Clustering Hybrid Detection Agent

Software Design Document (SDD)

Originator: Kalle Burbeck
Date: 18/05/2006
Version: 1.0

Synopsis

This document describes the detailed technical design of the Clustering Hybrid Detector Agent and its internal components.

Change history

Issue	Date	Editor	Reason
0.1	03/10/2003	Kalle Burbeck	First version.
0.2	01/12/2003	Kalle Burbeck	Minor updates
1.0	16/12/2003	Kalle Burbeck	Updates to reflect implementation
1.1	18/2/2004	Kalle Burbeck	Including user instructions as well as possible improvements
1.2	01/04/2004	Kalle Burbeck	Included section on possible commands to send to CHDA

Distribution list

Safeguard technical team.

Contents

1. Introduction

1.1. Goals and objectives

This document describes one of the Work Package 5 tasks within the Safeguard Project intended to develop the software agent called Clustering Hybrid Detection Agent (CHDA).

The main goal of this document is to identify the functional requirements and provide the technical design for the CHDA. The resulting design has been used as the foundation for implementing the agent.

This document is mainly based on the discussions held between SWISSCOM and LIU and is intended to be updated as new requirements are identified and existing issues are resolved.

1.2. Document scope

The document provides a description of the requirements, functionality, architecture, internal components and external interfaces of the agent as well as the CHDA preprocessor.

1.3. Related documents

For more information related to CHDA you may please refer to the following documents:

Safeguard Agent Tutorial that provides an overview of the platform, describes its functionality, programming model, and illustrates with examples how to begin developing Safeguard Agents.

Specification of SAP Communication Protocols – a detailed technical specification of low-level transport protocols and message encoding format used by SAP implementations.

WP4 Safeguard Architecture - Deliverable 6. In the Architecture document the CHDA is referred to as the TcpDump Hybrid Detection Agent. The name was changed since the agent may be connected to other data sources as well.

1.4. Definitions and acronyms

SAP – stands for Safeguard Agent Platform – and it is a particular implementation of Agent Platform developed to meet the goals of the Safeguard project.

CHDA – stands for Clustering Hybrid Detection Agent – and it is an implementation of the Hybrid Detector Agent for telecom using a clustering engine for anomaly detection.

LCCI – Large Complex Critical Infrastructure

Normal data – Data that is considered normal for the specific data source in the context of anomaly detection. Normal data does not include attacks or failures.

Feature – A feature is a data field in the input vector to the anomaly detection engine. A feature might be for example the port or ip adress of the destination host, the flags set in a TCP-packet, or the number of connections to the same host in the last 2 seconds.

CHDA preprocessor – Data cleaning and feature computation might be resource intensive. Most of this work is therefore done by an external preprocessor, before data is fed into the CHDA.

WEKA - An open source data mining framework. The current implementation of the clustering algorithm is compatible with the WEKA framework. More information on Weka is available online including source and API documentation: http://www.cs.waikato.ac.nz/~ml/weka/

2. Requirements overview

This section enumerates the software requirements for the CHDA.

2.1. Goals and responsibility

The main responsibility of the CHDA is to provide anomaly detection in the telecom domain to complement the existing commercial signature based engines. The agent will working in on-line mode and should have good performance. The normality model should be scalable, so that the agent may be trained on very large datasets.

The motivation for creating a specialized Hybrid detection agent for telecom is the hard requirements on scalability and performance of the selected data mining methods (clustering in this case) due to large data flows in telecom management networks.

The agent has two modes. Training mode and testing mode. In training mode a model of the input to the agent is built. The input is assumed to only consist of normal data. Incremental training makes online training possible.

In testing mode the agent matches new data against the internal model of normality. When data does not match, this is considered as an anomaly, and an alarm is sent to the Correlation Agent.

The state of the CHDA should be configurable by configuration messages from higher level agents.

In this version of the CHDA the input is assumed to be features built out of TCPdump data. The cleaning and feature computation is done by the CHDA preprocessor, built externally to the agent for performance and flexibility reasons.

2.2. System environment

The CHDA is written in Java and should be able to run on any platform supporting Java 1.4. Offline testing of agent using stored data files may therefore be done on any computer with Java 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 platform is Unix.

2.3. Users

CHDA is used within Safeguard. The results from the agent are accessed by correlation agents. Any agent may (e.g. MMI agent) may configure the CHDA.

2.4. Required agent characteristics

2.5. Assumptions and dependencies

The agent is dependent on output from TCPdump processed by the CHDA preprocessor into feature vectors.

TCPdump filters are assumed to be used to output only TCP and UDP data. Only the headers need to be output by TCPdump, since no features will be based on packet payload at this stage due to performance.

The agent and the preprocessor are not intended to be implemented at an industrial strength level. The basic goal in the Safeguard implementation is to demonstrate that the concept of the safeguarding of LCCI is valid.

The CHDA agent may report its alarms to any correlation agent as long as they support the communication interface (e.g. understand generic alarm or provide other means of accessing alarms.)

2.6. Performance requirements

The true performance requirements are unclear until full testing is done in the test network concerning maximum dataflow from tcpDump. Preliminary tests show data flow rates in the order of 100 sessions per second. Preliminary tests of CHDA indicates that it should be able to handle data flows in this order at least. If later evaluation shows that CHDA does not keep up with the data flow, it is straightforward to let multiple instances of the CHDA share the load.

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

The processing done by the agent is used by the correlation agent. The CHDA need to be fast enough to provide information to the correlation agent that is fresh enough to be useful for correlation. The requirements of the CHDA then depend on the time windows used by the correlator for detection.

