SECRET

SECurity of Railways against Electromagnetic aTtacks

Grant Agreement number: 285136
Funding Scheme: Collaborative project
Start date of the contract: 01/08/2012
Project website address: http://www.secret-project.eu

Deliverable D 4.4
Implementation of the Dynamic Protection System

Submission date: June 2015

Deliverable on Dynamic Protection System
Date: 24/08/2013
Distribution: All partners

Manager: Trialog

Document details:

Title	Implementation of the Dynamic Protection System
Work package	WP4
Date	05/06/2015
Author(s)	E Jacob (EHU), C Pinedo (EHU), C Gransart (IFSTTAR), A Kung (TRIALOG), M Sall (TRIALOG)
Responsible Partner	Trialog
Document Code	SEC-WP4-D44-Implementation of the Dynamic Protection System.docx
Version	1.3
Status	Final

Dissemination level:

Project co-funded by the European Commission within the Seventh Framework Programme

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission) Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document history:

Revision	Date	Authors	Description
0.1	05/06/2015	TRIALOG	Creation of the document.
0.2	23/06/2015	TRIALOG	Contribution from Trialog
0.3	24/06/2015	TRIALOG	Contribution from Trialog
0.4	29/06/2015	TRIALOG, Ifsttar	Contributions from Trialog and from Ifsttar
0.5	30/07/2015	EHU	Contribution from EHU, Implementation of the MCS
1.0	31/07/2015	Trialog	Taking into account the remarks of the internal reviewer
1.2	10/08/2015	Trialog	Addition of the contribution of Ifsttar (section 4)
1.3	01/09/2015	Trialog	Modification of the title
1.4	01/12/2015	Trialog	Finalisation of the Executive Summary and addition of a conclusion

Table of content

Table of figures

1 Executive summary

Railway communication systems are complex with relations to humans, organisational structures, legal and regulatory frameworks. And they will become even more complex and diverse including new technologies, new services, a greater variety and combination of systems and components. The main purpose of the Secret project through its dynamic protection system is to detect and dynamically cope with different EM attack conditions that may affect the communication among devices of the railway system.

The architecture of this system has been presented in the previous deliverables D4.1 and D4.2. Now, all the components that implement the Resilient Architecture (dynamic protection system above) are fully described together with the interactions that exist between some of them.

2 Introduction

2.1 Purpose of the document

The purpose of this document is to describe the implementation of the resilient architecture.

Previous deliverables described the design of the resilient architecture:

- D4.1 Preliminary specification of the dynamic protection system. It presents an overview of the protection system and gives a brief description of the components of the protection system.
- D4.2 Final specification of the dynamic protection system. It presents a final and complete view of the resilient architecture in order to face EM attacks. It served as the basis for the implementation of the resilient architecture.

Other deliverables are

- D4.3 Simulation and assessment results of dynamic protection system. In order to understand and predict the behaviour of the protection system, this deliverable has modelled all the components of this protection system with all their interactions and dependencies. Simulations have been made which gave good results
- D4.4 Implementation of the dynamic protection system. This implementation followed the architecture specified in D4.2
- D4.5 Validation of the implementation through use-case. As its name subjects, its objective is to validate the current implementation of the dynamic protection system.

In the next section the architecture of the protection system is recalled. The following sections describe the implementation of the three subsystems that compose the resilient architecture. These subsystems are:

- The Acquisition System Analyser (ASA) which is described in section 4,
- The Health/Attack Manager (HAM) which is described in section 5,
- And the Multipath Communication Manager (MCM) which is described in section 6.

2.2 Definitions and acronyms

	Meaning
API	Application Programming Interface
ASA	Acquisition System Analyser
BSC	Base Station Controller
BTS	Base Transceiver Station
CHAM	Central Health/Attack Manager
DS	Detection System
DSS	Detection SubSystem

EDGE	Enhanced Data Rates for GSM Evolution
EM	Electromagnetic
ERTMS	European Railway Traffic Management System
ETCS	European Train Control System
GSM	Global System for Mobile communications
GSM-R	Global System for Mobile communications - Railway
GPRS	General Packet Radio Service
HAM	Health/Attack Manager
ICT	Information and Communications Technology
IP	Internet Protocol
MCM	Multipath Communication Manager
MCS	Multipath Communication System
MPTCP	Multipath TCP
OHAM	On-board Health/Attack Manager
OBU	On-Board Unit
PDU	Packet Data Unit
RCA	Resilient Communication Architecture
RMS	Railway Management System
TETRA	TErrestrial Trunked RAdio
TCP	Transmission Control Protocol
THAM	Trackside Health/Attack Manager
TVRA	Threat and Vulnerability Risk Assessment
UMTS	Universal Mobile Telecommunications System
WiMAX	Worldwide Interoperability for Microwave Access
WP	Work Package

