Linköping Studies in Science and Technology

Thesis No. 1231

Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks

by

Kalle Burbeck

Submitted to Linköping Institute of Technology at Linköping University in partial fulfilment of the requirements for the degree of Licentiate of Engineering

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2006

Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks

by

Kalle Burbeck

February 2006
ISBN 91-85497-23-1
Linköping Studies in Science and Technology
Thesis No. 1231
ISSN 0280-7971
LiU-Tek-Lic-2006:12

ABSTRACT

Critical networks require defence in depth incorporating many different security technologies including intrusion detection. One important intrusion detection approach is called anomaly detection where normal (good) behaviour of users of the protected system is modelled, often using machine learning or data mining techniques. During detection new data is matched against the normality model, and deviations are marked as anomalies. Since no knowledge of attacks is needed to train the normality model, anomaly detection may detect previously unknown attacks.

In this thesis we present ADWICE (Anomaly Detection With fast Incremental Clustering) and evaluate it in IP networks. ADWICE has the following properties:
(i) Adaptation - Rather than making use of extensive periodic retraining sessions on stored off-line data to handle changes, ADWICE is fully incremental making very flexible on-line training of the model possible without destroying what is already learnt. When subsets of the model are not useful anymore, those clusters can be forgotten.
(ii) Performance - ADWICE is linear in the number of input data thereby heavily reducing training time compared to alternative clustering algorithms. Training time as well as detection time is further reduced by the use of an integrated search-index.
(iii) Scalability - Rather than keeping all data in memory, only compact cluster summaries are used. The linear time complexity also improves scalability of training.

We have implemented ADWICE and integrated the algorithm in a software agent. The agent is a part of the Safeguard agent architecture, developed to perform network monitoring, intrusion detection and correlation as well as recovery. We have also applied ADWICE to publicly available network data to compare our approach to related works with similar approaches. The evaluation resulted in a high detection rate at reasonable false positives rate.

This work has been supported by the European project Safeguard IST-2001-32685 and CENIIT (Center for Industrial Information Technology) at Linköping University.

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Acknowledgement

First of all I would like to thank Simin Nadjm-Tehrani, my advisor. Without your guidance and support, this work would not have been possible. I am also grateful for all the fun we have had together during the Safeguard project. Too bad I did not take a picture when we exited the subway in Barcelona. Or when the storm forced us to sleep on the floor at a London airport and we experienced an overload of a critical communication infrastructure first hand when everybody tried to call home.

Thanks to all colleges at RTSLAB for discussions and support. Keep the *fika* going or I will be forced to haunt you with my home made cakes. Special thanks go to Anne Moe, for your support with administrative problems, travels and organisation of events. Thanks also to Lillemor Wallgren, Britt-Inger Karlsson and Inger Norén for administrative help. Thanks to TUS for help with technical issues.

This work was financially supported by the European project Safeguard IST-2001-32685 and CENIIT (Center for Industrial Information Technology) at Linköping University. Taking part in a large international project has sometimes been frustrating but most often instructive, challenging and fun. I am glad that I got the opportunity to take part in Safeguard.

I would like to thank Tomas Lingvall, Thomas Dagonnier, Mikael Semling and Stefan Burschka and their colleagues at Swisscom for fruitful discussions and their many hours of work with the test network. Thanks also to Tomas for help with the Preprocessor and data generation.

The Safeguard agent architecture has been developed with the input from all the research nodes of the project, the cooperation of whom is gratefully acknowledged. Special thanks to David Gamez and John Bigham from Queen Mary, University of London and Oleg Morajko at AIA in Spain. Thanks to Wes Carter, our project coordinator.

Thanks to Daniel Garpe and Robert Jonasson for your work with the agent platform evaluation. Thanks to Tobias Chyssler for your work with alert correlation engines. Also thanks to Tobias and Daniel for your help with implementing the correlation agent and for your company and many discussions during those hectic months of implementation phase in the project. Thanks to Sara Garcia Andrés for your first implementation of the simulation for our initial work on survivability modelling. I would like to thank Henrik Larsson and Karin Ring for reading my thesis with fresh eyes.

Doing PhD-studies while being a father of two wonderful small girls is not always easy. You have to learn to work very focused to get the maximum out of

those hours in your office, so that you also have time to spend with your family at home. I would like to thank my dear wife and very best friend Malin for all her help and support. Not the least for those weeks when conferences and project meetings have taken me far away from home. I love you with all of my heart.

Thanks to Alva and Linnea for being such clever, cute and funny girls. Even when life sometimes is harsh, you often manage to make me smile. Thanks to my parents and Malin's for your help with the girls and your support. Thanks also to our cuddly black cats Mashlul and Korlash, for lying and purring in my lap while I was writing those last hard chapters in the thesis.

In context of my family I also would like to give special thanks to my advisor for her support not only with my licentiate studies, but also for supporting me in my private situation. Thanks for helping me being home those months with my girls in the middle of my studies.

In the end I would like to thank all my friends and my family for my years of fun in Linköping. I will always remember those years as a very good time of my life. I dedicate this work to you all.

Kalle Burbeck

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The number of computers on the Internet is steadily increasing. According to the Internet Systems Consortium Inc. (ISC) the number of advertised hosts on the Internet were approaching 320 000 000 in January 2005 [63]. Figure 1.1 shows the general trend from 1998 to 2005.

The introduction of technologies such as 3G and pervasive computing make even mobile phones and other devices connected to the net. Most organisations and companies depend heavily on the use of networking for their internal organizational processes as well as for providing their services. Important examples are governments, banks and E-commerce businesses. Some service sectors, not traditionally dependent on Internet are also foreseen to become more dependent on communication networks due to technology development. Examples are power networks, health systems and disaster relief management. Of course this implies that ordinary users are also increasingly dependent on Internet services. One study shows that the number of Americans using some form of on-line banking system has grown from 14 millions March 2000 to 53 millions as of September 2004 (44 percent of all U.S Internet users) [65].

1.1 Motivation

As the number of users and services on the Internet grows, the motivation for misuse grows accordingly. Unfortunately the effort required for compromising a computer system is decreasing by the use of automated attack tools. A common set of events following the detection of a vulnerability is as follows.

- A vulnerability is detected.

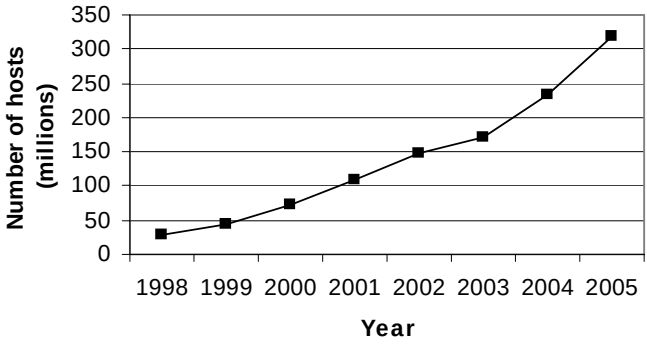

Figure 1.1: Number of hosts on the Internet advertised in domain name servers

- A malicious (or just curious) person/organisation obtains information on the vulnerability
- An attack tool is developed (requiring a considerable amount of technical insight)
- The attack tool is released and used by many (requiring little actual knowledge)
- The software vendor obtains information on the vulnerability
- A patch for the software is developed by the software vendor
- The patch is released and applied to a subset of systems removing the vulnerability from those systems.

The order of these events is very significant. The listed order is the most unfortunate since the attack tool is released before the patch is applied to end user systems resulting in a potentially very large number of compromised systems. Unfortunately the time from public discovery of a new vulnerability to the release of an attack tool is decreasing and is currently in the order of days. This means that the time window for developing and applying patches is becoming very short. One example is the Zotob-A worm [47] and its variants. On Tuesday 9th August 2005 Microsoft released a patch and less than three days [39] later exploit code was publicly available on the Internet. In four days (Saturday 13th) worms exploiting the vulnerability were spreading.

Rapid patching is important [76] but not sufficient. For a production system continuous patching may not be viable due to system complexity and diversity as well as compatibility requirements. For important systems, defence in depth is needed incorporating many different security technologies [112]. This may include firewalls at network boundaries and on individual hosts, removal of unused software and services, virus scanners and so on. To further harden the defence, intrusion detection systems may be applied.

Intrusion detection systems look for traces of computer misuse by examining data sources such as program or user behaviour, network traffic or logs. When traces of misuse are detected, alerts are produced and manual or automatic response may be initiated. Specific attacks may not be visible in every type of data source and diverse approaches may provide complementing information. It therefore makes sense to use multiple intrusion detection sensors either in isolation or preferably also combining their output by correlating the alerts.

The main detection scheme of most commercial intrusion detection systems is called misuse detection, where known bad behaviours (attacks) are encoded into signatures. Misuse detection is only able to detect attacks that are well known and for which signatures have been written.

An alternative approach is anomaly detection where good (normal) behaviour of users or the protected system is modelled, often using machine learning or data mining techniques. During detection new data is matched against the normality model, and deviations are marked as anomalies. Since no knowledge of attacks is needed to train the normality model, anomaly detection may detect previously unknown attacks. If an attack tool is published before a patch is applied and before attack signatures are developed or installed, the anomaly detection system may be the only remaining defence. Some attack types, including a subset of denial of service and scanning attacks, alter the statistical distribution of system data when present. This implies that anomaly detection may be a general and perhaps the most viable approach to detect such attacks.

1.2 Research challenges

A fundamental problem of intrusion detection research is the limited availability of appropriate data to be used for evaluation. Producing intrusion detection data is a labour intensive and complex task involving generation of normal system data as well as attacks, and labelling the data to make evaluation possible. If a real network is used, the problem of producing good normal data is reduced, but then the data may be too sensitive to be released to other researchers publicly.

Learning-based methods require data not only for testing and comparison but also for training, resulting in even higher data requirements. The data used for training needs to be representative for the network to which the learning-based method will be applied, possibly requiring generation of new data for each deployment.

Classification-based methods [40, 83] require training data that contains normal data as well as good representatives of those attacks that should be detected, to be able to separate attacks from normality. Producing a good coverage of the very large attack space (including unknown attacks) is not practical for any network. Also the data needs to be labelled and attacks to be marked. One advantage of *clustering-based methods* [57, 84, 90, 101] is that they require no labelled training data set containing attacks, significantly reducing the data requirement. There exist at least two approaches.

When doing *unsupervised anomaly detection* [57, 90, 101] a model based on clusters of data is trained using unlabelled data, *normal as well as attacks*. If the underlying assumption holds (i.e. attacks are sparse in data) attacks may be detected based on cluster sizes, where small clusters correspond to attack data. Unsupervised anomaly detection is a very attractive idea, but unfortunately the experiences so far indicate that acceptable accuracy is very hard to obtain. Also, the assumption of unsupervised anomaly detection is not always fulfilled making the approach unsuitable for attacks such as denial of service (DoS) and scanning.

In the second approach, which we simply denote *(pure) anomaly detection* in this thesis, training data is assumed to consist *only of normal data*. Munson and Wimer [84] used a cluster-based model (Watcher) to protect a real web server, proving anomaly detection based on clustering to be useful in real life. The anomaly detection algorithm presented here uses pure anomaly detection to reduce the training data requirement of classification-based methods and to avoid the attack volume assumption of unsupervised anomaly detection. By including only normal data in the detection model the low accuracy of unsupervised anomaly detection can be significantly improved.

In a real live network with connection to the Internet, data can never be assumed to be free of attacks. Pure anomaly detection also works when some attacks are included in the training data, but those attacks will be considered normal during detection and therefore not detected. To increase detection coverage, attacks should be removed from the training data to as large an extent as possible, with a trade-off between coverage and data cleaning effort. Attack data can be filtered away from training data using updated misuse detectors, or multiple anomaly detection models may be combined by voting to reduce costly human effort.

An intrusion detection system in a real-time environment needs to be fast

enough to cope with the information flow, to have explicit limits on resource usage, and adapt to changes in the protected network in real-time. Many proposed clustering techniques require quadratic time for training [69], making real-time adaptation of a cluster-based model hard. They may also not be scalable, requiring all training data to be kept in main memory during training, limiting the size of the trained model. We argue that it is important to consider scalability and performance in parallel to detection quality when evaluating algorithms for intrusion detection. Most work on applications of data mining to intrusion detection considers those issues to a very limited degree or not at all.

One fundamental problem of anomaly detection in general is the false positives rate. In most realistic settings normality is hard to capture and even worse, is changing over time. This implies that in addition to facilitate modelling the normality of a very complex system, an anomaly detection scheme needs to adapt over time.

1.3 Contribution

Many different anomaly detection schemes have been evaluated by other authors, but not all aspects of anomaly detection is getting the attention it deserves. Two such aspects are adaptability and performance. The primary contribution of this thesis is the design and implementation of the ADWICE (Anomaly Detection With fast Incremental Clustering) algorithm with the following properties:

- Adaptation - Rather than making use of extensive periodical retraining sessions on stored off-line data to handle changes, ADWICE is fully incremental making very flexible on-line training of the model possible without destroying what is already learnt. When subsets of the model are not useful anymore, those clusters can be forgotten.
- Performance - ADWICE is linear in the number of input data thereby heavily reducing training time compared to alternative clustering algorithms. Training time as well as detection time is further reduced by the use of an integrated search-index.
- Scalability - Rather than keeping all data in memory, only compact cluster summaries are used. The linear time complexity also improves scalability of training.

When performing anomaly detection and improving performance by using a search-index, detection accuracy can be influenced by the index. In this thesis we

discuss how, and to what extent, index errors influence anomaly detection results.

The application of ADWICE anomaly detection has been demonstrated in a test network setup at a telecom company (Swisscom), and its performance found to be satisfactory in the tested scenarios.

1.4 List of publications

The work that has resulted in this thesis has been presented in the following publications:

- K. Burbeck and S. Nadjm-Tehrani, **ADWICE: Anomaly Detection with Real-time Incremental Clustering**, in Proceedings of 7th International Conference in Information Security and Cryptology (ICISC 2004), Lecture Notes in Computer Science, Volume 3506, pages 407–424. Springer, 2004.

- K. Burbeck and S. Nadjm-Tehrani, **Adaptive Real-Time Anomaly Detection with Improved Index and Ability to Forget**, in Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops, Workshop on Security in Distributed Computing Systems, pages 195–202. IEEE Computer Society, 2005.

- K. Burbeck, D. Garpe, and S. Nadjm-Tehrani, **Scale-up and Performance Studies of Three Agent Platforms**, in Proceedings of International Performance, Communication and Computing Conference, Middleware Performance workshop, pages 857–863. IEEE Computer Society, 2004.

- T. Chyssler, S. Nadjm-Tehrani, S. Burschka, and K. Burbeck, **Alarm Reduction and Correlation in Defence of IP Networks**, in Proceedings of the 13th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2004), pages 229–234. IEEE Computer Society, 2004.

- D. Gamez, S. Nadjm-Tehrani, J. Bigham, C. Balducelli, T. Chyssler, and K. Burbeck, **Safeguarding critical infrastructures**, chapter 18 in the book *Dependable Computing Systems: Paradigms, Performance Issues and Applications* edited by H. B. Diab and A. Y. Zomaya. John Wiley & Sons, 2005.

The following paper was peripheral to the work in the thesis, written in collaboration with a masters project.

- K. Burbeck, S. G. Andres, S. Nadjm-Tehrani, M. Semling, and T. Dagonnier, **Time as a Metric for Defence in Survivable Networks**, in Proceedings of the Work in Progress session of 24th IEEE Real-Time Systems Symposium (RTSS 2003), 2003.

1.5 Thesis outline

The thesis is divided into 8 chapters, as follows:

- Chapter 2 presents basic terminology. It introduces many concepts related to intrusion detection, such as components of intrusion detection systems, different types of detection schemes and metrics used for evaluation. The chapter also introduces the notion of software agent and a number of basic data mining concepts.
- Chapter 3 presents the context in which ADWICE was developed. The Safeguard project as well as the Safeguard agent architecture is introduced. A short overview of different types of Safeguard agents is given. The Safeguard agent platform is presented and communication performance is compared with other platforms.
- Chapter 4 presents the first implementation of ADWICE using the original search index. Training and detection are described and evaluated.
- Chapter 5 presents ADWICE augmented with the grid index. The workings of the new index are described and evaluated. The notion of forgetting is introduced and adaptation of the ADWICE model described and evaluated.
- Chapter 6 presents the implementation of ADWICE as part of a Safeguard agent. The requirements and design of the agent are described together with practical problems, such as remote data access.
- Chapter 7 presents related published work. Alternative approaches to anomaly detection and are summarised and compared to ADWICE. In addition, a number of agent systems are discussed in the context of the Safeguard architecture.
- Chapter 8 concludes the thesis and indicates possible future directions of continued research.

Chapter 2

Background

In this chapter we explain the basic notions related to the work in this thesis. Sections familiar to the reader may be skimmed. Note however, that other authors may provide slightly different definitions due to the lack of consensus in security and intrusion detection terminology. In most cases this will not prevent understanding of the rest of this work.

2.1 Dependability and computer security

Since we use computer based systems in our every day life and for many critical applications, it is important that we can trust those systems to carry out their services in a dependable way. No complex system is perfect and we can not completely control the environment of a system. In other words, there will be *internal faults* (e.g. bugs) and *external faults* (e.g. attacks, accidents). Faults may lead to an *error*, a situation where the system state is no longer correct. Errors may cause *failures*. A failure is the event that occurs when the delivered service of a system, deviates from the correct service. Since we want the system to perform satisfactorily even in presence of faults there is a need for methods that tolerate faults or detect and recover from faults before they cause failures.

Dependability is "the ability (of a system) to deliver service that can be justifiably trusted" [8]. An alternative definition provided by Avizienis et al. [8] is "the ability of a system to avoid service failures that are more frequent or severe than is acceptable". Dependability is an integrated concept that encompasses the following attributes [8]:

- *Availability* - readiness for correct service.

- *Reliability* - continuity of correct service.
- *Safety* - absence of catastrophic consequences on the users and the environment.
- *Integrity* - absence of improper system alternations.
- *Confidentiality*[1] - absence of unauthorised disclosure of information.
- *Maintainability* - ability to undergo modifications and repairs.

Survivability is a concept very close to dependability and will be used interchangeably in this work. Ellison et al defines survivability as "the capability of a system to fulfil its mission in a timely manner, in presence of attacks, failures and accidents´´. [41] An alternative definition of survivability by Shirey [103] is "the ability of a system to remain in operation or existence despite adverse conditions, including both natural occurrences, accidental actions and attacks on the system".

Computer security is often considered the composite of the attributes confidentiality, availability and integrity (CIA) [53]. There are also a number of useful secondary criteria related to security [8]:

- *Accountability* - availability and integrity of the identity of the person who performed an operation.
- *Authenticity* - integrity of a message content and origin, and possibly some other information such as the time of emission.
- *nonrepudiability* - availability and integrity of the identity of the sender of a message or of the receiver.

When a user accesses a computer system, personal and sensitive information may be logged which raises privacy issues. This is true for intrusion detection systems [77] in particular, where data collection is necessary to detect intrusions. Below we define the important notion of privacy.

- *Privacy* - confidentiality of personal information.

A *security policy* is a set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical

[1] In the context of dependability, confidentiality has not received as much attention as other attributes [8].

system resources [103]. An *attack* is an *attempt* to violate the security policy of a system. Depending on the location where the attack is initiated it is denoted *external* or *internal*. Bace [11] defines an *intrusion* to be "intentional violation of the security policy of a system". This implies that the notion of attack, but not intrusion, encompass failed as well as successful attempts. However, in the area of intrusion detection the notions of attack and intrusion are sometimes used interchangeably [12]. In this thesis we will use the term attack, and explicitly state if the attack was successful or not if relevant.

2.1.1 Attack types

There are various types of attacks, and many taxonomies [109] of attacks have been proposed. We adopt the classification used by the DARPA/Lincoln Labs evaluation [75]. This classification does not cover all possible attack types but is enough for our purposes.

- *Scanning attacks* are commonly performed by human attackers or malicious code such as worms to gain information of network topology, what traffic firewalls will let through, open ports on individual hosts and software types and versions. A scanning attack is often the first step to gain enough information to continue the real attack.

- *System penetration attacks* involves obtaining unauthorised access to a system or network and possibly use this access to break the system (attack against availability), obtain secret information (attack against confidentiality) or altering sensitive data (attack against integrity). Some common system penetration attacks are:

  - *Remote to Local* (R2L) when the attacker obtains unauthorised local user access remotely on the network.
  - *User to root* (U2R) that involves a normal user obtaining unauthorised root access. For external attackers, gaining normal user access (e.g. R2L attack) is often a first step towards root access.

- *Denial of Service* (DoS) is when the attacker attacks the availability of a system, either by exploiting flaws to make software crash, or by flooding the target system with enough traffic so that normal service is no longer possible. When multiple hosts, possibly tens of thousands, are involved in a DoS attack, it is called Distributed Denial of Service (DDoS). DoS and

certainly DDoS may be very hard to defend against, unless countered close to the source (e.g. by the Internet Service Provider).

2.2 Intrusion detection

Amoroso [1] defines *intrusion detection* as "the process of identifying and responding to malicious activity targeted at computing and networking resources".

The use of the word *process* emphasises the involvement of technology, people and tools and the complex interaction between those entities. *Identification* may be done before, during or after the malicious activity proceeds. *Response* may be initiated by tools, people, or both. If the malicious activity is detected on an early stage, the activity may be prevented and the consequences avoided. If the activity is detected after it has completed, at least the amount of damage may be estimated and the need for future preventive actions analysed. Including *computing and networking resources* in the definition restricts the discussion to protection of such systems, even though the general principles may also apply in other settings, e.g. physical protection.

Note that failure potential is very high in the case of system security. A system may be very reliable and have a very small risk of failing due to non-malicious faults. However, when malicious intent is present, the attacker will make sure that the very rare failure condition is established [1].

A very simple illustration [1] of the intrusion detection concept is presented in figure 2.1. The attacker aims at breaking the security of a *target system* which is *monitored* by an intrusion detection system. When the intrusion detection system detects malicious activity the *intrusion detection infrastructure* is notified. The infrastructure may include additional software systems that collect, store, share, and use intrusion-related system information. Most often it also includes people and organisations. The infrastructure may then choose to *respond* to the intrusion. Response is a very complicated process, that needs to take into consideration a large number of technical and non-technical issues [1].

2.2.1 Components

Different types of intrusion detection systems have different strengths and weaknesses. It may therefore be useful to deploy multiple detection systems, to increase protection of the target system. Furthermore, intrusions often involve multiple organisations, people and systems increasing the need for a common information format for intrusion detection [118].

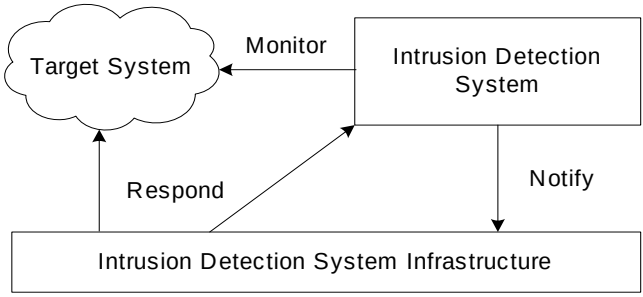

Figure 2.1: Basic intrusion detection concepts

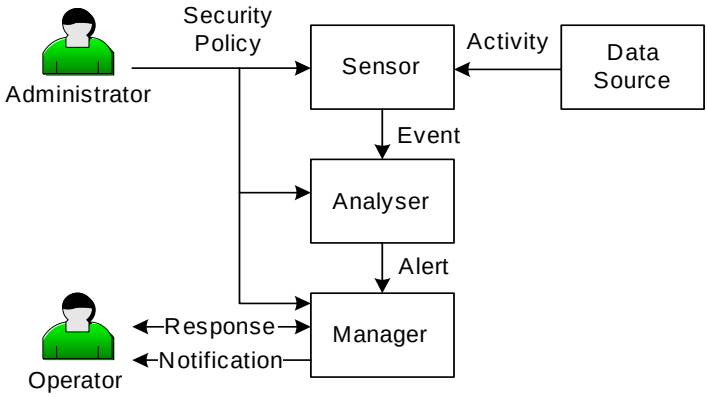

Figure 2.2: Terminology of IDMEF

The Common Intrusion Detection Framework (CIDF) [26] was an attempt by the US government's Defence Advanced Research Projects Agency (DARPA) to develop an intrusion detection system interchange language. Although its practical application has been limited to date, it has influenced later research and standardisation efforts.

The Intrusion Detection Exchange Format Working Group (IDWG) [66] has developed a common standard for intrusion detection data formats and exchange procedures known as the Intrusion Detection Message Exchange Format (IDMEF) [34, 118]. We adopt the terminology used by IDMEF illustrated by figure 2.2.

- An *administrator* is responsible for setting the security policy of the organisation, including deployment and configuration of intrusion detection systems.

- A *data source* is an access point in the target system where data is collected.
- A *sensor* collects data from data sources, detects events and forwards those to an analyser.
- An *analyser* processes the events and possibly produces alerts according to the security policy. The alerts may be formatted according to the IDMEF format.
- A *manager* handles the various components of the intrusion detection system. This may include configuration, data consolidation and notifications of the operator when certain alerts have occurred.
- An *operator* is the primary user of the manager and responsible for initiating responses to notifications and alerts.

Notice that in many implementations a sensor and analyser may be part of the same component. Furthermore, often there will be multiple sensors and analysers possibly accessing different data sources.

2.2.2 Taxonomy

Unfortunately there is still no consensus on the terminology when it comes to classification of various types of intrusion detection systems. Debar et al. [36] have however provided one of the most cited intrusion detection taxonomies up to date. We here adopt the revised version [35] [37] with some further extensions. The added notion of *modelling method* is inspired by the use of the notions *self-learning* and *programmed* in the taxonomy by Axelsson [10]. The order of categories has been updated according to Arvidson and Carlbark [5]. Figure 2.3 presents the resulting taxonomy. In the following sections we explain the taxonomy.

Audit Source Location

An intrusion detection system may collect data from many different audit sources. Often raw data is processed and specific features extracted, suitable for further analysis. The kinds of attacks that may be detected depend on the data source as well as the selection of features. Many audit sources contain significant quantities of noise. Sometimes audit sources are used because they are already present in the target system, rather than because they are very suitable for intrusion detection.

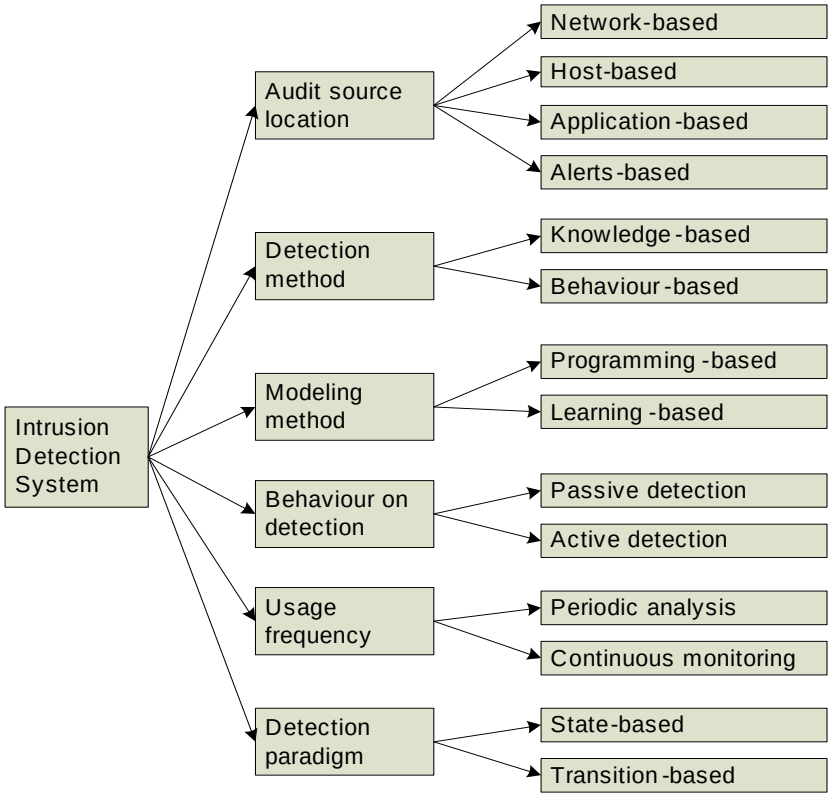

Figure 2.3: Intrusion detection taxonomy

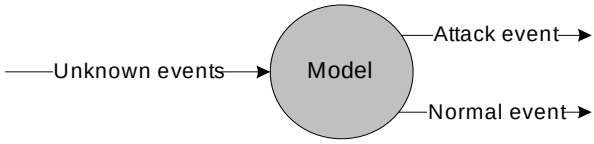

Figure 2.4: Categorisation of unknown events

Most computers are connected to local networks and/or the Internet and many applications are communication intensive. Whenever the attacker does not have physical access to the target system, she will need to access the system remotely using the network. This implies that attacks may be detected by analysing network traffic, denoted *network-based* intrusion detection. Raw network traffic (packets) may be captured by listening passively on all network traffic using a network card configured into promiscuous mode.

Modern operating systems offer different capabilities to collect logs of system and resource usage that may be analysed by a *host-based* intrusion detection system. Host-based data sources are the only way of gathering information of user activity on individual machines. A common source of host-based information is C2 security audits in UNIX corresponding to detailed information on the system calls of user processes.

An alternative to host-based detection is *application-based* intrusion detection. Application logs may provide more accurate and relevant information than low-level operating system data.

When intrusion detection systems are applied on a larger scale with many deployed systems, the resulting alerts may in turn be processed by *alert-based* intrusion detection systems. By *correlating* alerts relating to the same event detection accuracy can be improved by the combined information from multiple detectors. Furthermore, the total number of alerts that needs to be analysed may be decreased if alerts relating to the same event are grouped together.

Detection method

To successfully detect attacks, the intrusion detection system needs to be able to differentiate attacks from normal data using an internal *model* of attacks and/or normal data. Figure 2.4 illustrates how a model categorises unknown events into attacks or normal data. Naturally the model may be more or less specific, possibly being able to categorise subtypes of attacks and normal data, e.g. correctly separating a Blaster worm attack from a port scan.

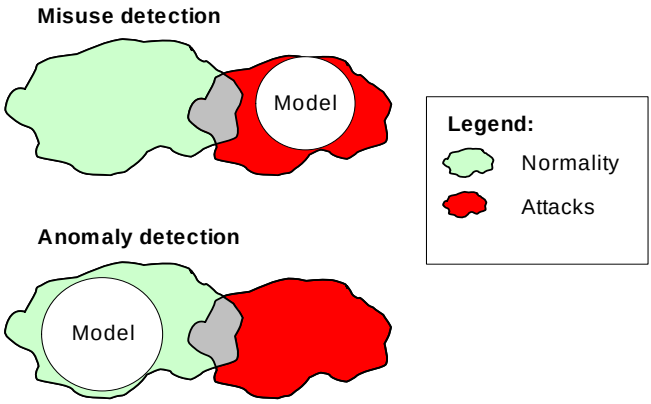

Figure 2.5: Misuse detection versus anomaly detection

Intrusion detection depends on the assumption that the data source has access to data where attacks are different from normal data. Moreover, this difference needs to be captured by the events produced by the sensor and by the model of the analyser to correctly produce alerts when attacks are present in the data. Two common approaches exist:

- *Misuse detection*, also known as *Knowledge-based intrusion detection*, which uses an internal model of attacks (misuse) to detect intrusions.
- *Anomaly detection*, also known as *Behaviour-based intrusion detection*, which uses an internal model of normal data to detect *anomalies* (i.e. deviations from normality).

Figure 2.5 illustrates the relation between misuse detection and anomaly detection. It is of course possible to include both normal and attack data in the model. We refer to such systems as *hybrid detectors*.

Modelling method

A *learning-based* intrusion detection system learns by examples how to build its internal model. For *programming-based* intrusion detection systems, the users of the systems specify how to separate attacks from normal events. Most learning-based systems perform anomaly detection, while misuse detection systems normally are programmed. We refer to programmed anomaly detection as *specification–based* intrusion detection [100].

Behaviour on detection

Passive intrusion detection generates alerts but does not actively apply countermeasures to thwart an attack upon detection. *Active* intrusion detection systems on the other hand are able to initiate countermeasures. Countermeasures may include cutting network connections that carry attacks, blocking traffic or reconfiguring equipment and software such as firewalls or routers. Active intrusion detection requires good detection accuracy, or else false alerts could cause the countermeasures to decrease the availability of the system.

Usage frequency

Intrusion detection may perform analysis periodically or continuously. Intrusion detection systems performing *continuous monitoring* acquire information of events immediately after they occur and process those events in real-time. We will refer to such analysis as *real-time* intrusion detection. Periodic analysis implies an additional time delay until the attack is detected but requires fewer resources such as processing power.

Detection paradigm

State-based systems detect intrusions by analysing system states. Examples of this could be a file integrity checker (e.g. Tripwire [64]) using checksums to confirm that no files have been changed and that files therefore are in their normal state. An alternative approach is to detect transitions from a normal state to a failure state, for example detecting the actual attack by using signatures (e.g. Snort[2]).

2.2.3 Evaluation metrics

Intrusion detection systems are often evaluated on data containing attacks as well as normal traffic. The data may be simulated or collected from real networks.

A number of commonly used metrics exist, and are described below.