2.7. Deployment

Deployment should be as simple as possible, preferable by simply unpacking the files and configuring the agent platform.

3. Functional requirements

3.1. CHDA Preprocessor

The basic requirements are the following:

3.1.1. Parsing

The CHDA needs to parse the packet headers output from TCPdump and produce a standardized format suitable for further computations.

3.1.2. Cleaning

Data not considered interesting should be removed as early as possible. First TCPdump filters is applied before even output is produced. But additional cleaning might be necessary in the preprocessor. At the moment no additional cleaning is defined.

3.1.3. Basic feature computations

The CHDA preprocessor needs to compute feature vectors suitable for anomaly detection. In a first step, sequences of packets belonging to the same TCP-connection from TCPdump are processed into connection records. Since UDP is connection-less each UDP-packet becomes a UDP-connection record.

3.1.4. Higher level feature computations

Out of the connection records higher level features might be computed (e.g. the number of connections to the same host as the current connection in the last two seconds). Time-to-detection suffers when the preprocessor needs to wait for a number of finished connections until computing features. Therefore higher level features will not be combined into the same model as the basic features. A separate CHDA could be trained on higher level features. However, this is left as feature work for now.

3.2. The Clustering Hybrid Detection Agent

3.2.1. Communication

The agent must be able to communicate with other agents within Safeguard (configuration).

3.2.2. Feature Selection

To ease experimentation it should be possible to use only a subset of the features provided by the preprocessor for anomaly detection.

3.2.3. Normalization

If a numerical algorithm is used for anomaly detection, non-numerical features need to be transformed into numerical ones. Numerical features need to be normalized on a scale from 0 to 1. The different features might also be given different weights.

3.2.4. Generality

The agent should be made as generic as possible, making it simple to connect the agent to new data sources without changing existing code.

3.2.5. Behaviour-based detection

The CHDA needs to be able to train a normality model, and later use this for anomaly detection. Results from anomaly detection should be passed on to the Correlation Agent as alerts.

3.2.6. Knowledge-based detection

The agent should have the possibility to combine the anomaly detection with signature based detection using human knowledge rather then training.

3.2.7. Persistence

The agent need to be able to save its data model to disk so that a trained model is not lost if the agent fails.

4. Design

4.1. Highlights

The agent is to be implemented in SAP Java. A separate thread is used for acquiring the data from the preprocessor, a part from that the agent is single threaded and messaging to other agents synchronous.

The preprocessor is implemented as a number of Perl-scripts due to Perl's text-processing features. Having the preprocessor as separate scripts also makes it easy to experiment, without need of changing the CHDA itself.

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

4.2. Basic assumptions

4.3. Changes from Architecture Specification (D6)

No significant changes are made from the TCPDump hybrid detection agent presented in the architecture specification. At the moment two engines, clustering based anomaly detection and a signature based white list, is designed and implemented.

4.4. Review of alternative solutions

Explanation of modules referred to in this section is further explained in the following chapter.

The pre-processor could be internal to the agent. However keeping it external is simpler and more flexible and easy to change. Also the Perl language provides good text processing features.

The agent could have a more parallel structure with asynchronous communication. This is considered unnecessary due 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 a separate

Other data sources may be used as input to the CHDA. If times allows, multiple instances of the CHDA working on different data could be realized.

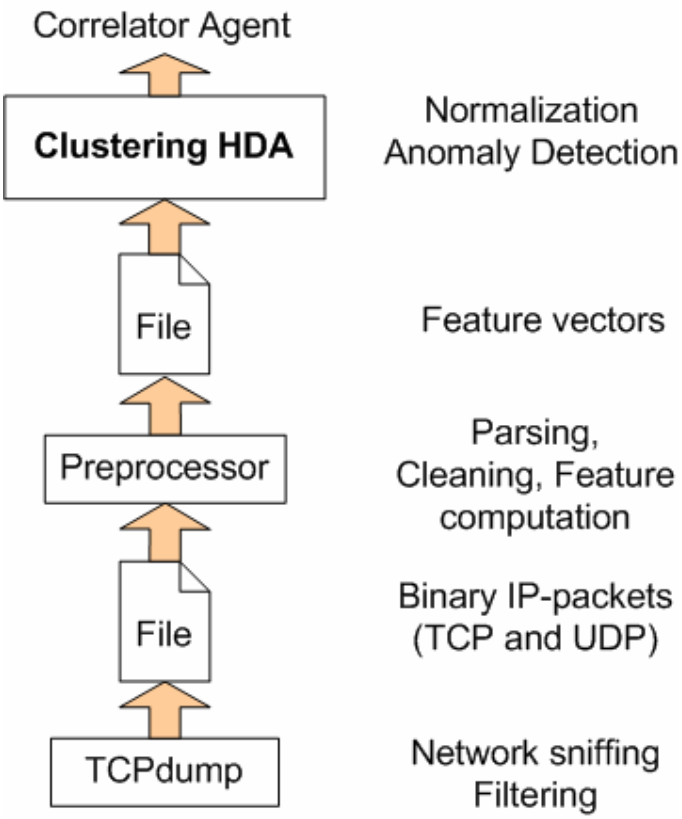
The anomaly engine of the CHDA could be implemented with other techniques then clustering. The design facilities simple exchange of the anomaly engine as long as it implements the AnomalyDetector interface. Other engines then Birch clustering may be tested in the future.

Other signature based engines could be included, facilitated by the use of the SignatureDetector interface.