3 Resilient architecture

3.1 Overview of the resilient architecture

The Resilient Communication Architecture (RCA) in the train is composed of the following main components:

- The Health Attack Manager (HAM)
- The Acquisition System Analyser (ASA)
- The Sensors which are connected to the ASA
- The Multipath Communication Manager (MCM)
- Several communication devices behind the MCM

The HAM and ASA components are part of what has been called the protection subsystem (see the figure below). The role of this subsystem is to continuously monitor the overall network for detecting EM attacks performed on the network. The MCM together with its communication devices is responsible for providing resilient communications between trains and the command centre located at ground. These subsystems and how they are interconnected are shown on the figure below:

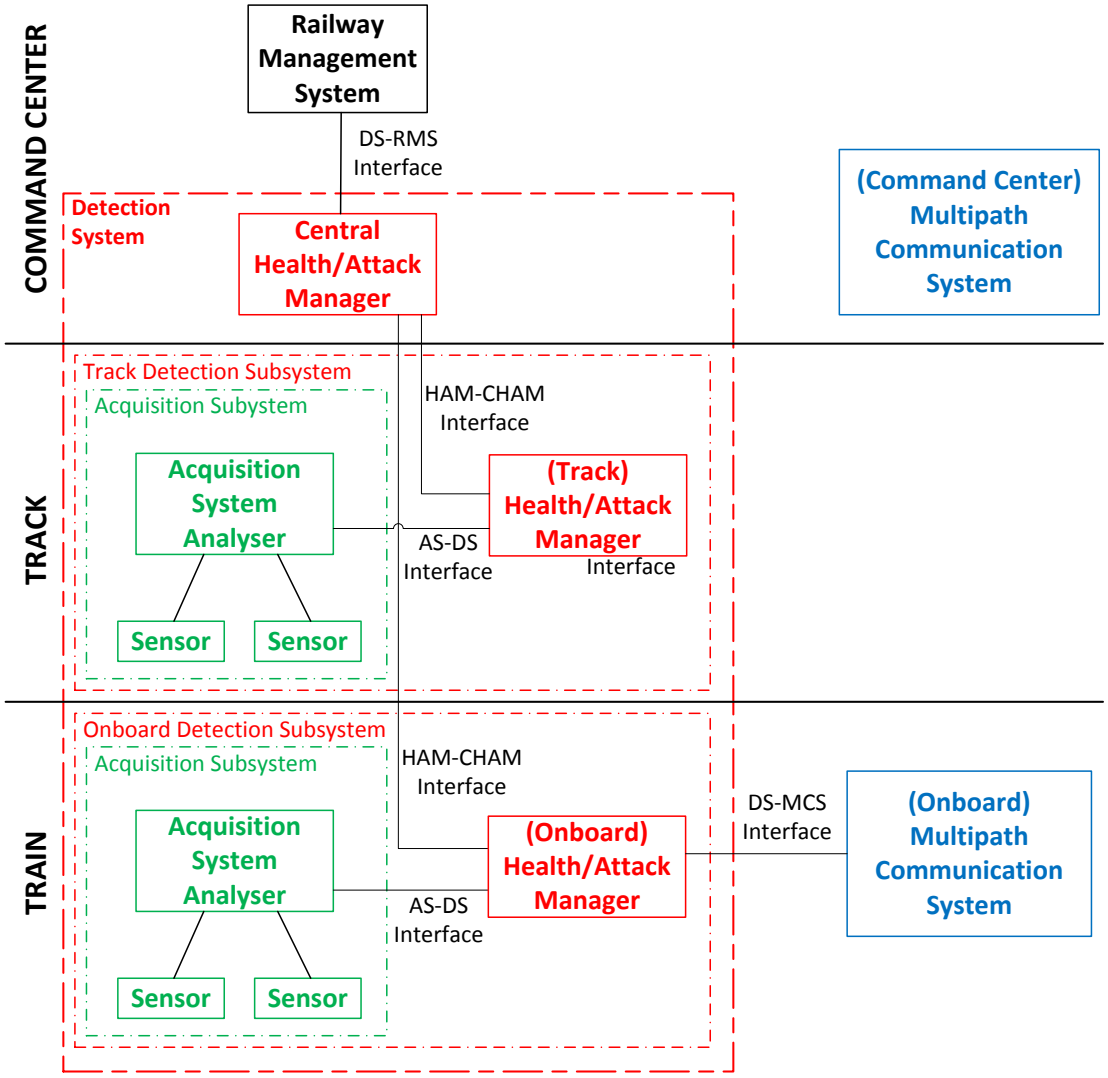


Figure 1: Architecture of the Detection Subsystem

The Resilient Communication Architecture along the track is the same as the one located in the train. Finally the Central Health Attack Manager (CHAM) is centralized and communicate with all the HAMs in the trains and along the tracks.

3.2 Implementation environment

The different components of the RCA have been implemented in Java under *git*[1] for the versioning management and maven[2] for the compilations and the build of the system.

The URL for the git repository is git@git.i2t.ehu.eus:secret/rca.git.

[1] Git is the most widely adopted version control system for software development based on a gnu licence.

[2] Maven is a build automation tool used primarily for Java projects

The version of the maven compiler is 2.5.1.

