primary differences between the Active probing implementation, described by Rish et al. (2005), and the one required for the thesis, was the focus in the article on detecting and localizing faulty nodes and localization of services, associated with these nodes. The output of the approach was the localization of a node within a fully mapped network. This means that the knowledgebase, guiding the probe selection and localization of the most likely fault, was aware of the environment.

In the thesis scope, the desired use was to detect faults in association with a single destination IP address, and associated interface. In the thesis specific scenario, previous knowledge of the characteristics of the environmental surroundings were low. So instead of troubleshooting the correct operation of nodes, within a fully known network, the desired use was to detect and locate possible culprits on the connections that was of relevance for the node in question.

An example of where the approach, in Active probing, was different from in the thesis context, was in the property of probe length; used in the article by Rish et al. (2005). The length of a probe was used to evaluate the amount of information gain, for a particular probe. Its value was a number of how many nodes were covered by the probe. This metric was not easily applied upon the situation; where, we were surveying the likeliness of different kinds of faults, instead of the amount of nodes, covered by one probe. Length of probes was a way to improve the choices done by the algorithm. But in the thesis context there was no clear use for it. Another example of mentioned work, that was to improve the efficiency of the active probing algorithm, was the active selection of probing station; the node within the network that would execute the probe, and then report back with an outcome. In this thesis only one node was considered, and by that, making the choice of probing station obsolete for the thesis situation.

The intent of the main implementation was to create a software tool that followed the ideas established in the theories about Active probing, not necessarily its direct implementation. A direct use of the software described in the article Rish et al. (2005) was not the goal, nor was it feasible. This due to the differences in constraints posed by the environments, the lack of concrete code available, and the previously highlighted differences.

One of the issues regarding the Active probing algorithm presented in the article by Rish et al. (2005) was the lack of code for the underlying logic. The technique was patented by the authors Brodie et al. (2008) and no open source implementation was found during intensive search. Only the pseudo code for the algorithm, concerning active probing, was given. This meant that the task of the main implementation was not only to apply and adapt the tool described by Rish et al. (2005) but also to provide, nearly all of, the code required, in order to create the functionality outlined in the active probing approach. The decision tree implementation was not given either, but as the structure for such an implementation would be far less complex than the Bayesian network one, it was not seen as an issue.

Within network communication one must pay regard to noise, in this thesis the assumption was made that the noise levels were low enough to overlook their effect on the performance of the fault detection and localization. According to Tang et al. (2005) the use of the active probing, in itself, may be a way to
mitigate the effect of noise. As for passive collection of information; the fact that a limitation to only look at troubleshooting from a single node perspective, in this thesis means, that we were not at the mercy of uncertainty caused by network instability. All passively available information required could be gathered on the node itself.

In the difference between the Bayesian network, and the decision tree implementation, there was a difference in the way they handle noise; in short, the decision tree generally was more affected by noise (Steinder & Sethi 2004).

As mentioned, the amount of implementation details was limited for given approaches; but a prototype software was required in order to satisfy the evaluation.

The work begun with what was known, the pseudo code for the active probing algorithm. Each step in the algorithm was then analyzed, and broken down, in order to extract the requirements of the independent parts; enabling the possible development of each function separately. When the task for each of the included functions was understood, the next step could be initiated.

In their work Rish et al. (2005) used Bayesian inference to provide the implementation of all the included functions. While searching back in the publication history of the researchers in an attempt to learn more about their implementation of the Bayesian inference, a parallel idea emerged. The idea, built on the same intent to detect and localize faults, also by using probes in an iterative process. The idea came from a previous work of the same research team, who wrote the active probing article (Rish et al. 2005). The article titled was “Test-based diagnosis: tree and matrix representations” (Beygelzimer et al. 2005) and described the use of decision tree implementation for detecting and localizing faults.

The implementation itself was clearly different from active probing, but from the perspective of overall thesis question; it was well within the scope. At this point, the decision to create two implementations with the same basic approach of using active probes, was taken. Alongside the Bayesian inference approach would also be a second implementation conforming to the same input, output, and knowledge.

The reason behind this was to get a more dynamic result, and discussion, regarding different implementations. This also gave the opportunity to better see the pros and cons of respective implementation. The development of two different implementations did work well into the setting constructed by the thesis. The same set of probes was used; so was the outcomes, and the knowledge guiding the implementation, in detecting and localizing different faults. The only difference between them, except for the actual code, was the work of creating the implementation specific knowledge representations. The Bayesian inference approach require a Bayesian network, while the decision tree require a tree representation of the knowledge. Which in itself, in the construction phase, awakened questions regarding the reasons for doing one way over the other.

In order to create the foundation for a relevant evaluation, the detection and localization of faults had to be performed on context relevant faults. In this thesis the knowledge comes originally from the informal interviews\footnote{According to the informal interviews performed internally on Ericsson.}. Two faults were chosen, which are described below.

![Illustration of the sync issue.](image-url)
Synchronization is a critical feature on any communication node and must therefore be monitored carefully. As the controls for the synchronization were based on the connection to a synchronization server, it was of interest as a connective related fault. If the connectivity to the synchronization server, in this case a NTP server, exhibit irregularities the synchronization can be affected with disturbances as a result. Naturally, the same approach could be applied to any of the other services relied upon on the node, but for simplicity, only one; NTP was chosen.

![Diagram showing auto negotiation issue](image.png)

**Figure 6 Illustration of the auto negotiation issue.**

Secondly, for a connection to work properly; the two terminations of the links must use the same speed and duplex configuration. In general, this is handled by the auto negotiation standard (IEEE 802.3 Ethernet working group 2015). In cases where this did not work properly, it was not always clear as to why that was the case. For example, the link may work fine for low amount of traffic, just to display serious congestion as the amount of traffic increases. This, could in some cases be detected by looking closely at the configuration and statistics of the interfaces, hence constituting a good choice for our intentions. But also to be clearly different from the affected services fault displayed earlier (the synchronization fault).

This knowledge about the example cases was translated into a dependency matrix. The dependency matrix as a knowledge representation system were utilized by both the Bayesian network (Rish et al. 2005) and binary decision trees (Beygelzimer et al. 2005). Hence, presenting the perfect starting point for the construction of the knowledge representations required in the implementations.

For the proceeding part of the thesis an example matrix was constructed. In order to assist in the understanding of future knowledge representations. The same knowledge has been presented in all upcoming knowledge representations, regardless of type. Thereby giving the reader a possibility to grasp the differences in how the same data was presented in different ways for different purposes.

Dependency matrices were, as pointed out by Beygelzimer et al. (2005), commonly used within the area of fault management in general for their capability of in a human accessible way represent the data required for fault correlation.