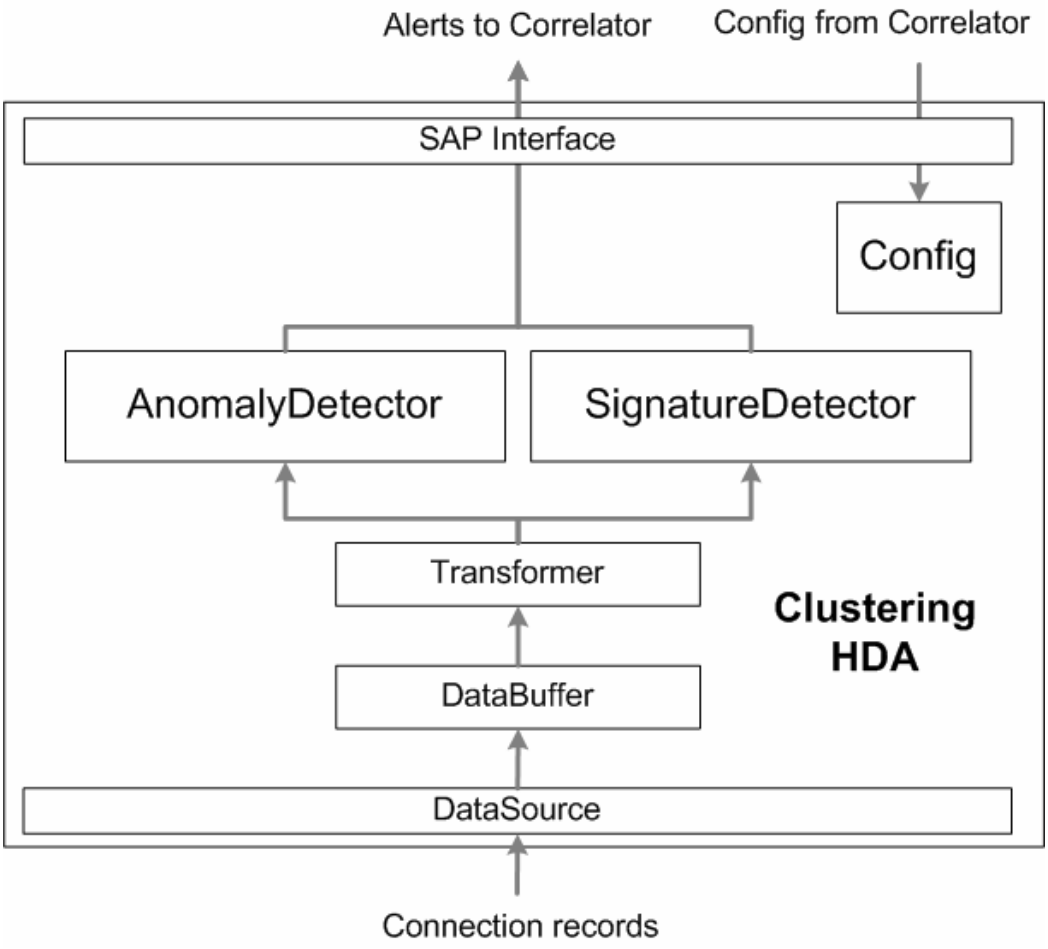
5. System overview

5.1. Architecture

Below is shown an overview of the main components related to the Clustering Hybrid Detector. TCPDump 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 featurevector contain basic data on a single TCP connection or UDP packet. The feature vector file is then processed by the Clustering Hybrid Detector who reports anomalies to a Correlator Agent.



In the figure below is the implemented design of the CHDA.



Below are the main modules of the CHDA agent (the DataBuffer do feature selection, normalization and alert construction):

- DataSource is the interface (and also a Java Interface) between the agent and the output of the pre-processor. The implementing class is loaded dynamically to facilitate simple exchange of data source. The implementation used normally is RemoteHostsDataSource which uses ssh to read session files from an arbitrary number of hosts, merging sessions based on time stamp.

- DataBuffer is a synchronous buffer where DataSource writes sessions. This is needed since DataSource executes by a separate thread for performance reasons.

- Transformer is a java Interface responsible for normalization and feature selection. The implementing class is loaded dynamically making it easy to connect the agent to another type of data.

- AnomalyDetector is a Java Interface at the moment implemented by the BirchAnomalyDetector providing anomaly detection using the BIRCH algorithm.