The implementation uses the following components developed by the open source community:

- jeromq version 0.3.4, for the communication between the different components
- JUnit version 4.11, for unit testing. It is a simple framework to write repeatable tests
- jackson-databind version 2.4.3, used by the components above. It is a multi-purpose Java library for processing JSON[3] data format

[3] JSON is short for JavaScript Object Notation, and is a way to store information in an organized, easy-to-access manner

4 Implementation of the Acquisition Subsystem

Two versions of the acquisition system were developed: one using the database and a second one using a real-time message queuing middleware. Whatever the technology used, after getting the current state of the sensors, a decision making algorithm is used next to evaluate the current wireless environment and determine of the equipment is under attack or not. In case of attack, the Acquisition System informs the Health Attack Manager to make a response to an attack.

4.1 Acquisition System using a database

The Acquisition system that used the database fetches the current value of all the sensors each second. Next, the Acquisition System applies the decision making algorithm the evaluate the threat. Figure x shows the class diagram of the Acquisition System

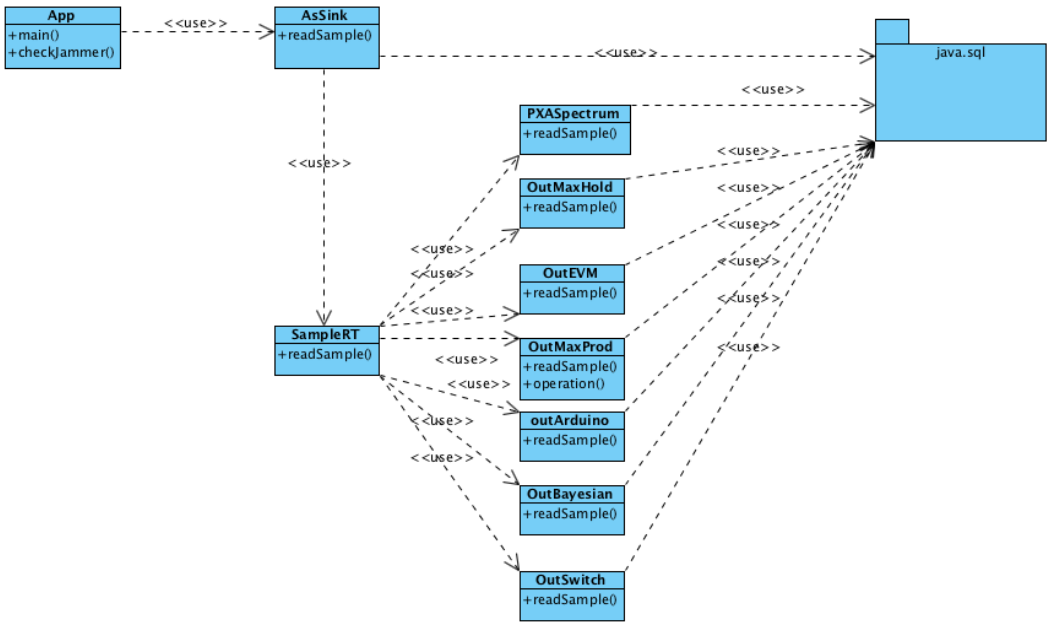


Figure x : Class diagram of the Acquisition System

4.2 Acquisition System using a middleware

The Acquisition System using a middleware is described into the livrable D4.2 (Final specification of the dynamic protection system). The data coming from the sensors are transported using the message format shown in section D4.2 – 4.2.2.2.