As previously mentioned, the Active probing approach consisted of an iterative hypothesis testing where a number of important states were defined as a combination of probes; this could be illustrated with a dependency matrix.
The dependency matrix was constructed with the knowledge and faults in mind, that was gathered from the informal interviews\textsuperscript{12}. The probes was denoted as $PX$, where $X$ was a number, that in this example was ranging from 1 to 6. The states representing the results of the troubleshooting was denoted in a similar way; but prefixed with a $S$ instead. A description of the states was also available.

The dependency matrix has shown to be a starting point for the construction of several different implementation specific data representations.

Within the matrix, an unsuccessful probe was denoted with a 1, while a successful one was left undefined. This worked since there were only two possible results of a probe, success or not, 0 or 1. Hence, we were only required to explicitly state one of them.

\begin{table}
\centering
\begin{tabular}{|l|c|c|c|c|c|c|}
\hline
for an ip and interface & s1 - destination unreachable & s2 - interface is down & s3 - Autoneg conf. & s4 - NTP server down & s5 - Autoneg conf. & s6 - Autoneg conf. & s7 - OK \\
\hline
p1 - ping & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
p2 - ifStatus & & & & & & & 1 \\
p3 - ifConfigured & & & & & & & \\
p4 - hasNTP & 1 & 1 & & & & & 1 \\
p5 - hasAutoNeg & & & & & & & 1 \\
p6 - collisionOk & & & & & & & 1 \\
\hline
\end{tabular}
\caption{Dependency matrix.}
\end{table}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{binary_decision_tree.png}
\caption{Binary decision tree from dependency matrix.}
\end{figure}

\textsuperscript{12} According to the informal interviews performed internally on Ericsson.
In this example (Table 2), that will be used through the report, the unsuccessful state was chosen as the "default" state. The fact that some zeros have been explicitly stated will be explained later in the report, when explaining the requirement for the construction of the Bayesian network.

From Figure 8 we can see that the outcome of the ping probe divides the state space in two ways; so does the outcome of the ifStatus probe. Iteratively, this approach narrows down the state space for every probe executed until only one state remains. The final state is the one most likely to represent the actual state of the system.

All nodes, except leafs, were represented by probes capable of returning a binary result. This is how the binary decision tree work in an iterative way. By being aware of probe results already acquired, no questions were asked more than once, and the implementation could simply walk down the tree until a leaf was found. The leaf represents a fault and a result in the process of fault localization.

This was a simple and straightforward representation of knowledge to express the relation between probe outcomes and states. Its connection to the dependency matrix was discussed by Beygelzimer et al. (2005). In the article the dependency matrix representation was attributed to be best suited to handle changes in knowledge, as well as being most manageable. Beygelzimer et al. (2005) also correlate the dependency matrix with the binary decision tree, called flowchart within the article.

For the implementation of the Bayesian inference a Python library was used (eBay Software Foundation 2013). This enabled the software to be implemented in a way that was conforming closely to the Bayesian network implementation of the Active probing algorithm (Rish et al. 2005). As for the construction of the knowledge representation, limited guidance was to be found for the particular area of decision networks.

Figure 10 shows the data model expressed as a Bayesian network, containing all probes and states.

When defining a Bayesian network, there was a need for probabilities for every connection in the network. This information was something that did not exist. The challenge, in this case was that there was no historical information; neither was there any good perceptions regarding the general frequency of the faults.

The explicit zeroes found in the dependency matrix were required for the Bayesian network to be complete. Unless the network was complete, we could not perform the inference.

The starting point for the construction of this data model was the assumption that all state were equally likely. Further, the assumption that the amount of probes supporting a state was an indication of high support for that particular state. For example; a first state (1), was defined by 3 probes that failed, a

![Figure 8 Bayesian network for the thesis taken from the dependency matrix.](image-url)
second state (2), was defined by 2 probes that failed. Given that all probes, for both (1) and (2) failed, the first state (1) was to be the most likely; since it got the higher total amount of probes supporting it.

The research on this field was scarce; hence, the choices that were made in defining the probabilities within the Bayesian network, were primarily shaped by the researchers assumptions and ideas.

To define each state the dependency matrix was used, each state was defined by a specific combination of probes. This was used to verify that the Bayesian network was capable of localizing all clear cases when one state was the right answer. When all clear states were found, given the probes defined, in accordance with the dependency matrix, the first adjustments was judged finished. The second part that required adjustment was the threshold used for knowing when the result was satisfying. Without a well-chosen value for this threshold, the active probing algorithm would not be able to decide when a good enough diagnosis had been reached. By using both manual testing, unit test, and simulation test runs of the algorithm; together with previously discussed knowledge representation, a threshold was eventually reached.

### 3.4 Evaluation

The structure of the way implementations will be evaluated are presented below.

#### 3.4.1 Prototype implementation

The evaluation of the prototype implementation was primarily concerned with the “Scenario 1 - Missing connectivity to remote client” (see Appendix A.1) that was used as the blueprint. Both during implementation and in the following evaluation. The primary target of the evaluation of the prototype implementation was thereby to verify the predefined scenarios’ conclusions in the node unit environment. By proving those on the node, the point of functionality for the prototype implementation would be confirmed.

The functional goals were mainly targeted on the interaction with the environment, and verifying the ability to run the software on the node unit, and secondary to be able to verify a fault. The most important part for the prototype implementation to confirm, was the possibility of an “on node”-software, with required capabilities; when considering information access.

#### 3.4.2 Main implementation

The evaluation of the main implementation was situated within the test environment, just as the development had been. The goal of the evaluation was to put the software developed through a series of tests, in order to answer the research questions in a satisfying way. This was done in three steps.

First, a set of unit test (Python Software Foundation 2015) were created to verify that the implementations were returning the right answers in accordance with the reference that was described as a dependency matrix, Table 2. Only the implementation algorithms were tested at this stage; not the code used for facilitating the “on node”-data access, or probe executions. The reason for this was the high dependency of such tests upon the environment. Also, for the prototype implementation, the focus upon environmental challenges was prominent. Hence, the liberty of omitting the test for that particular part of the code in the first stage of the main evaluation.

Secondly, there was a suite of java simulation test written to simulate the example faults in the test environment. The software was at this stage running on the node unit. And thanks to the prototype implementation no substantial amount of work had to be spent resolving basic implementation details. The majority of work at this stage was spend on the construction of test cases within the simulation environment. When such cases were showing satisfactory for our purposes; the thesis software was hooked into the execution chain of the test cases for testing.

Thirdly, based on the result requirements of the research questions; a collection of test cases were carefully chosen, based on their perceived capability to generate connectivity related faults. These test
cases were chosen to represent the most likely candidates to benefit from the capabilities of the developed software. The result from tests, at this stage, was to show the general impact of the usage of the thesis’ software in the test environment. The results found, during this test stage, would have substantial weight on the overall conclusion of the thesis.