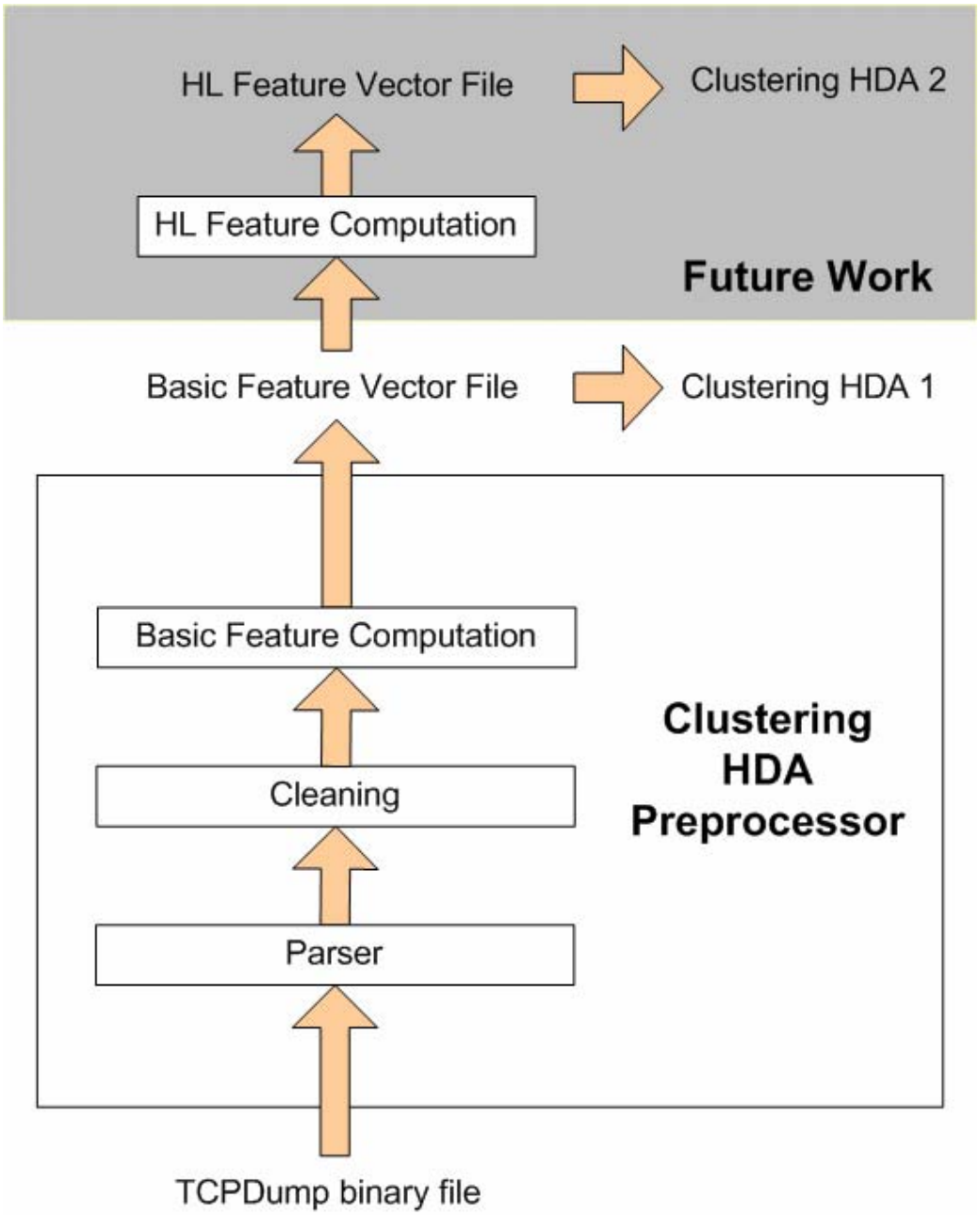
- SignatureDetector is a Java Interface, implemented by the WhiteListSignatureDetector. The white list engine recognizes sessions that should not reported as alarms, even when anomalous.

- Config handles configuration of the agent.

5.2. The Preprocessor

5.2.1. Description

The preprocessor is responsible for taking TCPDump output and producing a format suitable for the CHDA. It will be implemented as a number of simple scripts. The picture below shows the preprocessor and a possible future add on that computes higher level features (i.e. over multiple basic feature vectors in a specific time window).



5.2.2. Processing

The tcpDump process starts up with the parameters discribed below and is running continusly, writing binary packet data to small (>=1 MB) text data files.

tcpdump command: tcpdump -v -n tcp or udp -C 1 -w tcpdump.dat

The script sessionGenMain.pl is keeping track on input files (tcpdump.dat.N) and output files (session.dat.M). In each loop hdaMain executes the script hdaParser.pl on the latest tcpdump.dat file.

The sessionGenParser.pl script runs a separate tcpdump process with the flags below:

tcpdump –tttt -v -n tcp or urdp -r tcpdump.dat.N > tcpdump.txt

Now there is parsed packet headers in the tcpdump.txt file. The script parses the tcpdump.txt file, and stores complete sessions in the current session.dat.M file. Incomplete sessions are stored in a temporary file, to be completed when reading next tcpdump file.

The hdaParser script may sometimes start to read a tcpdump.dat.N file before tcpdump have finished it (1 MB limit is passed). To handle this case the scripts sometimes need to read a tcpdump.dat.N file multiple times. The script remembers how far in the latest tcpdump.dat.N file it has proceeded, and starts anew from this point.

Not all tcp sessions are ended correctly. The sessionGenParser scripts buffers a number of sessions waiting for final FIN-packets, however, when the buffer is full, the oldest sessions are assumed to have ended without final FIN. In this case the session is written out to the session file anyway, but with the FLAG field set to indicate non-complete ending of session.

5.2.3. Interfaces

5.2.4. Restrictions and limitations

5.3. DataSource

5.3.1. Description

This is a simple component for retrieving data from the preprocessor. The motivation to have a separate interface is to simplify future changes to data retrieval.

When the agent is started, the DataSource component starts looking for a feature vector file. Feature vectors represented as strings are input into the DataBuffer.

5.3.2. Processing

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

5.3.3. Interfaces

```
public interface DataSource {
      abstract public String getData() throws Exception;

}
```

5.4. DataBuffer

5.4.1. Description

This module is a synchronized buffer for String arrays.

5.4.2. Processing

The DataBuffer is a passive synchronized enity used by the DataAquirer thread and the main thread of the agent.