- A *true positive* (TP) is a real attack correctly categorised as an attack by the intrusion detection system.
- A *false positive* (FP) is a false alert, meaning that the intrusion detection system erroneously raised an alert for normal data.

[2]Snort is an open source network intrusion prevention system, capable of performing real-time traffic analysis and packet logging on IP networks. [105]

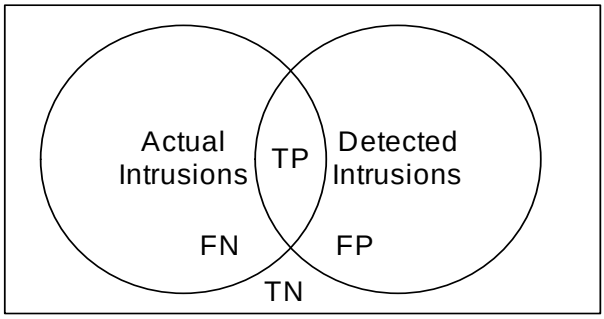

Figure 2.6: Evaluation metrics

- A *true negative* (TN) is normal data that correctly does not generate an alert from the intrusion detection system.
- A *false negative* (FN) is a missed attack, meaning that the attack was erroneously categorised as normal by the intrusion detection system.

Figure 2.6 shows the relation between true and false positives, and true and false negatives.

Relative metrics are often more useful for comparison and a number of derived metrics are therefore normally used for evaluation:

- *Detection rate* (DR) is the fraction of all attacks that are actually detected.

$$\text{Detection rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.1}$$

- *False positives rate* (FPR) is the fraction of all normal data that produces (false) alerts.

$$\text{False positive rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{2.2}$$

- *Overall accuracy* is the fraction of all data that is correctly categorised.

$$\text{Overall accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}} \tag{2.3}$$

Often a combination of detection rate and false positives rate is presented when evaluating an intrusion detection system. Accuracy or Detection rate in

isolation would be a useless metric. The straightforward method for producing 100 % detection rate would be to generate one alert for every encountered data. The intrusion detection system then detects all attacks, but of course the real attacks would be hidden in a huge number of false alerts.

Real computer or network data from normal usage is expected to contain only a small fraction of attack data. This means that a low false positive rate is critical. Assume that an intrusion detection system located inside a firewall processes 1 000 000 network packets each day and only 10 of those data are real attacks. Further assume that the false positive rate of the intrusion detection system is 1 % and the detection rate 90 %. During one day 10 009 alerts would be produced, 10 000 false positive and 9 true positive. It would take a significant amount of manual effort to find those 9 true alerts. This illustrates the difficulty of intrusion detection [9].

There is a well known trade-off between false positives rate and detection rate. Increasing the detection rate usually means also increasing the false positives rate. Flexible intrusion detection systems provide the opportunity for the user to adapt this trade-off (e.g. using a threshold) to obtain higher detection rate or lower false positives rate (not both) depending on the present need. Producing a diagram with detection rate on the Y-axis and false positives rate on the X-axis, known as a *ROC-curve*, is an illustrative way of presenting this trade-off. Figure 2.7 presents an example ROC curve. The form of the curve is typical for intrusion detection systems, where increasing false positives rates give diminishing returns in increasing detection rates.

When performing intrusion detection, not only accuracy is important, but also time. *Detection latency* measures the time from the attack to detection. For some detection schemes there may be a trade-off between detection latency and detection accuracy. Detection of a port scan may aggregate network packets over a specific time window. If this time window is large, this may increase detection latency. If the time window is small, stealthy (slow) port scans may not be detected because very little anomalous traffic is visible inside the time-window.

For use in practice, storing and processing efficiency is important. *Memory usage* measures how much memory the intrusion detection system requires for a specific model size. *Throughput* measures the amount of data analysed each time unit. To be effective for real-time detection, the trough-put needs to be higher than the input data-rate.

A sometimes neglected but important part of evaluation is *usability*. Various aspects of usability are ease of training and adaptability of learning-based systems, ease of deployment, configuration, and tuning as well as user support such as good

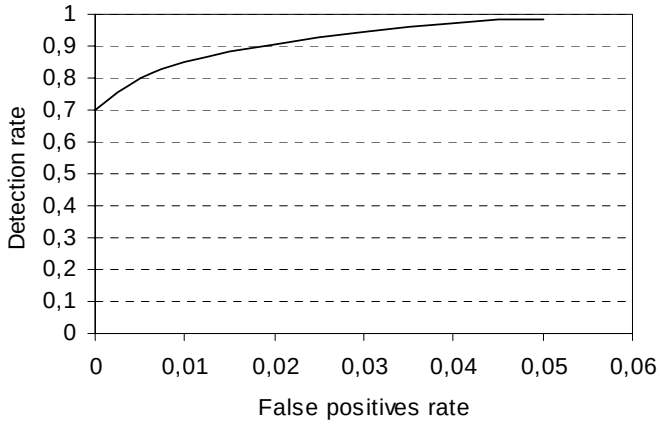

Figure 2.7: ROC-curve example

interfaces and management tools. A low false positives rate increases the usability of the intrusion detection system, since less effort is required to sort through the false alerts.

2.3 Software agents

According to Russell and Norvig [95], an *agent* is "anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors". Human agents use eyes and other organs as sensors, and hands, mouth and other body parts as effectors. Software agents exist only as encoded bit strings inside computers. Even though they interact directly only with the digital world, through their hardware they may also influence the real world (e.g. robots). But what makes software agents different from other program components? Comparing agents to objects, two important differences in general are:

- *Autonomy* - The object has state and so does the agent but the agent also incorporates the ability to initiate actions as what could be explained as *having behaviours*.

- *Independence* - In addition to having behaviours the agent is also communicating by passing messages. When using objects the communication is accomplished through method invocation with references. This means that

the control lies with the invoker, which is totally opposite the agent case where control lies with the receiver. The receiving agent decides whether to respond to a message or not.

Many computer-based systems of today are by necessity distributed. Data may be collected in one place, analysis of the data in a second, while the operator of the system is located at a third. For intrusion detection systems, analysing data from multiple hosts and even multiple networks, this is certainly true. To reduce complexity in the overall systems, traditional design principles suggest that each component should be realised as a reasonably independent subsystem implementing related functionality, communicating with related subsystems when necessary. In other words, the system should possess strong *cohesion* and loose *coupling* [19]. If subsystems are implemented as software agents, the complete system is denoted a *multi agent system* [119]. The autonomy and independence of agents map very well to the basic design principles of strong cohesion and loose coupling.

If a computer system is built using a set of agents, it makes sense to collect crosscutting functionality of agents into a middleware supporting the agent, rather than implementing this functionality as part of each agent. We call such agent middleware *agent platforms*.

2.3.1 Agent platforms

Agents need to communicate if they are to contribute to the purpose of the overall multi-agent system. They also need some other basic services like directory services to locate other agents and life-cycle management to be able to start and stop execution in a controlled way. *Agent platforms* are simply the basic middleware used by software agents.

The Foundation for Physical Intelligent Agents (FIPA) is a non-profit organisation setting up standards for agent applications and agent systems. The main contribution and goal of the organization is to achieve interoperability between agent platforms developed by different producers. Standards such as the Agent Communication Language contribute towards this goal. Another standard, the Agent Management specifications, introduces basic agent-platform concepts. The standard agent platform model set up by FIPA is pictured in figure 2.8. The platform can physically span multiple machines.

The *Directory Facilitator* (DF) is a place where agents can register their services for other agents to search for. The facility is also often named 'yellow pages' as in the phone book. Typical types of services advertised here are types of in-

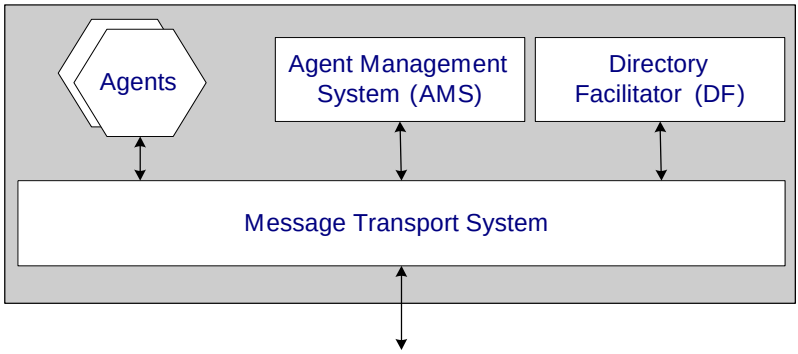

Figure 2.8: Reference architecture of a FIPA agent platform

teraction protocols supported and the language used for conversations as well as arbitrary user defined services. There is no restriction on the number of instantiated DFs on a platform.

The *Agent Management System* (AMS) is responsible for agent activities such as agent creation, life cycle and deletion as well as providing and keeping unique agent addresses. The analogy with the phone book's 'white pages' comes as no surprise. There can only be one logical AMS on an agent-platform even if it physically spans over multiple machines. We denote the platform part with AMS the *main container*.

The *Message Transport System*, also known as the *Agent Communication Channel* is where the control of message passing lies. The message exchanging parties could be agents living on the same platform but also on different platforms.

Many agent platform implementations are available on the Internet and it is no coincidence that many are implemented in Java. The platform independence and easy to use threads as well as dynamic loading of classes, make Java a suitable choice of language for many agent applications.

2.4 Data mining and machine learning

Data mining concerns "extracting or mining knowledge from large amounts of data" [69]. Data mining applies many basic learning techniques from the area of "Machine learning" [82]. A large number of different data mining and machine leaning techniques have been applied to the area of intrusion detection. Here we describe two important types of data mining approaches, *classification* and *clustering*, and how they have been used for intrusion detection.

The data source of an intrusion detection system provides some traces of system activity (see section 2.2.1). Not all system activity is relevant for intrusion detection and if the analyser was presented with unfiltered system activity, the intrusion detection task would be unnecessary hard. Therefore a number of measures or data *features* needs to be selected to represent system activity such that those features can be used to detect attacks. This initial data filtering is part of pre-processing in the domain of data mining. Pre-processing may in general include [69]:

- Cleaning - Data cleaning is the process of filling in missing values, smoothing noisy data, identifying or removing outliers and resolving inconsistencies.
- Integration and transformation - If data come from various sources, their format may be inconsistent and need to be harmonised. For some classification algorithms, numeric data need to be normalised, i.e. scaled to a specific range.
- Data deduction - Huge data sets may require a long time to analyse. By removing irrelevant attributes, aggregation, compression or generalisation of data reduction is possible.

In this thesis, we will denote the pre-processed data *feature vector*. Each dimension of the feature vector corresponds to a measure derived from system activity.

2.4.1 Classification

A classification algorithm is designed to learn (to approximate the behaviour of) a function which maps a feature vector into one of several classes by looking at several input-output examples of the function [82]. The resulting approximation of the function is denoted *classifier*. We call the function that is learnt, the *target function* and the resulting range of the function the *target concept*. The target function could in principle be specified manually for simple examples, but for large domains this will often be practically impossible.

There are many different approaches to classification, e.g. Artificial neural networks, decision trees, genetic algorithms, Bayesian learning, case based reasoning, fuzzy sets and so on. For description of various algorithms the reader is referred to text books on data mining [69] and machine learning [82].

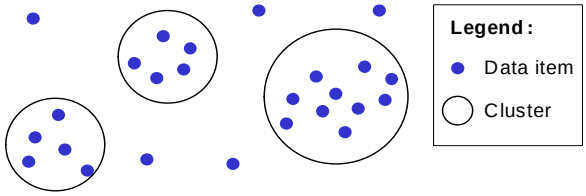

Figure 2.9: Clustering

An intrusion detection system, regardless if it is programming-based with rules or learning-based, can be considered a classifier. The classification problem of intrusion detection is to decide if data is normal or are symptoms of an attack. This implies that the straightforward target function of a classifier used for intrusion detection would have attacks classes and the notion of being normal as the target concept. The downside of this is of course that all training examples need to be labelled, involving a human expert.

An alternative approach is to use one attribute of the feature vector itself as target concept. One approach that has been evaluated is to learn to discriminate between different applications or network services during normal usage of the computer system. During detection, unknown data will be presented to the classifier and mapped into the target concept of valid network services. The proposed classification will be compared to the true service visible in the data. If the proposed classification does not match the true service of data, this tells us that the data is not similar to the training data of that service. If the training data corresponds to normal usage of the computer system this is certainly suspicious and may possibly imply that the data is part of an attack.

2.4.2 Clustering

A process of grouping a set of physical or abstract objects into classes of *similar* objects is called *clustering*. A *cluster* is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters [69]. Figure 2.9 illustrates clustering of points in a two-dimensional space.

Given two objects, represented as normalised feature vectors with continuous roughly linear variables, the similarities can be computed by considering the geometrical distance between the two vectors. A common distance measure is the well known Euclidian distance. Nominal and other types of variables can be handled using other types of distance measures.

In general, major clustering algorithms can be classified into the categories described below [69].

- *Partitioning methods* classify data into k groups, where each group or cluster contains at least one object and each object belongs to exactly one cluster. An initial partitioning into k clusters is created, after which iterative relocation is applied. The idea is to step by step improve the clusters by maximizing intra–cluster similarity, while minimizing inter–cluster similarity.

  One classical example is the *K-means* [69] clustering algorithms which represent each cluster by its mean, or *centroid*. The initial partitioning is created by randomly selecting k objects to represent the cluster means. All other objects are then assigned to the most similar cluster after which all cluster means are recomputed. This process iterates until a criterion function, such as the squared error criterion, converges.

- *Hierarchical methods* create a hierarchical decomposition of the given data objects. Cure [58] and Chameleon [71] are example of hierarchical methods. BIRCH [122] uses hierarchical clustering in its first phase, after which iterative relocation is applied to clusters rather than individual objects in subsequent phases. Two hierarchical approaches exist:

  - *The agglomerative* approach starts with each object forming a separate cluster. It successively merges similar clusters together until all clusters are merged into one at the top-most level of the hierarchy.
  - *The divisive approach* starts with all objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters until every object is in a single cluster or until a termination criterion is fulfilled.

- The general idea of *density-based methods* is to grow clusters as long as the density, the number of objects, in the neighbourhood of the cluster exceed some threshold. Contrast this to partitioning methods which are based on distance measures. DBSCAN [43] is an example of a density-based clustering method.

- *Grid-based clustering methods* quantise the object space into a finite number of cells that form a grid structure. Performing all clustering operations on the grid structure improves processing time. STING [116] is a typical example of a grid-based method.

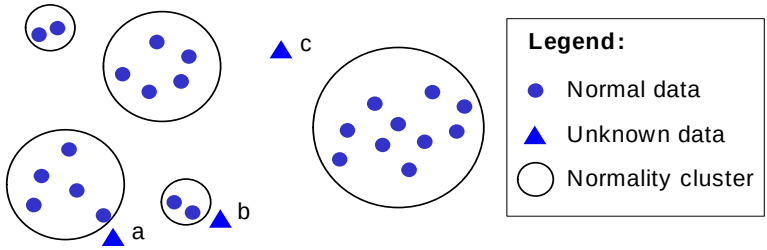

Figure 2.10: Pure anomaly detection using clustering

- *Model-based clustering* hypothesises a model for each cluster and finds the best fit of the data to that model. One approach is conceptual clustering, which given a set of unlabelled objects produces a classification of those objects. The classifications are often represented by probabilities, one example is the COBWEB [45] method. Another approach uses neural networks [82]. Self organizing maps (SOMs) [72] assume that there is some topology or ordering among the input objects and the SOMs try to take on this structure in space. SOMs have been used for intrusion detection and can also be used to visualise high-dimensional data.

There are at least three basic approaches to intrusion detection using clusters. In the first case, an example of *pure anomaly detection*, feature vectors representing normal system usage are clustered. The clusters then form a compact representation of normal system usage. When new system data are to be categorised, they are compared with the clusters. If unknown data is similar to a cluster in the model, the data is assumed to be normal. If not similar, the data is not similar to normality and therefore suspicious and may also be part of an attack. Figure 2.10 illustrates pure anomaly detection using clustering:

- Data point a and b will be considered normal because they are similar (i.e close) to clusters that represent normality.

- Data point c will be considered as part of a possibly *new* attack because it is not similar to any cluster in the model.

An alternative approach is unsupervised anomaly detection using clustering. System data, attacks as well as normal system usage data are clustered. The data points are assumed to be unlabelled, meaning that we do not know if one training data is normal or part of an attack. By assuming that attacks are far less common

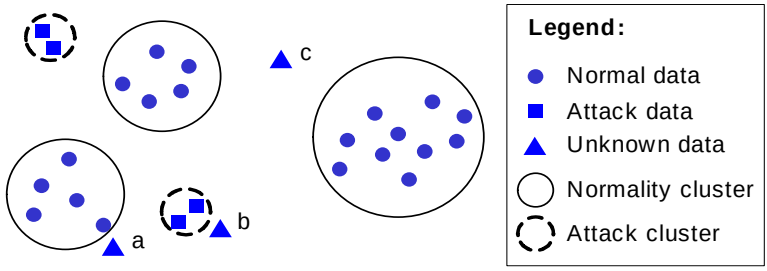

Figure 2.11: Unsupervised anomaly detection using clustering

than normal data, detection can be based on cluster size. Small clusters represent uncommon data which are suspicious and possibly part of an attack. Figure 2.11 illustrates unsupervised anomaly detection using clustering:

- Data point a will be considered normal because it is similar to a large cluster, which is assumed to be normal.
- Data point b will be considered as part of a possible attack because it is most similar to a small cluster, assumed to be part of an attack.
- Data point c can be considered as part of a *new* attack because it is not similar to any (large) cluster.

Clustering can also be used for straightforward classification. In contrast to pure anomaly detection and unsupervised anomaly detection, *labelled* training data is then clustered. Each cluster can be given the class of the majority of the contained data objects for example. When unknown data are classified, the class of the most similar cluster may be given to the unknown data. This method is a mix of misuse and anomaly detection since both attacks and normality are modelled. Figure 2.12 illustrates classification-based detection using clustering:

- Data point a will be considered normal because it is similar to a cluster where the majority of the data points are normal.
- Data point b will be considered as part of an attack because it is similar to a cluster where the majority of the data points are attacks.
- Data point c can be considered as part of a *new* attack because it is not similar to any cluster.

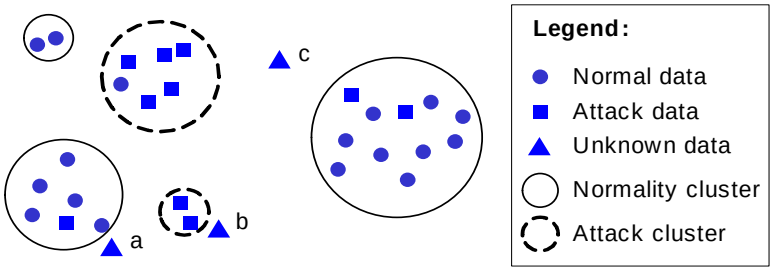

Figure 2.12: Classification based detection using clustering

Chapter 3

The Safeguard context

Most of the work in this thesis was done in the context of the European Safeguard project [96]. Safeguard (2001-2004) was a European research project aiming to enhance survivability of critical infrastructures by using agent technology. The telecom and electricity distribution domains were used to evaluate the Safeguard concepts.

This chapter gives a brief overview of the project and describes how the work in this thesis fits in the larger picture of Safeguard. The telecom domain will dominate this chapter, because ADWICE was demonstrated and successfully tested in that domain.

3.1 Critical infrastructures

Our heavily industrialised society would not survive without a generous and reliable supply of energy and communications. The energy and communications needs are supplied by what is known as *large complex critical infrastructures* (LCCIs). They are *large* because they consist of networks interconnected at local, regional, national and global levels. They are *complex* because they have to provide a range of services to a wide range of various costumers and there is rarely one single supplier. They are *critical* because our society would collapse if any of those infrastructures were unavailable an extended period of time.

However critical, those infrastructures are vulnerable to failures as well as deliberate attacks. Due to their complexity, the infrastructures depend heavily on computers and communications networks for monitoring, protection and management. The computer systems are in turn vulnerable.

In general, critical infrastructures can be divided into the following three layers:

1. *Physical layer*. This is the network of pipes, lines, cables, etc. that delivers the essential services. In the telecommunications domain the physical layer consists of the routers, switches and copper and fibre-optic lines that carry the telecommunications data. In the electricity domain the physical layer is the network of transmission lines, transformers, breakers and generators that create and transport the electrical energy.

2. *Cyber layer*. This is the computers, networks and data gathering sensors that are used to monitor and control the physical layer. In the telecommunications domain, the cyber infrastructure is used to monitor and control the many routers and switches within the system. In the electricity domain, the cyber infrastructure gathers information about power flows and breaker states and transmits the operators' control signals to the breakers and transformers. Although the cyber layer may share communications channels with the physical layer - in telecommunications, for example - the data that is transmitted within the cyber layer has a very different function from that within the physical layer.

3. *Human operations layer*. All the information gathered by the cyber layer is passed on to the human operators, who use it to manage the physical and cyber layers. The organizational processes that are in place for management, security enforcement, and recovery from failures are part of this layer.

In the past it has generally been the physical and human operations layers that have been the most vulnerable to attacks, failures and accidents. Accidents and failures in the physical layer have always been part of the daily running of the network and this layer has occasionally been subject to physical attacks as well. Within the human operations layer, operators inevitably make mistakes and they also have the specialised tools and knowledge that are needed to carry out malicious actions. It has also always been possible for attackers to gain physical access to the control room or to manipulate operators by social engineering. None of these problems have gone away in recent years, but a number of factors have contributed to a dramatic rise in the vulnerability of the cyber-layer, which has been the main area of focus for the Safeguard project.

3.1.1 Telecommunications vulnerabilities

In most European countries traditional circuit-switched telecommunication networks are being replaced by packet-based IP networks. It is therefore the manifestation of attacks, failures and accidents in IP networks that constitutes the most significant threat to telecommunications infrastructures. In recent years the dependability of IP infrastructures has decreased, due to the proliferation of worms, denial of service attacks, buffer overflow vulnerabilities and viruses. There are also the inevitable problems of hardware failures and software crashes. All of these problems are being exacerbated by the rapid deployment of hardware and software that have been designed primarily for other contexts (e.g. personal computing) within the patchwork that constitutes modern complex systems. This spread of commercial off-the-shelf hardware and software can also lead to a monoculture within which it is easy for malicious processes to spread.

In case of failures, accidents as well as attacks, it is important that problems are detected and resolved as soon as possible to increase service availability, and avoid further escalation of the problem. An important issue is to quickly and accurately identify the cause to a problem. In the context of a management network for telecom service providers we have identified the following needs:

- Reducing information overload on human operators (see sections 3.3.1 and 3.3.4). Many existing tools produce an abundance of alerts and logs.
- Increasing coverage by providing new sources of information such as anomaly detection with ADWICE (see chapter 4 and 5).
- Increasing information quality by reducing false positives (see section 3.3.4).
- Collating information, such as correlating alerts and combining with network topology information (see section 3.3.4).
- Presenting a global view of a network (see section 3.3.4).

3.1.2 Electricity vulnerabilities

The electricity cyber layer contains a number of control centres running workstations, energy management software (EMS) and databases over a local area network. These control centres interact with the Supervisory Control And Data Acquisition (SCADA) system that consists of a software interface and specialised hardware units (RTUs), which monitor sensors and interface with breakers and transformers (see figure 3.1).

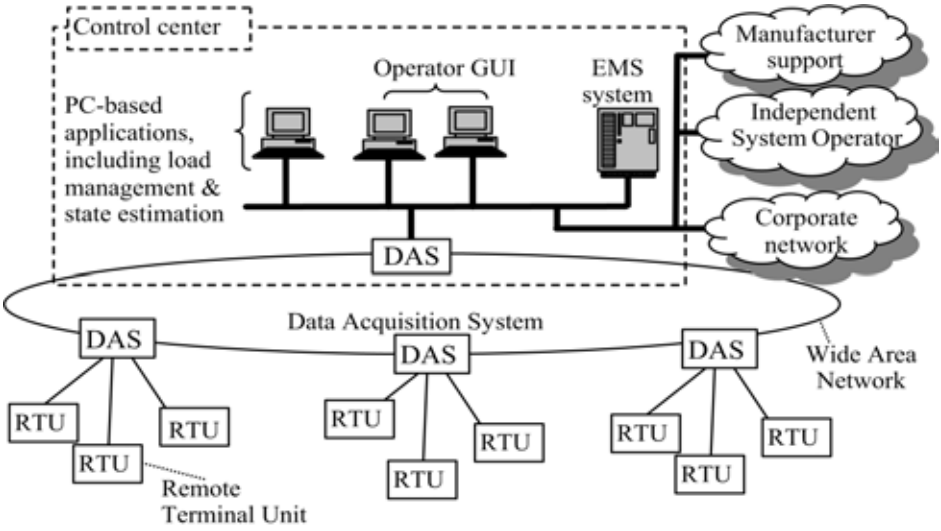

Figure 3.1: Electricity cyber infrastructure

Traditionally this cyber infrastructure was protected by its relative isolation and the non standard protocols that were used to communicate over it. However, with the deregulation of the electricity industry it no longer makes sense for each company to develop its own hardware and design proprietary protocols, and companies are increasingly looking to commercial products to solve their communication needs. A second consequence of this market orientation is the increased interconnectedness between the electricity management networks and other networks, most problematically between the corporate network and the control centre network. The dangers of this standardisation and interconnection became apparent in the recent Ohio nuclear incident [91] when the Slammer worm copied itself across from the corporate network into the plant network and disabled a safety monitoring system for nearly five hours, forcing the operators to switch to an analogue backup. In a separate incident the propagation of Slammer blocked SCADA traffic and impeded the ability of operators to monitor and control the electricity system [30].

3.2 Safeguard solutions

Within the Safeguard project we have aimed to produce a solution that can tackle some of these challenges, using an open architecture that can be extended to cope

with emerging threats. The core of this Safeguard solution is an agent system, which facilitates the distributed gathering and filtering of information from a number of different sources and the execution of rapid responses to attacks, failures and accidents.

3.2.1 Agents for increased dependability

Agents are semi-autonomous software entities that perceive their local environment, process this information in some way and carry out actions. By communicating with one another they can develop collective solutions to problems. Agents run on agent middleware (see section 3.2.2) that separates their generic services (such as communication capability, service discovery, heart beat emissions) from the specific "business" logic used for detection, correlation or action. This paradigm offers a number of advantages over a single centralised intelligence:

- *Scalability*. With a centralised system, communication bottlenecks can build up around the central controlling node as the network increases in size. Within a decentralised system the communication overhead depends upon the degree of coupling between the distributed components. However, since this overhead is distributed over the whole of the network, even quite tightly coupled distributed systems can generally be scaled more easily. The processing of information is also distributed across many nodes in agent systems.

- *Local detection and response*. The remoteness of centralised intelligence systems makes them useless in the event of a partial or complete network failure. On the other hand, a distributed agent system can carry out an autonomous rapid response in isolated parts of the network, which can perform definitive repair actions or simply "buy time" before a more lasting remedy is found. This is a substantial advantage in critical infrastructures, which are generally managed using a central control room (with backup control available at different locations).

- *Robustness*. Agent systems ideally do not have a single point of failure and can continue to operate even if a number of them are attacked or fail. This is a useful feature in dependability applications, where graceful degradation rather than rapid collapse is the optimal behaviour.

- *Emergent behaviour*. A lot of work has been done on the way in which certain forms of intelligence can emerge through large numbers of local interactions. A number of relatively simple agents can self-organise to perform

tasks that would be very difficult to create and coordinate from a central point.

- *Modularisation.* Whilst the object-oriented paradigm does facilitate a reasonable amount of modularisation, agent systems take this even further since each agent operates as an independent functioning system. Different agents can be designed by different people to handle specific tasks and only the messages between them need to be agreed. This makes it easy to set up communications between agent systems made by different companies for diverse critical infrastructures.

These advantages of agent-based systems make them a viable choice for the monitoring and protection of large critical infrastructures. However, it is the intelligence inside the agents that enables them to identify problems and react to them appropriately. The Safeguard approach to this will now be covered in the following sections starting with a description of the Safeguard agent platform.

3.2.2 The Safeguard agent platform

As explained in section 2.3, software agents have a number of common requirements. Some of the most important include life cycle management and communication. Because this functionality is common to all agents in a multi-agent system it can be provided by an agent platform.

Using an existing platform decreases the need of spending resources on developing basic agent functionality. Therefore a large number of existing agent platforms were evaluated in a preliminary agent platform survey [16]. At this point the specific requirements of the safeguard architecture were not known, rather a number of general requirements were used, including:

- *Maturity* - How mature is the platform? Is the software well used in a real world setting?
- *Activity* - How active is the project? Have new versions and bug-fixes been released recently?
- *Security* - What techniques are used to secure the agents?
- *Interoperability* - Can external software be integrated?
- *Scalability* - Can the platform manage large applications with many agents?

- *Footprint* - Are the memory and CPU usages reasonable? Can both light-weight and heavyweight agents be implemented with the platform?
- *Platform requirements* - Does the platform depend on specific operating systems and/or hardware?
- *Open source* - Is the platform implemented as open source? Can the platform be extended by a third party?
- *Documentation* - Is the platform well documented?
- *Support* -What extent of support is available?
- *Price* - Is the platform free? Are there non-commercial licences?
- *Availability* - Can an evaluation copy easily be downloaded?
- *Communication* - How is agent communication implemented?

Besides some notable exceptions, many of the surveyed agent platforms were deemed to be primarily research platforms, not mature enough to be used by the Safeguard project. Others lacked important functionality such as secure communication or were too heavy-weight for the Safeguard setting. The initial survey resulted in the selection of Jade [68] and Grasshopper 2 [55] for further evaluation. Due to very specific license restrictions concerning performance evaluations of Grasshopper, this platform was excluded from further evaluation and replaced by Tryllian ADK [108].

During the initial phases of the project a number of more specific requirements were discovered. Some wrapper agents had very high requirements on performance due to the high flow of network data in the telecommunication setting. This implied the need for an efficient platform implementation of agent communication. Other low level agents could have limited resources in terms of memory and processor speed. This led to the platform requirement that some low level agents were to be implemented in C while most high level agents were to be implemented in Java. Java was selected due to its platform independence reducing problems when agents are to be executed on different operating systems and hardware. Also it provides good support in terms of well documented libraries and also facilitates more efficient development than alternative languages.

The specific requirements led to the decision to abandon the idea to use existing agent platforms and instead implementing the Safeguard Agent Platform

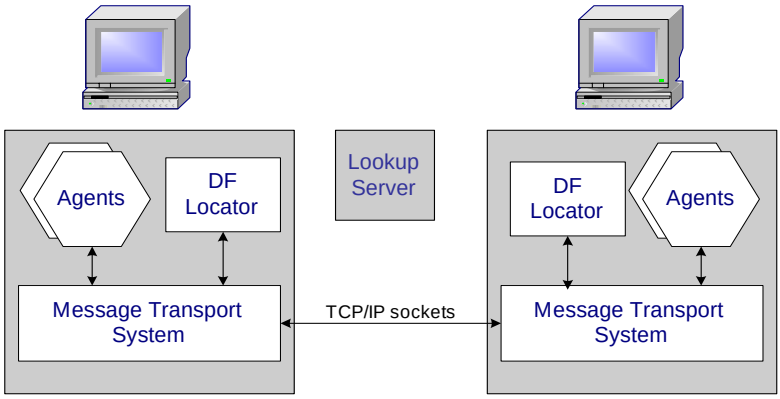

Figure 3.2: Safeguard agent platform architecture

(SAP). SAP was designed to be light-weight and also allow agents to be implemented in both C and Java. The platform provides the most important functionality needed by the agents including life cycle management, naming services, service discovery and communications. Figure 3.2 shows the architecture of SAP.

Compared to the abstract FIPA architecture presented in section 2.3.1, one difference is the lack of an explicit agent management system in the architecture. This functionality is provided by the separate lookup-server. To obtain good performance plain IP/TCP sockets are used for implementing the message transport system. The choice of implementation is abstracted away by the platform. Each agent finds other agents by searching for names and/or services. The local agent container needs only to be configured with the location of the lookup-server to facilitate this.

Performance evaluation

A final evaluation was performed to compare the implementation of communication, support for concurrency, security and performance of SAP with Tryllian and Jade. Communications performance was compared by computing round-trip time, that is, the time from Agent A sends a message to Agent B until Agent A have received an answer. Round-trip time was chosen as metric to keep time relative to a single agent. Figure 3.3 shows how the round-trip time increases for Tryllian, Jade and SAP when an increasing number of agent pairs are communicating located on separate hosts. The figure shows that SAP provides the best performance in this setting followed by Jade.