As there have been two clearly defined implementation with the same functionality, both of them will be evaluated in the perspective of finding the differences between them.
Chapter 4

Results

Since the method chapter was partitioned into three parts, so will the result chapter.

4.1 Pre study

The informal interviews\textsuperscript{13} gave two kinds of issues that were to be used as example cases throughout this thesis’ work. Those two were the detection and localization of faults associated with synchronization, and auto negotiation compatibility. Both were identified as existing problems where an automated solution would be of practical value. When comparing the faults, clearly defined; only general connectivity related faults could be provided from the test environment. Therefore, additional faults were gathered from what was observed in the employees experiences interacting with the customers; where there were considerably easier to locate sample faults.

Active probing (Rish et al. 2005) was found to be the most suitable approach in realizing the process of a manual troubleshooting workflow. In addition, the approach of active probing was also implemented with a decision tree (Beygelzimer et al. 2005). Both implementations providing comparable implementations with different theoretical complexity. These two implementations were chosen for answering the research questions of the thesis.

In the implementation of the prototypes, only manually acquired knowledge regarding the localization of the different faults was used; meaning the tool was always aware of what it was looking for.

4.2 Prototype implementation

One of the targets of the prototype implementation was to prove whether the overall approach to the problems was sound. As far as the prototype implementation reached, the architecture can be said to satisfy that target. The implementation was confirming the predicted functionality in accordance with the scenario.

The association of one destination IP address to an interface was also confirmed as the most useful and conclusive way of ordering the troubleshooting by.

The implementation environment was found to pose some practical issues that was realized through the prototype implementation. During most of the time spent on working with the prototype implementation, there were some limitations in regards to the access to libraries. Result of this was that constructions used had to be adapted, and could not rely on most external libraries; due to their dependency of the standard libraries. This gave a slow and cumbersome prototype implementation process.

The constraints posed by the environment were solved by the end of the prototype implementation; showing potential of a less cumbersome main implementation process.

\textsuperscript{13} According to the informal interviews performed internally on Ericsson.
Looking at the road towards a more profound solution, the findings of the prototype implementation seemed to be promising. The basic functions to survey the connectivity from a node at one point in time were working, as shown with “Scenario 1 - Missing connectivity to remote client”. The information, like routing tables, configurations, etc., was available for the continued work of the main implementation.

### 4.1 Main implementation

A working piece of software was created to be the main implementation. Within the same software, two different knowledge representations and probe evaluation approaches were implemented. A Bayesian inference- and the binary decision tree implementation. Both of these were subjects to the three steps of testing; only, in the second and third step, also tested in sequence.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayesian network</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
</tr>
<tr>
<td>Binary decision tree</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
<td>Verified</td>
</tr>
</tbody>
</table>

**Table 3 Result for step 1 testing**

When testing only the result evaluation code with the predefined probe result sets, both implementations were showing precision and recall score of 1.0 in the process of classifying the faults explicitly recreated; this, according to the unit tests in step 1. Given the defined input to the algorithms, all states could be verified; but, when the states became inconclusive the different implementations could give different results. The probe result sets not representing a clear state output was not tested.

<table>
<thead>
<tr>
<th>Custom test case</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>testStateDestinationNotReachable</td>
<td>Verified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Verified</td>
</tr>
<tr>
<td>testStateInterfaceIsDown</td>
<td></td>
<td>Verified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testStateNTPServerNotReachable</td>
<td></td>
<td></td>
<td>Verified</td>
<td></td>
<td></td>
<td></td>
<td>Verified</td>
</tr>
<tr>
<td>testStateAutonegotiationNotConfigured</td>
<td></td>
<td></td>
<td>Verified</td>
<td></td>
<td></td>
<td></td>
<td>Not replic.</td>
</tr>
</tbody>
</table>

**Table 4 Result for step 2 testing**

In step 2 the similar result could be seen, both ways of implementing the evaluation code in the software were capable of localizing all but one of the defined faults. The fault that was not able to be recreated was the auto negation mismatch fault.

There were differences between the implementations when looking at the amounts of probes they were using in order to localize the same fault. The Bayesian inference implementations were using an, on average higher, amount of probes while reaching the same classification as the decision tree implementation. As a direct result of this, the time used for detection and localization was higher than for the same result reached with the decision tree implementation. The variation in the amount of probes from state to state also varied for both of the implementations. The decision tree implementation was using a constant amount of probes for each individual state localized, while the Bayesian inference implementation could use different amounts of probes, depending on the probing order and results.

For step 3, the task was to run the software together with normal test cases used in the verification of the Ericsson software. The result of running the software, regardless of algorithm after test cases, showed that the amount of faults detected and localized were very low. The faults that were identified did not directly relate to the test cases. Over 30 test cases were used to begin, and 3 cases were ran extensively in an effort to have a fault occur. The only results from these tests where the localization of a few unreachable destinations and warnings for lack of auto negotiation on interface.
When comparing the two implementations, evaluation was carried out by running them in parallel; which showed that the same result was reached in the majority of cases. But the amount of probes required to reach the same result, did differ frequently; a result, also seen in step 2.

<table>
<thead>
<tr>
<th>Normal test cases</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPForwardingTest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testConfigIPForwardingWithMultipleRoutes</td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testNullRoutingByStack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testRoutingLoopPreventionDroppedPackets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testStaticRouteSelectionBasedOnAdminDistance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testStaticRouteSelectionBasedOnPrefixLengthOfDestinationSubnet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testStaticRouteStillWorksWhenNextHopIsChanged</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSPFv2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testHoldDownTimer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
</tr>
<tr>
<td>testVerifyOSPFDoesNotReduceArpingPerformance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
</tr>
<tr>
<td>testHandlingOfMtuMismatchDutUsingJumboFrames</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testLosDetectionP2P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testManyNegotiatedProcessesAndManyConfiguredInterfaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VirtualRouter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>testMaxInterfaceAndVrConfiguration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testPingFromAndToLegacyAndVirtualRouter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testReconfigurationVr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testTracerouteFromAndToLegacyAndVirtualRouter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testUntaggedAndTaggedInterfacesInLegacyAndVirtualRouter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testForwardingInSeveralRouters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testHttpsAccessUsingOamAPAndVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testNetconf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testDeleteIPInVRAndDefaultRouterAndDeleteVRWithIPConfigure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testMoveIPvInterfaceBetweenVirtualRouterAndDefaultRouter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testOverlappingStaticRouteTableInDefaultAndVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testStatePropagation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>testArpToTheSameIPAddressInDifferentVirtualRouter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 6 Result for step 3 testing, all test cases**

<table>
<thead>
<tr>
<th>Normal test case</th>
<th># of runs</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>testCorrectRouteUsedAfterPortFlapping</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
</tr>
<tr>
<td>testRunningMultipleProcessesIndependently</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
</tr>
<tr>
<td>testForwardingInSeveralRouters</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Found</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5 Result for step 3 testing, selected test cases**
Chapter 5

Discussion

5.1 Results

The intent with the pre study was to gather connectivity related faults, both directly and indirectly, that would persist in the test environment. For example; the direct fault that a connection has been lost, and the indirect fault where a service is affected by an issue with a connection. This was found to be more complex than anticipated. The faults could neither be gathered from logs nor from employees working with the maintenance of the test environment. The connectivity faults persisting in the test environment was basic in nature so to deepen evaluation of the thesis software, additional faults had to be found, with the risk of them not being common in the test environment. The underlying cause for what was found to be a “too” good and stable test environment. Historically there have been a lot of relevant faults in the test environment, but this situation changed during the time spend working with the thesis\(^4\).