5.4.3. Interfaces

```
public interface DataBuffer {
```

```
        abstract public void putData(String data) throws
           InterruptedException, Exception;
        abstract public int getNumberOfData();
        abstract public String[] getData() throws NoDataException;
        abstract public String[] peekData() throws Exception;
        abstract public void removeData() throws NoDataException;
}
```

5.5. Transformer

5.5.1. Description

This module is transforming strings arrays available in DataBuffer to Instance-objects suitable for feeding the Detector modules. It performs feature selection and normalization. What features to select as well as max-values for the features need to be supplied in the CHDA configuration file. Also the Transformer will construct an alert object upon request. There are multiple implementation of the Transformer object, for the swisscom network there is the Transformer1 class.

5.5.2. Processing

The DataBuffer is a passive synchronized enity used by the DataAquirer thread and the main thread of the agent.

5.5.3. Interfaces

```
public interface Transformer {
        abstract public void init(Properties cfg, Logger log) throws
           Exception;
        abstract public Instance transform(String[] data) throws
           Exception;
        abstract public Alert constructAlert(String[] data, int belief)
           throws Exception;
        abstract public String getLabel(String[] data) throws
           Exception;
        abstract public boolean isNormalLabel(String label) throws
           Exception;
        abstract public boolean isLabelPresent();
}
```

5.6. AnomalyDetector

5.6.1. Description

This is the main module of the agent, performing anomaly detector. 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.

5.6.2. Processing

The AnomalyDetector is a passive entity used only by the main thread of the Agent.

5.6.3. Interfaces

```
public interface AnomalyDetector {
        abstract public void setTrainingMode(boolean trainingMode)
        abstract public boolean isTrainingMode();
        abstract public void setSensitivity(double sensitivity) throws
           Exception;
        abstract public double getSensitivity() throws Exception;
        abstract public void reset(Properties prop) throws Exception;
        abstract public void train(Instances data) throws Exception;
        abstract public void train(Instance data) throws Exception;
```

```
        abstract public double detect(Instance data) throws Exception;
        abstract public void loadModel(File f) throws Exception;
        abstract public void saveModel(File f) throws Exception;

}
```

5.7. SignatureDetector

5.7.1. Description

The SignatureDetector performs signature based detection. Currently the WhiteListSignatureDetector implements this interface, recognizing sessions that should not cause alarms even when anomalous.

5.7.2. Processing

The SignatureDetector is a passive entity used only by the main thread of the Agent.

5.7.3. Interfaces

```
public interface SignatureDetector {
        abstract public void addSignature(Signature rule) throws
           Exception;
        abstract public void deleteSignature(Signature rule) throws
           Exception;
        abstract public ArrayList getSignatures() throws Exception;
        abstract public Signature getSignature(int signatureID) throws
           Exception
        abstract public double detect(String[] data) throws Exception;
        abstract public void loadModel(File f) throws Exception;
        abstract public void saveModel(File f) throws Exception;


}
```

5.8. Control flow

5.8.1. Main loop and events.

Main loop will consist of:

Check for messages, if there are messages available, handle messages synchronously.

Do detection/training (depending on mode) of one instance.

If there are no messages or data instances available, sleep for a few milliseconds.

5.8.2. Startup process

First a startup script is run which:

The startup script should:

Start the agent. Initial configuration is read from a file. Some parameters is possible to later change via messages from other agents to the running CHDA others will need to be fixed during execution.

Start the preprocessor, reading from the latest available tcpDump data unless otherwise specified.

Start a new TCPdump process with suitable flags set.

When the agent is instantiated the following events take place.

Read initial configuration from file

Instantiate all objects, such as DataBuffer, AnomalyDetector, DataSource

Load initial AnomalyDetector model from file, if model file exist.

Instantiate DataAcquirer thread.

5.8.3. Shut down process

Finish the ongoing work with the last message or data instance.

Stop DataAcquirer thread.

Save all Detector models.

Return from main thread.

5.8.4. Threads and synchronization

Besides main thread there is one additional thread:

DataAquirerer – Collects data from the DataSource

The main thread and the DataAcquirerer communicate with SyncEvent objects and through the synchronized DataBuffer.

5.9. Configuration

5.9.1. External

The following configuration is accessible by other agents.

Training mode (true|false)

Sensitivity (0-1)

5.9.2. Internal

The following configuration is internal to the CHDA and is set at startup and will not change during execution.

Dataset definition for the anomaly engine in WEKA arff-file format. This defines the feature names (attribute names), the feature types (e.g. numerical or non-numerical) as well as the max and min values of the attribues. It is the responsibility of the Transformer to convert data from the Feature Vector file into the format defined by the arff-file.

FeatureWeights could be used to give different features different weigths. The weights are applied on the data as it is specified in the arff file.

FeatureSelector select what attributes in the dataset definition that should be used as input for the anomaly detection engine. Selection is done after weighting.

6. Data design

This section includes a description of all data structures including internal, global, and temporary data structures.

6.1. Agent data model

The main data structure is the DataBuffer, which contains a buffer of feature vectors ready for processing by the AnomalyDetector. The Anomaly Agent uses Instance-objects, compatible with the WEKA data mining framwork.

6.2. Database description

The CHDA needs no data base. The normality model needs to be kept in main memory to increase detection speed. For persistence files are used.