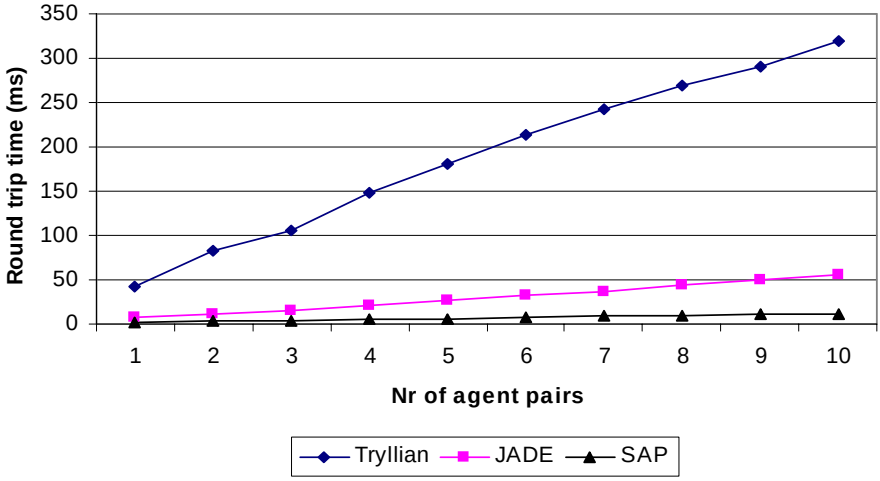

Figure 3.3: Performance of Safeguard agent platform

Figure 3.4 shows how the round-trip time increases when a very large number of agent pairs communicate. Also here SAP provides the best performance while Tryllian was not able to handle the large number of agents in the final experiment.

Full results from the platform evaluation can be found in the original paper [18] and reports [17, 51]. In general SAP provided the best performance, sometimes closely followed by Jade.

3.2.3 The Safeguard agent architecture

The safeguarding agents executing on a common agent platform are added as a fourth layer interacting primarily with the cyber and organizational levels of the critical infrastructures. Figure 3.5 shows how this is done at a conceptual level.

The Safeguard agent architecture is presented in figure 3.6. The generic roles can be described as follows:

- *Wrapper agents* wrap standard devices and existing diagnosis mechanisms in the protected network, and provide their outputs after some filtering and normalisation for use by other agents.
- *Hybrid detector agents* utilise domain knowledge for a given infrastructure, but combine it with behavioural detection mechanisms (e.g. anomaly detection with white lists) to detect known and unknown problems.

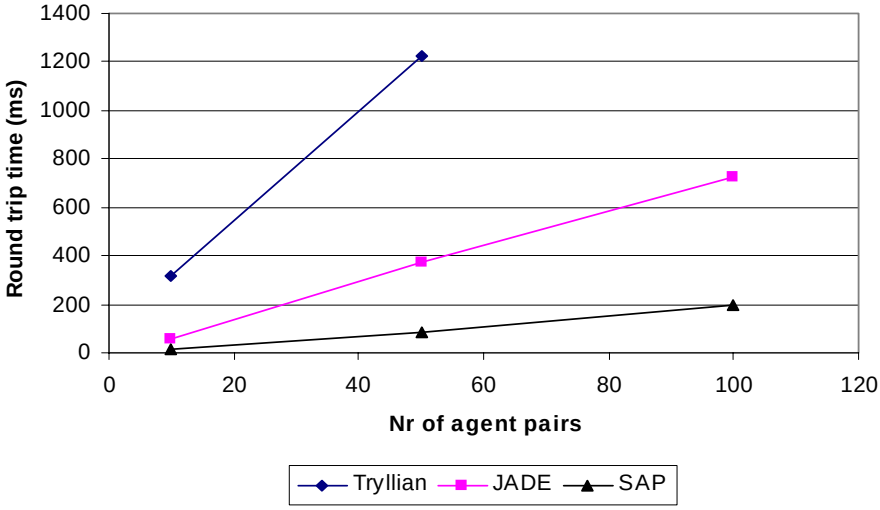

Figure 3.4: Scalability of Safeguard agent platform

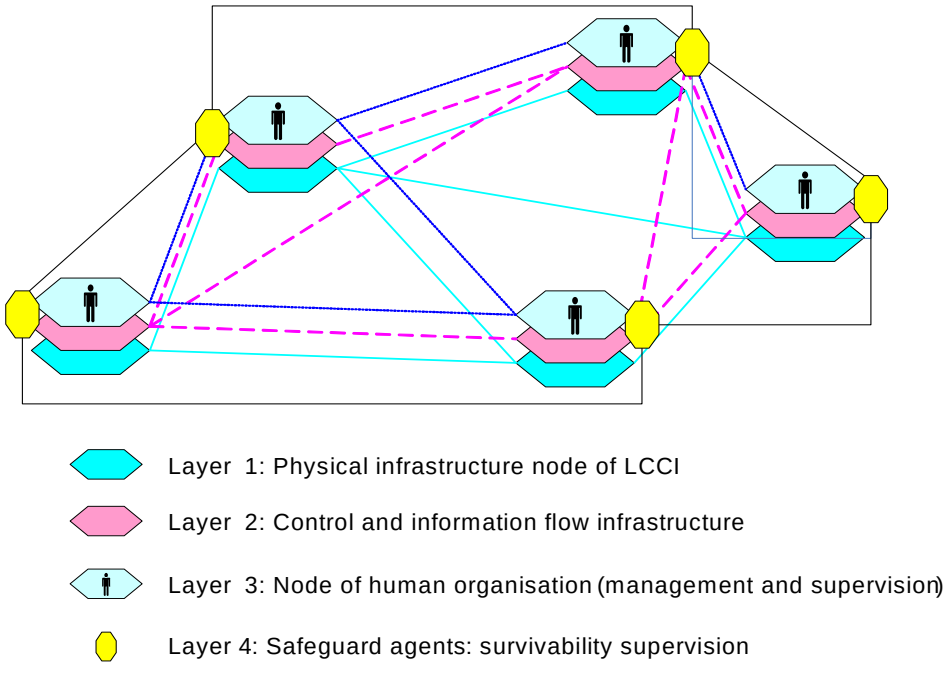

Figure 3.5: Conceptual view of Safeguard

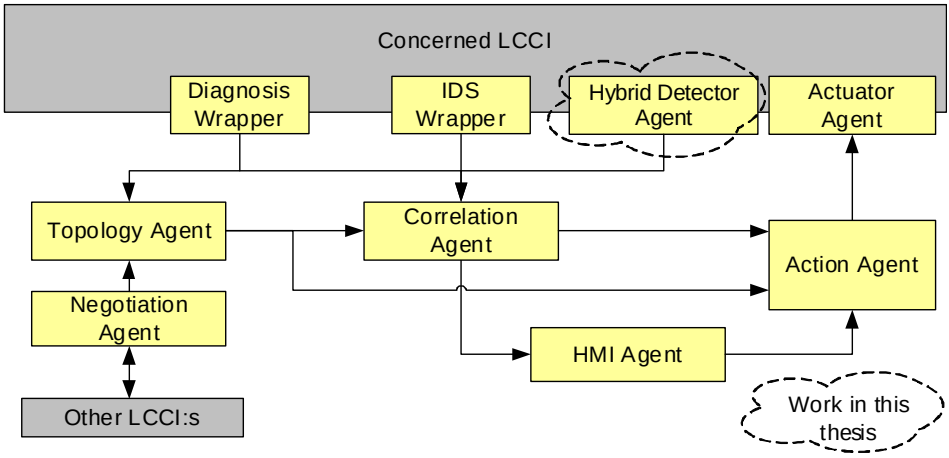

Figure 3.6: The Safeguard agent architecture

- *Topology agents* gather dynamic network topology information, e.g. host types, operating system types, services provided, known vulnerabilities.

- *Correlation agents* identify problems that are difficult to diagnose with one source of information in the network, by using several sources of information from wrapper, topology, or hybrid detector agents. They use the data sources to order, filter and focus on certain alerts, or predict reduced availability of network critical services. One type of correlation agent performs adaptive filtering and aggregation to further reduce the alert rates.

- *Action agents* enable automatic and semi automatic responses when evaluation of a problem is finished.

- *Actuator agents* are wrappers for interacting with lower layer software and hardware (e.g. changing firewall rules).

- *HMI (Human-Machine Interface) agents* provide an appropriate interface, including overview, for one or many system operators.

- *Negotiation agents* communicate with agents in other LCCIs to request services and pass on information about major security alerts.

3.3 The Safeguard agents

In this section we give an overview of the agents in the Safeguard Architecture. Some high level agents, including the topology agent, the HMI agent and the negotiation agents are generic and used in both the telecommunication and power domains. In other cases, where more specialised agent instances are needed for the two domains, we present examples of the telecommunication instances. The reader interested in specific agents of the power domain are referred to related Safeguard publications [14, 48, 97, 98].

3.3.1 Wrapper agent

A number of diagnostic tools were already available and used in both the telecom and power domains. To provide the agent system with access to those existing systems, light weight wrapper agents were devised. Examples of wrapped systems of telecom include:

- Snort - A well-known open source intrusion detection system
- Samhain - A file integrity checker
- Syslog - A very general logging tool in Unix.
- Firewalls

All those tools output large numbers of alerts. One important goal of the Safeguard agent system is to identify problems by providing a diagnosis to humans and the action agents. However, the raw output from existing tools, millions of alerts, would very quickly overload a human operator as well as higher level agents such as the action agent. The low level agents therefore need to reduce the total number of alerts as close as possible to the source, before forwarding the remaining alerts to higher levels. In particular there were a lot of uninteresting messages coming from untuned Syslog and Samhain. Practically in LCCIs, tuning requires a static network, were changes almost never happen or it requires a non-affordable effort for the administration. The resulting amounts of uninteresting messages have a low severity, though through their number of occurrence, they cannot be neglected. Therefore static filters exclude data from Samhain and Syslog that does not carry any valuable information, such as messages that Samhain checks a file or that Syslog is doing a garbage collection.

Static filters are implemented either as an ignore filter or as a delete filter. The ignore filters keep the alerts in the database to be used for forensic investigation if

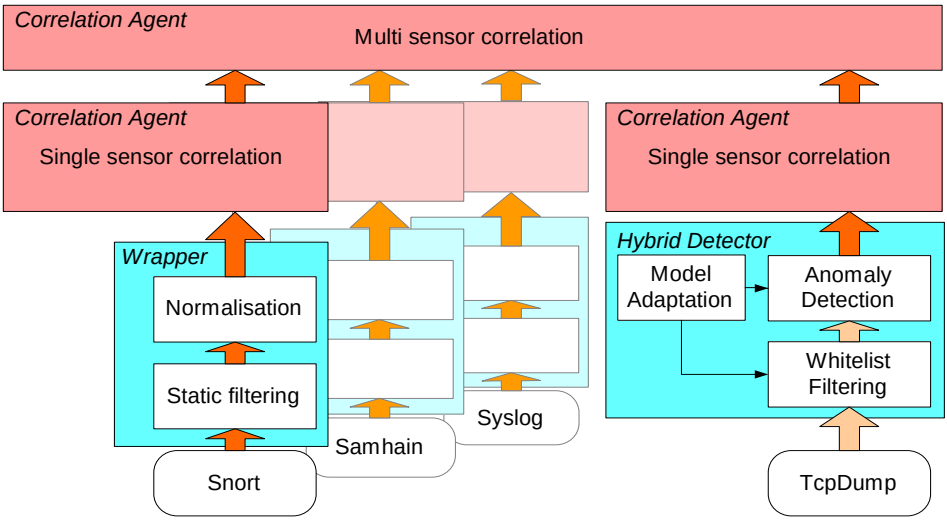

Figure 3.7: Functional overview of wrappers and related agents

needed, but they are not forwarded to high level agents. The delete filters remove alerts permanently from the database. All static filters have a limited duty cycle defined by the operator and have to be reactivated afterwards.

Another severe practical problem is the disagreement in message format of different tools. Before alerts from diverse tools can be analysed by higher level agents, their format needs to be normalised. This is another task of the wrapper agents. In the Safeguard prototype implementation a simple agent message format was devised to handle the many diverse messages handled by the agents (e.g. alerts, commands, information). In a non-prototype implementation available standards should be used where applicable, such as IDMEF (see section 2.2.1) for intrusion detection alerts.

Figure 3.7 shows how the wrapper agents take output from existing tools, perform normalisation and static filtering and forward the remaining alerts to the correlation agents. In parallel the hybrid detector agent increases coverage by providing additional types of alerts. The processing of the hybrid detection agents and correlation agents are further described in the following sections.

3.3.2 Hybrid detector agent

The hybrid detector agent combines learning-based anomaly detection with programming–based detection using signatures. Signatures can be used to quickly

detect and categorise known problems, while the anomaly detection engine can detect new unknown attacks and problems.

In the telecom domain the data rate can be very high, compared to the electricity domain. This puts hard requirements in terms of performance and scalability of the hybrid detector agent. ADWICE, the anomaly detection scheme presented in the next chapters of this thesis, as well as a separate instance of the hybrid detector agent, was implemented to fulfil those needs. Due to the cluster-based engine the agent instance was named Clustering Hybrid Detection Agent (CHDA). Further details of the implementation of CHDA are provided in chapter 6.

In addition to CHDA, a number of alternative anomaly detection approaches was evaluated by other nodes in the project in the power domain. Those include an invariant-based detection scheme [14] and a case-based reasoning [48] scheme.

3.3.3 Topology agent

The topology agent (TA) stores information about the cyber infrastructure and makes it available for the other agents running on the system. This information is gathered from databases and human operators and includes the hardware and software components, their interconnections, importance and vulnerabilities, the services they are running and the people responsible for them. This agent also provides a measure of health for the components that it is monitoring - a computer with a high CPU load and high temperature is rated as healthier than one that has a low load and is cool running. The component health measures can then be used by the correlation agent to compute network health (i.e. overall service level).

3.3.4 Correlation agent

The correlation agent ties all other agents together and has a number of complex tasks to perform. To avoid scalability problems in the telecom domain hierarchies or distribution of correlation agents can be used. In our case the first level of correlation agents (ARCA - Alert Reduction Correlation Agent) implements adaptive filtering and aggregation (i.e. single sensor correlation). The higher level agent (TACA - Topology Alert Correlation Agent) correlates the filtered alerts from different sensors as well as performs network health monitoring. Figure 3.8 shows a functional overview of the correlation agents. We now go on to explain the functions involved.

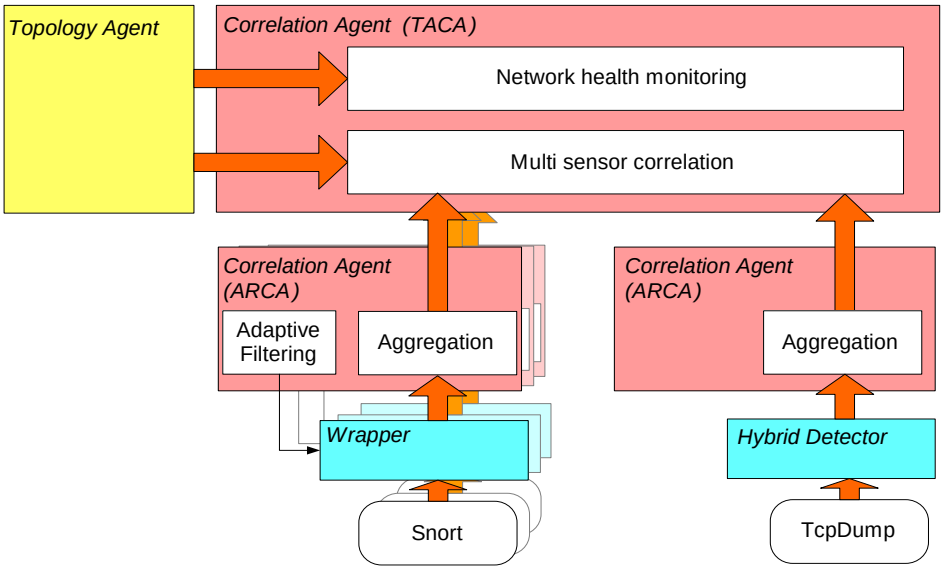

Figure 3.8: Functional overview of correlation agents

Aggregation

Repeated, identical alerts do not provide any additional information. It would reduce the information overload if all of these alerts were represented in only one alert including the number of its frequency. The relevant fields for aggregation have to be defined for each source of alerts individually. For Snort the source and destination IP, as well as the server ports and the messages are relevant. Ephemeral ports[1] should be ignored because they will vary between connections. For Syslog, the process identification number is unimportant but the program and the message field is relevant. All data is aggregated within a given time window.

Adaptive filtering

Static filtering provided by wrapper agents deals with known problems. However, since it is not possible to foresee all future misconfigurations, adaptive filtering algorithms were implemented to suggest new filter rules to the operator. This is based on Naïve Bayesian (NB) learning [61]. The idea is to train a NB text classifier to classify messages by looking at word statistics of messages. The

[1] Ephemeral ports are temporary ports assigned by the client's IP stack, and are assigned from a designated range of ports for this purpose.

NB-classifier has to be trained first with a subset of messages labelled as interesting or not. During training, a knowledge base is automatically acquired, enabling the NB-classifier to assess whether unknown messages are interesting [22]. The meaning of interesting in our approach is a message that can be useful when analysing it for signs of an attack. For example, a message saying "telnetd: Successful login as user root" or "File changed" is classified as interesting, but messages like "Syslog-ng: 54 Objects alive. Garbage collecting while idle" or "Could not resolve host name" will be classified as uninteresting (the last message is an example of a misconfiguration that should be detected and reported separated from the intrusion detection process).

The adaptive filters are used in the following workflow:

1. For performance reasons, the algorithm for adaptive filtering is launched on a periodic basis.
2. The algorithm suggests filter rules for top scoring events to a human expert via HMI.
3. The reply is also used to optimise the training corpus, achieving on-line retraining.
4. In order to avoid over learning effects, features that hardly ever occur in time are reduced in their significance and are finally forgotten.

Multi-sensor correlation

Figure 3.9 shows the high level design of TACA. Aggregated alerts from low level correlation agents are first correlated with topology information. For example: Snort analyses a network packet targeting a host and finds out that it contains a buffer overflow attack taking advantage of a Windows vulnerability. When the IP of the target host is correlated with topology information of this host, the host is found to run Unix rather than Windows. This implies that the attack failed and the severity of the alert downgraded. The resulting alerts are stored in persistent storage in the InAlert data base.

In the prototype implementation two types of correlation engines were used:

- A rule-based correlation engine was implemented to provide the agent with expert knowledge.
- A learning-based correlation engine was implemented to help focus on relevant alerts for cases when rules do not exist.

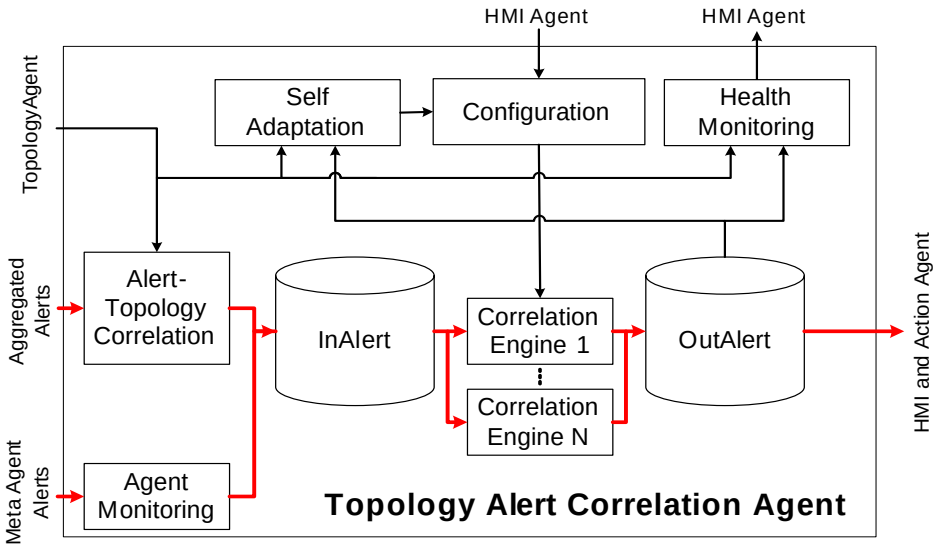

Figure 3.9: Design of TACA (Topology Alert Correlation Agent)

The learning-based engine regards all alerts concerning one host during a time-window. In other words, the selected correlation parameters are time and IP address. Given a time window, a time slot of a three dimensional[2] vector is produced by taking the sum of severities from the three sensors. Based on the input pattern, the correlator decides whether this is an interesting alert or not. For further processing just one single alert is generated. The idea is illustrated in Figure 3.10.

Three different engines were evaluated [22, 23] using time-slots:

1. A simple additive correlator with the parameter 'added severity threshold'
2. A neural network [104] with a variable set of neurons and hidden layers. Training is achieved by back propagation.
3. K-Nearest Neighbour [38] with K as a variable parameter.

The additive correlator simply builds a weighted sum of the number of occurrences of the severities within a given timeslot. The sum is then compared to a threshold generating an alert if the chosen limit is exceeded. By varying the threshold different detection rates and false positive rates can be achieved. The

[2] During evaluation three alert sources were used including Snort, Syslog and Samhain.

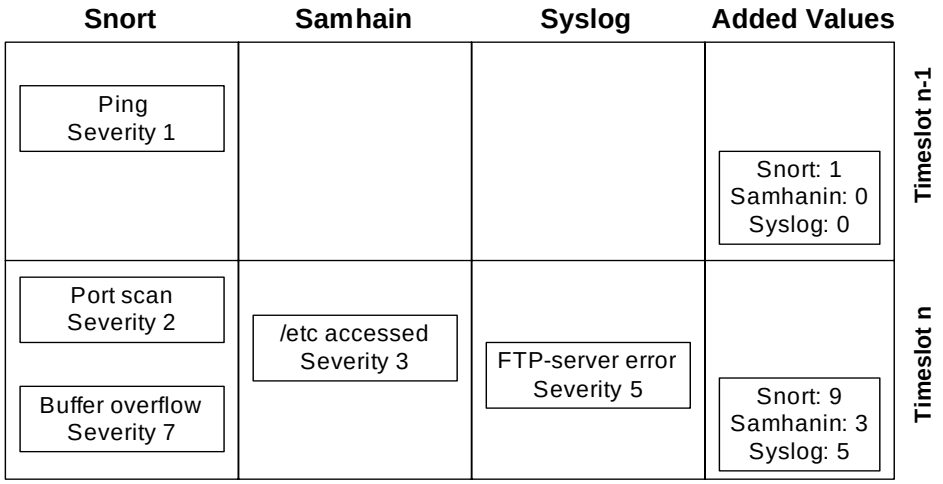

Figure 3.10: Timeslot based alert correlation

other two correlation algorithms were first trained on a subset of data before it is applied to the new real time dataflow. Further details on those correlation techniques and their evaluation can be found in work by Chyssler et. al [22, 23].

Network health monitoring

The topology agent collects and stores a lot of information. Sometimes it is useful for the operator to get a more condensed view of a host or network. This is the purpose of the network health metric. Figure 3.11 shows an example of how a condensed view of network health can be realised. The purpose of the network is to provide a number of services. Individual hosts providing services contribute to the overall service provided by the network. Network health is a condensed measure on the availability of service or service level.

For individual hosts health is in this example computed out of:

- Online/Offline status. A host that is down is considered dead with health equal to zero.
- Measures of load and memory usage. Memory consumption vary a lot during a program's life-time and therefore such measures are computed over time to even out the spikes during normal operation.

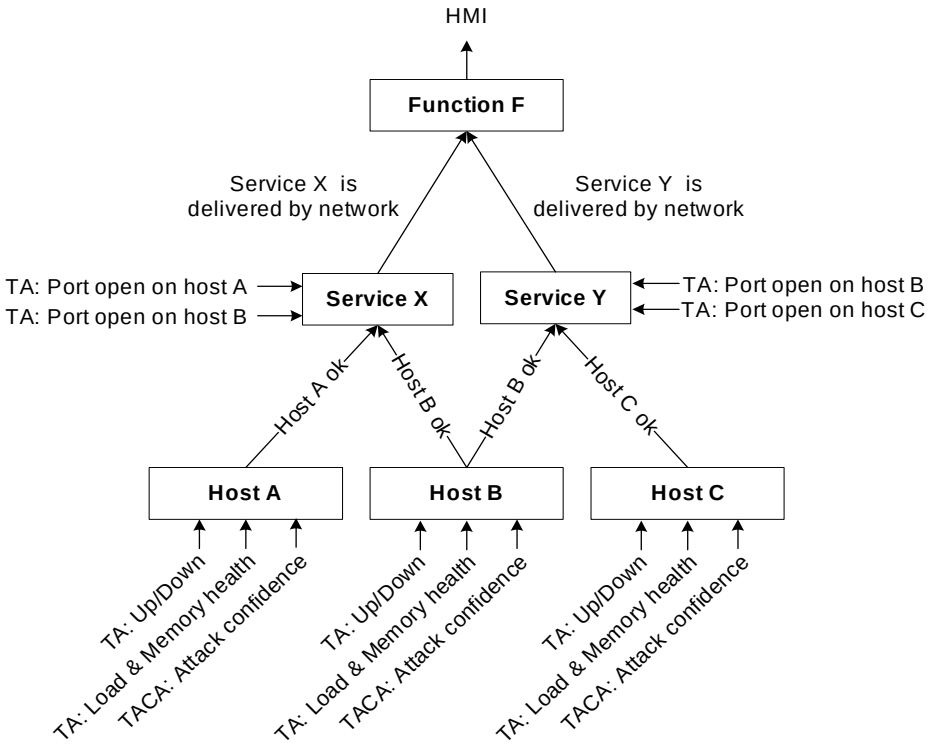

Figure 3.11: Principle of global monitoring

- Attack confidence. What is the probability that the host is currently compromised or attacked?

Individual service health is computed by taking into account if their designated ports are open or closed. A closed server port provides no service. In addition to this, the health of the hosts providing the service contributes to the overall service health. The network health is computed as a weighted sum of all services provided by the network. The HMI agent described in the next section can then show network health as well as status for individual hosts.

3.3.5 Human-machine interface agent

The Safeguard agents' autonomous actions are carried out under the supervision of a human operator. In some situations the agents will need to ask human operators for permission to carry out actions. In other situations the agents will be

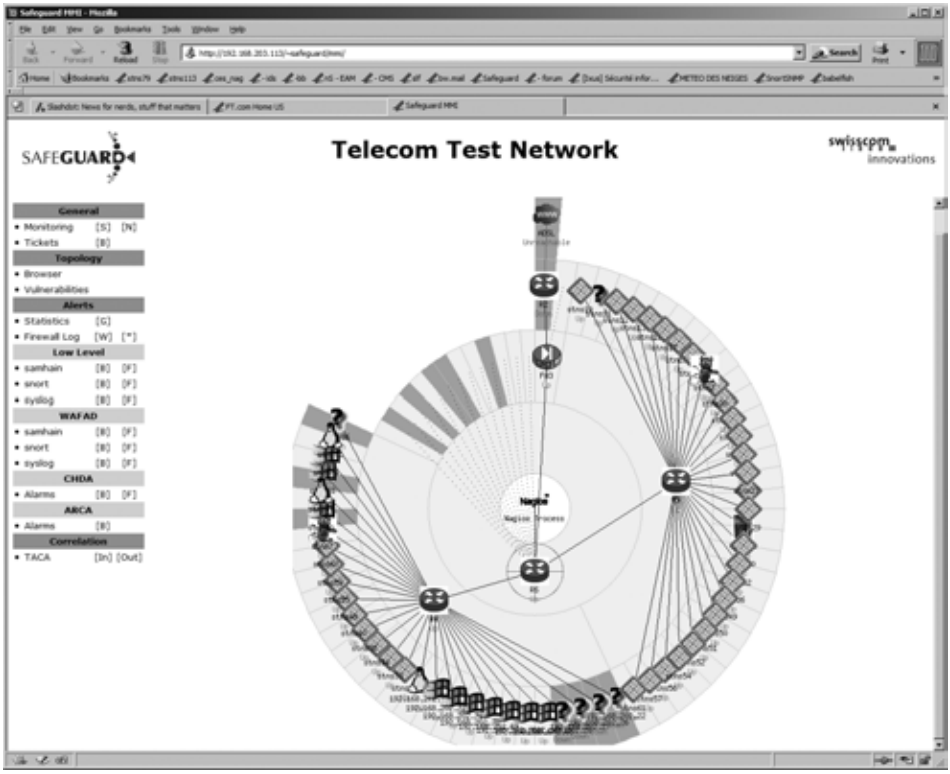

Figure 3.12: Network overview as presented by the HMI agent.

unable to take action and therefore alerts or requests for help need to be sent to the human operators. The human-machine interface agent manages all these forms of communication between Safeguard and the human operator. Figure 3.12 shows the network overview presented by the human-machine interface agent using the naglos network management tool.

HMI displays the ranking of the filtered alerts that were processed by the correlation agent and enables the human operator to monitor and control the autonomous actions of the agents. The human-machine interface agent also displays the information gathered by the topology agent about the cyber infrastructure. Figure 3.13 shows network health over time for a number of network zones.

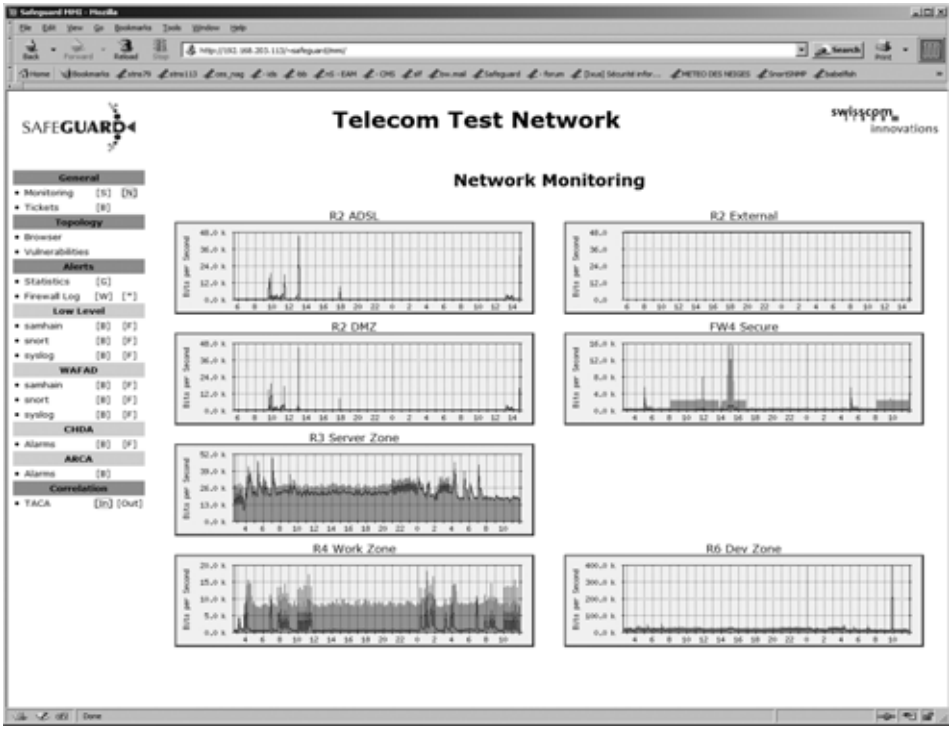

Figure 3.13: Network health monitoring of the HMI agent

3.3.6 Action agent

The primary task of the action agent is to obtain a problem description from the correlation agent, and choose a suitable action that may resolve the problem. The first implementation of the action agent consists of a two-dimensional action matrix. The action agent provides a list of capabilities in terms of possible actions. Specific known problems can then be mapped to suitable actions.

Especially in the IP domain, the actions are limited to passive defence due to the possibility of IP spoofing. In severe cases immediate clearance of a human operator is required. In general three classes of passive defence actions are possible [97]:

1. Perimeter defence: router, firewall configuration:
   - In case of configuration error:
     - Correct entry.
     - Find the SNMP script that is responsible.
   - Decrease traffic volume (graceful degradation).
   - Block or drop traffic.
   - Switch off routing protocols or static routes.
   - Redirect traffic.
   - In case of element failure:
     - Reboot.
     - Switch over to backup element.
2. Internal router and switch reconfiguration:
   - Decrease traffic volume.
   - Block or drop traffic.
   - Switch off rotten services.
   - Switch off troublesome interfaces.
   - Reboot element.
3. Host based countermeasures:
   - Reduce process priority.
   - Kill process.

- Redefine priorities of users and processes.
- Disrupt inbound or outbound traffic.
- Reboot machine.
- Switch off machine.

The action agent can be configured to perform automatic actions for situations where the administrator decides this is viable. In other cases the action agent can propose actions for the operator. Operators as well as agents sometimes make mistakes. An important functionality of the action agent is therefore to provide the opportunity to undo previous actions whenever possible.

3.3.7 Actuator agent

The actuators are low level agents which can be either controlled by the action agent or the HMI when direct human control is necessary. The latter is an ultimate requirement of operators, being in need of direct control, because the last instance is the human if everything else fails.

Actuators were in the telecom domain prototype implemented for firewalls, routers, switches, and individual hosts. Actions included are for example:

- Host shut down and reboot.
- Various file actions including removing and changing attributes.
- Various process actions including signalling and renicing[3].
- Drop, block and rate limiting of traffic.
- Adding/Removing Cron-jobs[4].

3.3.8 Negotiation agent

The negotiation agent handles the interactions with other critical infrastructures. As was discussed in section 3.1, one of the emerging problems with critical infrastructures is their increasing interdependency and the negotiation agent's main task is to minimise the risk of cascading failures by keeping the operators and

[3] Renice is a Unix utility for changing system scheduling priorities for processes.
[4] Cron is a Unix tool that facilitates periodical running of processes (i.e. jobs).

agents in other critical infrastructures informed about attacks, failures and accidents. The negotiation agent also exchanges information that can help in the day-to-day running of the networks.