When looking at the research questions it is clear that the focus generally was to evaluate the use of active probes in a troubleshooting software situated on one node, with an emphasis upon the factors of active probes and one node. As this was the goal this thesis was set out to research, the lack of clearly defined fault in the test environment was categorized as a distinct, yet not fatal, obstacle to the thesis work overall. The focus was still to evaluate the node perspective. Even though the test environment was not the most optimal for evaluation.

One option considered was the use of another environment for development and evaluation. Since that option was researched it was clear that it was neither possible to develop nor perform the actual evaluation in any other environment\(^4\).

The faults identified were, synchronization server unreachable and configurations related to auto negotiation. These faults were gathered from the informal interviews\(^4\) carried out with experienced troubleshooters mainly working with the customer environment. Hence problems was not particularly associated with the test environment. The attempted reproduction of these faults in the test environment did show good results in the case of synchronization faults but also revealed inability to replicate all the auto negotiation related faults. This was a result of the latest software subjected to testing did not allow the underlying misconfigurations for the duplex mismatch to occur\(^5\). Specifically it did not allow the configuration of half-duplex mode at all. This lead to fewer practically verified fault for the software developed. The underlying cause for this is the inability of the node to, in current software, operate in a half-duplex mode\(^5\). If this would have been discovered earlier other faults would have been research as well.

The choice to create two parallel implementations was taken to enable a comparison of how active probes could be used on a single node. Facilitating a better discussion about how the active probes were

---

\(^4\) According to the informal interviews performed internally on Ericsson.

\(^5\) Internal documentation
used. The aspects of interest was the amount of probes required, the computational requirements and the accuracy of the output.

Active probing according to Rish et al. (Rish et al. 2005) was using Bayesian inference, which was implemented in accordance with the article. Additionally a second way of implementing the active probing using decision trees was found in an earlier work (Beygelzimer et al. 2005) by the same group of researchers. This was intended to facilitate a valuable discussion in regards to how two, clearly different implementations can be used to work in accordance with the ideas of using active probes for detection and localization of connectivity related faults.

The prototype implementation was intended to be a “pre runner” for the main implementation. The main objective was detection of all the issues in the test environment, and also finding ways to handle those, before initiating the main implementation. Further this meant the prototype did not have to do the exact same thing as well as the main implementation software should. As long as the overall operation of the prototype included all environmentally dependent functions as the main implementation would need. To do this the “Scenario 1 - Missing connectivity to remote client” and “Scenario 2 - Missing connectivity to remote client, point to fault location” were used, to ensure that the features required were included.

A lot of environmental issues were found during the prototype implementation. One of the main ones were the absence of the python standard library. This caused great issues when attempting to access the Ericsson configuration for the routing features of the node. The solution was to use a parallel installation running on node, within which full access was available, and thereby the ability to allow access to the full standard library.

The issues detected had their origin in the fact that no other software but the Ericsson routing software should be running on the node. This was a suitable and logical standpoint yet problematic when attempting to do new, non-intended things on the node as was required for this thesis.

The choice of associating one destination IP address to an interface was made already during the prototype implementation. We were looking for connectivity related faults, the kind of faults that always will be associated with a destination IP address as well as an interface. Hence it was the natural choice to define those two values for the entity to troubleshoot. In that way, the tool could work in a systematic and practical way by using the full connectivity of the node, and at the same time be assured that all connectivity related faults were detected and localized. The backside of this follows that troubleshooting a node will, in most cases, be proportional to the amount of destinations it communicates with. The alternative option would have been to look at a selection of values, for example, abnormalities in counters that would indicate issues, and then locate the associated destination and interface. An increase reliance of the knowledge would be required for such approach, but the advantage would be a potentially faster, detection and localization of faults.

Conclusively the prototype implementation paved the way for the main implementation allowing more attention to focus on the actual software construction.

As previously discussed in association with the pre study, two implementations were created and evaluated. This was done because the ability to compare two different implementations enriches the conclusion regarding the use of active probes. Mainly in regarding to how a solution should be constructed in order to mitigate different kinds of disadvantages and unfavorable characteristics of the evaluation implementation itself. A comparison of the two implementations will be disused later when the test results have been discussed.

The unit test in step 1 filled two important roles for the implementation. First, it allowed the fine tuning needed in order for the Bayesian inference implementation to work satisfactory. Secondly it was a way of confirming that both algorithms did arrive at the state predefined when given the qualifying set of
probe result for that particular state. This was important since it allowed the continued testing to have a solid core that could be trusted in the capability to detect the defined states.

Following the step 1 unit tests was the step 2 functional tests. The outcome of these tests that were based upon the execution in the real environment and detecting simulated faults in an actual network. By looking at the result we were able to tell whether the software was capable of detecting faults for which it was built. The destination unreachable result found on the NTP synchronization server destination was accurately presented as NTP server unreachable state. This was indicated for both of the implementations.

From the creation phase of the test cases used it was evident that one of the previously defined states would not be possible to test. The reason for this were some rather late identified constraints in the node software. Which further resulted in an inability to provide the required setup within the environment to verify the software’s capability to localize the state. The fault in question was the auto negotiation mismatch situation described earlier and by Shalunov & Carlson (Shalunov & Carlson 2005). Since the discovery of this issue concerning the testing environment occurred at a rather late stage during the thesis work, no other reasonable alternative was found.

The reason this became problematic derived from the general absence of connectivity related faults in the test environment. Which was essential for the evaluation of the implementation. The auto negotiation mismatch fault was regularly occurring in the customer environment16 and the process of detecting it known (Shalunov & Carlson 2005). In this particular case the trouble of testing this fault in the test environment comes from the difference in the software versions between the test- and the customer environment. A later version of the software was used in the test environment, completely logical since testing will always be on the latest software version. In this instance the configuration that was capable of causing the issue was not possible in the new version. This meant that in order to test the fault, duplex mismatch, in the test environment, an older software would have had to be installed. The endeavor of accomplishing this was not prioritized in the thesis’ work due to the fact that the fault would not be occurring in the current test environment which was the essential target for this thesis.