6.3. Temporary data

6.3.1. Basic Feature Vector file

This section defined the format of the Basic feature vector file. First in the file there is a header, to distinguish different files from each other during testing:

```
#Connection records from the Swisscom Test-network
#Startdate: <date>
#Startime: <time>
#<commaseparated list of feature-names>
```

Each of the following lines in the file contains a commaseparated list of features with no extra space between values. Real numbers should use dot rather then comma. (e.g 3.5 rather then 3,5). Below is a list of names of Basic features. The order of the features in the file should follow the list.

```
StartTime
EndTime
Protocol (1=tcp, 0=udp)
SourceIP
SourcePort
DestinationIP
DestinationPort
SourceBytes
DestinationBytes
ErrorFlag (Signals a session error by the preprocessor)
```

For the swisscom testnetwork the Transformer1 class transforms those features to:

```
Time (hour of day)
Length (milliseconds at the moment)
Protocol
SourceNet
SourceIP
SourcePort
DestinationNet
DestinationIP
DestinationPort
SourceBytes
```

```
DestinationBytes
ErrorFlag (Signals a session error by the preprocessor)
```

Note that Transformer1 is specificly implemented for swisscom testnetwork, handles those IP:s separatly, and uses hardcoded knowledge of network topology to deduce network (number 0-3 internal networks, 4 the rest.)

7. External interfaces

The software's interfaces to the outside world are described in this section.

The exact definition of messageobjects will be available in the Safeguard message defintion directory together with java implementations.

7.1. Agent identification

The agent name is “ClusteringHybridDetectorAgent”.

7.2. Input messages

7.2.1. Config Message

The CHDA agent understands the following configuration messages:

GetLearningMode (boolean mode)

SetLearningMode (boolean mode)

GetSensitivity(Integer amount)

SetSensitivity(Intger amount)

7.3. Output messages

7.3.1. Config Message

The CHDA agent sends the following configuration messages as response to the GetLearningMode and GetSensitivity messages:

Return learning mode (boolean mode)

Return sensitivity(Integer amount)

7.3.2. Alert Message

The fields of the alert message are:

```
StartTime (String)
EndTime (String)
SourceIP (String)
DestinationIP (String)
SourcePort (Int)
DestinationPort (Int)
Protocol (String)
AlertBelief (Real [0,1])
```

AlertBelief is output from the AnomalyDetector, the rest of the field is the same as in the Basic Feature vector file.

8. Testing

Test strategy and preliminary test case specification are presented in this section.

The anomaly detection engine will be evaluated off line on public data sets (KDDCUP 99).

Blackbox testing will be done with a simple dummy agent testing reconfiguration and receiving alerts. A basic feature vector file will be generated in the telecom test network and supplied offline to the agent for initial testing.

The agent need to be tested on-line, running for some period of time, to confirm some measure of reliability and if performance is good enough in real setting.

9. User instructions

Unfortunalty the agent need some adaption before executed in new environment due to dependance of data source as well as normalization and data transformation used in a new setting. The Agent directory consists of those directories:

- Src – Agent source
- Test – Agent testing code
- Lib – Jar-files used by agent. The following JARs need to be available: mysql-connector-java-3.0.9-stable-bin.jar, sap.jar, weka_slim.jar (subset of WEKA classes)

9.1. Installation

- Install TCPDUMP according to its documentation. Note that it need to run with root priveleges.
- Install Java 1.4
- Install Perl
- Confirm shh, ls and tail unix commands are available or install those (if using remote data source)
- Copy agent directory to host

9.2. Adaptation to network

9.2.1. Accessing data

If no suitable Transformer implementation and DataSource implementation is available, new implementations need to be provided. For

Available transformers:

Transformer1: First try for Swisscom testnetwork

KDDTransformer: For KDD data set (Uses `SymbolicNumericMapper which might save some effort handling new non-numeric features with limited amount of distinct values.`)

Transformer2Dlabel: Simple transformer for 2 dimensional data for initial evaluation

Available data sources:

RemoteHostsDataSource: Implemented for swisscom testnetwork to merge sessions files from multiple remote hosts.

SingleFileDataSource: Straightforward file read from one local file

CMDDatasource: Simple data source. Assumes datasource.command is set to a command line command that outputs session vectors one line at a time. Could be used to read a file remote or local, or could call a script reading multiple files for example.

9.2.2. Connection to higher level agent

The sendAlert(Alert a) message in main agent class is responsible to reporting alarms to higher level agents. Here new ways of reporting may be implemented if necessary.

9.3. Compilation

- Run the build_all.sh script in bin directory. Note that all file specifications configuration are relative the bin directory.