In the event of a significant failure, the negotiation agent can also request services from other critical infrastructures of a similar or different type. For example, if a telecommunications network experiences a major failure, the negotiation agent can arrange for its calls to be switched through the most suitable alternative network at an appropriate price. If the data communications in an electricity network fail, more services can be requested from a telecommunications network.

3.4 Safeguard test beds

To be able to evaluate the project, two test environments were developed in parallel with the safeguard system. In the electricity distribution setting the evaluation was partially based on simulation with the safeguards running on real hosts interacting with a simulator. For evaluation in the telecommunication setting, a real test network consisting of multiple sub networks and about 50 machines was constructed, mimicking a real telecommunication management network. In this test-network real attacks and faults can be initiated.

Basically the test network consisted[5] of four zones plus additional security lab zones for the secure development and test of attacks:

- External zone: simulating the Internet in order to avoid contamination of the Internet (e.g. worms) when external attacks are to be evaluated. The external zone also provides the real Internet connection when attacks are suspended implying external partners are able to use the Safeguard test network for development and testing purposes. When the Internet connection is enabled attacks may come from the Internet and data can be collected according to the current security policy (e.g. router, firewall configuration). The external zone is connected to the Safeguard Demilitarised Zone (Safeguard DMZ) by a state-full firewall, allowing only SSH (Secure Shell) protocols into the Safeguard DMZ.

- Safeguard DMZ: In this zone all the following sub networks come together. All services relevant for inter to intra net connection are located here. All

[5]This thesis describes the test bed implementation at the end of the Safeguard project. After the project ended, the test bed was developed further and more machines included

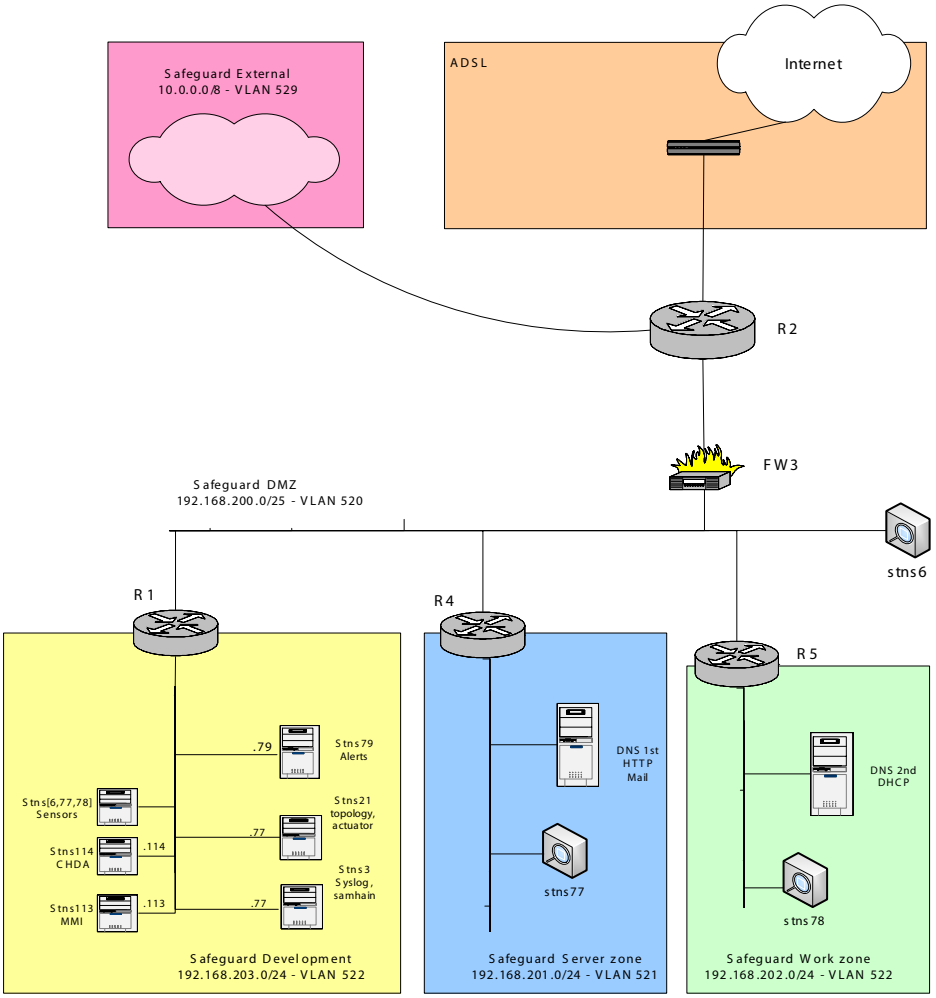

Figure 3.14: The Safeguard telecom test network

traffic coming from via the Internet is routed into the DMZ via the firewall to a specific jump machine. This hardened machine offers a SSH service where partners are supposed to log in. After a successful login, being contained in a jailed shell where only SSH to the test network is allowed, partners are able to transport data from or into the test net in a safe way.

- Work zone: A subnet where employees of a company or administrative personnel are working. This zone encompasses a vast variety of different hardware (Sparc 5, 10,20; Ultra 10, PC) and computer systems, such as Unix Solaris (2.6-2.10) all patch levels, NetBSD, Windows, (2000, NT, XP) different patch levels, (VM ware).

- Server zone: In this zone servers of all kinds are located, utilizing Sparc 5, 10, 20, ULTRA10 hardware) for e-mail, web, file, applications, administration and control. Deployed operating systems are Solaris (2.6-2.10) all patch levels, Linux, Windows server (VMware) and OpenBSD. This zone represents a realistic operating system distribution and structure for Swisscom's FX and Mobile administrative server environment.

- Safeguard development zone: This is a more protected sub network. If the 203 router is disconnected, this zone is sealed and impenetrable. All data from the other zones are gathered by a hardened machine - Unix Sol9 STNS3 having two separate network cards. In this zone, all network administrative related services are resident such as Emergency Jumpstart and management services and the actual Safeguard agent system, being distributed on eight machines. Those machines have been hardened LINUX 9.0, Solaris 2.8+ machines.

All zones are protected and monitored by Safeguard utilizing various sensors delivering their pre–processed data via wrappers and distributed small helper programs, residing on the hosts themselves. These helper programs are running as Cron–jobs or periodic or triggered demons with high priority and protection.

All zones are synchronised via the NTP protocol thus assuring that the final correlation of alerts in the different IDS alert channels is timely synchronised and actions can be synchronised. If synchronization is lost in one part of the network the machines synchronise themselves on their next gateway, thus ensuring that over a day's period proper synchronization is in the seconds regime.

To describe the full evaluation of Safeguard is outside the scope of this thesis. The interested reader is referred to the report on validation and testing of Safeguard [98].

Chapter 4

ADWICE

The basic idea of ADWICE, is to represent normality by the compact cluster summaries of the BIRCH clustering algorithm [122]. As explained in the introduction we perform pure anomaly detection, assuming the model is trained only on normal data.

BIRCH builds a tree structure used to guide the search for similar clusters while clustering a set of data points. We take this idea and apply it to the cluster-based normality model. An index tree can speed up training as well as detection. This chapter describes and evaluates the initial version of ADWICE using a search-index similar to the original BIRCH clustering algorithm.

4.1 Basic concepts

The full BIRCH algorithm is presented in the original paper [122] and only the parts relevant in the context of ADWICE are summarised here. The BIRCH clustering algorithm requires data to be numeric. Non-numeric data is therefore assumed to be transformed into a numeric format by pre-processing. How this is done will be explained in the context of evaluation later.

Given n d-dimensional data vectors $\boldsymbol{v}_i$ in a cluster $CF_j = \{\boldsymbol{v}_i | i = 1 \ldots n\}$ the centroid $\boldsymbol{v_0}$ and radius $R(CF_j)$ are defined as:

$$\boldsymbol{v_0} = \frac{\sum_{i=1}^{n} \boldsymbol{v}_i}{n} \qquad R\left(CF_j\right) = \sqrt{\frac{\sum_{i=1}^{n} \left(\boldsymbol{v}_i - \boldsymbol{v_0}\right)^2}{n}} \tag{4.1}$$

R is the average distance from member points in the cluster to the centroid and is a measure of the tightness of the cluster around the centroid.

A fundamental idea of BIRCH is to store only condensed information, denoted *cluster feature*, instead of all data points of a cluster. A cluster feature is a triple $CF = (n, \boldsymbol{S}, SS)$ where n is the number of data points in the cluster, $\boldsymbol{S}$ is the linear sum of the n data points and SS is the square sum of all data points. Given the CF for one cluster, the centroid $\boldsymbol{v_0}$ and radius R may be computed. The distance between a data point $\boldsymbol{v}_i$ and a cluster CF_j is the Euclidian distance between $\boldsymbol{v}_i$ and the centroid, denoted $D(\boldsymbol{v}_i, CF_j)$ while the distance between two clusters CF_i and CF_j is the Euclidian distance between their centroids, denoted $D(CF_i, CF_j)$. If two clusters $CF_i = (n_i, \boldsymbol{S_i}, SS_i)$ and $CF_j = (n_j, \boldsymbol{S_j}, SS_j)$ are merged, the CF of the resulting cluster may be computed as $(n_i + n_j, \boldsymbol{S_i} + \boldsymbol{S_j}, SS_i + SS_j)$. This also holds if one of the CFs is only one data point making incremental updates of CFs possible.

An *index tree* is a tree with three parameters, leaf size (LS), threshold (T), and maximum number of clusters (M). The purpose of the index tree is to guide the search for similar clusters in the normality model. A leaf node contains at most LS entries, each of the form (CF_i) where $i \in \{1, \ldots, LS\}$. Each CF_i of the leaf node must satisfy a threshold requirement (TR) with respect to the threshold value T. Two different threshold requirements have been evaluated with ADWICE. The first threshold requirement $R(CF_i) < T$ corresponds to a threshold requirement that was suggested in the original paper and is therefore used as base line in this work (ADWICE–TRR). With this threshold requirement, a large cluster may absorb a small group of data points located relatively far from the cluster centre. This small group of data points may be better represented by their own cluster because detection is based on distances. A second threshold requirement was therefore evaluated where $D(\boldsymbol{v}_i, CF_i) < T$ was used as decision criteria ($\boldsymbol{v}_i$ is the new data point to be incorporated into the cluster). This version of the algorithm will be referred to as ADWICE–TRD.

Each non-leaf node contains a number of entries of the form (X, $child_i$) where $child_i$ is a pointer to the node's i-th child. X provides information to guide a top-down search of the tree to find the closest cluster (i.e. centroid) given a data vector v. The following section will explain what information X that is used for in the first implementation of ADWICE. Chapter 5 describes an alternative implementation of the index-tree, with other information used to guide the search.

4.2 Training

During training the model is presented with new training data one item at a time. Given a model and a new data vector $\boldsymbol{v}$, a top-down search of the index tree for

the closest cluster is performed. If the threshold requirement is fulfilled, the new data point may be merged with the closest cluster, otherwise a new cluster needs to be inserted into the index tree. If the size (number of clusters) of the model has reached the maximum M, the threshold T is increased, the model rebuilt, and then v is inserted in the new model. When the number of clusters in the model increases, new leaf nodes may need to be created, and the index updated accordingly.

Below is an algorithmic description of the training phase of ADWICE, in which only the main points of the algorithm are presented and some simplifications made to facilitate presentation. Note that we have abstracted away the use of the search index that will be presented in more detail in the next section.

Algorithm 1 ADWICE Training

```
 1: procedure TRAIN(v, model)
 2:     closestCF = findClosestCF(v, model)
 3:     if thresholdRequirementOK(v,closestCF) then
 4:         merge(v,closestCF)
 5:     else
 6:         if size(model) ≥ M then
 7:             increaseThreshold()
 8:             model = rebuild(model)
 9:             train(v, model)
10:         else
11:             leaf = getLeaf(v,model)
12:             if spaceInLeaf(leaf) then
13:                 insert(newCF(v), leaf)
14:             else
15:                 splitLeaf(leaf, newCF(v))
16:             end if
17:         end if
18:     end if
19: end procedure
```

Rebuilding means that existing clusters are removed from the model and reinserted. Because the threshold is increased, some of the clusters may now be merged, thereby reducing the size below the limit M. Rebuilding the model requires much less effort than the initial insertion of data because the number of clusters is significantly less than the number of data points.

If the increase of T is too small, a new rebuild of the tree may be needed

to reduce the size below M again. A heuristic described in the original BIRCH paper [122] may be used for increasing the threshold to minimize the number of rebuilds, but in this work we use a simple constant to increase T conservatively to avoid influencing the result by the heuristic.

If it is possible to add clusters to the model (the size is still below M), we find the leaf where the new cluster should be included and insert the cluster if there is still space in the leaf. Otherwise we need to split the leaf, and insert the new cluster in the most suitable of the new leaves.

4.2.1 Using the original BIRCH index

The original BIRCH index consists of a CF tree, which is a height balanced tree with four parameters: branching factor (B), threshold (T), maximum number of clusters (M), and leaf size (LS). Each non-leaf node contains at most B entries of the form (CF_i, $child_i$), where $i \in 1, \ldots, B$ and $child_i$ is a pointer to the node's i-th child. Each CF at non-leaf level summarises all child CFs in the level below.

- Finding the closest cluster is done by recursively descending from the root to the closest leaf, and in each step choosing child i such that $D(\boldsymbol{v}, CF_i) < D(\boldsymbol{v}, CF_j)$ for every other child j.

- When inserting new data into the tree all nodes along the path to the root need to be updated. In absence of a split, the CFs along the path to the updated leaf need to be recomputed to include v by incrementally updating the CFs. If a split occurred, we need to insert a new non-leaf entry in the parent node of the two new leafs and re-compute the CF summary for the new leafs. If there is free space in the parent node (i.e. the number of children is below B) the new non-leaf CF is inserted. Otherwise the parent is split in turn. Splitting may proceed all the way up to the root in which case the depth of the tree increases when a new root is inserted.

- When splitting a leaf, the two farthest CFs of the leaf are selected as seeds and all other CF_j from the old leaf are distributed between the two new leafs. Each CF_j is merged with the leaf with the closest seed.

The setting of the different parameters of the algorithm is further discussed in the context of evaluation in section 4.4.1.

4.3 Detection

When a normality model has been trained, it may be used to detect anomalies in unknown data. When a new data point $\boldsymbol{v}$ arrives detection starts with a top down search from the root to find the closest cluster feature CF_i. This search is performed in the same way as during training. When the search is done, the distance $D(\boldsymbol{v}, CF_i)$ from the centroid of the cluster to the new data point $\boldsymbol{v}$ is computed. Informally, if D is small, i.e. lower than a limit, $\boldsymbol{v}$ is similar to data included in the normality model and $\boldsymbol{v}$ should therefore be considered normal. If D is large, $\boldsymbol{v}$ is an anomaly.

Let the threshold T be the limit (L) used for detection. Using two parameters E_1 and E_2 where $E_1 \geq E_2$, we can compute $MaxL = E_1 * L$ and $MinL = E_2 * L$. Then we compute the belief that $\boldsymbol{v}$ is anomalous using the formula below:

$$anomaly\ belief = \begin{cases} 0 & \text{if } D \leq MinL \\ 1 & \text{if } D \geq MaxL \\ \frac{D-MinL}{MaxL-MinL} & \text{if } MinL < D < MaxL \end{cases} \tag{4.2}$$

A belief threshold (BT) is then used to make the final decision. If we consider $\boldsymbol{v}$ anomalous we raise an alert. The belief threshold may be used by the operator to change the sensitivity of the anomaly detection. For the rest of the thesis to simplify the evaluation we set $E_1 = E_2 = E$ so that $\boldsymbol{v}$ is anomalous if and only if $D > MaxL = MinL$. Note that clusters are spherical but the area used for detection of multiple clusters may overlap, implying that the clusters may be used to represent also non-spherical regions of normality.

4.4 Evaluation

Performing attacks in real networks to evaluate on-line anomaly detection is not realistic and our work therefore shares the weaknesses of evaluation in somewhat "unrealistic" settings with other published research work in the area. Our approach for dealing with this somewhat synthetic situation is as follows. We use the KDDCUP99 data set [78, 80] to test the real-time properties of the algorithm. Having a large number of attack types and a large number of features to consider can thus work as a proof of concept for the distinguishing attributes of the algorithm (unknown attacks, fast on-line, incremental model building). We then go on to evaluate the algorithm in the Safeguard test network built with the aim of emulating a realistic telecom management network (see section 3.4).

Despite the shortcomings of the DARPA/Lincoln Labs related datasets [78] they have been used in at least twenty research papers and were unfortunately at the time of this work the only openly available data sets commonly used for comparison purposes.

The original KDD training data set consists of almost five million session records, where each session record consists of 41 fields (e.g. IP flags set, service, content based features, traffic statistics) summarising a TCP session or UDP connection. Since ADWICE assumes all training data is normal, attack data are removed from the KDD training data set and only the resulting normal data (972 781 records) are used for training. All 41 fields[1] of the normal data are considered by ADWICE to build the model.

The testing data set consists of 311 029 session records of which 60 593 are normal and the other 250 436 records belong to 37 different attack types ranging from IP sweeps to buffer overflow attacks. The use of the almost one million data records for training and more than 300 000 data for testing in the evaluation presented below illustrates the scalability of ADWICE. Other cluster-based work uses only small subsets of data [57].

Numeric features are normalised to the interval [0,1]. Some features of KDD data are not numeric (e.g. service). Non-numeric features ranging over n values are made numeric by distributing the distinct values over the interval [0, 1]. However, two distinct values of the service feature (e.g. http, ftp) for example, should be considered equally close, regardless of where in the [0, 1] interval they are placed. This intuition cannot be captured without extending the present ADWICE algorithm. Instead the non-numeric values with $n > 2$ distinct values are scaled with a weight w. In the KDD dataset $n_{protocol} = 3$, $n_{flag} = 11$ and $n_{service} = 70$. If $w/n >= 1$ this forces the algorithm to place two sessions that differ in such non-numeric multi-valued attributes in different clusters. That is, assuming the threshold condition requiring distance between two vectors to be less than 1 to insert merge them. This should be enforced because numerical values are scaled to [0, 1]. Otherwise a large difference in numerical attributes will anyway cause data to end up in the same cluster, making the model too general. If multi-valued attributes are equal, naturally the difference in the numerical attributes decides whether two data items end up in the same cluster.

We illustrate the transformation of categorical values with an example, in this case the service attribute of the KDD data set. Let L be an arbitrary ordered list of all $n_{service} = 70$ different values of the service attribute. Assume $L[0] = http$ and

[1]The complete KDD dataset as well as documentation of all features are available for download from the UCI KDD Archive [110].

$L[1] = ftp$ to exemplify this. We select a weight such that $w_{service}/n_{service} >= 1$, in other words $w_{service} >= n_{service} = 70$. Let $w_{service} = 70$ which fulfils this condition. Now map the values of L into the interval [0,$w_{service}$] by for each value $L[i]$ computing the numeric value $x_i = (i/n_{service}) * w_{service} = i/70 * 70 = i$. This imply that for every pair of value x_i and x_j $|x_i - x_j| >= 1$. In our example $x_{http} = 0$ and $x_{ftp} = 1$ and of course $|x_{http} - x_{ftp}| = |0 - 1| >= 1$. The distance based threshold state that $D(\boldsymbol{v}, CF) < T$ must be fulfilled to allow a data point to be absorbed by a cluster. We remind the reader that D is the Euclidian distance in the current implementation and T is the threshold. Because we make sure $T << 1$ to avoid a too general model the threshold condition will never be fulfilled when two service values differ, because the Euclidian distance between such vectors will always be at least 1. In our example, assume two different vectors $\boldsymbol{v}$ and $\boldsymbol{u}$. Assume the extreme case when all dimensions except the service dimension are equal and $\boldsymbol{v}[service] = x_{http}$ and $\boldsymbol{u}[service] = x_{ftp}$. Then the Euclidian distance will be $D(\boldsymbol{v}, \boldsymbol{u}) = \sqrt{(0^2 + ... + (x_{http} - x_{ftp})^2 + 0^2 + ...)} = |x_{http} - x_{ftp}| = 1$. This does not fulfil the threshold condition and therefore merging of two such vectors will be avoided. If other dimensions differ too, this will only increase the Euclidian distance between the vectors.

4.4.1 Determining parameters

Of the three parameters T, B and M the threshold T is the simplest to set, as it may be initialised to zero. The other parameters are discussed below.

The M parameter needs to be decided using experiments. Because it is only an upper bound of the number of clusters produced by the algorithm it is easier to set than an exact number of clusters as required by other clustering algorithms. As M limits the size of the CF-tree it is also an upper bound on the memory usage of ADWICE.

In one extreme case we could set M to the total number of training data resulting in one cluster for each unique training data. We would then have a model that marks all data that are not included into training data as attacks. Of course we need the model to be much more general than that, or every new normal data not included in the training set would result in a false positive. This situation is called over-fitting, when the model is too specific and producing very good accuracy for training data without being able to handle anything else. The conclusion is that in general M needs to be set much lower than the number of data represented by the normality model to avoid over-fitting.

The algorithms strive for covering all training data by the available number of clusters, growing the clusters by increasing the threshold whenever required. If

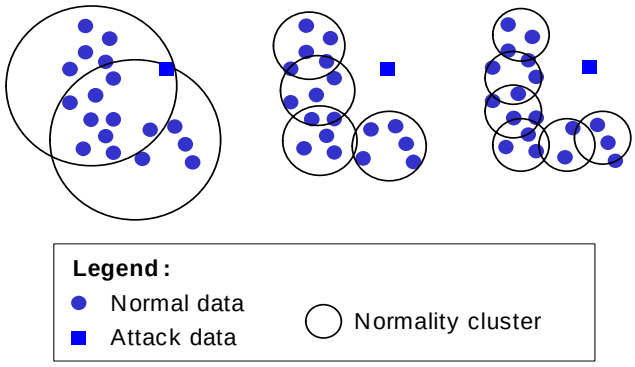

Figure 4.1: Models with different M of the same data

the clusters are too large, they will cover not only normal data, but also possibly attack data. In the other extreme case, when M is set to one, there will be one cluster basically covering all space where there are training data. This imply that M also needs to be set high enough to avoid too general (large) clusters. How high depends on the distribution and number of training data although some general guidelines could be produced given more experience with the algorithm in the context of real networks.

Figure 4.1 shows three different cluster models of the same data using two, four or six clusters. Representing normality by only two clusters makes the model too general because also the distant attack data are covered and much of the empty space around the normal data. However, both four and six clusters provide good representations of the normal data. In general, the exact number of clusters is not important.

The above reasoning was confirmed in experiments where M was increased from 2 000 to 25 000 in steps of 1 000. When setting M above 10 000 clusters the accuracy reaches a stable level meaning that setting M at least in this range should be large enough to represent the one million normal data points in the training set. In the forthcoming experiment M is therefore set to 12 000.

Naturally increasing the branching factor also increases the training and testing time. The extreme setting $B = M$ would flatten out the tree completely, making the algorithm linear as opposed to logarithmic in time. Experiments where the branching factor was changed from 20 to 10 improved testing time by roughly 16 %.

Experiments where the branching factor was increased from 2 to 2 048 in small steps showed that not only time but also accuracy is influenced by the

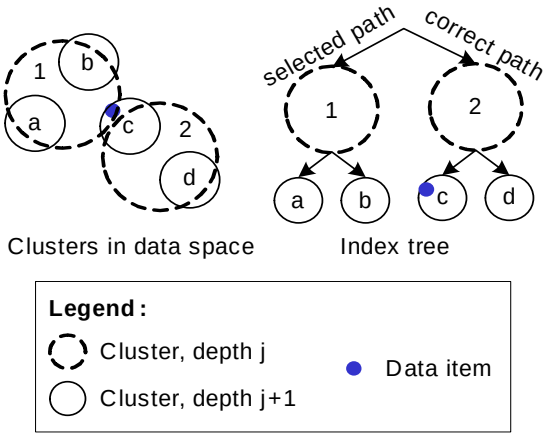

Figure 4.2: Example of index error

branching factor. More specifically, accuracy improves when the branching factor increases. Why is this the case?

The index tree is used for searching the model. Intuitively the algorithm tries to group clusters that are close (or similar) in the same branch of the tree. This is true for each level of the index tree. The cluster features at higher levels are only summaries of all clusters in the branches below. Therefore, the search may sometimes choose the wrong path and select a branch which does not contain the closest cluster. We call this an *index error*. If the branching factor is very small, the number of choices increases. At the same time, the probability that two close clusters are located in different branches at low levels in the tree increases. This increases the risk of ending up in the wrong branch.

Figure 4.2 illustrate a situation where the closest cluster is not found due to an index error. To the left in the figure we see clusters in data space and to the right the corresponding index tree. Consider the clusters to be a subset of a larger model. We now consider the new data point. To classify the data point, we search the model for the closest cluster. At some level j in the tree, the distance between the data and the centre of clusters 1 and 2 are computed. As visible in the figure the distance to cluster 1 is shorter than to cluster 2, and therefore the branch summarised by cluster 1 will be chosen when we descend the tree. Unfortunately this will lead the wrong part of the model and we will have an index error.

The experiments showed that the false positives rate stabilised when the branching factor is increased above 16. In the forthcoming experiments the branching factor is therefore set to 20, a reasonable trade-off between accuracy and training

time.

A situation where the index may influence detection accuracy is not ideal. In chapter 5 we further discuss the implications of index errors and introduce an algorithm without this property, thereby improving accuracy. But before that, evaluation of the initial index is described in the following sections.

4.4.2 Detection rate versus false positives rate

Figure 4.3 shows the trade-off between detection rate and false positive rate on an ROC diagram (see section 2.2.3). To highlight the result we also compare our algorithm ADWICE–TRD with ADWICE–TRR which is closer to the original BIRCH algorithm. The trade-off in this experiment is realised by changing the E-parameter from 5 (left-most part of the diagram) to 1 (right-most part of the diagram) increasing the detection space of the clusters, and therefore obtaining a better detection rate while the false positives rate also increases.

The result confirms that ADWICE is useful for anomaly detection. With a false positives rate of 2.8 % the detection rate is 95 % when E = 2. While not conclusive evidence, due to short-comings of KDD data, this false positives rate is comparable to alternative approaches using unsupervised anomaly detection [57, 90]. On some subsets of KDD data Portnoy et al [90] report 1–2 % false positives rate at 50–55 % detection rate, but other subsets produce considerably inferior results. The significantly better detection rate of ADWICE is expected due to the fact that unsupervised anomaly detection is a harder problem than pure anomaly detection.

Since the KDDCUP data initially was created to compare classification schemes, many different classification schemes have been applied to the KDD-CUP data set. Classification implies that the algorithms were trained using both normal and attack data contrasted to ADWICE which is only trained on the normal training data. The attack knowledge makes differentiating of attack and normal classes an easier problem, and it was expected that the results [40] of the winning entry (C5 decision trees) should be superior to ADWICE. This was also the case[2] regarding false positives (0,54 %), however detection rate was slightly lower, 91,8 %. Due to the importance of low false positives rate we indeed consider this result superior to that of ADWICE. We think the other advantages of ADWICE

[2] In the original KDDCUP performance was measured using a confusion matrix where the result for each class is visible. Since ADWICE does not discern different attack classes, we could not compute our own matrix. Therefore overall false positives rates and detection rates of the classification scheme were computed out of the result for the individual classes.

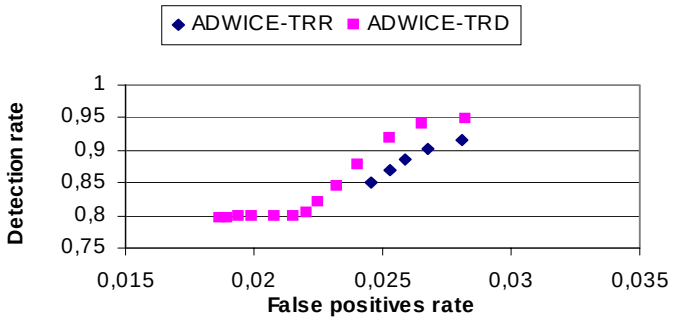

Figure 4.3: Detection rate versus false positives

make up for this. Also, we recall that ADWICE is one element in a larger scheme of other Safeguard agents for enhancing survivability.

The result shows that for values of E above 4.0 and values of E below 1.75 the false positives rate and detection rate respectively improve very slowly for ADWICE-TRD. The comparison with the base-line shows that using R in the threshold requirement (ADWICE–TRR) implies higher false positives rate. Section 4.4.4 describes further reduction of false positives in ADWICE–TRD by aggregation.

4.4.3 Attack class results

The attacks in the test data can be divided into four categories (see section 2.1.1):

- *Probe* - 6 distinct attack types (e.g. IP sweep, vulnerability scanning) with 4 166 number of session records in total.
- *Denial of Service (DOS)* - 10 distinct attack types (e.g. mail bomb, UDP storm) with 229 853 number of session records in total.
- *User-to-root (U2R)* - 8 distinct attack types (e.g. buffer overflow attacks, root kits) with 228 number of session records in total.
- *Remote-to-local (R2L)* - 14 distinct attack types (e.g. password guessing, worm attack) with 16 189 number of session records in total.

Since the number of data in the DOS class is much larger than other classes, a detection strategy may produce very good overall detection quality without handling other classes that well. Therefore it is interesting to study the attack classes

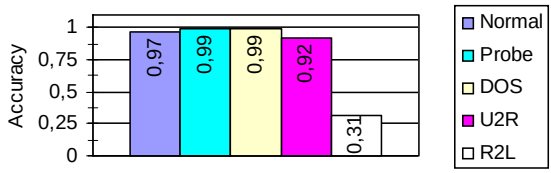

Figure 4.4: The accuracy for attack classes and the normal class

separately. Note that since ADWICE is an anomaly detector and has no knowledge of attack types, it will give the same classification for every attack type unlike a classification scheme.

Figure 4.4 shows the four attack classes Probe, DOS, U2R and R2L as well as the normal class (leftmost column) for completeness.

The results for Probe, DOS and U2R are very good, with accuracy from 92 % (U2R) to 99 % (DOS). However, the fourth attack class R2L produces in comparison a very bad result with an accuracy of only 31 %. It should be noted that the U2R and R2L classes are in general less visible in data and a lower accuracy should therefore be expected. The best entries of the original KDD-cup competition had a low detection rate for U2R and R2L attacks, therefore also a low accuracy for those classes.

4.4.4 Aggregation for decreasing alert rate

While the 2–3 percent false positives rate produced by ADWICE may appear to be a low false positive rate in other applications, in practice this is not acceptable for network security as explained in section 2.2.3. Most realistic network data is normal, and if a detection scheme with a small percent of false positives is applied to millions of data records a day, the number of false alerts will be overwhelming. In this section we show how the total number of alerts can be further reduced through aggregation.

An anomaly detector often produces many similar alerts. This is true for new normal data that is not yet part of the normality model as well as for attack types

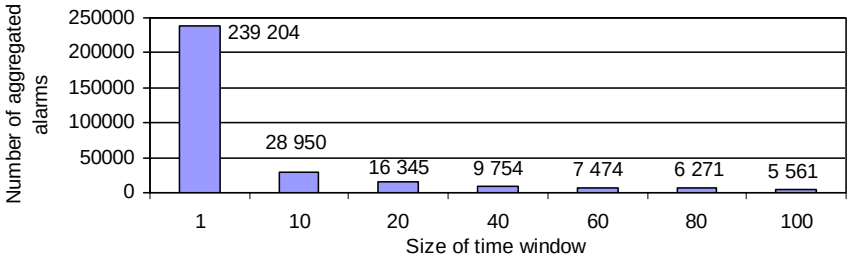

Figure 4.5: Aggregated alerts for different time windows

like DOS and network scans. Many similar alerts may be aggregated to one alert, where the number of alerts is represented by a counter. In the Safeguard agent architecture aggregation is one important task of the alert reduction correlation agent, see section 3.3.4. By aggregating similar alerts the information passed on to higher-level agents or human operators becomes more compact and manageable. Here we evaluate how aggregation would affect the alert rate produced from the KDD data set.

The KDD test data does not contain any notion of time. To illustrate the effect of aggregation we make the simplifying assumption that one test data is presented to the anomaly detector each time unit. All alerts in which service, flag and protocol features are equal are aggregated during a time window of size 0 to 100. Of course aggregation of a subset of features also implies information loss. However, an aggregated alert, referring to a certain service at a certain time facilitates the decision for narrowing down to individual alerts for further details (IP-address should have been included if present among KDD features). The result is shown in figure 4.5.

When a new alert arrives, it is sent at once, to avoid increasing time to detection. When new alerts with the same signature arrive within the same time window, the first alert is updated with a counter to represent the number of aggregated

alerts. Without aggregation ADWICE produces 239 104 alerts during the 311 029 time units. Using a short time window of 10 time units, the number of aggregated alerts becomes 28 950. Increasing the time window to 100 will reduce the original number of alerts to 5 561, an impressive reduction of 97,7 %. The explanation is that many attacks (probes, DOS) lead to a large amount of similar alerts close in time. Note that aggregation also reduces false positives, since normal sessions belonging to a certain subclass of normality may be very similar. While it might seem that aggregation only makes the alerts less visible (does not remove them) it is in fact a pragmatic solution that was appreciated by our industrial partners, since it significantly reduces the time/effort at higher (human-intensive) levels of investigation. The simple time slot based aggregation provides a flexible system in which time slots can be adaptively chosen in response to different requirements.