Conclusion of this is that two of the states, s3 - Autoneg conf. CC ok-state and s6 - Autoneg conf. CC not ok-state were not possible to verify in the current test environment. And thereby, these states will not be given weight in the evaluation step 2 and successive step 3.

When looking at the difference between the two implementations we can observe that the used amount of probes differentiate them, even when both implementations present similar results. The Bayesian inference implementation uses more probes than the decision tree implementation. The cause for this was found to depend on how the different implementations knowledge representations were constructed. In the case of the decision tree implementation, the task of manually constructing the full tree, was a fairly straight forward task. While on the contrary the construction of the Bayesian network that were used for the Bayesian inference was substantially more complex. The complexity lead to the need for more optimization and fine tuning in the instance of the Bayesian network. Due to the overall scope of this thesis not being to optimize data structures for the implementation, the data structure that generally required the most tuning would be the one using the highest amount of probes. In this case it was clear that the more complex one, Bayesian network, was using more probes than the less complex one, decision tree. Mainly since the amount of parameters having to be estimated for the Bayesian inference was so much greater.

Take the threshold for example, in order for the Bayesian inference to produce a fast and accurate result the threshold would have to be tuned together with the conditional probabilities between all the probes and states. The threshold would have be tuned until a valued were found that provided the best average performance for all different situations. The threshold tuning in this thesis was only aimed at making sure that all clear states could be identified. Leaving all the combinations not contained within the precise qualifying set, sub optimized. Worth mentioning, provided a great amount of samples, the

16 According to the informal interviews performed internally on Ericsson.
Bayesian inference could be much better tuned. Leading to the belief that manual construction was not a particularly suitable way of using it.

In short, the decision tree implementation did use less probes and delivered the same result given the setting and methods used in this thesis. Hence leaning to the conclusion that it, the decision tree implementation, must be the preferred knowledge representation in this particular case.

In step 3 the software was run after the test cases had been executed and before the test-case-specific environment configuration was torn down. The reason it could not be hooked in right after every failing test case, at the best possible place, was constraints in the test environment. This stopped the evaluation of the solution from participating in the normal test case runs in the test environment. Instead the evaluation was carried out by manually modifying teardown methods for handpicked test cases. The test cases were chosen based on their perceived capability to produce connectivity related faults. But by doing this modification we always ran the software upon test case environment teardown. This factor affected the amount of tests that could be included at this stage. Hence only choosing a limited but relevant number of test cases for evaluation.

The factors necessary for the use of the software on the general test case were not accessible simultaneously. The two factors were, the result of the test case and the intact environment used by the test case. If they would have been accessible simultaneously the software would have been able to hook into most test cases, and had the ability to only attempt troubleshooting when the test case failed. In the current test environment the intersection between these two requirements does not exist hence the capability to execute the software at the most optimal place could not be done.

When looking at the results from step 3 evaluation, two results were observed, the warning for auto negotiation not configured and destination down. The s5 - Autoneg not conf-state was frequently reported in the test environment. The reason for this was, auto negotiation upon the interfaces were not required in the test environment in general. For example when the eth0 interface was used, no auto negotiation configuration could be found. This because the eth0 interface was not managed by the routing software and thereby it could not be surveyed for the information regarding whether auto negotiation was enabled or not by the software.

The s1 - destination unreachable-state was observed on rare occasions, it could be explained in a number of ways. Either it was a fault, the result of problem in the connectivity of the node, which was the case representing the highest value from a fault detection perspective. Or it was a residual destination that had not been properly cleared from sources used for basis of detection. Or a third option, could be the imperfect timing for the software execution, in such a case, the destination unreachable state would be explained by the failure to get access to the intact execution environment.

The test environmental logs could be used in resolving this issue. Note, however, the absence of these logs in anything but the test environment.

In this particular case the logs revealed that we had two different scenarios. (1) the test case succeeded and (2) the test case failed. In case (1) the observed state was not an indication of a fault within the test environment. The explanation for the particular s1 - destination unreachable-state was the test case only using the destination during part of the execution. And in the following parts of the test case reconfigured the connectivity. Hence the detected connectivity faults were not of importance for the troubleshooting of the environment.

In case (2) failure in pinging a previously defined destination generated a failed test case. Which was accurately detected by the software. But as no additional information was available regarding the unreachable destination, no additional information could be given by the software. Traceroute was not checked as the network was constructed by switches, meaning no information could be acquired by using traceroute as the traffic where only terminated at the destinations.
The amount of faults located were low. The background for this could also be observed in the test case logs maintained by the test framework. There were very few connectivity related faults to be found on the test case executions that were run before the software. Thereby explaining the result from the testing of the software. This was a general trend, and it is important to note, that there have been a clear recession from the beginning of this thesis work up until this day. And that is, in the amount of connectivity related faults that do occur in the test environment. The development in this regard also appear steady for the foreseeable future.

The results from the software execution can be enforced by the observation of fault logs from the test environment. Logs that, at a deeper inspection, shows us that there were in fact close to no faults that could have been detected. And due to the framework not allowing the software to run after each test the chance of having one occur during test case executions with the software were very low. This did not only show the issue to acquire good test results, but also the general state of the need for a single node troubleshooting software within the test environment. Hence proving the conclusion that the software show low additional value when used within the test environment as it is today. Not just because of the result form evaluation step 3 but also due to the overall presence of faults within the test environment.

Evaluation results show the absence of faults, which also analyzing the test case logs revealed. The absence of faults prove two things. One, that there were very few connectivity related faults in the test environment. And two, that the approach of using a single node troubleshooting within a test network may not be the best utilization of resources.

5.2 Method

The initial work in this thesis was to a large extent centered around surveying the area that is fault management. It began with a summary (Steinder & Sethi 2004) and eventually ended up in the theories of active probing (Rish et al. 2005). A parallel way of implementing active probing was also found in the use of decision trees (Beygelzimer et al. 2005). Both of these implementations were adopted as the most promising ways of utilizing the active probes for the task of fault detection and localization. The work on building a broad understanding of the field was useful but the narrowing down could have been made earlier. This would have been more appropriate in order to allocate more time to the later phases of the thesis work, which was found to require extensive knowledge in the areas of node- and test environment management.

One standing challenge throughout the whole thesis work was the testing environment. Where the networks around the nodes were quite small. This affected the complexity of the faults capable of occurring, thereby making some faults identified within the customer environment difficult, or even impossible to evaluate in the test environment. Initially the focus of the software was purely on the node in the test environment. But as the work progressed the connectivity related faults became fewer and fewer in the test environment¹. The result of this was the need to bring in faults from the customer environment, where the identical node was also used. Unfortunately, the intersection between faults in the test and the customer environments were low. This lead to issues, since we were only able to evaluate the solution in the test environment. Leading into a situation where we had faults that was not occurring in the environment we could test and evaluate the solution within. In hindsight it is not clear how this issue could have been avoided as there were no other possible evaluation environments, and very few connectivity related faults in the test environment. This fact had a great impact upon the result on both step 2 and step 3 testing, as previously discussed.