9.4. Configuration

Important properties to consider is Transformer-related, DataSource releated, Whitelist related, Anomaly Detector related as well as birchfm.static.maxNumClusters and birchfm.dynamic.e*.

```
// ************** CHDA properties *************
// // Arrays specified as {value1, value2, ... }
//
// If not specified, use stdout
//chda.log.filename = log.txt
// ALL_SEVERITY_MASK,DEFAULT_SEVERITY_MASK,DEBUG, WARNING,INFO,ERROR,
FATAL
chda.log.mask = {ALL_SEVERITY_MASK}

// Default 20
chda.databuffer.size = 500
// Default 50 ms
// If nothing to do sleep this time
chda.agent.sleeptime=10
// Quit if no data source available for maxsleep time
// Good when testing to quit when no more data
chda.agent.dataerror.quit = true

//Minimum interval between heartbeats in milliseconds
//If heartbeats are used they are sent in this interval
//Need tacaAvailable=true
//If set to -1 heartbeats are never sent regardless of
//if TACA is available
chda.agent.heartbeatPeriod=5000

//Used for hot backup agent when illustrating use of heartbeats
//Start one agent in active mode and one in passive mode
//When active agent is crashed and stops sending heartbeats
//TACA will search for other agents registered with CHDA service
//And if found send activate command (GenericCommandMessage class) to
one of those
//ACTIVE=0, PASSIVE=1
chda.agent.initialState=0
chda.agent.passiveSleepTime=5000
```

```
//Set to false to avoid errors if no taca registered
//Avoid all search for TACA and do not try sendining messages
chda.agent.tacaAvailable=false
//Default ARCA (direct data base), could be TACA also in future
//NONE = report to NONE, used when testing
chda.agent.reportTo=NONE

//When reporting alarm to ARCA database, use this url to get
connection
//LIU
chda.agent.outAlarmDB.url=jdbc:mysql://mir30.ida.liu.se:3306/chda?use
r=Y&password=X
// Laptop
chda.agent.outAlarmDB.url=jdbc:mysql://localhost:3306/chda?user=root
// test network
//chda.agent.outAlarmDB.url=jdbc:mysql://stns79:3306/global?user=root
&password=X
//Remove old data on start if true
chda.agent.outAlarmDB.dropOnStart=true

//Files with drop and create sql statements
chda.agent.outAlarmDB.dropTables.path=../conf/DropTables.txt
chda.agent.outAlarmDB.createTables.path=../conf/CreateTables.txt

// ************** DataSource properties *************
// data source (Should implement se.liu.rtslab.chda.DataSource
interface
// is loaded dynamically. As long as the implementation is on the
classpath
// it should work fine
datasource.class = se.liu.rtslab.chda.RemoteHostsDataSource

//datasource.class = se.liu.rtslab.chda.SingleFileDataSource

// Rest of properties is used to init the above dataSource and should
match
// properties uset in init() method of the data source implementation

// *** Common
// Default 50 ms
datasource.sleepperiod=50
// Default 60000 ms
// Send error after this long time
datasource.maxsleep=50000000

// *** Used by RemoteHostsDataSource
datasource.hosts = {mir41.ida.liu.se}
datasource.users = {kalbu}

// dirs should end with slash
datasource.dirs =
{/home/kalbu/COMPAQ/java/Safeguard/data/sessions031219_subset/}
// File assumed to end with .<INT> or '*'
// where * means start with most current file
// Observe that DataSourceThread will stand and wait until ALL first
files
// are available.
datasource.firstFiles = {session.dat.1}

// *** Used by CMDDataSource
// datasource.command = /bin/sh -c ssh -l root inossmtgdlito1 ls -al;
```

```
// *** Used by SingleFileDataSource
// Default ../data/session.dat.1
datasource.firstdatafile=../data/sessions031219_subset/test/session.d
at.1000

// ************** WhiteListEngine properties *************
// Input and output text files for whitelist models
whiteListEngine.modelfilein=../models/scWhiteListEngine.txt
whiteListEngine.modelfileout=../models/scWhiteListEngine_out.txt


// ************** AnomalyDetector properties *************
// Either true or false. Sets initial agent mode.
// anomalydetector.trainingmode=true
// Starting anomaly model
// Default empty string,
// if not set start with new model
// anomalydetector.modelfilein = ../models/birchModel.bin
// Model file to store updated model
// Default empty string.
// If not set, do not save model
// If same as modelfilein, save over old model file
//anomalydetector.modelfileout = ../models/birchModel.bin

// ************** Transformer properties *************
// Transformer class
transformer.class = se.liu.rtslab.chda.Transformer1
// Dataset definition
// max and min vector used for normalization
// Should contain no class label
// Default ../conf/dataSetDef.arff
transformer.dataformatFile = ../conf/scSessionDataSetDef.arff

// Weight vector, normally applied after normalization to [0,1]
interval
// Default all one
transformer.weigths={1,0.1,1,10000,10000,1,1,10000,10000,0.1,0.1,1}

// Selector vector
// Default all true
// After transformation, provide possibility to select subset of
transformed features
transformer.selector={true,true,true,true,true,true,true,true,true,tr
ue,true,true}

// If specified, then label is present, this may be used for
evaluating
// Label is assumed to be last attribute.
// Label is not used as input for anomaly detection
// Length of selector/wiegths should be length of data without label
// Default is empty label, meaning no label present
// transformer.normalLabel = normal.

// ************** Birch Framework properties *************
//Overrules by anomalydetector.trainingmode
birchfm.dynamic.trainingmode = true
// everything on the inside of r*e1 is normal
// default 1 (very pessimistic)
birchfm.dynamic.e1 = 2.5
// everything on the outside of r*e2 is abnormal
```

```
// default 1 (very pessimistic)
birchfm.dynamic.e2 = 2.5

// Set when clusterer is instanciated
// Never reset.
birchfm.static.maxLinearRegression = 10

// IMPORTANT
// Maximum number of clusters, default 100.
birchfm.static.maxNumClusters = 1700
birchfm.static.randomizedBatchMode = false

// Branching Factor, default 3 (more suitable 20)
birchfm.static.branchingFactor = 20
// Starting threshold, default 0
birchfm.static.startThreshold = 0
birchfm.static.debug = false

// Use simple additative thresholdUpdate instead of heuristic
// Default false
birchfm.static.simpleThresholdUpdate =true
//IMPORTANT: Set small enough so that threshold does
//not grow to quickly
birchfm.static.thresholdStep = 0.00001
// Instead of using normal radius, use threshold (max radius)
// for all clusters, default false.
birchfm.static.thresholdAsRadius = true

//What condition to use for decide if a cluster may grow
//Default 1
birchfm.static.absorbConditionChoice = 2

//1: normal distance to cluster center
//2: Include cluster sizes in consideration, larger cluster is closer
birchfm.static.nonLeafClosestCFChoice=1

//Default false
//Store away all data in clusters.
//Used for debugging mostly, do not use with large data sets
birchfm.static.storeDataInClusters = false

// ************** Birch Framework Offline testing properties
*************
// Only used for off line training and testing
// replaces some properties for chda agent

//If files end in .arff thay are assumed to be arff-files following
//WEKA format.
//birchfm.offline.trainingFile =
../data/sessions031219_subset/session.dat.10
//birchfm.offline.testingFile =
../data/sessions031219_subset/session.dat.10
birchfm.offline.trainingFile
=/.scratch/tmp/kalbu/sessions031219/sessions031219.dat.all.wl2
birchfm.offline.testingFile
=/.scratch/tmp/kalbu/sessions031219/sessions031219.dat.all.wl2

// Default true
birchfm.offline.outDir = ../out
birchfm.offline.outputClusterFile = false
// Output all instances after testing, with there anomaly level
```

```
// and closest cluster number.
// Default true
birchfm.offline.outputResultFile = false
//default true
birchfm.offline.outputStatsFile = true
//Default true. Append rather then overwrite statsFile
birchfm.offline.outputStats.append = true
birchfm.offline.statsFile = exerimentResults.txt
```