4.4.5 Safeguard scenarios

One of the main efforts of the Safeguard project is the construction of the Safeguard telecom management test network, presented in section 3.4, used for data generation for off-line use as well as full-scale on-line tests with the Safeguard agent architecture. Development of the Safeguard test network and further generation of intrusion detection data is ongoing work. Here we present some initial results from tests performed over a total time period of 36 hours. The ADWICE model was trained using data from a period known to contain only normal data. To keep parsing and feature computation time low to facilitate real-time detection, features were only based on IP-packet headers, not on packet content (e.g. source and destination IP and ports, time, session length). This means of course that we at this stage can not detect events that are only visible by analysing packet content. The purpose of this instance of the hybrid detection agent is to detect anomalies, outputting alerts that can be analysed by high level agents to identify time and place of attacks as well as failures or misconfigurations.

In Scenario 1 an attacker with physical access to the test network plugged in a new computer at time 15:33 and uploaded new scripts. In Scenario 2 those scripts are activated a few minutes later by the malicious user. The scripts are in this case harmless. They use HTTP on port 80 to browse Internet, but could just as well have been used for a distributed attack (e.g. Denial of Service) on an arbitrary port. The scripts are then active until midnight the first day, producing traffic considered anomalous for their respective hosts. During the night they stay passive. The following morning the scripts become active and execute until the test ends at 12:00 the second day.

Figure 4.6 illustrates the usefulness of the output of the clustering hybrid de-

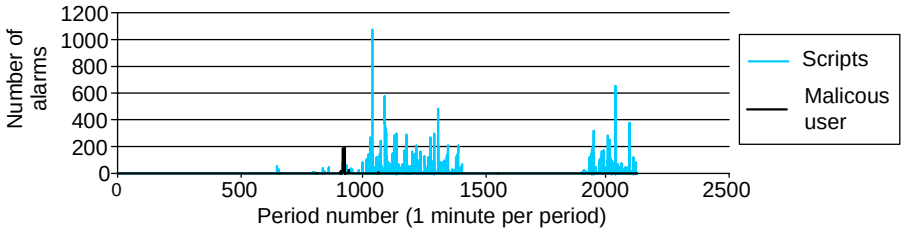

Figure 4.6: Distributed malicious scripts cause alerts

tection agent. The 36 hours of testing were divided in periods of one minute and the number of alerts for each time period is counted.

For Scenario 1, all alerts relating to the new host (IP x.x.202.234) are shown. For Scenario 2 all alerts with source or destination port 80 are shown. The figure shows clearly how the malicious user connects at interval number 902 (corresponding to 15:34), when the scripts execute, wait during the night, and then execute again. Some false alerts can also be noted, by the port 80 alerts occurring before the connection by the malicious user. This is possible since HTTP traffic was already present in the network before the malicious user connected.

Chapter 5

ADWICE with grid index

This chapter describes the extended version of ADWICE, using a new grid-index. The basic algorithm is the same, as presented in the previous chapter, and rather then reiterating that we focus on the index itself. Adaptation of the normality model using incremental training as well as forgetting is also discussed.

5.1 Problems of the original BIRCH index

The original BIRCH index is not perfect. That is, the search for the closest cluster sometimes selects the wrong path, and we do not find the relevant cluster if it exists. This notion of *index error* is explained in the previous chapter in section 4.4.1.

5.1.1 Influence of index errors

Index errors may influence the detection quality (and evaluation) of an algorithm. Table 5.1 shows how index errors influence anomaly detection results in general when a testing data vector is compared to the model.

If the type of data is not included in the trained normality model an anomaly detection algorithm with index errors returns the correct result, since there is no close cluster to find anyway. However, there are two cases where index errors may produce erroneous results. If the new data point is normal, and the model includes a close cluster, an index error results in a false positive instead of a true negative, thereby decreasing detection quality. On the other hand, if the new data is anomalous and such data has previously been (erroneously) included into the model, or if the attack data is very similar to normal data, the index error

Data is	Model covers data	Expected evaluation	Evaluation with index error
normal	yes	true negative	false positive
normal	no	false positive	false positive
attack	yes	false negative	true positive
attack	no	true positive	true positive

Table 5.1: Consequences of index errors for anomaly detection

may result in a true positive, improving detection quality. In other words, index errors make the behaviour of the detection scheme unpredictable, since quality may both improve and degrade. The index errors do not completely invalidate evaluation, since if training data is to a large extent normal and attacks are not normally included in the model, the second case of error (causing improvement) is less likely. The first type of error, when the testing data is normal and the model subsumes this data, is most common. This reasoning seems to imply that detection accuracy may be improved using an index that does not cause index errors. But how much will the detection accuracy be influenced?

As shown in section 2.2.3 the false positives rate (FPR) is the number of false positives (FP) divided by the number of normal data ($FP+TN$). Detection rate (DR) is the number of detected attacks (TP) divided by the total number of attack data ($TP + FN$).

Assume that IE is the rate of index errors and we use the index to find the closest cluster of X data items. Then $IE * X$ data items will result in failure to find the closest cluster.

The index errors then contribute to errors in detection result, in those cases where the data is included in the model. As explained previously this is the case for true negatives, where index errors for normal data included in the model in case of index errors will result in additional false positives. In the case of false negatives, the index error will contribute with additional true positives.

The final detection rate and false positives rate in presence of index errors can be computed as shown below:

$$\begin{aligned} DR(IE) &= \frac{TP}{TP+FN} + \frac{FN * IE}{TP+FN} \\ &= DR + (1 - \frac{TP}{TP+FN}) * IE \\ &= DR + (1 - DR) * IE \end{aligned} \tag{5.1}$$

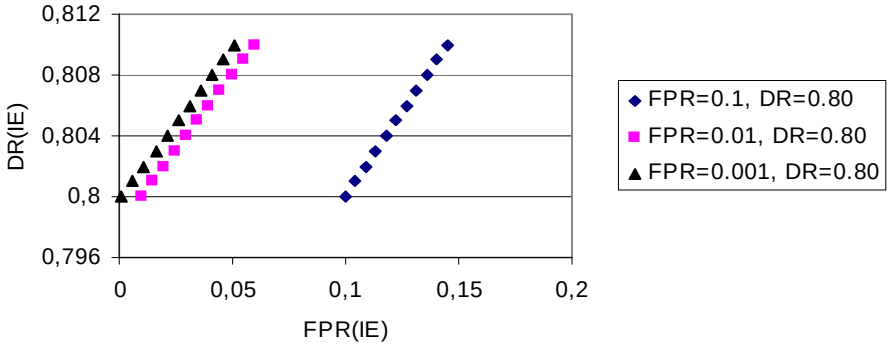

Figure 5.1: Influence of index errors for detection rate 80%

$$\begin{aligned} FPR(IE) &= \frac{FP}{FP+TN} + \frac{TN*IE}{FP+TN} \\ &= FPR + (1 - \frac{FP}{FP+TN}) * IE \\ &= FPR + (1 - FPR) * IE \end{aligned} \tag{5.2}$$

We conclude that we can compute $DR(IE)$ and $FPR(IE)$ using only knowledge of DR, FPR and IE. Figure 5.1 shows $DR(IE)$ and $FPR(IE)$ for three different values of FPR where DR equals 80% and IE ranges from 0 to 0.5%. At $IE = 0\%$ detection rate are exactly 80%. When IE increases, detection rate as well as false positives rate increases too. Figure 5.2 shows the same range of FPR and IE, but with detection rate 90%.

The general conclusion is that as long as detection rate is much higher than the false positives rate, index errors will significantly increase the false positives rate, while the improvement in detection rate is less visible. Removing index errors will therefore typically improve overall detection results significantly due to the importance of a low false positives rate.

The implications of index errors during training, is that clusters trained early into the model can be 'lost'. If we assume that the model is trained only on normal data, as in the case of ADWICE, clusters getting lost in the model potentially imply the following:

- If the model later is trained on similar data, the new data will result in creation of new clusters rather than being merged with the old lost clusters.

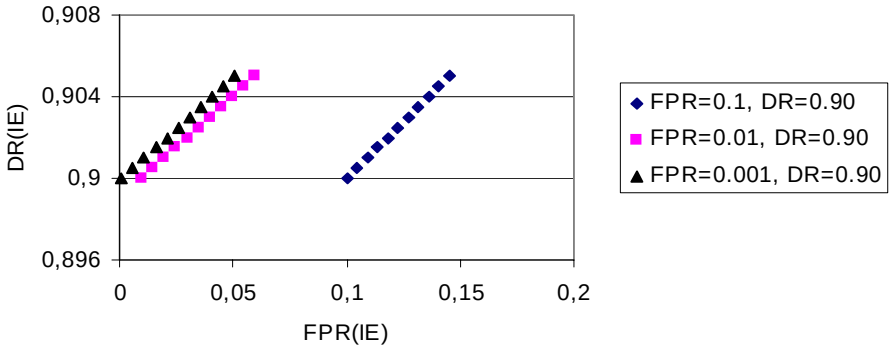

Figure 5.2: Influence of index errors for detection rate 90%

This implies that those clusters contain less data points than expected. If the lost data are not evenly distributed around the cluster, the incomplete cluster may have smaller area and/or possibly have the cluster centre slightly relocated. This means that the cluster possibly does not cover a part of the model that should be normal, leading to additional false positives during detection. In principle a slightly relocated cluster could also influence detection accuracy by covering new normal data, not previously included in the training data. This would actually decrease the false positives. But this case is less probable.

- If the model is not later trained on similar data, this will lead to additional false positives if testing data contains such normal data.

The above reasoning shows that it is reasonable to expect that removing index errors should improve accuracy. When selecting a search index for use with a model for anomaly detection, inexact indexes should in most cases be avoided due to the potential increase of false positives. If inexact indexes are anyway preferred, it is useful to evaluate how much the index influences accuracy.

The theoretical reasoning cannot easily be used to compute the index error of a specific algorithm. A sometimes more practical solution is to remove the index temporarily and use the model without the index and compare the accuracy with and without the index. In the case of ADWICE, changing the branching factor of the index to the maximum number of clusters would result in linear search removing the influence of the index. Depending on the algorithm and available

processing power, this may limit the size of the data set that can be used for evaluation.

5.2 The grid-index

Because index errors may potentially decrease detection accuracy, we designed an alternative algorithm using a new search index, ADWICE-grid. The requirements for the new index are:

- There should be no index errors.
- Adaptation of the BIRCH-index should be preserved (incremental training).
- Performance should be similar to that of the BIRCH-index.

There are two possibilities for handling the adaptation requirement. The index may be updated together with the clusters for each change to the model or the index may be more independent of changes to the model. Since BIRCH follows the first principle we wanted to explore the second type of index. Minimizing the need for continuously updating the index, may potentially further improve training performance. Rather than guiding the search using cluster summaries, ADWICE-grid divides space into a hierarchy of smaller regions (i.e. subspaces), called a grid.

A *subspace* of d-dimensional space is defined by two vectors, $\boldsymbol{Max}$ and $\boldsymbol{Min}$ for each dimension, specifying for each dimension an interval or *slice*. A grid is a division of space into subspaces. In two dimensions this results in a space divided into rectangles. The idea of the new grid index is to use a grid-tree where each node specifies a subspace. The subspaces decrease in size along the path from root to leaf, thereby guiding the search. Leaves contain the clusters as in the case of the original BIRCH index.

The division into subspaces can be done using different schemes:

1. At each level in the tree *all* dimensions could be divided into smaller slices. The left example of figure 5.3 shows an abstract example of this for a two-dimensional space. Empty (white) subspaces do not need to be represented by leaves in the tree.

2. Alternatively only one dimension is used for slicing at each level of the tree. The right example of figure 5.3 illustrates this.

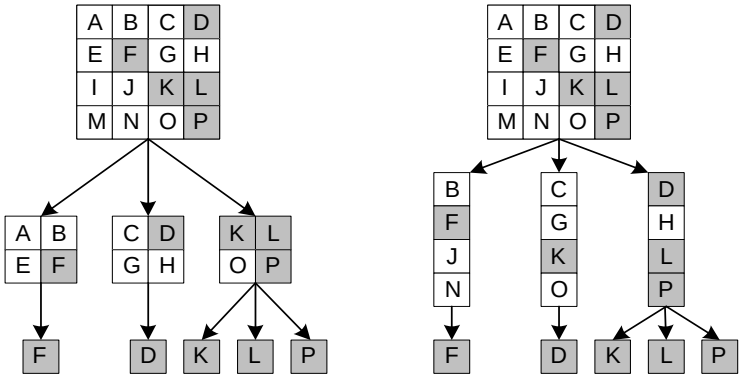

Figure 5.3: Alternative grid schemes

At each node in the tree, some computing needs to be done in order to find a path to the closest cluster. Because performance is very important in real-time intrusion detection, we want the search to proceed as quickly as possible. Slicing using only one dimension at each level requires less computing than the alternative, and the second grid scheme was therefore selected for evaluation.

Figure 5.4 shows a 2-dimensional grid divided into subspaces together with the corresponding grid tree using grid-scheme 2 (the tree is further explained below). Note that:

- Not all leaf subspaces need to be at the same level.
- Empty subspaces have no corresponding leaf.

Our intuition and experience tells us that such a grid is sparsely populated. This means that suitable programming primitives such as hash tables (rather than vectors) should be used to avoid empty subspaces taking up space in the index tree.

Before starting with the implementation we tested the performance of the primitive operations used in an index tree. In case of the BIRCH-index, the primitive operation is the distance function computing the Euclidian distance in multi dimensional space. A performance evaluation shown in figure 5.5 revealed that a hash function call is 6-15 times faster than one call to the distance function depending on the number of dimensions (60-20). Because at each node of the CF tree of BIRCH, up to B (normally set to 10-20) branches may exist, using a hash table could be 100 times faster when the distance function is applied multiple times during linear search of a non-leaf node to find the correct child. This means that there is room for additional processing with the new index.

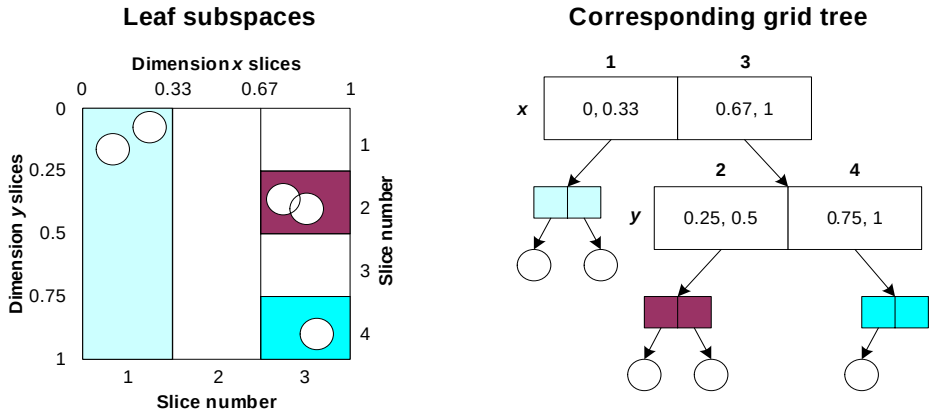

Figure 5.4: Basic notions of the grid

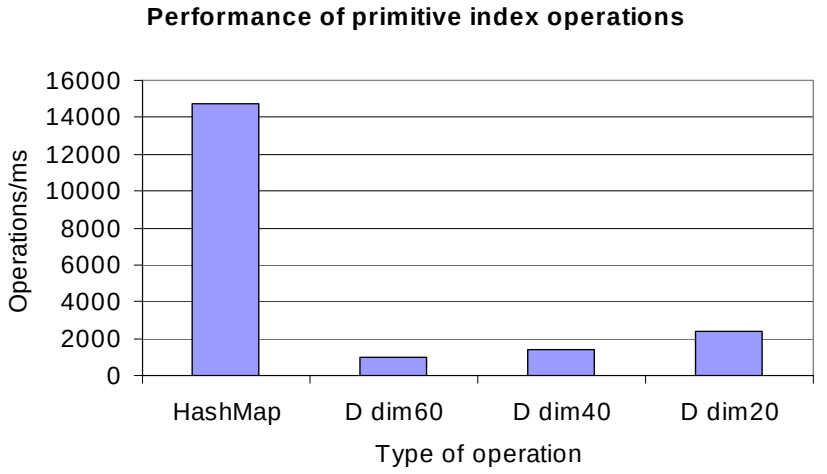

Figure 5.5: Performance of primitive index operations

Our grid index consists of a grid-tree, which is a sparse, possibly unbalanced tree with three parameters, threshold (T), leaf size (LS) and maximum number of clusters (M). Each dimension i of the d-dimensional space is assumed to have a maximum (Max_i) and a minimum (Min_i). These are in practice realised during feature extraction for unbounded domains, and lead to the division of the current dimension i in a certain number of intervals ($NumSlices_i$) with a certain width ($SliceWidth_i$). A slice is one such interval in some dimension. A function `getSliceDimension(depth)` is devised which maps a node depth to one dimension. Each non-leaf node of depth j contains at most $NumSlices_i$ children where each child of a non-leaf node is an entry in a hash table. The hash table is a mapping from an interval number to a child node.

To find the closest cluster of a new data $\boldsymbol{v}$ in a node with depth j (starting with the root), we first compute the slice dimension
$i =$`getSliceDimension(`j`)`. In the current node we then only consider dimension i of $\boldsymbol{v}$. Given the value of $\boldsymbol{v}$ in dimension i ($\boldsymbol{v}[i]$) the number of the interval into which $\boldsymbol{v}$ fits is computed. This interval number is mapped to a child using the hash table. In this way we find our way down to a leaf, where the data is located. In this leaf we may then do linear search among the clusters to find the closest.

Unfortunately there is a complication that makes the index more complex. There is no guarantee that the closest cluster actually is located in the same leaf as the data point itself. Merging of a data point with the closest cluster CF_i may be performed if $D(\boldsymbol{v}_i, CF_i) < T$. This means that the closest CF_i may be located inside any of the subspaces reachable within a distance T of the data point $\boldsymbol{v}$. Accordingly we possibly need to search multiple paths at each node in the tree, depending on the value of T. At each non-leaf node at depth j we compute a *neighbour space* $[\boldsymbol{v}[i] - T, \boldsymbol{v}[i] + T]$. T is the radius (R_{ns}) of this space and the vector v the center. All slices that overlap the neighbour space are searched to find the closest cluster. Because space is sparse, many slices are not occupied. Since only children for occupied intervals exist as hash table entries, empty intervals do not need to be searched.

One should notice that the number of neighbour subspaces will increase exponentially with the number of dimensions. In practice only a small fraction of neighbour subspaces is populated with clusters and because multiple clusters are contained by each leaf in the index tree, many subspaces are not represented by the index tree. As proved by the evaluation, this makes the search feasible also for the case of KDD data where the number of dimensions is as high as 40. However, one may construct data sets where an increasing number of neighbours will

cause performance of search to degrade. The full implications of this remain to be evaluated and is left as future work.

There exist two cases when the nodes of the grid tree need to be updated:

- If no cluster is close enough to absorb the data point, $\boldsymbol{v}$ is inserted into the model as a new cluster. If there does not exist a leaf subspace where the new cluster fits, a new leaf is created. However, there is no need for any additional updates of the tree, since nodes higher up do not contain any summery of data below.

- When the closest cluster absorbs $\boldsymbol{v}$, its centroid is updated accordingly. This may cause the cluster to move in space. A cluster may potentially move outside its current subspace. In this case, the cluster is removed from its current leaf and inserted anew in the tree from the root, since the path all the way up to the root may have changed. If the cluster was the only one in the original leaf, the leaf itself is removed to keep unused subspaces without leaf representations.

Compared to the continuously updated original BIRCH index, the need for grid index updates are very limited, because the first case above requires only insertion of one new leaf, and the second case occurs infrequently.

In case of a split of a leaf, the original leaf is transformed to a non leaf node. The new node computes its split dimension according to its depth in the tree, and the clusters of the original leaf are inserted in the new node resulting in creation of leaves as children to the node.

The `getSliceDimension(depth)` should be defined manually or automatically according to properties of the input data to avoid a tree with a large depth. For example, if all data have the same or almost the same value in a certain dimension, this dimension is not useful for slicing, since the depth of the tree will increase without distributing the clusters into multiple children.

In most cases not all dimensions need to be used for slicing (thus limiting the height of the tree), assuming that the function `getSliceDimension(depth)` is selected appropriately. However, if the situation arises that all dimensions have been used for slicing, and still the number of clusters to be inserted does not fit in one leaf due to the limited leaf size (LS), then the LS parameter for that leaf can be increased, affecting only that leaf locally. This is the approach that our current implementation adopts. An alternative or complementary approach to handle this situation is to rebuild the tree using a smaller width of the intervals for each dimension.

5.3 Adaptation of the normality model

5.3.1 Incremental training

As described earlier, agents need to adapt in order to cope with varying LCCI conditions including changing normality. Here we describe two scenarios in which it is very useful to have an incremental algorithm in order to adapt to changing normality.

In some settings, it may be useful to let the normality model relearn autonomously. If normality drifts slowly, an incremental clustering algorithm may handle this in real-time during detection by incorporating every test data classified as normal with a certain confidence into the normality model. If a slower drift of normality is required, a random subset of encountered normal data based on sampling could be incorporated into the normality model.

Even if autonomous relearning is not allowed in a specific network setting there is need for model adaptation. Imagine that the ADWICE normality model has been trained, and is producing good results for a specific network during detection. At this point in time the operator recognises that normality has changed and a new class of data needs to be included as normal. Otherwise this new normality class produces false positives. Due to the incremental property, the operator can incorporate this new class without the need to relearn the existing working normality model.

In neither case there is need for retraining the complete model or to take the model off-line. The operator or agent may even interleave incremental training of individual or small sets of new training data with detection. This means on-line training requires additional processing power, but does not require that detection is stopped.

5.3.2 Forgetting

Incremental training of ADWICE corresponds to extending the model with additional training examples as discussed above. Old clusters will obtain more data members and this may cause the clusters to move or grow if the threshold is increased and new training data is similar to existing clusters. The size of the model, corresponding to the number of clusters, will remain the same as no new clusters are created, due to the representation in form of cluster summaries.

In cases where new training data differ more significantly from the old model, new clusters will be created. If this continues over time, the threshold will be increased until the model is too general. If M is increased when needed to avoid a

too general model, the model may at least in theory grow very large after extended periods of time which may also become a problem. Additionally, normality does not only change by the appearance of new types of data. Old services will stop being used, users may no longer have access to the system and old hosts may be removed from the network. Such events cause normality to decrease. Some types of data will simply not be present any more.

Should data that once where present in the system, but that is not present anymore still be part of the normality model? In most cases no. For example, a user is getting a new working position in another company. Some time passes and then data corresponding to that user is again present in the system. This may correspond to the user herself misusing old credentials, or another person using an inactive account. Or maybe less probable, the user has returned to her old working position. If the normality model still incorporates the data corresponding to the user, misuse of her previously inactive account will not be detected.

To adjust the model for such changes and to prevent the model to become too general or from growing indefinitely in case of incremental training, there is need for a removal of subsets of the model. We use the term forgetting because the process resembles to the forgetting in the human brain. Unless memory is reinforced, it often fades over time until it is forgotten completely or becomes very hard to recall. Reinforcing in the case of ADWICE, corresponds to *using* a certain cluster. Using in this setting correspond to one of the following events.

- During training, new training data was inserted into the cluster.
- During detection, encountered data was close enough to the cluster to be classified as normal.

Based on the notion of cluster usage, rules can be defined when to forget or fade out subsets of the model. In our initial implementation of forgetting each time a cluster is used it receives a tag with the current time. Periodically the model will be checked for unused clusters. If the time since last usage is longer than a threshold, the cluster will be removed from the model. Two parameters are used in the current implementation. *CheckPeriod* specifies how often forgetting will take place. *RememberPeriod* specifies for how long unused clusters will remain in the model. Those parameters need to be set by the operator.

In the final chapter of this thesis we will discuss alternative approaches to forgetting.

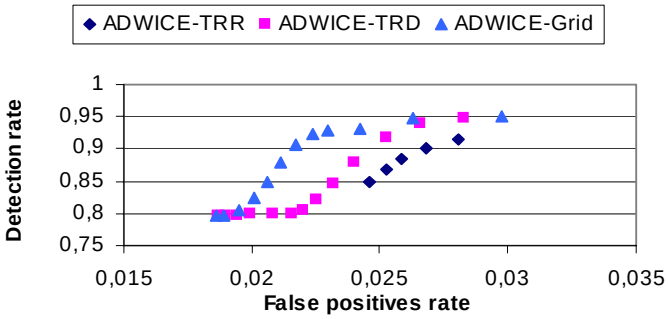

Figure 5.6: Detection rate versus false positives using ADWICE-grid

5.4 Evaluation

5.4.1 Detection rate versus false positives rate

The grid index is expected to reduce the number of false positives, as implied by the analysis in section 5.1.1. Figure 5.6 confirms this, as ADWICE-grid improves the false positives rate from 2,5% to 2,2% at 92% detection rate.

5.4.2 Incremental training

To evaluate the incremental training of ADWICE we treat an arbitrary abnormal class as normal and pretend that the normality model for the KDD data should be updated with this class. Without loss of generality we choose the IP sweep attack type and call it 'normal-new'; thus, considering it a new normal class detected by the operator. The model only trained on the original normal data will detect the normal-new class as attack, since it is not included in the model. This produces 'false positives'. The old incomplete normality model is then incrementally trained with the normal-new training data producing a new normality model that incorporates the normal-new class. Our evaluation showed that (without aggregation) the old model produced 300 false positives, whereas the new retrained model produced only three.

An important point of this work is that the original model, known to truly reflect recent normality, does not need to be retrained as soon as new cases of normality are encountered. We evaluate this on data generated in the Safeguard test network. The three days of training data is used to train an initial model which is then used for detection on the seven days of testing data. When certain types

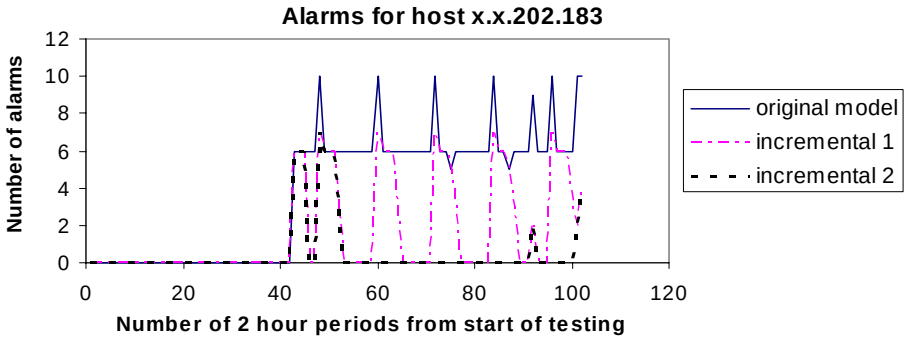

Figure 5.7: Adapting using incremental training

of traffic (new cases of normality) start producing false alerts the operator may tell ADWICE to incrementally learn the data causing those alerts to avoid similar false alerts in the future. Figure 5.7 shows alerts for host x.x.202.183 in three scenarios. Period number 1 starts at time 2004-03-11 00:00 (start of testing data) and each two-hour period presents the sum of alerts related to host x.x.202.183 during the corresponding time interval. At 2004-03-14 12:00, corresponding to period number 43, the host is connected to the network.

In the first scenario, no incremental training is used, and the testing is performed on the original model. This corresponds to the first curve of figure 5.7. We see that when the host connects, ADWICE starts to produce alerts and this continues until the testing ends at period 102.

In the second scenario the operator recognises the new class of alerts as false positives. She tells ADWICE to learn the data resulting in those false alerts at time 2004-03-14 17:55 (end of period 45). The second curve shows that many of the original false alerts are no longer produced. However, at regular intervals there are still many alerts. Those intervals correspond to non-working hours.

In the third scenario incremental training is done in two steps. After the first incremental training 2004-03-14 a second incremental training is initiated at 2004-03-15 07:55 (end of period 52) when the operator notices that false alerts related to host x.x.202.183 are still produced. Figure 5.7 shows how almost all alerts now disappear when the model is updated for the second time.

The need for the two-step incremental learning arouse since the model differs between working hours and non-working hours. The alerts the operator used for initial incremental training were all produced during working hours (2004-03-14 12:00 -2004-03-14 17:55).

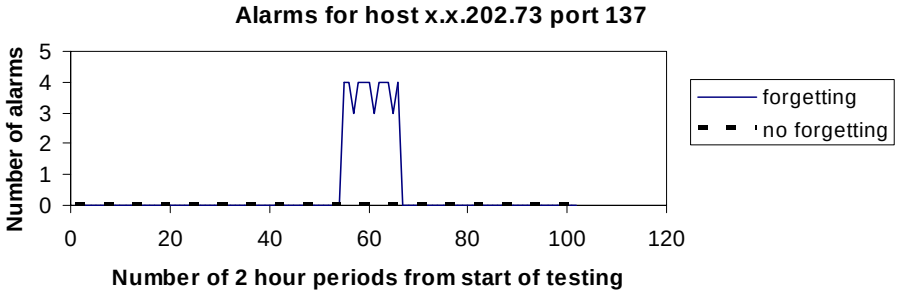

Figure 5.8: Adapting using forgetting

5.4.3 Forgetting

In this section we illustrate the use of forgetting. A model is trained on tree days of data and is then used for detection with and without forgetting on the following seven days of data. Figure 5.8 shows alerts for one instance of traffic (host x.x.202.73, port 137) that ceases to be present in the (normal) testing data, making that kind of traffic anomalous. With forgetting this fact is reflected in the normality model. In this experiment a CheckPeriod of 12 hours and RememberPeriod of 3 days (72 hours) are used.

When traffic from host x.x.202.73 on port 137 is again visible in data (periods 55-66) the traffic is detected as anomalous. Without forgetting these anomalies would go undetected.

Chapter 6

Clustering hybrid detection agent

This chapter describes the design and implementation of the Clustering Hybrid Detection Agent (CHDA), an instance of the hybrid detection agent, introduced in the Safeguard architecture in section 3.2.3. The primary purpose of CHDA is to perform anomaly detection and send alerts concerning suspicious activity to a correlation agent for aggregation and further processing.

Figure 6.1 shows an overview of the main components related to the CHDA. Tcpdump[1] is sniffing the network. Tcpdump filters are applied so that Tcpdump only outputs UDP and TCP binary data. The Tcpdump output is processed by the Preprocessor into feature vectors, where each feature vector contains basic data on a single TCP connection or UDP packet. The feature vector file is then processed by the CHDA which reports anomalies to a correlator agent.

Files are used to move data between Tcpdump, the Preprocessor and the CHDA because it is fast and simple. Also since data is stored on disk, it is easy to save data for later and to do off-line experimenting on old data.

As long as the agent works on the level of sessions rather than individual packets the agent needs to wait until a session is ended before doing detection. The hardest performance requirements will then be on Tcpdump itself and the Preprocessor handling Tcpdump data. If data flow increases, the time to detection may increase due to the processing of the Preprocessor and Tcpdump may start to lose packets. This resides outside the agent itself.

[1] Tcpdump is a program that prints network packets that match a boolean expression.

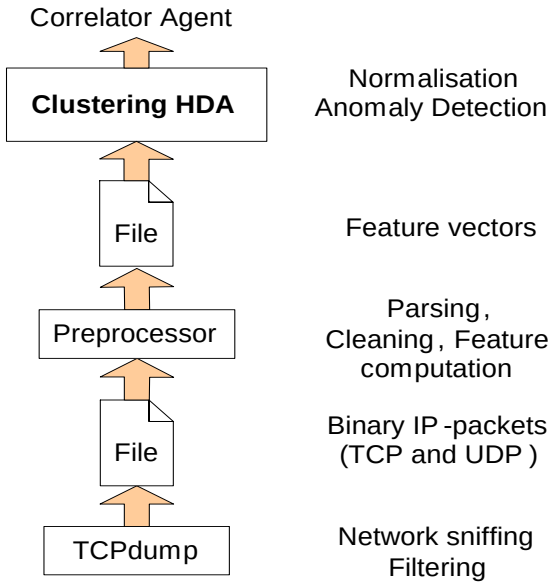

Figure 6.1: Data flow of CHDA

6.1 Design

The main modules of the CHDA agent are presented here. Figure 6.2 illustrates the main data flows between the modules. The picture shows the use of SAP (Safeguard Agent Platform) for communication, but alerts can also be written to a data base or file. In the following sections each module will be described in further detail.

- `DataSource` is the interface between the agent and the output of the Preprocessor.
- `DataBuffer` is buffer where DataSource thread writes data. The buffer is needed to avoid delays caused by the data source.
- `Transformer` is responsible for normalisation and feature selection. The Transformer is also responsible for creating alerts.
- `AnomalyDetector` performs anomaly detection.
- `SignatureDetector` performs signature-based detection.