The choice to implement the active probing in two ways was found to be useful in the understanding of how to handle the potential offered by the active probes. And also to show how the complexity of the software can be adapted to simplify and improve the most common tasks of troubleshooting.
An example of when the implementation can affect the usefulness of the software is in how knowledge can be imported and modified. In the perspective of a human providing information the decision tree implementation must be considered easier to work with, hence more beneficial. As the translation from a dependency matrix format into a decision tree have been investigated (Beygelzimer et al. 2005). But on the other hand, if the knowledge can be extracted from a big amount of data and the human interaction with it can be limited to a minimum then the Bayesian network approach would be more suitable. This since such case could utilize Bayesian learning.

Another factor that could have made the Bayesian network approach more efficient would be the added knowledge of the present state of the node. Better performance may be possible by including the potential knowledge about the state of the node into the algorithm. This is one piece of information that was not relevant in the test environment but could prove to be useful in the customer environment.

There are a number of practical conclusions gathered from implementing this solution on the node unit. In order for the implementation to be possible a lot of work were spent on resolving unexpected issues occurring on the node. Examples are the lack of python standard library, cumbersome access to channels to reach configuration information and instablity of software when reboot of the node. All matters taken into account, following can be concluded; in order for a more stationary deployment of a software like the one developed in this thesis, some additional work is required.

5.3 Test environment

The test environment that was described in the theory background have been further developed and improved over the span of this thesis work. When looking back on the time when defining the research questions, there have been development primarily in regards to stability. And also the amounts of connectivity related faults regularly occurring have declined.

From the perspective of detection and localization of faults on the node the test environment pose some unique challenges. In the result from step 3, we could observe a clear lack of faults localized, with the exception of auto negation not present on interfaces, which could be explained by auto negotiation not being explicitly required for interfaces within the test environment. The near absence of connectivity related faults was verified by consulting the logs for the relevant test cases used in the testing of the software. Very few faults could be found there. Hence the result received could be confirmed from a second source. The explanation for this result was, as previously explained, the lack of faults to localize.

Conclusively, the level of uncertainty, regarding the results given by the software, should be considered low; due to the amount of localized faults seen in the test logs.

The amount of information regarding the test environment is vast, from the test framework perspective all configurable parameters of all connected units are known. But this information is not accessible to the software executing on the node, leading to a situation where the software have very little information in comparison with other potential troubleshooting methods, like manual troubleshooting for example. Due to the single node restriction nor could the software be allowed to access this information.

One of the practical implications of this restriction appeared when the software was tasked with troubleshooting a failed test environment setup. Since no traffic was passed through the test case environment there were no traces of used destinations in the arp cache\textsuperscript{17}, and thus no way to find out which connections to test. Such information is located in the framework, configuring the environment, hence it would be beneficial to place a software with similar capabilities as we have on node there. With access to all available information. Currently the software is limited to detect and localize disruptions on already configured or used destinations. Which severely limits its practical usefulness of the software within the test environment.

\textsuperscript{17} http://man.cx/arp
5.4 Customer environment

The third research question was related to the customer environment where the situation was very different from the one observed in the test environment. The network was significantly larger, the hardware and software were more heterogeneous and very little were known of the node surroundings. These factors affects the outlook of the approach taken in this thesis. The software could not be evaluated within the customer environment.

The results, seen from the evaluation of the software in the test environment, showed less than initially anticipated usefulness. But several of the factors limiting the use of the software in the test environment were not present in the same degree in the customer environment. Some other factors identified suggest even greater potential in the customer environment. Factors such as the aforementioned heterogeneity and the complete lack of information about, and control over, the surroundings of the node.

One of the most important differences between the test- and the customer environment, from the software execution perspective is how it is triggered. In the test environment the trigger is naturally after a failed test case (a custom that is currently impractical for reasons concerning test case construction and environment setup/teardown standards). But, in the customer environment another kind of triggers are required. Informal interviews and internal documentation suggest two active and three passive ways. The active ways are TWAMP (Network Working Group et al. 2008) and Ping measurement. And the passive ways are alarm threshold hooks, KPI (Key Performance Indicator) threshold hooks, and the possible use of an interval based execution. TWAMP and Ping measurement both require manual setup which makes them more suitable for targeted efforts of troubleshooting, which is not the main intent for this thesis. The internal based execution, the threshold hooks for alarms, and KPI values are more suitable for continuous and unmanaged troubleshooting, which makes them preferable for this thesis software. Primarily due to the low operational overhead they require.

5.5 Source criticism

Mainly primary sources have been used and the research area has been referred along with what has been discovered by talking to Ericsson employees possessing knowledge related to the area touched by the thesis work. The assumption throughout the work have been that not all knowledge has been available beforehand. Therefore making the work of collecting additional information an ongoing process. Leading to some dead ends and sharp turns along the way towards finalizing the thesis.

In regards to the informal interviews performed, the work has always strived to verify the information gathered. Either by using multiple internal sources and documentation or by referring to academic research within the area. If the reliance upon the informal interviews had been known prior to the thesis a more formal and documented approach would have been employed.

5.6 The work in a wider context

The software developed in this thesis may not be of outer importance to the test or customer environment in its current form. But the conclusion of this thesis could in the greater perspective be used to improve the stability and reliability of communication infrastructure, critical to the modern society. By enhancing the capability for outer edge communication nodes to implement their own troubleshooting, greater accuracy when emitting alarms can be achieved. Resulting in shorter response times when faults occur and in the long term more reliable communication networks.

---

18 According to the informal interviews performed internally on Ericsson.
19 Internal documentation
20 http://en.wikipedia.org/wiki/Performance_indicator
Chapter 6

Conclusions

After the analysis of the results, together with the knowledge gathered in associated areas, following conclusions in regards to the research questions for this thesis can be presented.

Regarding the first research question; Can the Active Probing approach for fault detection and localization be applied on a single node in an IP network? we can say yes. Based on the result found in the step 2 evaluation of the main implementation and general experience with the development of a software. Active probing can be used from the perspective of a single node to detect and localize connectivity related faults in an IP network. Two things to remember is; the limiting of information access by confining the gathering process to one node and, the loss of multiple probing perspectives.