9.4.1. Whitelist engine (optional)

It is possible to filter away data producing a lot of abnormal behaviour, known to be normal such as DNS traffic or use whitelist to filter away unwanted traffic. (Also consider possibility to use TCPdump filters for lower level filtering)

Each signature implements the Signature interface and is assumed to have a one line string representation possible to read and write from/to text files. String representation should be commaseparated with first value the full classname (including packets) and other values parameters of the signature.

9.5. Execution

- Start up session generation scripts or use available scripts off-line. The start_hda_tcpdump.pl scripts is used for closing down previous processes if any is still running, as well as starting up the tcpdump process and session parser scripts.

- Start up agent using the start_chda.sh script (in which properties for INI as well as CFG file locations are specified)

9.6. Online reconfiguration

It is possible to reconfigure the agent on-line, using another agent such as the simple se.liu.rtslab.msgagent.MsgAgent.

The message agent can be started with the start_msgagent.sh script available in bin-directory. It is possible to let the agent read from a file using ./start_msgagent.sh < msgfile.txt where the message file contains one command on one line followed by the argument (or “noargs”) on the next line, followed by the next command and so on.

CHDA understands the se.liu.rtslab.messages.GenericCommandMessage which have two fields, command and args. CHDA understands the following commands. (Commands are case insensitive)

Command	Args	Location (Class)	Explaination
activate	none	ClusteringHDAAgent	Activates the agent if passive
passivate	none	ClusteringHDAAgent	Passivate the agent if active
stop	none	ClusteringHDAAgent	Stop the agent
setTrainingMode	True false	ClusteringHDAAgent	Setting trainingMode to true or false
logStats	none	ClusteringHDAAgent	Log all statistics of all modules
logStatsToFile	none	ClusteringHDAAgent	Append all stats to defult statsfile (../out/results.txt)

logStatsToFile	<filename>	ClusteringHDAAgent	Append all stats to argument file
trainFromFile	None <file> <startline>|<endline>|<file>	ClusteringHDAAgent	Trains model incrementally from file. Note that no online data and no messages is processed during this.
testFromFile	None <file> <startline>|<endline>|<file>	ClusteringHDAAgent	Test model incremental from file. Note that no online data and no messages is processed during this.
addWhiteListRule	<signature class>,<signature string representation>	ClusteringHDAAgent	Add filter to whitelist
saveWhiteListModel	<filename>	ClusteringHDAAgent	Saveing whitelist to file
loadWhiteListModel	<filename>	ClusteringHDAAgent	Loading whitelist from file
cfg	<property>=<propertyvalue>	The class beeing reconfigured	Change configuration
saveBirchModel	<filename>	BirchAnomalyDetector	Save birch model
loadBirchModel	<filename>	BirchAnomalyDetector	Load new birch model
rebuild	none	BirchFramework	Rebuild birch tree
filterAllClusters	<filter class>,<filter string representation>	BirchFramework	Filters all clusters in model using the given filter
setAllNumUsed	<integer>	BirchFramework	Sets all numUsed fields of clusters to the argument integer. Should normally be used to reset all values to 0.
setAllTimeAccessedNow	noargs	BirchFramework	Sets time access to present time in every cluster
setAllTimeAccessedTo	<time in milliseconds>	BirchFramework	Set time accessed to time provided in argument in every cluster
setThreshold	<threshold>	BirchFramework	Set threshold to specified value

10. Possible improvements

- Configuration messages –The only implemented communication with other agents is alarms. Suitable reconfiguration messages is changing sensitivity, switch from training to testing mode and vice versa, and command to save the model.

- Regular saving the model or improved shut-down procedure– The model is saved when quitting the agent. A known issue is that saving the model fails sometimes, if storing to disk is not finished when agent closes down.

- Higher level features is a possible extension.

- Incorrectly finished sessions are buffered until buffer is full. However, also at regular (short, some seconds) intervals we may need to clean out the buffer of still unfinished sessions to minimize time-to-detection when sessions are not ended correctly. (fixed?)

- There is no header on feature vector files, but on the other hand every session contains start and end timestamp.

- Transformer class Transformer1 is implemented to handle swisscom network, a more general implementation with configuration possibilities could be useful for applying the agent in other settings.

- At the moment RemoteHostsDatasource waits until all first remote files are available. This may cause some delays if starting data generation from scratch, as it may take a while for parserscripts to start up and start producing first session files. (However, nowadays, data generation is started independent of CHDA and CHDA starts from last data file, so this is no problem)