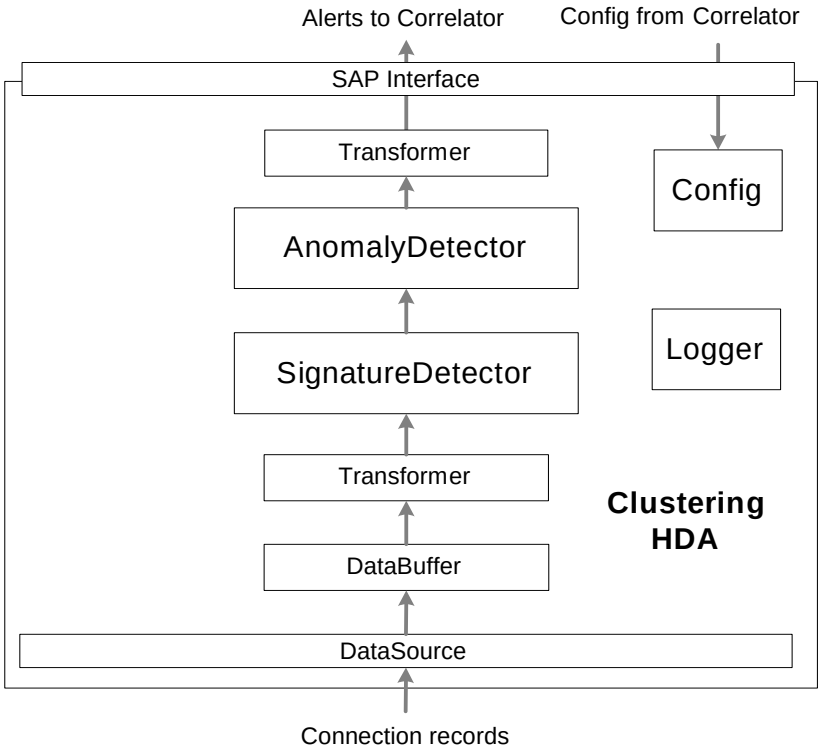

Figure 6.2: CHDA design

- `Config` handles configuration of the agent. Configuration was not implemented as a separate module, but rather cross-cutting the other modules. The introduction of the Module interface presented in section 6.1.7 improved the design.

- `Logger` handles logging.

6.1.1 Preprocessor

The Preprocessor is responsible for taking Tcpdump output and producing a format suitable for the CHDA. In the Safeguard prototype the Preprocessor is implemented as a number of Perl-scripts due to Perl's flexible text-processing features.

The output of the CHDA Preprocessor is a basic feature vector file which contains features computed out of a single TCP connection or one UDP packet.

A discussion with security personnel led to the selection of the following basic features to be computed by the Preprocessor:

- The start-time of the connection.
- The end-time of the connection.
- The connection length in milliseconds.
- The protocol type (UDP or TCP).
- The source IP address.
- The destination address.
- The source port.
- The destination port.
- The number of bytes transferred to the destination.
- The number of bytes transferred to the source.
- A flag, indicating if the connection was ended legally using a FIN flag or not.

All features are based on information available in the IP header. This decision was made to minimize processing time, and avoid the non-trivial problem of feature computation for the non-structured information of packet content.

For certain attacks, such as denial of service and scanning attacks, it is interesting also to compute statistics over time, related to many different TCP connections or UDP packets. Such high level features could be computed periodically out of the basic feature vectors. The KDD data set described in section 4.4 included a number of high level features computed over two seconds for example. To minimize the time to compute basic feature vectors, high level feature vectors could be processed by a separate Preprocessor as well as a separate agent. In the prototype no high-level feature vector was included. Figure 6.3 gives an overview of the CHDA Preprocessor and the possible extension with high level feature computation.

To increase performance, data is to be filtered as close to the source as possible. The Tcpdump filters are used to filter away duplicate traffic from different subnets and other data not to be included in the CHDA model.

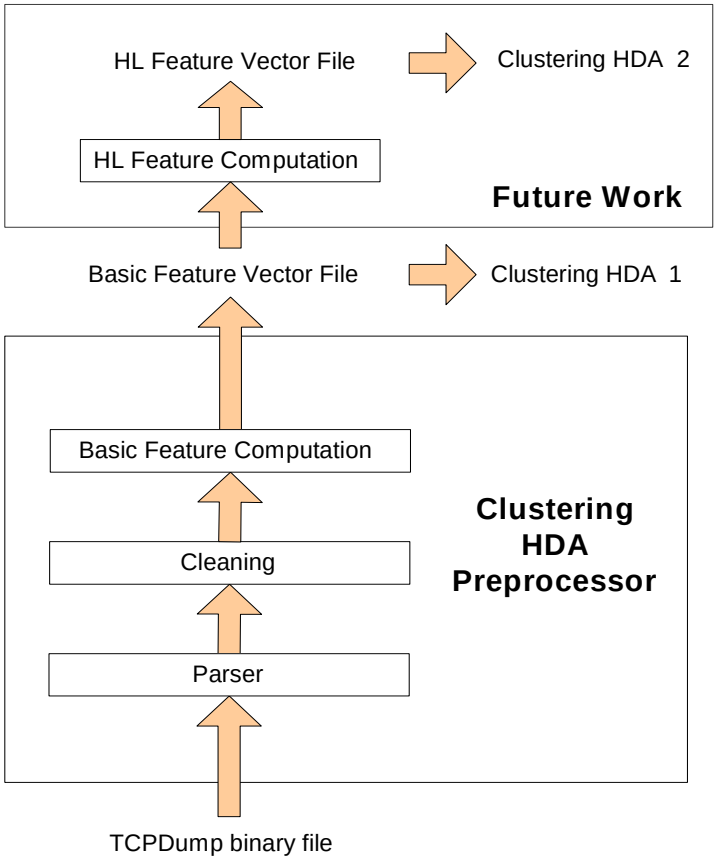

Figure 6.3: CHDA Preprocessor

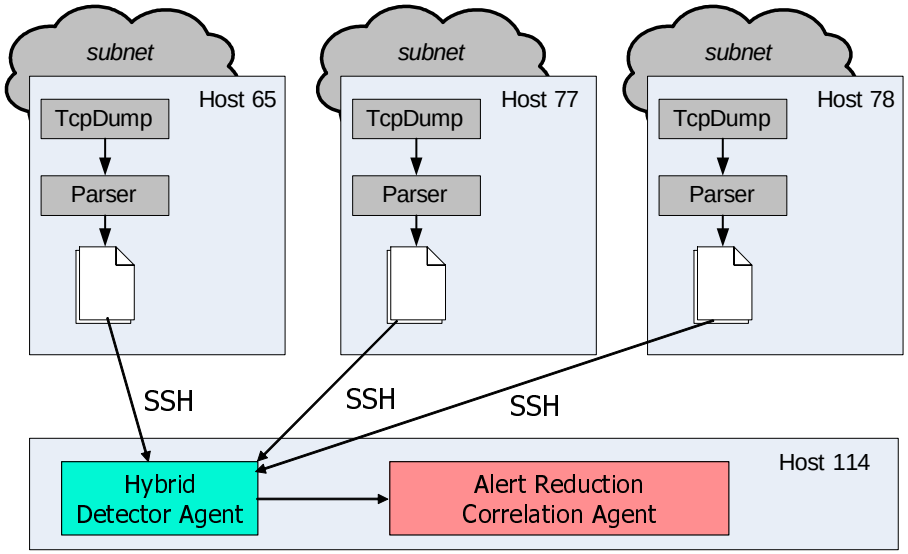

Figure 6.4: Remote data sources

6.1.2 DataSource

DataSource is used for retrieving data from the Preprocessor. The motivation to have a separate Java interface is to simplify future changes to data retrieval. The implementation used in the Safeguard context is RemoteHostsDataSource which uses SSH to read session files from an arbitrary number of hosts, merging sessions based on time stamps. Remote access is needed, since data is produced in multiple subnets and the agent is located in the special Safeguard zone subnet. Figure 6.4 illustrates remote data access.

When the agent is started, the DataSource class starts looking for one or more feature vector files. Feature vectors represented as strings are input into the DataBuffer.

During agent initialisation, a separate DataAquirer thread is started. Internally it accesses a DataSource interface to acquire data, which is then put into the DataBuffer. When there is no data available, the DataAquirer will sleep. If there is no data available for a long time, the thread will notify the main agent thread.

The DataSource interface has only one method:

- `String getData()` - Returns one data item from the data source.

6.1.3 DataBuffer

The DataBuffer is a passive synchronised first in first out buffer used by the DataAquirer thread and the main thread of the agent. The interface has the following methods:

- `putData(String data)` - Add one data item to the end of the buffer.
- `int getNumberOfData()` - Return the number of data items currently in the buffer.
- `String[] getData()` - Return and remove the first data item in the buffer.
- `String[] peekData()` - Return a copy of the first data item in the buffer.
- `removeData()` - Remove the first data item from the buffer.

6.1.4 Transformer

The Transformer transforms string arrays available in the DataBuffer to Instance-objects suitable for feeding the detector modules. It performs feature selection and normalisation. What features to select as well as maximum and minimum values for the features need to be supplied in the CHDA configuration file. In addition the Transformer will construct an Alert object upon request. The Transformer is a passive entity used only by the main thread of the agent providing the following methods:

- `Instance transform(String[] data)` - Transforms one data string into the internal representation object called Instance.
- `Alert constructAlert(String[] data, int belief)` - Construct an Alert object including belief, which is a measure of anomaly for the data item. The data should be understandable by human operators and possible to correlate with alerts from other agents. The alert must therefore be constructed using the original data, such as IP addresses and ports, rather than the internal representation (i.e. the Instance object).
- `Alert constructAlert(String[] data, ADResult r)` - Constructs an alert using an ADResult object. ADResult objects is produced by the AnomalyDetection module during detection and includes

a message for the operator as well as the anomaly belief. The message to the operator can for example state what features contributed most when the data was rated as anomalous.

There are multiple implementations of the interface including Transformers for:

- The KDD data set.
- The Safeguard prototype.
- 2 dimensional numerical data for testing and illustrating clustering.

Using the CHDA agent to process other types of data (e.g. host-based data rather than network connections) is very simple. Only a corresponding implementation of the Transformer is required. Of course this assumes that the alternative data has already been pre–processed into feature vectors which may be more or less complex.

6.1.5 AnomalyDetector

This is the main module of the agent, performing anomaly detection. The module works in either training or testing mode. It takes Instance-objects as input. In training mode it only updates the internal model without giving any output. In testing mode it returns the anomaly belief which is a measure on how anomalous the data is. It may also return a message for the operator further describing the anomaly. The AnomalyDetector is a passive entity used only by the main thread of the agent and provides the following methods:

- `void setTrainingMode(boolean trainingMode)` - Set training mode on/off. This method provides the anomaly detection engine a possibility to perform processing to prepare the model to work in a new mode. For ADWICE such a mode change is transparant and requires no extra processing except for non-mandatory logging.
- `boolean isTrainingMode()` - Return true if training mode is set.
- `void setSensitivity(double sensitivity)` - Set the sensitivity of the model. Decreasing sensitivity will decrease false positives rate at the cost of decreasing detection rate and vice versa.
- `double getSensitivity()` - Return the sensitivity of the model.

- `void train(Instances data)` - Train the model on all Instance objects included in the Instances collection.
- `void train(Instance data)` - Train the model using one Instance object.
- `ADResult detect(Instance data)` - Use the current model to decide how anomalous the Instance object is. Returns an ADResult object containing anomaly belief and a message for the operator.
- `void loadModel(File f)` - Load a new model from file.
- `void saveModel(File f)` - Store the current model to file.

Currently there are two implementations of the AnomalyDetector interface, ADWICE-birch and ADWICE-grid. It is straightforward to replace one anomaly detector engine with a new one, without any changes/recompilation of the agent itself due to dynamic loading of the detector modules.

6.1.6 SignatureDetector

The SignatureDetector performs signature-based detection. Currently the signature detector is only used as a white-list filter, filtering data that are known to be normal. Adding a programmed signature-based misuse engine is straightforward, except for creating the rule-base which may be more or less complex depending on the extent and purpose. The SignatureDetector is a passive entity used only by the main thread of the agent providing the following methods:

- `void addSignature(Signature rule)` - Adds a signature to the rule-base.
- `void deleteSignature(Signature rule)` - Removes a signature from the rule-base.
- `ArrayList getSignatures()` - Returns all signatures.
- `Signature getSignature(int signatureID)` - Returns one signature.
- `double detect(String[] data)` - Checks if one data item matches any signature.
- `void loadModel(File f)` - Load a set of signatures from a file.
- `void saveModel(File f)` - Save all current signatures to a file.

6.1.7 Module

All modules of the CHDA agent need to be initialised, reconfigured and logged. Rather than handling each module individually, we abstract away their primary functionality and consider them as generic modules whenever possible. The Module interface provides the following methods, implemented by all major modules of the agent:

- `void init(Properties props, Logger log)` - Handles initialisation of the module. The method is called by the owning object before the executing thread is started.
- `void start()` - Prepares the module for use, called by the executing thread before calling other methods of the module.
- `void reconfigure(Properties props, Logger log)` - Reconfigures one or many properties of the module and may reset the Logger.
- `void stop()` - Cleans up the module and closes resources.
- `void logStatistics(Logger log)` - Logs current statistics of the module to the given Logger.
- `void logProgress(Logger log, long prev, long now)` - Logs module progress. Called periodically.
- `void collectStats(Vector statNames, Vector stats)` - Returns all statistics collected by the module.
- `String getName()` - Returns the name of the module.

6.2 Life cycle

The CHDA is implemented in Java and should be able to run on any platform supporting Java 1.4. Offline testing of the agent using stored data files may therefore be performed on any computer with as least Java version 1.4.

For online training or testing the system running the agent needs also to support Tcpdump and the CHDA Preprocessor. Therefore the primary target platforms are Unix and Linux.

6.2.1 Startup process

The CHDA agent and related processes are executed through a start-up script:

- Initial configuration is read from a file. Some parameters are possible to change later via messages from other agents to the running CHDA, others will need to be fixed during execution.
- The Preprocessor is started, reading from the latest available Tcpdump data unless otherwise specified.
- A new Tcpdump process is started with suitable flags set sniffing data accordingly.

When the agent is instantiated the following events take place:

- Initial configuration is read from file
- Modules such as DataBuffer, AnomalyDetector and DataSource are dynamically instantiated based on the initial configuration read from file.
- Initial detector models are loaded from files, if such model files exist.
- The DataAcquirer thread is instantiated.

It would be easy to start up multiple instances of the CHDA agent, even on the same host, due to the agent paradigm. Different instances could process different types of data, or be used to share the load executing on multiple hosts if the data flow is too large to be handled by one agent and host.

6.2.2 The main loop

The main loop of the agent consists of:

- If there are messages waiting (e.g. reconfiguration), those are handled.
- If there are data available in the DataBuffer, detection/training (depending on mode) is performed on one data instance.
- If there are alerts ready, those are sent to the configured receiver, normally a correlation agent. If a receiver is not available, the alerts are stored locally.

- If forgetting is enabled, the detector modules are told using a reconfiguration method call, to look for unused subsets of their models and forget those.

- If there are no messages or data instances available, the main agent thread will sleep for a few milliseconds.

In parallel the DataAquirer thread will try to access data from its data sources and store them in the DataBuffer.

The agent could have had a more parallel structure with for example a third thread handling communication. This was considered unnecessary since the agent only provides two basic services, detection and reconfiguration. Reconfiguration may not be done on the anomaly engine during detection even if using separate threads for this. The DataSource module is executed by the separate DataAquirer thread to decouple the agent from delays related to the data source. Data will not always arrive at a constant rate.

Note that it is possible to quickly switch to training mode while performing detection, adapt the model by training on one or a few additionally known data items, and switch back to detection mode without significantly delaying detection. This is possible since the anomaly detection model using ADWICE is completely incremental.

6.2.3 Shut down process

When the agent is shut down the following events take place:

- Ongoing work is finished with the last message or data instance currently processed.

- All remaining alerts are sent. If the receiver is not available, the alerts are stored locally.

- The DataAcquirer thread is stopped.

- All Detector models are saved.

- The main thread returns.

6.3 Performance and scalability

The theoretical training and detection time of ADWICE using the index of BIRCH is $n * \log c$ where n is the number of input data and c is the number of clusters. Because $c << n$ this time is close to linear. In practice, the overhead of data access and agent processing need to be taken into account.

During our experiments we concluded that the agent was able to process 1 400 to 1 900 session records per second. This was the case when data were read from a single local file and using a model size of 12 000 clusters. During the experiments the agent was executing on a host with a Pentium 4 1,8 GHz processor with 512 MB memory.

In the Safeguard network multiple network sniffers where used as illustrated by figure 6.4 in section 6.1.2. Because one agent needs information from multiple hosts, remote data access have to be used. If the hosts at which the sniffers executed would be overloaded, the sniffers would start dropping packets and attacks could pass by without detection. Executing the agent on its own host, avoids the danger of having the agent sharing processing resources with a sniffer. The downside of this setup is the additional time required to process data. When accessing multiple files remotely encrypting traffic using SSH, the performance decreased, resulting in about 500 session records processed each second.

Considering this is a first prototype implementation and that the parallel remote data access of multiple files is quite intricate, the obtained performance is good. In addition it was enough for keeping up with the data rate in the safeguard test network, with hundreds of session records produced per second during evaluation.

The current implementation is not optimised in any way to increase performance. Some possibilities for improving performance include:

- Reduce the amount of logging currently used for evaluation purposes.
- Redesign the agent and algorithm implementation now when the implementation has stabilised somewhat.
- Keep the connection between the agent and the data source open, rather than setting up new SSH connections periodically.
- Split the agent into multiple instances sharing the load.
- Add more processing power and memory.

- Consider alternative methods of data access. Splitting the agent into multiple instances would for example remove the need for accessing multiple remote files in parallel.
- Improve the performance of the algorithm used, for example by splitting the model in smaller subsets, thereby reducing the time used to search the model. We will discuss this further in the final chapter of the thesis.

The linear time for training the model and the use of compact cluster summaries make the algorithm very scalable. We have trained models on millions of sessions, resulting in model sizes ranging from 5 000 to 60 000 clusters without scalability problems.

Chapter 7

Related work

This chapter discusses how Safeguard and ADWICE relate to other research in the area of intrusion detection. We start by describing a number of interesting agent architectures and compare those to the Safeguard architecture in section 7.1. In section 7.2 we change the focus to learning-based intrusion detection.

7.1 Agents for intrusion detection

This section briefly presents architectures and systems utilising agents for intrusion detection. The benefits of using agents in this setting have been briefly presented in section 3.2.1. Unlike the approaches presented below, Safeguard takes a much wider view. In addition to attacks, Safeguard aims to defend also against failures. Also, not only traditional communication networks are considered, but also the power domain, aiming to apply similar techniques across the different domains.

AAFID

AAFID [106](Autonomous Agents For Intrusion Detection) is a distributed intrusion detection system and architecture developed at Purdue University. AAFID was the first architecture to propose the use of autonomous agents for intrusion detection. Similar to the Safeguard architecture, there is a hierarchy of entities. *Filters* give agents access to various data sources providing a subscription based service of data selection and data abstraction. *Agents* receive data from local filters searching for intrusions and generating reports to the *transceiver* on the local host. Agents may be simple and specialised or large and complex, but they do not

have the authority to generate alerts, that is the task of higher level entities like the transceivers or *monitors*. One transceiver is executing on each host, collecting and processing reports from agents and redistributing them to other agents or monitors. Transceivers also control the local agents. Monitors are the highest-level entities in the architecture. Unlike transceivers they can control entities executing on many different hosts. They receive information from transceivers and lower level monitors and detect events that involve multiple hosts. Monitors may also communicate with one or more user interfaces.

Most agent architectures today are developed in Java, but AAFID was implemented using the programming language Perl [89]. During the project some of the disadvantages of Perl at that time were discovered, including big resource usage, lack of strict type checking and security problems.

Micael

Micael [33] is an intrusion detection architecture implemented using IBM Aglets [107] and Java. The architecture is developed at Universidade Federal do Rio de Janeiro in Brazil and makes use of mobility, unlike the AAFID and the Safeguard architecture.

The *head quarter agent* centralises the system's control functions, maintains copies of other agent's executable code and is responsible for agent creation. The head quarter agent does not normally move, but may do so if the load of the local host is very high or if the host is compromised.

Sentinels are agents that remain residents in each of the protected hosts, collecting information. If an arbitrary level of anomaly is detected, the agent requests that the head quarter creates a *detachment agent* that travels to the anomalous host. Depending on the type of anomaly, a specific detachment agent will be selected to verify with greater detail the anomaly and if appropriate perform counter measures.

Periodically the head quarter may create an *auditor agent* that will travel between hosts and check the integrity of other agents in the system. The auditor agent uses precompiled internal checksum tables, and may request that missing sentinels are restarted by the head quarter.

SANTA

SANTA (Security Agents for Network Traffic Analysis) presented by Dasgupta and Brian [32] is another project using the IBM Aglets platform [107]. Santa is

developed by ISSRL (Intelligent Security Systems Research Laboratory) at Memphis University. Four types of low-level *monitoring agents* are presented. They detect deviations at packet, process, system and user level by learning a model of normal behaviour using neural networks. The monitor agents send their reports to the *communicator agents*, a builtin agent type in the Aglets agent platform, that will in turn inform the *decision/action agents*. Each decision/action agents has a fuzzy controller, representing five severity levels using five corresponding fuzzy sets. The threat level reported by the monitor agents will be used to decide upon the fuzzy membership of those sets. If the threat level are medium-high or high, a *killer agent* will be dispatched, killing the responsible process. If the threat level is lower, a *helper agent* will provide status information to the graphical interface. The killer agent will also dispatch a helper agent to provide information about the handled threat.

CIDS

CIDS [31, 67] (Cougaar-based Intrusion Detection System), is an agent-based intrusion detection system also presented by ISSRL, using the Cougaar [29] agent platform and Java. In CIDS a *security node* consists of four different agents. The *manager agent* is responsible for giving commands and providing feedback to agents. The *monitor agent* detects anomalies on the packet, process, system and user level. The anomalies are reported to a *decision agent* that uses three decision engines, including one based on known attacks and one based on fuzzy inference. Each engine generates a bid with a corresponding action, and the action with the largest bid is selected and forwarded to the *action/response agent*. One can conclude that the functionality of the four monitor agents in the SANTA system has been merged into one and that the combined decision/action agent has been split into separate agents while adding additional functionality like the three decision engines. The action agent will build an IDMEF (see section 2.2.1) object, describing the current anomaly, the probable cause and the proposed action. The IDMEF object will then be presented to the operator through a separate user interface.

Experiments using an Nmap[1] port scan and a user to root (see section 2.1.1) attack are presented, and detected utilizing the agents. Multiple security nodes may exist in a network, each consisting of the above four agents and communicating through the manager agent. This level of the architecture is not evaluated.

Unlike the Safeguard architecture there is only one monitor agent responsible for collecting information from various data sources. Some data sources may pro-

[1] Nmap (Network mapper) [86] is a free open source utility for network exploration or security auditing.

duce a large amount of information in a short time, especially network traffic. In the Safeguard implementation a separate agent is normally responsible for each data source. In this way data from each data source can be reduced before correlation between separate data sources. The Safeguard solution will reduce the risk of information overload and performance problems at low levels in the architecture.

Compared to the agent hierarchies of AAFID and Safeguard, it is unclear how multiple security nodes of CIDS should be organised when large numbers are needed to protect many hosts in larger networks. It is also not discussed how and what information the manager agents should share.

PAID

Gowadia et al [54] from University of South Carolina present PAID, a probabilistic agent-based intrusion detection system. The idea is that agents should share their knowledge. This is implemented using Bayesian Networks [61], where each agent has three sets of variables. The *input set* are those variables of which other agents have better knowledge and the *local set* are used only internally. Finally the *output set* are those variables of which the agent has the best knowledge, which may in turn be shared with other agents by exchanging messages. The Bayesian networks in the current implementation are built by a domain expert, and the probabilities are assessed manually. It is assumed that the graph of agent communication corresponds to a tree, implying equilibrium will be reached and global consistency is assured. How this tree of agents should be organised for large networks is not specified.

PAID supports three types of agents. The *system-monitoring agents* perform online or offline processing of log data, communicate with the operating system and monitor system resources. Each *intrusion-monitoring agent* computes the probability for one specific intrusion type. Those agents subscribe to variables and/or beliefs published by system monitoring agents or other intrusion-monitoring agents. As an agent receives new information, its beliefs are updated accordingly. The *main IDS agent* is a singleton agent that supervises the working of the entire system and provides results to the user.

The agent system is now being implemented using the JADE [68] agent platform and Java. The directory facilitator (see section 2.3.1) of JADE is responsible for maintaining information about the published variables and monitored intrusions.

Multi-agent systems require agents to share information to reach a common goal. In Safeguard and many other approaches, this information often consists of human readable messages, like intrusion detection alerts. In the case of PAID,

messages are more closely related to the internal model of the agents which reduces the need for each agent to process the message before using the information. On the other hand, this type of message representation builds on the assumption that all agents use the same type of model which is not the case of Safeguard and many other agent systems with heterogeneous agents.

A denial of service resistant architecture

Mell et al [81] at NIST (National Institute of Standards and Technology) propose an agent-based intrusion detection architecture that is resistant to flooding denial of service attacks. In their model an *enterprise network* is built using multiple *domains* connected by the *enterprise bus*. Each domain consists of a number of *regions*, connected by a *backbone*. The backbone is assumed to consist of machines secured from penetration including firewalls, routers, switches and hardened security devices like IDSs. *Critical hosts* in the backbone house agents performing intrusion aggregation, analysis and control. One central idea is to limit the information the attacker may gain by attacking the individual regions. The IDS sensors located in the vulnerable regions are not allowed to connect directly to the agents in critical hosts. Rather a proxy agent is used. By allowing agents to move inside a set of critical hosts, and inside a set of proxy hosts, the IDS will be a moving target for the attacker. Also, the critical hosts will ignore network traffic if not coming from specific IDS sensors, limiting the information that can be gained by scanning the network. Because there is no communication directly from the vulnerable region to the critical hosts, the attacker can not target an attack directly towards critical hosts, and will only have knowledge of a subset of proxy hosts.

Each domain should function as a standalone IDS. Even if the enterprise bus is attacked, in many cases corresponding to the Internet, each domain will continue functioning and a DoS attack will be stopped at the domain fire wall. If a critical agent is attacked even though it is not visible, backup agents should be present to take up its responsibility. Agents can also move away from an attacked host.

Safeguard handles the threat of denial of service attacks against the IDS by letting higher level Safeguard agents execute in a separate hardened network. Similar to the work by Mell et al, it is necessary that host-based sensors reside in the more vulnerable regions to access local data. However, disabling local sensors will only reduce the information available to higher level agents. A missing agent will soon cause an alert to be raised, due to missing heartbeat messages in the case of Safeguard.

IDA

IDA (Intrusion Detection Agent System) [6, 7] is an intrusion detection system developed at the Information-technology Promotion Agency (IPA) in Japan using the D'Agents platform [56]. It is a host-based IDS, primarily intended to detect intrusions that deprive root privileges in Unix.

At every protected host, *sensors* are monitoring system logs in search of *Marks Left by Suspected Intruder* (MLSI). The MLSIs used in the presented work are root shell start-up and modification of critical files for local attacks, and start-up of shells and specific critical commands for signs of remote attacks. The sensor reports the discovered MLSIs and their types to the *manager*. The manager is responsible for the mobile agents, analysing the gathered information and providing an interface to the operator. The manager will launch a *tracing agent* to the host where the MLSI was found. The tracing agent will move from host to host tracing the intrusion independently of the manager. At each host the tracer agent dispatches an *information-gathering agent* that will collect further information on the local host concerning the MLSI and returns to the manager to report.

At each host there is a *message board* that agents use to communicate, to avoid that multiple tracing agents follow the same track. The manager provides a *bulletin board*, where all information gathering agents report their findings.

Of course only attacks giving rise to the presented MLSIs can be detected. On the other hand, concentrating only on those MLSIs implies that only a subset of data needs to be processed. Safeguard, rather than concentrating on specific subsets of attacks, filters irrelevant data as early as possible to avoid unnecessary processing at higher levels.

ANT

ANT [44, 46] is an intrusion detection framework based on mobile agents mimicking behaviours of social insects developed at Université Claude Bernard Lyon. Mobile *intrusion detection agents* randomly travel between hosts monitoring local machines trying to discriminate between normal and abnormal behaviour. If an anomaly is detected, the intrusion detection agent will send an alert by building and spreading *artificial pheromone*. The pheromone consists of a number of fields including a gradient that can be used by agents to trace the source of the alert. The pheromone field is detected by *intrusion response agents*. Those will climb the pheromone gradient to find the target of the attack and implement various response behaviours.

The alert distribution and response of ANT is very different from Safeguard

and most other approaches. It is important that the administrator trusts and can predict the behaviour of protective systems in the network. This may be easier using traditional communication means, rather than using artificial pheromone. ANT has been tested in a simulation, illustrating that agents can find the source of attacks using the pheromones. The emergent behaviour if applied in a real network as well as a detailed comparison with traditional communication means would be very interesting to study.

Lightweight agents

Helmer et al [62] present an agent architecture in multiple levels developed at Iowa State University. At the bottom level stationary agents are used at each data source to collect and clean data. At the next level, mobile agents travel between subsets of data cleaning agents to monitor and classify ongoing activities. At the highest level data mining and data fusion agents are located together with databases and an interface agent.

The agent architecture is implemented using the Voyager [113] agent platform and Java. One feature of this platform is the possibility to dynamically aggregate agents. Such added functionality called *facets* is used to extend the behaviour of agents dynamically providing significant flexibility.

Agents in general provide a very flexible implementation of an intrusion detection system. Using most agent platforms, it is easy to start, stop, upgrade and clone individual agents without shutting down the complete system. Using Java, regardless of the choice of agent platform, it is easy today to load classes dynamically and update different aspects of agents without recompilation. This is actually implemented in our clustering hybrid detection agent (see chapter 6) to remove the need for recompilation when alternating between different module implementations. Because the Safeguard agents do not move, the amount of code used by the agents is less relevant than for architectures making use of mobility.

7.2 Learning-based anomaly detection

Significant effort has been devoted to learning-based anomaly detection and countless papers have been published in this area of research. In the following sections we present a range of different approaches that have been proposed. Of course this list is not complete, but it is representative and will provide some insights about work done in this field. In the first section we start by describing approaches that like ADWICE make use of clustering. In section 7.2.2 we describe other

techniques for learning-based anomaly detection and finally we discuss important properties of those alternative techniques and compare them with ADWICE.

7.2.1 Clustering-based anomaly detection

ADWICE builds a model using clustering. In this section we present some other approaches using clustering-based data models, and discuss how they relate to ADWICE.

Watcher

Munson and Wimer [84] have used a clustering-based pure anomaly detection technique to protect a real web server executing on an instrumented Linux system. Each c-function of the Linux kernel is given an identity, and each time a c-function is called a corresponding counter is increased. Because there are 3 300 instrumentation points, this results in a 3 300 dimensional vector of counters. During a well defined time-interval, an *epoch*, the counters are increased, resulting in a profile vector at the end of each such time interval.

To reduce the number of dimensions, statistical filters are applied. If two different c-functions are always called together, including both in the model does not add information. The complete number of modules is thus mapped into a much smaller set of 100 virtual modules.

During normal execution profile vectors are collected and together form a model of normal behaviour. Munson and Wimer noted that the vectors when plotted together in a 100 dimensional space formed natural clusters. Each such cluster was then represented by its centroid, a centre point and a radius, resulting in a very small model compared to the large set of original vectors.

Watcher is evaluated by protecting a real computer connected to Internet and inviting interested people to try to get root access. Execution profiles are compared to the model in real-time, and by including real-time responses like killing processes and banning IP-addresses, attacks can be stopped autonomously before succeeding. The experiment resulted initially in a 100 % rejection rate of port scans, buffer overflow attacks and worms.

The authors state that the threshold for rejection was set quite low. This of course makes detection of attacks more probable while increasing the number of false positives. The authors do not include evaluation of false positives in their experiment.

The Watcher model representation is very similar to that of ADWICE. Even if current evaluation of ADWICE makes use of network data, the algorithm could

also be applied to application data similar to Watcher. Unlike Watcher that in the presented implementation builds a static model, ADWICE provides multiple possibilities for adaptation and a search index to increase performance.

Intrusion detection with unlabeled data

Intrusion detection in general builds on the assumption that intrusions are qualitatively different from normal data. Portnoy et al [90] build a model of *unlabelled* training data, adding the assumption that the number of normal instances vastly outnumbers the number of intrusions. Building on this assumption, large clusters of data are given the classification normal, while small clusters will be considered intrusions. Based on initial experiments, a fixed percentage N is used to decide how many large clusters that will be labelled normal.

Portnoy et al use single linkage clustering, similar to BIRCH and ADWICE, implying one pass through training data and linear training time. Feature vectors are normalised using mean and standard deviation. Euclidian distance is used for continuous values, adding a constant for each disagreeing categorical value. Unlike ADWICE a fixed cluster width is set, rather than step-wise increasing this width. Adaptability is not considered, but periodical retraining on two-week basis is discussed as a possibility to keep the model updated.

The detection method is evaluated on the KDD-99 data set. Cross validation was used, meaning the original data set was divided in multiple parts, each containing about 490 000 data items. Only four out of ten subsets were used, because those contained sufficient variations of different types of intrusion and normal data. Training with the raw data set yielded very poor performance, because the KDD-99 data set consists to a large extent of intrusions, thereby breaking the assumption that normal data should be much more common. Therefore the original data set was filtered, resulting in data sets with only about 1 % attacks. Multiple tests were then performed, choosing one subset for training and another for testing. For some subsets a 1–2 % false positives rate at a 50–55 % detection rate were obtained, but other subsets produce considerably inferior results. Even though the detection rate is lower than some other approaches, one should keep in mind that this was achieved without need for complete data cleaning or labelling. One should however note the need for filtering of the KDD training data to fulfil the assumption that normal data outnumbers attack data. Such filtering would also to some extent be needed if the approach is applied to real data. Flooding denial of service attacks and scanning attacks will otherwise give rise to large clusters breaking the underlying assumption. This filtering may however be easier than the need for removal of possible less visible attacks required by pure anomaly

detection such as ADWICE.