Which leads us to the second research question; How useful is the information from the Active Probing approach on a single node in a IP network? as recently mentioned, the amount of information will be reduced by the single node perspective. The question then become; how much will it be reduced and in relation to what? The work has shown that given the occurrence of connectivity related faults the usefulness of the approach exists. But as the amount of faults decline, so does the value of it put in relation to the time and effort required to acquire it. The usefulness of the information can in the test environment be compared to corresponding information from test logs. Step 3 showed additional information given was marginal. On the other hand, when looking at the second context included in this thesis, the customer environment, the outlook was quite different. In the customer environment, little to none information regarding the connectivity of the node is available. Hence, all information gathered possess a higher value in such situation. Therefore, the usefulness of the information depend largely on the knowledge we already possess in the current environment, conclusion being that software located on the node in the test environment cannot compete with the information advantage of the simulation machinery.

The final question; How would a potential solution be applied within a customer network? has been previously discussed. Despite the absence of tests in a customer network, the majority of findings points towards a promising adaptation of the researched approach within the customer environment. The heterogeneity of the network, lack of current information, the general physical inaccessibility to faulty units and sheer size of the environment would all promote a single node troubleshooting software in addition to the already existing centralized solutions. It is a reasonable conclusion but we cannot be fully assured until such a study has been performed.

Regarding the usefulness of this approach in the test environment; the action of deploying this kind of solution would require a fair amount of adaptation of the current test cases. This due to the complexity and the dispersion of the way tests are written. There is no single construction which allows the software to be run at optimal time and without causing extensive amount of extra work. In a future where the practical hurdles of using the tool have been overcome it could prove to be valuable. But until that point when the structure of the test environment, as a whole, may allow the precision calling only when faults occur. The software would likely not perform sufficiently to compensate for the increased resource
usage in regards to time and network load. Hence, the use of this tool in the test environment can, at present, not be clearly justified currently.

To summarize, the node is not the most suitable place utilized for troubleshooting within the test environment, due to the vast amount of information available, outside the node. A more preferable troubleshooting can be made outside of the node to a greater extent, by locating the implementation approach inside test library itself, with full access to all existing information.

As future work several suggested paths can be of interest. In order to make use of the conclusions of the thesis within the test environment the implementation could be adapted to fit within the test library for troubleshooting connectivity related faults. Making use of the active probing approach, except, make the implementation within a test library class which is connected to the unit and performs the troubleshooting with the information accessible both on the node and in the test environment. A solution like this would overcome a multitude of issues present in the current incarnation of the software.

A second possible continuation could be evaluation of the approach within the customer network. And further, focus on the use of probes for creating sustainable knowledge gathering of feature status. For example use the probe implementations from SmokePing (Oetiker & Tyni 2013) and look at NTP and LDAP services but also analyzing additional services relevant from the single node operational perspective. Additionally the Bayesian network implementation could be improved also by adding knowledge of the current state of the node into the algorithm; in order to further streamline the iterative fault detection and localization process.
References


IEEE 802.3 Ethernet working group, 2015. IEEE 802.3 Ethernet. Available at: http://www.ieee802.org/3/.


Python Software Foundation, 2014. Python 2.7.8. Available at: https://www.python.org/download/releases/2.7.8/.

Python Software Foundation, 2015. unittest — Unit testing framework. Available at: https://docs.python.org/2/library/unittest.html.


Appendix A

Scenarios

A.1 Scenario 1 - Missing connectivity to remote client

Setup:
- Two client machines are set up along with a node unit.
- Each client machine are connected over a network of L2 switches (will not show up when using traceroute) to the node over separate VLANs. This means that the clients cannot communicate directly, but traffic them between have to be routed via the node.
- The node is configured to be able to forward traffic between the two separate VLANs.
- The clients are configured to use default route over the same VLAN for all traffic.

Test:
- A ping is sent from client A to client B, returns true with low latency.
- Connectivity engine is run once on the node to explore the connectivity.
- Client B sets its VLAN interface connecting it to the node, to down.
- (A ping is sent from client A to client B, returns false.) Not req.
- Connectivity engine is run once on the node to explore the connectivity.

Expected result:
When comparing the connectivity of the node a difference will be present, the node cannot ping client B. Traceroute returns nothing since we have a level two network. And the interface previously used to connect to B from the node is still up on the node.

Conclusion:
This scenario shows that the connectivity engine together with a matching engine can localize fault outside of the node by using the tools available to the node itself.
A.2 Scenario 2 - Missing connectivity to remote client, point to fault location

Setup:
- One client machine is set up along with a node unit and a router.
- The client and the node are connected via the router.

Test:
- A ping is sent from client to node, returns true with low latency.
- Connectivity engine is run once on the node to explore the connectivity.
- The client sets its vlan interface connecting it to the node, to down.
- Connectivity engine is run once on the node to explore the connectivity.

Expected result:
When comparing the connectivity of the node a difference will be present, the node cannot ping client. Traceroute returns reachability to the router and the interface previously used to connect to communicate between the node and client is still up on the node.

Conclusion:
This scenario shows that the connectivity engine together with a matching engine can propose an localization of where a fault outside of the node exists by using the tools available to the node itself.
Appendix B

Produced code Prototype implementation

This appendix contains an overview of the code produced in the prototype implementation.

B.1 Connectivity engine
The connectivity engine should help to figure out as much information as possible about the connection. It should, in a performance optimized way, use all the tools available on the node to do so.

B.2 Demo runner
The demo runner is written to be able to demonstrate how the connectivity engine, together with the stub of the matching engine, fulfill Scenario 1.

B.3 Matching engine
The matching engine provides a way of comparing two vectors generated by the connectivity engine.
Appendix C

Main implementation evaluation tests

This appendix contains an overview of the test used to verify and evaluate the main implementation.

C.1 Step 1 – Unit test
The unit tests (Python Software Foundation 2015) verify that, according to the dependency matrix, both implementations output the state, defined as right for the exact probe outcome. For example, if the result of ping-probe = 1, hasNTP-probe = 1, and the remaining probes are implicit 0; then, the output is s1 - destination unreachable.

The same principle is used to verify each state. A unit test suite was constructed, one for each of the two implementations.

C.2 Step 2 – Verification test
The verification tests were written in java, using the test- and lab APIs. The verification tests were written by me for the single purpose of verifying the specified features of the thesis’ software in the test environment.

- testStateDestinationNotReachable, should verify the ability to detect when a destination found could not be reached, and also verify that everything was OK on a parallel destination. Referring to testing “s1 - destination unreachable”. The fault was induced by disabling the interface on the destination side.

- testStateInterfaceIsDown, should verify that the “s2 - interface is down” was properly detected. This was done by disabling an interface on the node. And thereafter, check if the software was reporting this issue on the correct interface and associated destinations.

- testStateNTPServerNotReachable, should verify that a “NTP server destination down” could be detected. The actions were to configure the NTP server properly, then run the software. Verifying that “s4 - NTP server destination down” was not reported; and then, run a second time with the NTP server destination misconfigured.