Y-means

Guan et al [57] use a clustering approach called Y-means to model unlabelled network data. Based on the population in a cluster, the cluster will be labelled as normal or abnormal. Large clusters are considered normal as in the work by Portnoy et al [90].

Y-means overcomes two shortcomings of the original K-means algorithm. K-means requires the user to choose a suitable number of clusters (K) which is critical to the clustering result. In addition some of the K-clusters may be empty after clustering. Y-means starts like K-means by assigning all instances to K clusters. If there are empty clusters, new clusters are created using data points that are very far from their cluster centres. When empty clusters have been eliminated, K can be adjusted by splitting and merging clusters. If δ is the standard deviation of data, 99 % of the data points will statistically be inside a $2,32\delta$ radius of a cluster. The authors call this area the *confident area* of the cluster. Data points outside this area are deemed outliers, and the remotest outliers will be used first when creating new clusters by splitting. Splitting continues until no outliers exist. To avoid over splitting, clusters with overlapping confident areas will be merged.

Y-means is like ADWICE evaluated using the KDD-99 data set. However, only a small subset consisting of 12 000 data points are used to create the model. On this small subset, the authors present a detection rate of 82,32% at 2,5% false positives rate. This result is, somewhat unexpectedly, significantly better than the result by Portnoy et al [90] also performing unsupervised anomaly detection using clustering but on other larger subsets of data.

The KDD-data consists of some types of attacks (scanning, DoS) that consist of a large number of data records, while other attacks may consist of only one packet. This means that a small random subset may not truly be representative of the complete data set, making such experiments hard to compare with experiments on the complete data set or other subsets.

Similar to the work by Portnoy et al [90], Guan et al perform unsupervised anomaly detection modelling both attacks and normal data detecting intrusions based on cluster sizes. ADWICE models only normal data, thereby performing pure anomaly detection. ADWICE, designed to be scalable, was able to model all 1 000 000 normal data points without difficulty rather than only a small subset. Unlike ADWICE and the work by Portnoy et al, Y-means requires a large number of passes through training data.

Clustering packet payload

The meaning of a network packet header is well known, but the payload varies a lot and consists of significantly less structured data. Many attacks are not visible if considering only packet headers. This calls for general methods ignoring the abundance of payload content types, while searching for defining differences between normal and attack payload.

Many proposed detection methods, including our application of ADWICE in the Safeguard context, ignore the payload and apply learning only to header-based features or a small number of selected content-based features. Zanero and Savaresi [121] on the other hand use clustering in a first stage to group payload into a number of different classes represented by a single byte, thereby significantly compressing the information. In a second tier an anomaly detection algorithm based on self-organising maps (SOMs) [72] is applied to payload classes together with the header features.

Experiments were performed with three different clustering algorithms for the first stage, of which the authors consider SOM to be the most successful. The authors used the DARPA/Lincoln Labs 1998 data set for their proof-of-concept evaluation, which corresponds to the data used to produce the KDD-99 data set. They used a 10×10 SOM, thereby organising data in 100 different clusters. The network was trained in 10 000 epochs over a representative subset of network packets. The authors do not state how many packets were used for training, or how much time was needed to train the model.

2 000 packets are classified using the trained SOM. When the number of packets in each class is presented for normal and Nessus[2] traffic, it is clear that the distribution of packets into classes is different. However, some Nessus packets will be classified into the same classes as normal packets and vice versa. It is shown that normal FTP traffic is classified into a small set of 8 classes out of total 100. Examples of FTP-based attacks are given, of which two were classified into classes not containing any FTP-traffic, and one into a class that is an uncommon FTP-class. This suggests that the compressed information can indeed provide some useful contribution to the detection of payload-based attacks.

If further evaluation of the approach proves that the results hold in general, much work on anomaly detection including ADWICE could be extended with such compressed payload-based features. However, unlike ADWICE, SOMs requires significant amounts of training, and is not as easily adapted when normality changes.

[2]Nessus [85] is a security scanner for various flavours of Unix. It performs a large number of remote security checks, and suggests solutions for security problems.

The authors claim that adding the payload-based feature to the second stage increased detection rate by 75% at the cost of an unspecified but limited increase in false positives.

Detection of modified files

When having compromised a machine, the attacker may install a root kit to facilitate easy access in the future. By using data bases of hashes of known normal files, and comparing those to hashes of potentially compromised hosts, intrusions can be detected. However, for large networks with many different types of systems and different patch levels, maintaining a complete data base of benign files may not be practical. Carrier and Matheny [20] propose a method to use when the original hashes are unknown or hash data bases incomplete.

Hashes of a set of relevant system files are collected from all hosts in the network. The vectors of hash values are then clustered. The distance between hosts is defined to be the number of hash values that are different. In the presented experiments all hosts were clustered into sets, such that the distance between all hosts in a single cluster was zero. This corresponds to all hashes being equal.

If one assumes, that only a minority of the hosts in a large network are compromised, and many uncompromised hosts have the same patch level, suspected hosts may be detected by examining the size of host clusters. Small clusters of hosts are more probable to be compromised in this case.

By handling host clusters rather than individual hosts, false positives are more easily handled. If a cluster of hosts results in a false positive, examining only one of the hosts in the cluster suffices to discard this false alert. The authors therefore also count false positives as the number of erroneously detected clusters, rather than individual hosts. Experiments using a simulated data set, show that the method has a potential to help identify compromised hosts and reducing the work of examining suspected hosts.

Clustering normal user behaviour

Oh and Lee [87] cluster each data feature by itself, while taking user transactions into account when creating user profiles. A user transaction is represented by a list of occurrence frequencies of kernel signals for all the commands of a user during a session. Similar data objects of each feature are clustered together and clusters represented by a minimum, maximum, count of the represented transactions, the centre and ratio together with their standard deviations. The centre is computed by taking the average of the data objects in each transaction, and then

computing the average over the number of transactions in the cluster. Similarly, the *individual repetitive ratio* of a transaction in a cluster is the number of data objects in the present cluster and transaction, divided by the complete number of data in the transaction. The overall ratio is the average of individual ratios over all transactions.

The user profiles consist of internal cluster summaries as explained above, and external summaries representing statistics of the noise data objects not present in clusters. During detection four different anomaly measures are computed for each feature when comparing a new transaction with existing user profiles. The internal distance is the difference between the centre of a cluster and the average of data objects in the user transactions that are in range of the cluster. The global internal distance is then computed by taking the average of all individual internal distances. The other three measures are called internal ratio, external difference and external ratio. If the anomaly measures are larger than a threshold, the user transaction is suspicious.

Two sets of experiments are presented. In the second user logs from the DARPA/Lincoln Labs 1998 data set were used and the results compared to the classic statistics-based NIDES intrusion detection system [2]. At about 10% false positives, all anomaly measures resulted in almost 100% detection rates. NIDES had a higher false positives rate and a lower detection rate.

Unlike this work, ADWICE clusters all features together. This facilitates detection based on combinations of anomalous values that may go undetected if only considering features one by one. On the other hand, considering features together makes it more difficult to understand what features that were the primary cause to the anomaly. ADWICE provides some help in this context, by including into the alert the individual features that contributed most to the overall anomaly level.

ADWICE in its current form does not represent outliers separately (i.e. external summaries). Normal outliers either are present in the model as sparsely populated clusters, or they may be removed if forgetting is applied.

Fuzzy clustering

Shah et al [102] use fuzzy clustering to detect anomalies in low level system data including statistical network data, process information and global system data such as memory usage and CPU load. Samples are collected at a rate of once per second. In total 189 features are collected, but using principal component analysis the number of dimensions are reduced to 12. The resulting vectors are clustered using the Fuzzy mediods algorithm [73].

Unlike traditional clustering approaches, including ADWICE, data may belong to more than one fuzzy cluster. Membership is not a binary function. The Fuzzy mediods algorithm outputs n representative objects (mediods), such that the total fuzzy dissimilarity within each of the n clusters is minimized.

The diameter of the clusters is computed as the average pair-wise distance between members of the cluster, and the radius is half this distance. A threshold based on the radius is compared to the distance to an unknown data point to decide if the data is anomalous or not.

A model is trained on data collected in an attack-free environment, using a Linux kernel running a vulnerable Apache server [3]. A number of different attacks are used for testing. One of the best results was an attack that was detected with a 43,53 % detection rate at a 16,12 % false positives rate. The authors state themselves that the false positives rate is high, but they assume that higher level reasoning may mitigate that to some extent.

Unlike ADWICE, clustering of Fuzzy mediods requires multiple iterations over data objects increasing training time and making incremental training and adaptation less viable.

Seurat

Xie et al. [120] collect system file change data from multiple hosts and use clustering to detect common changes at multiple hosts at specific time intervals, possibly indicating distributed attacks such as worms. Each host is equipped with a light weight tool that collects a daily summary of file update attributes. The attributes include file name, type, device number, permissions, size, time-stamps and MD5 checksum. Currently only a binary bit representing whether the file was changed or not during the present day is used.

Data vectors from the detection window are clustered and compared with clusters from previous t days (e.g. the *comparison window*). The clustering algorithm is a simple iterative algorithm normally used for K-means initialization, using no prior knowledge of the number of clusters. A cluster is called a *new cluster* if it consists of multiple vectors only from the detection window. This indicates abnormal file updates and causes generation of an alert.

7.2.2 Other selected techniques

Of course there are countless other approaches to model intrusion detection data besides clustering and below we take a broader view of learning-based intrusion

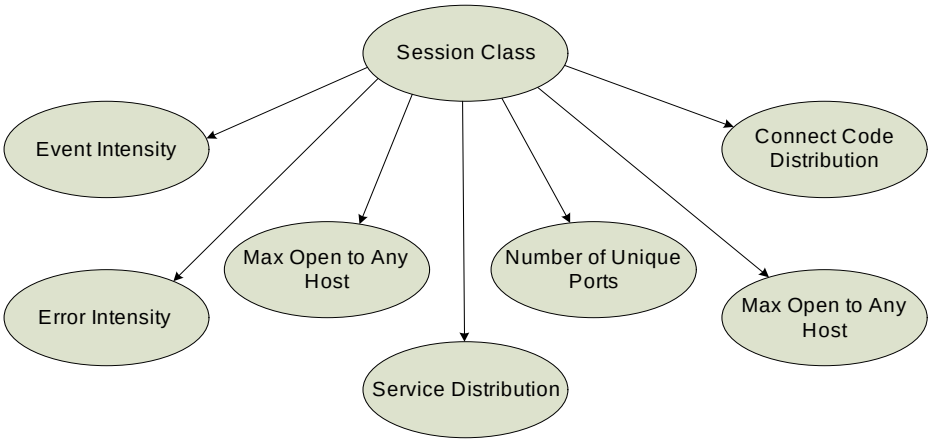

Figure 7.1: eBayes TCP model

detection. In this section we broadly present a number of different such approaches by briefly explaining them and in the next section we summarise some of their key properties in the context of ADWICE.

EBayes TCP

Valdes and Skinner [111] present eBayes TCP, a component of a larger system called EMERALD [42]. The model is a Bayesian tree where the root node contains the different hypotheses and the leaves contain observable events as shown in figure 7.1.

Two possibilities for model adaptation are presented:

- If a single hypothesis is dominant at the root node this dominant hypothesis will be reinforced.
- A dummy state is included in the root node and the belief of this state is computed as for other normal states. If the dummy becomes the dominating state, it is promoted to a normal state. Dynamic hypothesis generation realises use of unlabelled data for training the model but may then not be able to discriminate between different attack classes.

A real-time IDS based on learning program behaviour

Ghosh et al. [52] describe three machine learning techniques to model normal program behaviour using sequences of system calls from the Basic Security Module

(BSM) in Solaris. The presented approaches are:

- An Elman neural network is used to predict the next sequence of events at any point in time. The n:th input sequence I_n is presented to the network which produces an output O_n. O_n is then compared to I_{n+1} to obtain the prediction error (i.e. the anomaly measure).
- A finite automata where states correspond to n-grams in the BSM data and output states correspond to sets of l-grams (where $l < n$) that may be seen when the automata is in a given state. During training statistics of successor l-grams are collected through counting. During testing the deviation of the successor l-grams from expected values is used for anomaly scores.
- A finite automata that represents program behaviour is automatically constructed from normal training data. One or more n-grams are assigned to each state. The automata predicts an n-gram N if there is a transition from the current state to the state corresponding to N and if the transition is labelled with the first l events of N. If there is no matching transition labels no prediction is made. If the matching transition leads to a state not containing N this is considered a prediction error (i.e. an anomaly).

Intrusion detection applied to CORBA objects

Marrakchi et al. [79] perform application level intrusion detection by modelling client request at a CORBA [27] server. Request interceptors of the VisiBroker ORB [28] are used to collect request names and their parameter values. Each client connection is considered one data item, where bind/unbind between server and client object corresponds to connect/disconnect.

The normality model is a tree structure where nodes correspond to requests, the root marks connection and leafs disconnection. Each node also contains information of the parameter values. During training all client requests are assumed to be attack-free and new branches are added to the tree as new connections are presented to the system. In a second manual phase during training a tolerance interval is constructed for each numerical parameter value.

During detection incoming requests are matched against the model following a path from the root to a leaf. During tree traversal the similarity between the present connection and the model is computed. Similarity starts at 1 and receives a penalty at each node where parameter values do not match normality. Alerts are generated if an unexpected request is detected during tree traversal or if a leaf node is reached and the similarity is less than a threshold.

CDIS: Computer defence immune system

Williams et al [117] present an anomaly detection scheme based on the computer immune system approach. The data used are 28 header attributes from individual TCP-packets, and 16 attributes from UDP and ICMP which are processed into binary strings of 320 bits. Each non-binary attribute is represented by an interval rather than a fixed value, thereby realising imperfect matching.

Initially the system goes through a training phase (see figure 7.2). Antibodies are created as random binary strings and matched against a corpus of self-packets thereby performing negative selection with removal of matching antibodies. The model may then optionally be tuned by *affinity maturation* to obtain more general antibodies. The attribute intervals are then increased until they start matching self after which they are decreased again. The initial training is a very time-consuming task performed off-line.

During detection the model of antibodies is matched against network packets. After an initial match against one antibody, an extended search for other matching antibodies is performed. If additional matches are found on the local host or another remote host, the antibody is said to be *costimulated* and only then an alert is generated to reduce false positives.

The dotted parts of figure 7.2 are not yet implemented. This includes removing (forgetting) antibodies after a certain life-span and to reinforce those antibodies that generate alerts to handle changing system behaviour.

Data mining methods for detection of new malicious executables

Schultz et al [99] have evaluated the application of several data mining methods to the problem of detecting previously unknown malicious executables.

The authors compare a base-line signature-based approach against the RIPPER algorithm [24] and two approaches based on the naive Bayes approach. The RIPPER algorithm used character strings within the executables as features for generating classification rules. The Naive Bayes algorithms used character strings as well as sequences of byte codes from the executable files. In addition to the standard Naive Bayes the authors introduced the so-called multi-Naive Bayes, which basically splits the features in different sets and then applies the standard naive Bayes to each set of features, and then makes a vote based on the prediction for each set of features.

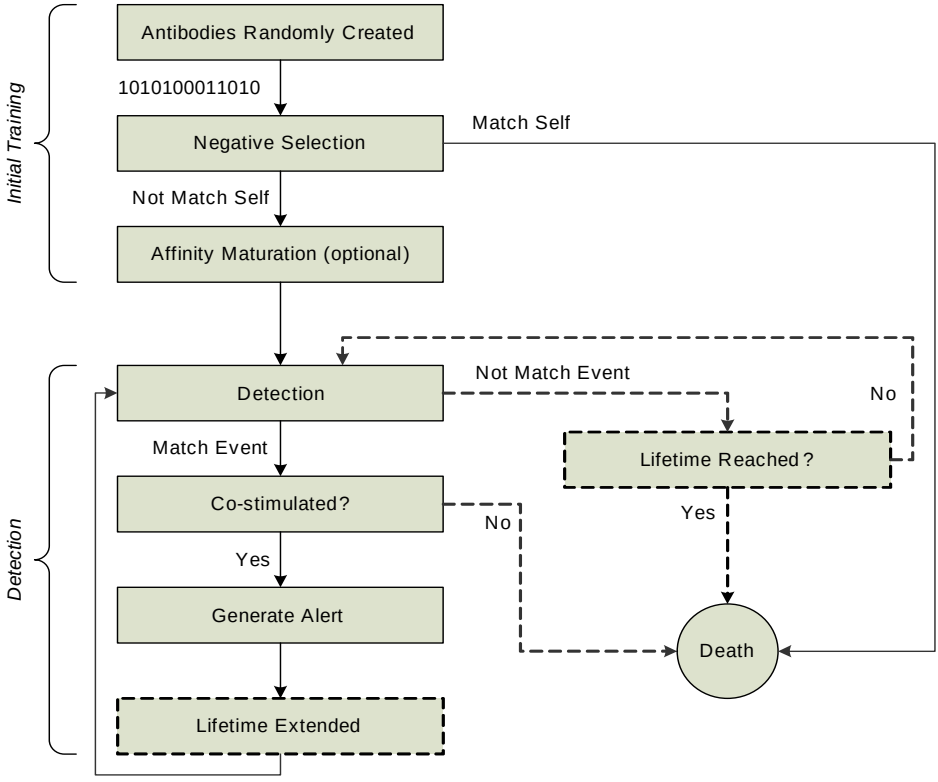

Figure 7.2: CDIS antibody life cycle

Monitoring anomalous windows registry access

Apap et al [4] introduce RAD (Registry Anomaly Detection) which uses a probabilistic normality model to detect anomalous Windows registry access. From each registry access five features are extracted, namely the process name, query (set, get ...), key, response (success, not found, buffer overflow ...) and result value. Since the Windows event logger causes a major resource drain if used to log all registry accesses, the *basic audit module* was implemented to create and store registry-based feature vectors in a data warehouse.

A set of consistency checks is applied to detect anomalies. $P(X)$ for each feature and $P(X|Y)$ for each pair of features are computed resulting in 25 consistency checks for each registry access. A normality model is computed out of the collected normal data. For each data record, an anomaly score is computed and then compared with a user threshold to decide whether to send an alert or not. Training is performed off-line, while detection is performed real-time, keeping up with the 50 000 records per hours produced during experiments

Detecting anomalous network traffic with self-organizing maps

Ramadas et al. [94] use self organizing maps (SOMs) [72] as normality models for network data. Network packets are captured with the tcpurify program, which captures only the first 64 bytes and has the option of obfuscating the sender and receiver addresses. The tcptrace program then reports a number of periodic messages during a TCP or UDP connection (for UDP defined using time-outs). 6 features are used (and reported from tcptrace) including questions per second, average question size and idle time between questions.

A self organizing map converts non-linear statistical relationships between data points in high dimensional space to relations between points in two dimensional space. During training all neurons are presented with each training sample, and for each sample the closest neuron and its neighbours are adjusted closer to the sample. During detection a sample is compared to all neurons and if there is more than two units of standard deviation from the closest neuron it is considered anomalous.

Detecting self-propagating email using anomaly detection

Gupta and Sekar [59] use a state machine and statistics to detect increasing volumes of email traffic. The data source is email server logs, but the current evaluation simulates the users.

The basis is a state machine for email server operation with one send event and an arbitrary number of deliver events. Each email creates a new instance of the state machine. Four frequency statistics are captured at the email server, ranging over all clients as well as individual clients.

Each statistic was maintained at multiple time-scales ranging from about one second to one hour. Rather than using averages the frequency distribution was captured using time windows and histograms using geometric bins (since value ranges are not known). The severity of an anomaly is defined to be the difference between the highest non-zero bin at detection and training.

HMM-based intrusion detection

Cho and Han [21] propose two techniques to improve HMM-based (Hidden Markov Model [93]) detection applied to events from the basic security module in Unix. To simplify modelling, the amount of training data is reduced by concentrating on events related to privilege flows. This includes change from user and group ID to execution ID root when setuid processes[3] are used. In this way only about 25% of the original events are needed. Performance is improved by combining multiple HMM-models.

The system call related information, such as file paths and system call return values, is too large and varying to be directly modelled using HMMs. The many different measures are reduced by applying a self-organised map so that all measures are converted into one representative.

Training is performed by observing training sequences and updating the model until probabilities for the normal sequences are maximised. During detection the probability that a specific sequence is normal (generated by the HMM) is calculated.

Anomalous payload-based network intrusion detection

Wang and Stolfo [115] present PAYL, a fully automatic, simple, unsupervised, and incremental anomaly detector. The data source is binary payload content where the payload may optionally be truncated. The simplest approach considers payload from individual packets but payload from complete or partial connections may also be used. To obtain a good model, the complete data stream needs to be divided into smaller subsets so that similar payloads are modelled together.

[3] In some applications, a user process needs extra privileges, such as permission to read the password file. Unix systems offer a set of system calls, called the uid-setting system calls, for a process to raise and drop privileges. Such a process is called a setuid process.

Usually the standard network services have fixed pre-assigned ports (e.g. port 80 for HTTP, 21 for FTP). Each application has its own payload distribution. The distribution of payload also varies due to payload length where longer payload may indicate non-printable characters indicative of media files or executables. Also the direction of the data stream is relevant.

A sliding window of one byte is passed over the complete payload and the occurrence for each byte is counted. For a payload the feature vector is calculated by dividing each byte count with the total number of bytes. Each byte is treated as a variable and its mean and standard deviation computed.

Given a training data set a model M_{ij} is computed for each observed length i and port number j. M_{ij} stores the mean byte frequency and standard deviation of each byte's frequency. Incremental updating of the model is possible. During detection the distribution of the tested payload is computed and compared with the model.

Anomaly detection based on eigen co-occurrence matrix

Oka et al. [88] introduce a new method, Eigen Co-occurrence Matrix (ECM) that models system call sequences and extracts their principal features to be used for anomaly detection. For each sample user sequence of length l a *co-occurrence matrix* is constructed by counting the occurrence of every event pair within a certain distance. This results in an $m \times m$ matrix M_i for each sequence where m is the number of unique events. By finding the principal components the very large matrix can be reduced. A feature vector is obtained by projecting M_i into the reduced co-occurrence matrix space. The feature vectors are used to construct multiple layered networks with events as nodes and connections as edges. Unknown data are then compared to those networks.

An empirical analysis of target-resident DoS filters

Collins and Reiter [25] analyse the performance of several previously proposed techniques for denial of service filtering.

Four different filtering techniques are evaluated. Two of the techniques make use of source address data, and the other two make use of coarse indications of the path that each packet traverses on its way to the destination node.

Network-based	Packet headers	4
	Complete packets	1
Host-based	System calls	3
	Windows registry	1
	Files	1
Application-based	CORBA requests	1
	Email server	1

Table 7.1: Data sources

7.2.3 Discussion of key properties

This section discusses some key properties of the presented non-clustering based techniques and relates those to ADWICE.

Data source

If we classify the papers according to what data sources they use we get the result shown in table 7.1. We can conclude that all three considered types of data sources (see section 2.2.2) are present. The most common ones are packet headers and system calls. Using packet headers rather than complete packets implies that a majority of attack classes, those visible only in payload, will not be detected. On the other hand, using payload may decrease performance if the payload needs to be parsed and may aggravate the privacy problem due to user data processing. Deciding how to use the unstructured packet content or what features to extract is not straightforward. This may explain why more effort has been spent on processing packet headers.

In the Safeguard setting we used only packet-based features for ADWICE to minimize processing time and avoid the feature selection problem for payload data. Selecting a specific number of specific features will make the detection approach less general. It will possibly reduce the probability that new attacks will be detected in the case when they are not visible in the selected features. General approaches to payload does not have this problem, including the work by Wang and Stolfo [115].

Detection method

In this section we summarise what subclasses of anomaly detection are present in the surveyed work. Pure anomaly detection means that only normality is modelled while hybrid anomaly detection provides explicit models also of attacks.

Pure anomaly detection	10
Hybrid anomaly detection	3

Table 7.2: Detection methods

Labelled attack and normal data	3
Only normal training data	10
Unlabelled normal and attack data	1

Table 7.3: Training data

This means that a model built of normal usage statistics will be considered pure in this sense, even though the model may also contain some attacks. On the other hand, if a learning algorithm has knowledge about attack classes in addition to normality, this approach will be considered hybrid. Table 7.2 shows the resulting classifications for the survey. Note that the work by Williams et al. [117] represents normality by its inverse by using antibodies.

In table 7.3 the training data used by the different approaches is summarised. We can conclude that most approaches assume that training data is normal and therefore will consider attacks included in the training data as normal. This is the approach taken by ADWICE too. There are also a number of supervised approaches [25, 99, 111], where data has to be labelled. One interesting approach is the work by Valdes et al. [111]. The learning scheme is able to adapt the existing model with new attack hypothesis using unlabelled data. No approach performs full unsupervised anomaly detection or is trained only on unlabelled data containing attacks. In the section covering cluster-based techniques both Portnoy et al. [90] and Guan et al. [57] are applying unsupervised anomaly detection.

Detection algorithm

Below is a list of the different algorithms used for learning:

- Bayes learning [99, 111]
- Neural networks [52, 94]
- Finite automata [52]
- Computer immune system [117]
- Decision trees [99]

	Yes (Training)	Yes (Detection)	No
Performance evaluated	3	7	4

Table 7.4: Performance evaluations

- Probabilistic model [4]
- Hidden Markov Model [21]
- Statistics [59, 115]
- Other methods [79, 88]

As we can see, many different approaches have been evaluated, and even though this list is representative, there are certainly other methods not included here. Unfortunately the motivation behind choosing a certain learning scheme is not always presented. More explicit discussions of alternative learning schemes and why they have been considered inferior would have been helpful for directing future research. Some authors do not consider the use of a specific learning scheme the primary contribution. This may to some extent explain why the specific properties of the selected learning scheme are not discussed more.

When it comes to on-line detection (and adaptability) execution performance is very important. Table 7.4 shows to what extent performance is measured. Considering training time is less common than time for detection, because many approaches assume off-line training and no on-line adaptation. In many cases where performance is considered the authors only conclude that the detection scheme is fast enough to keep up with the data processed during evaluation. Few papers [88, 94, 115] present the maximum throughput of the detection scheme. In our work with ADWICE we consider time for detection as well as training, because we want to provide the possibility of on-line adaptation.

Usage frequency

To minimize time to detection, the intrusion detection system needs to be vigilant at all times, using continuous on-line detection. If the detector is executed only periodically, attacks may be missed. On the other hand, periodic usage requires significantly less resources in terms of processing power than continuous detection.

For real applications, what is considered normal will vary over time. New users and services will be added to the system and old ones stop being present in

	Yes	No
Continuous detection	12	0
Continuous model adaptation	1	11

Table 7.5: Usage frequency

the system. In such a dynamic environment the model needs to adapt to changing behaviour. This may be performed continuously or by periodic retraining. If continuous training is implemented, it is possible to always keep the model up to date and minimize the need for off-line periodic retraining. Adaptation may also be problematic, since frequent normality may become very dominant in the model depending on which techniques that are used. Adaptation may also give rise to new risks, because an attacker, at least in theory, for certain implementations may slowly adapt the model to include also attacks.

Table 7.5 shows to what extent the different approaches implement on-line detection and continuous model adaptation. Most surveyed research targets on-line detection, although in many cases only off-line evaluation has been performed and full continuous detection prototypes have not been implemented.

Valdes et al. [111] present two possibilities for adaptation, including new attack hypothesis generation. The other surveyed approaches perform training off-line and do not consider continuous adaptation or leave this as future work. Some approaches may be able to be extended to continuous adaptation but such extensions are not discussed.

Alert granularity

An intrusion detection system needs to provide enough information on detected events, so that other systems or humans may respond to the attacks appropriately. The alerts produced by the intrusion detection system may consider the attacks on different levels and thereby limit the specific information that can be provided to the human. Most surveyed approaches can provide information of the host/user targeted by the attack, and in many cases also about the specific process or application.

When it comes to the type of attack, for most approaches the operator will get no specific information on the type of attack detected. This holds also for ADWICE. This is implied from the fact that most approaches are pure anomaly detection systems and are trained only on normal data.

Some approaches [99, 111] have hybrid models, modelling both normal and

attack data. Those may provide more specific information on known attacks at the cost of higher training data requirements or attack knowledge.

Anomaly detection is primarily used for detecting new attacks. Often misuse detection will be used in parallel providing the user with very specific information on the attack type of known attacks.

Attack resistance

The attacker may want to cover her tracks, thereby giving the motivation for attacking the intrusion detection system itself or hide the attack in normal data. Examples of attacks against intrusion detection system are for example evasion [92] and mimicry attacks [114]. It is important to consider how vulnerable the intrusion detection system is to such attacks.

Very few papers consider the security or insecurity of their detection approach. Some authors claim that they will consider at least mimicry attacks in future works [4, 115].

In our work, attack resistance was mostly considered on the global level of the agent system. We expect that individual agents with some difficulty can be disabled or evaded, but the complete system should still be able to detect the attack with high probability. Applying multiple detection schemes in parallel as in the case of Safeguard, makes it much harder to avoid detection than evading a single anomaly detector.

Evaluation data

There exist a number of different approaches to intrusion detection evaluation. To avoid practical problems data may be simulated, using more or less complex models of users and systems. This makes evaluation of very complex systems possible and avoids the difficulties of collecting real data. Unfortunately it is hard to prove that the simulation results are valid in a real world setting.

Collecting or generating good realistic data for intrusion detection system evaluation is a difficult task. Ideally real normal data produced by real users in a real system should be used. Then regularities arising from use of simulated data can be avoided. This is especially important for learning-based systems since they otherwise may learn properties of data not present in the real world. Using real data may however introduce other problems, e.g. related to privacy. [77]

To evaluate detection accuracy, attacks need to be included in the data. In the most realistic setting, the attacks are performed by real possibly unknown as well as malicious adversaries. Unfortunately this implies a number of difficulties and

		Attacks		
		Simulated	Real attacks	Real attackers
Normality	Simulated	2	4	
	Real data	1	4	3

Table 7.6: Use of real world data

is therefore not very common. For evaluation we need to know what attacks are present where in the data and this may require extensive analysis if the intent and even presence of the attacker is unknown. We also do not have any control over to what extent different attack types are included in the data, and suitable attack coverage may be hard to obtain.

An alternative way of collecting attack data is to let non-malicious people perform the attack in a well-controlled setting. This means we have real attacks, but not real attackers and reasonable realistic attack data may be produced. Now we may have full control on what attacks are present in data and where. A problem with this approach is if the normal data is collected in a real world network. Then performing attacks targeting the real network will in most settings not be a viable option, even when the attacker is known and not truly malicious. A way around this may be to inject the attacks in data off-line or during replay of normal data. This may however again introduce features in data not corresponding to the real world setting.

In table 7.6 we classify the surveyed evaluations according to what extent real world data is used. If the system is evaluated using only real data, normally the experiment is very limited when it comes to attack coverage [25,111,115]. Using real data may however increase the confidence that the approach is able to model real complex systems. A good trade-off is to use real users to produce real normal data and real but controlled attacks [4,21,94,99]. In those cases often a reasonable number of attacks have been performed. If simulated data is used for the most extensive experiments, the confidence in the approach may be increased by at least performing feasibility tests with real users. Combining different evaluation approaches will increase the confidence with the results.

When it comes to detection rate and false positives, it is not possible to compare the different approaches considered here. In some cases it does not make sense to compare, due to the fact that different problems are being addressed, e.g. performing supervised learning or pure anomaly detection or detecting different types of attacks. In other cases comparisons are impossible due to different evaluation data or even different metrics of performance (e.g. alerts/day versus false

positives rate). In many cases the evaluation could be improved, although it may not always be straightforward to do so due to the difficulty of obtaining good evaluation data. Overall the false positives rate is still a significant problem for learning-based approaches including ADWICE.

Chapter 8

Conclusions and future work

In this thesis we have described ADWICE, a pure anomaly detection scheme using incremental clustering. The use of incremental clustering results in scalability, facilitating the use of very large training data sets and building of large normality models. This was confirmed in experiments building models out of millions of training data resulting in normality models with tenths of thousands of clusters.

Incremental clustering provides the opportunity to adapt a cluster-based model online, possibly interleaving adaptation with detection. We have explained how incremental training can be used to reduce the need for complete periodical retraining of the model and illustrated how this can be used to reduce false positives. In the context of adaptability we also introduce the use of forgetting for a cluster-based model. This prevents the model to grow indefinitely and may increase detection of new anomalies that once have been considered normal.

Training time is linear due to the use of incremental clustering. The search index further improves performance of training as well as detection by providing logarithmic search times of the model. In related work, real-time detection and indices for fast matching against the normality model are seldom considered together with the basic detection approach. We think, however, that it is important to include the index in the detection scheme from the start, since the index may influence not only performance, but also other properties such as adaptation and even accuracy as shown in this thesis. We explained how the initial index could contribute to false positives providing insights that can be useful also in the context of other detection schemes using indices.

A software agent was developed using ADWICE as primary detection engine. This agent was fully integrated into the prototype of the Safeguard agent architecture. The complete Safeguard architecture, including the agent using ADWICE,

was installed and used in the Safeguard test network. This illustrated the use of ADWICE in a larger context, protecting a network from attacks and failures and confirmed that scalability and performance of the agent and ADWICE were good enough to be useful in a realistic environment.