- testStateAutonegotiationNotConfigured, should verify the auto negotiation mismatch issue; this, by configure the destination interface to fixed full duplex and the node unit side with auto negotiation.

<table>
<thead>
<tr>
<th>Verification test case</th>
<th>Verified state</th>
</tr>
</thead>
<tbody>
<tr>
<td>testStateDestinationNotReachable</td>
<td>s1 - destination unreachable</td>
</tr>
<tr>
<td></td>
<td>s7 – OK</td>
</tr>
<tr>
<td>testStateInterfaceIsDown</td>
<td>s2 - interface is down</td>
</tr>
<tr>
<td>testStateNTPServerNotReachable</td>
<td>s4 - NTP server destination down</td>
</tr>
<tr>
<td>testStateAutonegotiationNotConfigured</td>
<td>s3 - Autoneg conf. CC ok</td>
</tr>
<tr>
<td></td>
<td>s5 - Autoneg not conf</td>
</tr>
<tr>
<td></td>
<td>s6 - Autoneg conf. CC not ok</td>
</tr>
</tbody>
</table>
C.3 Step 3 – Evaluation test

Test cases were chosen for evaluating the software in the test environment. They were used in the continuous verification of developed software and are targeted at verifying the features being developed. The tests were written in java using the test and lab APIs; just like the ones used in C.2.

There were many parameters to consider before choosing the test cases for this step; this, so that we could get as good chance as possible in the evaluation of the tool. The test should make use of ping probes, configure and send traffic on at least one link, include one or more virtual clients and preferably, not require node reboots.

Approximately 30 test cases from 3 suits were chosen; each containing all tests for verifying one feature. The test suits were associated with the node features IPForwarding, OSPFv2 and VirtualRouters. Due to the time required; in addition to run the full suits multiple times, one test from each test suite was chosen for 100 invocations each.

Those cases were:

- `testCorrectRouteUsedAfterPortFlapping` from IPForwarding.
- `testRunningMultipleProcessesIndependently` from OSPFv2.
- `testForwardingInSeveralRouters` from Virtual router.
Appendix D

Results main implementation

The results below are the evaluation outcomes of the software from the test environment.

D.1 Step 1 – Unit test
All clear states were accurately localized by providing the probe results associated with them.

D.2 Step 2 – Verification test
Each test case was executed multiple times and all but one was a successful.

- testStateDestinationNotReachable was showing great accuracy in localizing, both for the “destination unreachable”, and the “OK” state.
- testStateInterfaceIsDown, was showing great accuracy in localization of the “interface down state”.
- testStateNTPServerNotReachable was showing great accuracy in the localization of both “OK” state, and “NTP server destination down” state.
- testStateAutonegotiationNotConfigured, “Autoneg not conf”- and “Autoneg conf. CC ok” were verified; but the case of “Autoneg conf. CC not ok” could not be verified within the test framework, due to inability of the current software version to operate in half duplex mode1. Meaning that the duplex mismatch scenario could not be replicated with current lab setup in a test case.

D.3 Step 3 – Evaluation test
Both implementations were showing identical results, hence the distinction between the implementations will be omitted in the presentation of the results.

<table>
<thead>
<tr>
<th>test case</th>
<th>invocations</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>testCorrectRouteUsedAfterPortFlapping</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>testRunningMultipleProcessesIndependently</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>testForwardingInSeveralRouters</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In the second table we can observe the full set of tests that were used when evaluating step 3. Only one, “testOverlappingStaticRouteTableInDefaultAndVR”, of all 32 test cases showed deviation from the fifth state.
<table>
<thead>
<tr>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPForwardingTest</td>
</tr>
<tr>
<td>testConfigIPForwardingWithMultipleRoutes</td>
</tr>
<tr>
<td>testDefaultGateway</td>
</tr>
<tr>
<td>testDutDiscardsPackets Destined To Network And Directed Broadcast Addresses Of Own Interface</td>
</tr>
<tr>
<td>testDutDiscardsPackets For Mac Broadcast Not Specifying Ip Broadcast Or Ip Multicast</td>
</tr>
<tr>
<td>testDutDiscardsPackets With Invalid Destination Address</td>
</tr>
<tr>
<td>testDutDiscardsPackets With Invalid Source Address</td>
</tr>
<tr>
<td>testNullRoutesDefaultRoute</td>
</tr>
<tr>
<td>testNullRoutesOrderWithCOMCli</td>
</tr>
<tr>
<td>testNullRoutingByStack</td>
</tr>
<tr>
<td>testRoutingLoopPreventionDroppedPackets</td>
</tr>
<tr>
<td>testStaticRouteSelectionBasedOnAdminDistance</td>
</tr>
<tr>
<td>testStaticRouteSelectionBasedOnPrefixLength Of Destination Subnet</td>
</tr>
<tr>
<td>testStaticRouteStillWorks When NextHop Is Changed</td>
</tr>
<tr>
<td>OSPFv2</td>
</tr>
<tr>
<td>testHoldDownTimer</td>
</tr>
<tr>
<td>testVerifyOSPF Does Not Reduce Arping Performance</td>
</tr>
<tr>
<td>testHandling Of Mtu Mismatch Dut Using Jumbo Frames</td>
</tr>
<tr>
<td>testLosDetection P2P</td>
</tr>
<tr>
<td>testRunning Multiple Processes Independently</td>
</tr>
<tr>
<td>testManyNegotiated Processes And Many Configured Interfaces</td>
</tr>
<tr>
<td>VirtualRouter</td>
</tr>
<tr>
<td>testMaximum Interface And Vr Configuration</td>
</tr>
<tr>
<td>testPing From And To Legacy And Virtual Router</td>
</tr>
<tr>
<td>testReconfiguration Vr</td>
</tr>
<tr>
<td>testTraceroute From And To Legacy And Virtual Router</td>
</tr>
<tr>
<td>testUntagged And Tagged Interfaces In Legacy And Virtual Router</td>
</tr>
<tr>
<td>testForwarding In Several Routers</td>
</tr>
<tr>
<td>testHttpsAccess Using Oam Ap And VR</td>
</tr>
<tr>
<td>testNetconf</td>
</tr>
<tr>
<td>testDeleteIP In VR And Default Router And Delete VR With IP Configure</td>
</tr>
<tr>
<td>testMoveIP Interface Between Virtual Router And Default Router</td>
</tr>
<tr>
<td>testOverlapping Static Route Table In Default And VR</td>
</tr>
<tr>
<td>testState Propagation</td>
</tr>
<tr>
<td>testArp To The Same IPAddress In Different Virtual Router</td>
</tr>
<tr>
<td>testMove Address Between Virtual Router And Default Router</td>
</tr>
</tbody>
</table>
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Christopher Pettersson