The DARPA/Lincoln Labs related data sets have been widely used but also criticised [78]. The normal traffic regularity as well as distribution of attacks compared to distribution of normality does not correspond to network data in a real network. With this in mind, our DARPA/Lincoln Labs based evaluation with the KDD data set still shows feasibility of ADWICE given the assumptions that relevant features are used and that those features separate normal data from attacks.

For full evaluation of forgetting and incremental training, data over a longer period needs be collected. We see the experiments used in this thesis as a proof of concept. It is improbable that parts of normality should be forgotten already after a few days in a real network. Producing good public data for intrusion detection evaluation including anomaly detection and correlation is still an important task for the intrusion detection community and we think collaboration on this task is very important. Two main approaches exist:

- A test network or simulator can be used for generation of data, thereby realising a fully controlled environment. Generation of more extensive data sets in the Safeguard test network is ongoing work but requires more resources, primarily to generate good normal data, but also to perform attacks.

- Data can be collected from real live networks. Here, normality is no longer a problem, but privacy is, and so is the issue of attack coverage. Emerging tools can be used to sanitise [15] data with some limitations. Sanitising data while keeping relevant properties of data so that intrusion detection analysis is still valid is not easy, especially for unstructured information, for example, network packet content. When it comes to attacks, they may have to be inserted into data offline if the real attack present in data is not enough. Performing attacks online in a real network is typically not a viable option.

If correlation is to be evaluated, such as the correlation agents of Safeguard, data generation becomes even more complex. Many different data sources may be relevant to the detection process. Although some work has been performed in this area [60] further efforts are needed creating open data sets to make comparisons between different approaches easier.

Pure anomaly detection is useful even if training data is not free from attacks. We can then detect changes in behaviour, and the old attacks already present in training data should be handled by misuse detection systems to a large extent. What is the implication of training online on both attack and normal data? As long as statistics is used, attacks may be assumed to be outliers in normal data and detected based on this assumption. In the case of ADWICE and similar approaches, training online on data not assumed to be normal is probably not a good idea. Then we may learn attacks as normality the first time they occur. By only adapting the model using data that is classified as normal (by ADWICE itself, another detection scheme or a human expert) this problem can be reduced.

8.1 Future work

The current implementation only handles numerical data and the mapping from categorical values to numerical is not a general solution. Future work includes development of a detection scheme that is more general and able to handle categorical data as well as numerical. Using categorical values means that representatives of data need to be stored inside a cluster, rather than just the CF used by BIRCH and currently ADWICE. To what extent this use of categorical data would influence performance, memory usage, and detection quality needs to be investigated. Work by Ganti et al. [49,50] may be used as a starting point for scalable clustering of categorical data.

We have here evaluated two kinds of threshold requirements using radius or distance. Using distance produces better detection quality but using the radius is more resistant to outliers of data raising the question whether a combination of these two could lead to the best of both worlds.

Forgetting may be used for handling outliers. Future implementations should study cluster size when deciding whether a cluster ought to be forgotten. Large clusters, corresponding to very common normality in the past, should be very resistant against forgetting. Also, rather than forgetting clusters periodically, the influence of clusters could be reduced step by step until the point where it is completely forgotten. This could be realised by defining an influence radius to complement the normal radius of the cluster when performing detection. The influence radius could be defined as a percentage of the original radius for example, where the percentage will decrease over time.

Global clustering will produce a more optimal clustering than the local incremental clustering used by ADWICE. One approach to get the best of both worlds may be to use non-incremental global clustering periodically, and incremental

clustering online for adaptability. This requires the global clustering algorithm to be scalable enough to handle large training data sets in the context of network data.

In our work service names and host addresses are included as data features. This means each host/service will be modelled by a separate set of clusters, but that all clusters even representing separate hosts/services will be collected into one global network model. This implies that the model only has global parameters, thereby removing the need to set model parameters for individual hosts/services. However, this approach results in a single large tree of clusters. An alternative approach would be to represent each host/service by a separate tree of clusters. The global network model would then consist of a forest of trees, rather than one large tree. This would significantly reduce the number of clusters in each model, thereby reducing the height of the tree implying improved performance. The correct tree could be found in constant time using hashing, similar to the hash-based tree used in the grid-index. Such a forest of trees could use global parameter settings to avoid the need for handling a large set of individual parameter settings.

In the Safeguard prototype only features based on packet headers were included. Many attacks require also payload-based features to be detected. Like in the case of the KDD data set, such features can be selected by a human expert and pre-processed before feature vectors are processed by an intrusion detection scheme. An alternative more general approach is to compute statistics based on the binary packet content without caring about semantics of the packet data similar to the work by Wang and Stolfo [115]. More work is required to analyse and collect statistics on the usefulness of individual features for detection of specific attacks and attack types. Lundin-Barse and Jonsson [13] present a framework for determining exactly which log data can reveal a specific attack. Larson et al [74] continue this work by presenting a tool that can be used to find differences between normal and attack data in log files. Knowledge of such differences is required for selection of relevant features for intrusion detection. Knowledge of specific features relevant for existing attacks will not be enough for detection of all new attacks, but could provide a starting point.

Our experience with incremental training indicates the need for new techniques to complement the anomaly detector. Julish [70] describes how clustering of alerts is used to identify root causes (e.g. misconfiguration of a system resulting in false alerts). This technique could be used to suggest to the operator that large classes of similar alerts may possibly be false positives that should be included into the model by incremental training.

We have further noted that there may be a need for the operator to tell the

anomaly detector to learn not only the data producing the alert, but also a generalisation of data. For example, the two step incremental training during evaluation of adaptation would not have been necessary if the operator could have told the system to learn that the data producing alerts was normal both during working and non-working hours. Those experiences call for intelligent tools to help the operator using pure anomaly detection with adaptation in a real environment.

8.2 Final words

Anomaly detection in general is a difficult problem, as illustrated by the high false alert rate produced by most presented approaches. Knowing this, is then anomaly detection something we want to include in the defence of our networks? We argue that it is already useful but of course no silver bullet. Anomaly detection should be used as a complement to other defences and for solving specific problems.

The complexity of the general problem of anomaly detection can be significantly reduced using the following principles:

- Model something simple not the complete world.
- Detect simple things not everything.
- Do not use a complex model if a simple one is enough.

In our work with ADWICE we can be accused to break all those principles. We model a complete computer network which is certainly complex. We try to detect many types of attacks. We use a quite complex cluster-based model, rather than simple statistics like mean and standard deviation. It should come as no surprise that like many related approaches this results in a relatively high false alert rate. When exploring new approaches and designing new algorithms hard problems are a welcome challenge. The easiest path is not the one most probable to result in new knowledge. However, when using anomaly detection in the real world, we need to be much more pragmatic. Many commercial approaches include anomaly detection to some extent today. Approaches in commercial tools include detection of attacks resulting in significant statistical deviations, like flooding denial service attacks and scanning attacks and modelling relatively simple things like network protocols. This reduces the problem of false positives. Unfortunately this also implies that the goal of using anomaly detection to detect new attacks is only partially fulfilled. The trend of using more general rule-based approaches, detecting attack types, rather than individual attack instances may to some extent compensate for this.

The topic of this thesis is intrusion detection. One should in this context remember that many problems and attacks can be completely avoided or reduced using suitable prevention techniques. Some may go further and argue that prevention is the only thing we need. We do not think that is the case. Prevention is of course very important, but where prevention fails, we need also intrusion detection. Using distributed network defence systems like Safeguard, attacks that slip through our other defences can be detected and with suitable response in place also stopped.

Bibliography

[1] Edward Amoroso. *Intrusion Detection An introduction to Internet Surveillance, Correlation, Trace Back, Traps and Response*. Intrusion.net Books, 1999.

[2] Debra Anderson, Teresa F. Lunt, Harold Javits, Ann Tamaru, and Alfonso Valdes. Detecting unusual program behavior using the statistical components of NIDES. Technical Report SRI-CSL-95-06, SRI Internation, Computer Science Laboratory, May 1995.

[3] Apache, 2005. http://www.apache.org/ Acc. December 2005.

[4] Frank Apap, Andrew Honig, Shlomo Hershkop, Eleazar Eskin, and Salvatore J. Stolfo. Detecting malicious software by monitoring anomalous windows registry accesses. In *Recent Advances in Intrusion Detection (RAID'02), Proceedings*, volume 2516 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2002.

[5] Martin Arvidson and Markus Carlbark. Intrusion detection systems - technologies, weaknesses and trends. Master's thesis, Linköping university, 2003. LITH-ISY-EX-3390-2003.

[6] Midori Asaka, Takefumi Onabuta, Tadashi Inoue, and Shigeki Goto. The use of mobile agents in tracing an intruder in a local area network. In *Pacific Rim International Conference on Artificial Intelligence (PRICAI'00), Proceedings*, volume 1886 of *Lecture Notes in Computer Science*, pages 373–382. Springer, 2000.

[7] Midori Asaka, Takefumi Onabuta, Tadashi Inoue, and Shigeki Goto. Remote attack detection method in ida: Mlsi-based intrusion detection using discriminant analysis. In *Symposium on Applications and the Internet (SAINT'02), Proceedings*, pages 64–73. IEEE Computer Society, 2002.

[8] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[9] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and Systems Security*, 3(3):186–205, 2000.

[10] Stefan Axelsson. Intrusion detection systems a survey and taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University of Technology, 2000.

[11] Rebecca Gurley Bace. *Intrusion Detection*. Macmillian Technical Publishing, 2000.

[12] Emelie Lundin Barse. *Logging for intrusion and fraud detection*. PhD thesis, Chalmers University of Technology, 2004.

[13] Emilie Lundin Barse and Erland Jonsson. Extracting attack manifestations to determine log data requirements for intrusion detection. In *20th Annual Computer Security Applications Conference (ACSAC 2004), Proceedings*, pages 158–167. IEEE Computer Society, 2004.

[14] John Bigham, David Gamez, and Ning Lu. Safeguarding scada systems with anomaly detection. In *Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS'03), Proceedings*, volume 2776 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2003.

[15] Matt Bishop, Bhume Bhumiratana, Rick Crawford, and Karl N. Levitt. How to sanitize data. In *13th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises (WETICE'04), Proceedings*, pages 217–222. IEEE Computer Society, 2004.

[16] Kalle Burbeck. A preliminary agent architecture survey. Technical report, Real-Time Systems Laboratory, Linköping University, October 2002. http://www.ida.liu.se/∼rtslab/publications/2002/burbeck2002Agent.pdf.

[17] Kalle Burbeck. Agent platform evaluation. Technical report, Real-Time Systems Laboratory, Linköping University, April 2003. http://www.ida.liu.se/∼rtslab/publications/2003/burbeck2003Agent.pdf.

[18] Kalle Burbeck, Daniel Garpe, and Simin Nadjm-Tehrani. Scale-up and performance studies of three agent platforms. In *23rd IEEE International Performance, Computing, and Communications Conference (IPCCC'04), Proceedings*, pages 857–863. IEEE, 2004.

[19] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley, 2001.

[20] Brian D. Carrier and Blake Matheny. Methods for cluster-based incident detection. In *Second IEEE International Workshop on Information Assurance (IWIA'04), Proceedings*, pages 71–78. IEEE Computer Society, 2004.

[21] Sung-Bae Cho and Sang-Jun Han. Two sophisticated techniques to improve hmm-based intrusion detection systems. In *Recent Advances in Intrusion Detection (RAID'03), Proceedings*, volume 2820 of *Lecture Notes in Computer Science*, pages 207–219. Springer, 2003.

[22] Tobias Chyssler. Reducing false alarm rates in intrusion detection systems. Master's thesis, Linköping University, 2003. LiTH-IDA-Ex-03/067-SE.

[23] Tobias Chyssler, Simin Nadjm-Tehrani, Stefan Burschka, and Kalle Burbeck. Alarm reduction and correlation in defence of ip networks. In *13th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises (WETICE'04), Proceedings*, pages 229–234. IEEE Computer Society, 2004.

[24] W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference, Proceedings*. Morgan Kaufmann, 1995.

[25] Michael Collins and Michael K. Reiter. An empirical analysis of target-resident dos filters. In *IEEE Symposium on Security and Privacy (S&P'04)*, pages 103–114. IEEE Computer Society, 2004.

[26] Common intrusion detection framework (cidf), 1999. http://www.isi.edu/gost/cidf/ Acc. March 2005.

[27] Corba, 2005. http://www.corba.org/ , Acc. December 2005.

[28] Borland Software Corporation. Visibroker, 2005. http://www.borland.com/visibroker/ Acc. March 2005.

[29] Cougaar agent architecture, 2005. http://www.cougaar.org/ , Acc. December 2005.

[30] North American Electricity Reliability Council. Sql slammer worm: Lessons learned for consideration by the electricity sector slammer, June 2003. available at: http://www.esisac.com/publicdocs/SQL_Slammer_2003.pdf, Acc. August 2005.

[31] D. Dasgupta, F. Gonzalez, K. Yallapu, J. Gomez, and R. Yarramsettii. Cids: An agent-based intrusion detection system. *Computers & Security*, 24(5):387–398, 2005.

[32] Dipankar Dasgupta and Hal Brian. Mobile security agents for network traffic analysis. In *DARPA Information Survivability Conference and Exposition II (DISCEX-II), Volume 2, Proceedings*, pages 332–340. IEEE Computer Society, 2001.

[33] Jose Duarte de Queiroz, Luiz F. Rust da Costa Carmo, and Luci Pirmez. Micael: An autonomous mobile agent system to protect new generation networked applications. In *Recent Advances in Intrusion Detection (RAID'99), Proceedings*, 1999.

[34] Hervé Debar and David A. Curry. The intrusion detection message exchange format. Internet-Draft: work in progress, January 2005. http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt Acc. March 2005.

[35] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion detection systems. Technical Report 3176, IBM Research, 1999.

[36] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

[37] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion detection systems. *Annales des Télécommunications*, 55(7-8):361–378, 2000.

[38] Luc Devroye and László Györfi Gábor Lugosi. A probabilistic theory of pattern recognition. *Applications of Mathematics*, 31, 1996.

[39] eeye digital security - pr20050812, 2005. http://www.eeye.com/html/company/press/PR20050812.html Acc. August 2005.

[40] Charles Elkan. Results of the kdd'99 classifier learning. *ACM SIGKDD Explorations*, 1(2):63 – 64, 2000.

[41] B. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, and N. R. Mead. Survivable network systems: An emerging discipline. Technical Report CMU/SEI-97-TR-013, Carnegie Mellon University, Software Engineering Institute, November 1997.

[42] Emerald, 2005. http://www.sdl.sri.com/projects/emerald/ Acc. March 2005.

[43] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd International Conference on Knowledge Discovery and Data Mining (KDD'96), Proceedings*, pages 226–231. AAAI Press, 1996.

[44] Serge Fenet and Salima Hassas. A distributed intrusion detection and response system based on mobile autonomous agents using social insects communication paradigm. *Electronic Notes in Theoretical Computer Science*, 63, 2001.

[45] Douglas H. Fisher. Improving inference through conceptual clustering. In *Sixth National Conference on Artificial Intelligence, Proceedings*, pages 461–465. Morgan Kaufmann, 1987.

[46] Noria Foukia, Salima Hassas, Serge Fenet, and Paul Albuquerque. Combining immune systems and social insect metaphors: A paradigm for distributed intrusion detection and response system. In *Mobile Agents for Telecommunication Applications, 5th International Workshop (MATA 2003)*, volume 2881 of *Lecture Notes in Computer Science*, pages 251–264. Springer, 2003.

[47] F-secure, 2005. http://www.f-secure.com/ Acc. August 2005.

[48] David Gamez, Simin Nadjm-Tehrani, John Bigham, Claudio Balducelli, Tobias Chyssler, and Kalle Burbeck. Safeguarding critical infrastructures. In Hassan B. Diab and Albert Y. Zomaya, editors, *Dependable Computing Systems: Paradigms, Performance Issues and Applications*, chapter 18. John Wiley & Sons, 2005.

[49] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Cactus - clustering categorical data using summaries. In *Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99), Proceedings*, pages 73–83. ACM, 1999.

[50] Venkatesh Ganti, Raghu Ramakrishnan, Johannes Gehrke, Allison L. Powell, and James C. French. Clustering large datasets in arbitrary metric spaces. In *15th International Conference on Data Engineering, Proceedings*, pages 502–511. IEEE Computer Society, 1999.

[51] Daniel Garpe. Comparison of three agent platforms performance, scalability and security. Master's thesis, Linköping University, 2003. LiTH-IDA-Ex-03/070-SE.

[52] Anup K. Ghosh, Christoph Michael, and Michael Schatz. A real-time intrusion detection system based on learning program behavior. In *Recent Advances in Intrusion Detection (RAID'00), Proceedings*, volume 1907 of *Lecture Notes in Computer Science*, pages 93–109. Springer, 2000.

[53] Dieter Gollman. *Computer security*. John Wiley & Sons, 1999.

[54] Vaibhav Gowadia, Csilla Farkas, and Marco Valtorta. Paid: A probabilistic agent-based intrusion detection system. *Computers & Security*, 24(7):529–545, 2005.

[55] Grasshopper. http://www.grasshopper.de/index.html, Acc. August 2002.

[56] Robert S. Gray, George Cybenko, David Kotz, Ronald A. Peterson, and Daniela Rus. D'agents: Applications and performance of a mobile-agent system. *Software - Practice and Experience*, 32(6):543–573, 2002.

[57] Yu Guan, Ali A. Ghorbani, and Nabil Belacel. Y-means a clustering method for intrusion detection. In *Canadian Conference on Electrical and Computer Engineering (CCECE 2003), Proceedings*, volume 2, pages 1083 – 1086. IEEE, 2003.

[58] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 1998), Proceedings*, volume 27 of *SIGMOD Record*, pages 73–84. ACM Press, 1998.

[59] Ajay Gupta and R. Sekar. An approach for detecting self-propagating email using anomaly detection. In *Recent Advances in Intrusion Detection (RAID'03), Proceedings*, volume 2820 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2003.

[60] J. Haines, D. Kewley Ryder, L. Tinnel, and S. Taylor. Validation of sensor alert correlators. *IEEE Security and Privacy*, 1(1):46–56, 2003.

[61] D. Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, March 1996.

[62] Guy G. Helmer, Johnny S. Wong, Vasant Honavar, Les Miller, and Yanxin Wang. Lightweight agents for intrusion detection. *Journal of Systems and Software*, 67(2):109–122, 2003.

[63] Internet Systems Concortium Inc. Internet domain survey, 2005. http//www.isc.org/index.pl?/ops/ds/ Acc. September 2005.

[64] Tripwire Inc. Tripwire, 2005. http://www.tripwire.com/ Acc. Feb. 2005.

[65] Pew Internet and American Life. Online banking 2005 a pew internet project data memo, 2004. http//www.pewinternet.org/ Acc. Feb. 2005.

[66] Intrusion detection exchange format working group(idwg), 2005. http://www.ietf.org/html.charters/idwg-charter.html Acc. March 2005.

[67] Issrl : Intelligent security systems research laboratory. http://issrl.cs.memphis.edu/, Acc. November 2005.

[68] Jade. http://sharon.cselt.it/projects/jade/, Acc. August 2005.

[69] JiaweiHan and Micheline Kamber. *Data Mining - Concepts and Techniques*. Morgan Kaufmann, 2001.

[70] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and Systems Security*, 6(4):443–471, 2003.

[71] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999.

[72] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[73] R. Krishnapuram, A. Joshi, O. Nasraoui, and L. Yi. Low-complexity fuzzy relational clustering algorithms for web mining. *IEEE transactions on fuzzy systems*, 9:595–607, 2001.

[74] Ulf Larson, Emilie Lundin Barse, and Erland Jonsson. Metal - a tool for extracting attack manifestations. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'05), Proceedings*, volume 3548 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2005.

[75] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.

[76] Richard Lippmann, Seth E. Webster, and Douglas Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *Recent Advances in Intrusion Detection (RAID'02), Proceedings*, volume 2516 of *Lecture Notes in Computer Science*, pages 307–326. Springer, 2002.

[77] Emilie Lundin and Erland Jonsson. Anomaly-based intrusion detection: privacy concerns and other problems. *Computer Networks*, 34(4):623–640, 2000.

[78] Matthew V. Mahoney and Philip K. Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection (RAID'03), Proceedings*, volume 2820 of *Lecture Notes in Computer Science*, pages 220–237. Springer, 2003.

[79] Zakia Marrakchi, Ludovic Mé, Bernard Vivinis, and Benjamin Morin. Flexible intrusion detection using variable-length behavior modeling in distributed environment: Application to corba objects. In *Recent Advances in Intrusion Detection (RAID'00), Proceedings*, volume 1907 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2000.

[80] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[81] Peter Mell, Donald G. Marks, and Mark McLarnon. A denial-of-service resistant intrusion detection architecture. *Computer Networks*, 34(4):641–658, 2000.

[82] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[83] Srinivas Mukkamala, Guadalupe Janoski, and Andrew Sung. Intrusion detection using neural networks and support vector machines. In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, pages 1702–1707. IEEE, 2002.

[84] J.C. Munson and S. Wimer. Watcher the missing piece of the security puzzle. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC'01)*, pages 230–239. IEEE Computer Society, 2001.

[85] Nessus, 2005. http://www.nessus.org/ Acc. December 2005.

[86] Nmap, 2005. http://www.insecure.org/nmap/ Acc. December 2005.

[87] Sang Hyun Oh and Won Suk Lee. An anomaly intrusion detection method by clustering normal user behavior. *Computers & Security*, 22(7):596–612, 2003.

[88] Mizuki Oka, Yoshihiro Oyama, Hirotake Abe, and Kazuhiko Kato. Anomaly detection using layered networks based on eigen co-occurrence matrix. In *Recent Advances in Intrusion Detection (RAID'04), Proceedings*, volume 3224 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2004.

[89] The perl directory, 2005. http://www.perl.org/ , Acc. December 2005.

[90] Leonid Portnoy, Eleazar Eskin, and Salvatore Stolfo. Intrusion detection with unlabeled data using clustering. In *ACM Workshop on Data Mining Applied to Security (DMSA-2001)*. ACM, 2001.

[91] K. Poulsen. Slammer worm crashed ohio nuke plant network, August 2003. SecurityFocus News: http://www.securityfocus.com/news/6767, Acc. August 2005.

[92] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion and denial of service: Eluding network intrusion detection. Technical Report T2R-0Y6, Secure networks Inc., 1998.

[93] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 – 286, 1989.

[94] Manikantan Ramadas, Shawn Ostermann, and Brett C. Tjaden. Detecting anomalous network traffic with self-organizing maps. In *Recent Advances in Intrusion Detection (RAID'03), Proceedings*, volume 2820 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2003.

[95] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice-Hall, 1995.

[96] Safeguard. http://www.ist-safeguard.org/ Acc. May 2004.

[97] Final report on architecture. Deliverable D6, The Safeguard project, July 2003.

[98] Validation, test beds and results. Deliverable D9, The Safeguard project, 2004.

[99] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy (S&P'01), Proceedings*, pages 38–49. IEEE, 2001.

[100] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *9th ACM conference on computer and communications security (CCS'02), Proceedings*, pages 265–274. ACM Press, 2002.

[101] Karlton Sequeira and Mohammed Zaki. Admit anomaly-based data mining for intrusions. In *8th ACM SIGKDD international conference on Knowledge discovery and data mining, Proceedings*, pages 386–395. ACM Press, 2002.

[102] Hiren Shah, Jeffrey L Undercoffer, and Anupam Joshi. Fuzzy clustering for intrusion detection. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems (FUZZ '03)*, pages 1274 – 1278. IEEE, 2003.

[103] Robert W. Shirey. Internet security glossary. RFC 2828, May 2000. http://www.ietf.org/rfc/rfc2828.txt Acc. March 2005.

[104] J. Sima. Introduction to neural networks. Technical Report V-755, ICS CAS, Prague, 1998.

[105] Snort, 2005. http://www.snort.org/ Acc. September 2005.

[106] Eugene H. Spafford and Diego Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, 2000.

[107] Hideki Tai and Kazuya Kosaka. The aglets project. *Communications of the ACM*, 42(3):100–101, 1999.

[108] Tryllian. http://www.tryllian.com, Acc. August 2005.

[109] Jeffrey Undercoffer, Anupam Joshi, and John Pinkston. Modeling computer attacks: An ontology for intrusion detection. In *Recent Advances in Intrusion Detection (RAID'03), Proceedings*, volume 2820 of *Lecture Notes in Computer Science*, pages 113–135. Springer, 2003.

[110] Irvine University of California. The uci kdd archive, 2003. http//kdd.ics.uci.edu Acc. February 2004.

[111] Alfonso Valdes and Keith Skinner. Adaptive, model-based monitoring for cyber attack detection. In *Recent Advances in Intrusion Detection (RAID'00), Proceedings*, volume 1907 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2000.

[112] H. S. Venter and Jan H. P. Eloff. A taxonomy for information security technologies. *Computers & Security*, 22(4):299–307, 2003.

[113] Voyager, 2005. http://www.recursionsw.com/mobile_agents.htm , Acc. December 2005.

[114] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *9th ACM conference on Computer and communications security (CCS'02), Proceedings*, pages 255–264, New York, NY, USA, 2002. ACM Press.

[115] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection (RAID'04), Proceedings*, volume 3224 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.

[116] Wei Wang, Jiong Yang, and Richard R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *23rd International Conference on Very Large Data Bases (VLDB'97), Proceedings*, pages 186–195. Morgan Kaufmann, 1997.

[117] Paul D. Williams, Kevin P. Anchor, John L. Bebo, Gregg H. Gunsch, and Gary B. Lamont. Cdis: Towards a computer immune system for detecting network intrusions. In *Recent Advances in Intrusion Detection (RAID'01), Proceedings*, volume 2212 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2001.

[118] Mark Wood and Michael Erlinger. Intrusion detection message exchange requirements. Internet-Draft: work in progress, October 2002. http://www.ietf.org/html.charters/idwg-charter.html Acc. March 2005.

[119] Michael Wooldridge. *An introduction to multi agent systems*. John Wiley & Sons, 2002.

[120] Yinglian Xie, Hyang-Ah Kim, David R. O'Hallaron, Michael K. Reiter, and Hui Zhang. Seurat: A pointillist approach to anomaly detection. In *Recent Advances in Intrusion Detection (RAID'04), Proceedings*, volume 3224 of *Lecture Notes in Computer Science*, pages 238–257. Springer, 2004.

[121] Stefano Zanero and Sergio M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC´04)*, pages 412–419. ACM, 2004.

[122] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch an efficient data clustering method for very large databases. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 1996), Proceedings*, volume 25 of *SIGMOD Record*, pages 103–114. ACM, 1996.

LINKÖPINGS UNIVERSITET

Avdelning, institution
Division, department

Institutionen för datavetenskap

Department of Computer and Information Science

Datum
Date

2006-02-28

Språk
Language

[ ] Svenska/Swedish
[X] Engelska/English
[ ] _____

Rapporttyp
Report category

[X] Licentiatavhandling
[ ] Examensarbete
[ ] C-uppsats
[ ] D-uppsats
[ ] Övrig rapport
_____

URL för elektronisk version

ISBN 91-85497-23-1

ISRN LiU-Tek-Lic-2006:12

Serietitel och serienummer
Title of series, numbering

ISSN 0280-7971

Linköping Studies in Science and Technology

Thesis No. 1231

Titel
Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks

Författare
Kalle Burbeck

Sammanfattning
Abstract

Critical networks require defence in depth incorporating many different security technologies including intrusion detection. One important intrusion detection approach is called anomaly detection where normal (good) behaviour of users of the protected system is modelled, often using machine learning or data mining techniques. During detection new data is matched against the normality model, and deviations are marked as anomalies. Since no knowledge of attacks is needed to train the normality model, anomaly detection may detect previously unknown attacks.

In this thesis we present ADWICE (Anomaly Detection With fast Incremental Clustering) and evaluate it in IP networks. ADWICE has the following properties:

(i) Adaptation - Rather than making use of extensive periodic retraining sessions on stored off-line data to handle changes, ADWICE is fully incremental making very flexible on-line training of the model possible without destroying what is already learnt. When subsets of the model are not useful anymore, those clusters can be forgotten.

(ii) Performance - ADWICE is linear in the number of input data thereby heavily reducing training time compared to alternative clustering algorithms. Training time as well as detection time is further reduced by the use of an integrated search-index.

(iii) Scalability - Rather than keeping all data in memory, only compact cluster summaries are used. The linear time complexity also improves scalability of training.

We have implemented ADWICE and integrated the algorithm in a software agent. The agent is a part of the Safeguard agent architecture, developed to perform network monitoring, intrusion detection and correlation as well as recovery. We have also applied ADWICE to publicly available network data to compare our approach to related works with similar approaches. The evaluation resulted in a high detection rate at reasonable false positives rate.

Nyckelord
intrusion detection, anomaly detection, real-time, clustering, adaptation, IP networks

Department of Computer and Information Science
Linköpings universitet

Linköping Studies in Science and Technology
Faculty of Arts and Sciences - Licentiate Theses

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.

No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.

No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.

No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agentteoretiskt perspektiv, 1994.
No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482 **Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598 **Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hiera rchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16 **Marie-Therese Christiansson:** Inter-organistorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader,1999.
No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.
No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800 **Anders Subotic:** Software Quality Inspection, 1999.
No 807 **Svein Bergum**: Managerial communication in telework, 2000.

No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
No 823 **Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
No 832 **Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
FiF-a 34 **Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
No 842 **Magnus Kald:** The role of management control systems in strategic business units, 2000.
No 844 **Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
FiF-a 37 **Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.
FiF-a 40 **Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
FiF-a 41 **Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
No. 854 **Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
No 863 **Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
No 881 **Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
No 882 **Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
No 890 **Annika Flycht-Eriksson:** Domain Knowledge Management inInformation-providing Dialogue systems, 2001.
Fif-a 47 **Per-Arne Segerkvis**t: Webbaserade imaginära organisationers samverkansformer, 2001.
No 894 **Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
No 906 **Lin Han**: Secure and Scalable E-Service Software Delivery, 2001.
No 917 **Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
No 916 **Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
Fif-a-49 **Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
Fif-a-51 **Per Oscarsson:**Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
No 919 **Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
No 915 **Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
No 931 **Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
No 933 **Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
No 938 **Bourhane Kadmiry**: Fuzzy Control of Unmanned Helicopter, 2002.
No 942 **Patrik Haslum**: Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
No 956 **Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
FiF-a 58 **Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
No 964 **Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
No 973 **Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.
No 958 **Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.
Fif-a 61 **Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.
No 985 **Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.
No 982 **Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.
No 989 **Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.
No 990 **Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.
No 991 **Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.
No 999 **Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002
No 1000 **Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.
No 1001 **Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
No 988 **Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
FiF-a 62 **Lennart Ljung:** Utveckling av en projektivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
No 1003 **Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
No 1005 **Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
No 1008 **Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
No 1010 **Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
No 1015 **Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
No 1018 **Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
No 1022 **Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
FiF-a 65 **Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.
No 1024 **Aleksandra Tesanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.

No 1034 **Arja Vainio-Larsson**: Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
No 1033 **Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befarade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
Fif-a 69 **Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
No 1049 **Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
No 1052 **Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
No 1054 **Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
Fif-a 71 **Emma Eliasson:** Effektanalys av IT-systems handlingsutrymme, 2003.
No 1055 **Carl Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
No 1058 **Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
FiF-a 73 **Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
No 1079 **Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
No 1084 **Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
FiF-a 74 **Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
No 1094 **Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
No 1095 **Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, 2004.
No 1099 **Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
No 1110 **Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
No 1116 **Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
FiF-a 77 **Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
No 1126 **Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
No 1127 **Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
No 1132 **Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
No 1130 **Ioan Chisalita:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
No 1138 **Thomas Gustafsson:** Maintaining Data Consistency im Embedded Databases for Vehicular Systems, 2004.
No 1149 **Vaida Jakoniené:** A Study in Integrating Multiple Biological Data Sources, 2005.
No 1156 **Abdil Rashid Mohamed**: High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
No 1162 **Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
No 1165 **Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
FiF-a 84 **Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
No 1166 **Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
No 1167 **Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
No 1168 **Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
FiF-a 85 **Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
No 1171 **Yu-Hsing Huang:** A systemic traffic accident model, 2005.
FiF-a 86 **Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
No 1172 **Petter Ahlström**: Affärsstrategier för seniorbostadsmarknaden, 2005.
No 1183 **Mathias Cöster**: Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
No 1184 **Åsa Horzella**: Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
No 1185 **Maria Kollberg**: Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.
No 1190 **David Dinka**: Role and Identity - Experience of technology in professional settings, 2005.
No 1191 **Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.
No 1192 **Nicklas Bergfeldt:** Towards Detached Communication for Robot Cooperation, 2005.
No 1194 **Dennis Maciuszek:** Towards Dependable Virtual Companions for Later Life, 2005.
No 1204 **Beatrice Alenljung:** Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005
No 1206 **Anders Larsson:** System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.
No 1207 **John Wilander:** Policy and Implementation Assurance for Software Security, 2005.
No 1209 **Andreas Käll:** Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.
No 1225 **He Tan:** Aligning and Merging Biomedical Ontologies, 2006.
No 1228 **Artur Wilk:** Descriptive Types for XML Query Language Xcerpt, 2006.
No 1229 **Per Olof Pettersson:** Sampling-based Path Planning for an Autonomous Helicopter, 2006.
No 1231 **Kalle Burbeck:** Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.