For all kind of sensor, the description of the data transported is the same:

```
package SECRET_Acquisition is
```

```
--
-- Generic Message see Livrable D4.2
--
      ...

    --
    -- Example a data transmitted by the middleware (MaxHold detector)
    --
    type Max_Hold is
        State : Current_State;
        Value : Integer; -- Sensor current value between [0..100]
    end Max_Hold;

end SECRET_Acquisition;
```

The Decision making algorithm is executed into the AS_Sink process.

4.3 Decision making algorithm

The decision making algorithm takes the values generated from the different algorithms and decide if the situation is correct or if there is a potential jamming. For all the methods developed we present the decision logic. Next, we make a sum of all the algorithms that have reacted. The more high value we have, the more detection of a jammer is correct.

4.3.1 Algorithm Max Hold

For this algorithm, we are using a sliding window of two seconds. If the value given by the sensor is greater than 50 during at least two seconds then we are under an attack.

```
//
//         Max Hold
//
           if (previous.outMaxHold.value > 50 &&
               current.outMaxHold.value > 50)
           {
                result = true;
                System.out.print(" MaxHold ");
                nbSensors = nbSensors + 1;
           }else{
                System.out.print(" -------- ");
           }
```

4.3.2 Algorithm EVM

For this algorithm, we are using a sliding window of three seconds. If the value given by the sensor is equal to 100 all the time then we are under an attack.

```
//
// EVM
//

          if (secondPrevious.outEVM.value == 100 &&
              previous.outEVM.value == 100 &&
              current.outEVM.value == 100)
          {
               System.out.print(" EVM ");
               result = true;
               nbSensors = nbSensors + 1;
          }else{
               System.out.print(" --- ");
          }
```

4.3.3 Algorithm Max Prod

For this algorithm, we are using a sliding window of two seconds. If the value given by the sensor is greater than 70 during two at least seconds then we are under an attack.

```
//
// MaxProd
//
          if (previous.outMaxProd.value > 70 &&
             current.outMaxProd.value > 70)
          {
               result = true;
               System.out.print(" MaxProd ");
               nbSensors = nbSensors + 1;
          }else{
               System.out.print(" ------- ");
          }
```

4.3.4 Algorithm Secret Arduino

We are under an attack if the current value of the sensor is equal to 100. That's means that no SSID is available.

```
//
// Arduino
//
          if (current.outArduino.value == 100)
          {
               result = true;
               System.out.print(" Arduino ");
               nbSensors = nbSensors + 1;
          }else{
               System.out.print(" ------- ");
          }
```

4.3.5 Algorithm Bayesian

For this algorithm, we are using a sliding window of two seconds. If the value given by the sensor is greater than 70 all the time then we are under an attack.

```
//
// Bayesian
//
          if (previous.outBayesian.value > 70 &&
              current.outBayesian.value > 70)
          {
               result = true;
               System.out.print(" Bayesian ");
               nbSensors = nbSensors + 1;
          }else{
               System.out.print(" -------- ");
          }
```

4.3.6 Algorithm commercial product

We are under an attack if the current value of the sensor is equal to 100. The startup time for this equipment is 10 seconds, so this equipement is not qualified as real-time. Moreover, this sensor react as soon as there is some hyperfrequencies traffic whatever the frequency used. So this

algorithm is just used as a confirmation of an attack already detected by at least one of the above methods. This detector can generate false positive in case of there is some traffic on a particular frequency but which is not the one that we are using.

```
//
// Commercial product
//
          if (current.outSwitch.value == 100)
          {
               if (result != true)
               {
                    falsePositiveSwitch = true;
               }else{
                    positiveSwitch = true;
               }
               System.out.print(" Switch ");
               // nbSensors = nbSensors + 1;
          }else{
               System.out.print(" ------ ");
          }
```

The following table presents an example of the trace of the execution of the detection system. In this trace, the EVM method detects first the jammer; The next second, the MaxHold algorithm detects also the jammer. Few seconds later, the commercial product confirms that there is a jammer.

seconds	MaxHold	MaxProd	EVM	Arduino	Bayesian	Switch	MaxHold	MaxProd	EVM	Arduino	Bayesian	Switch	
1	7	0	0	0	0	0	------	------	---	------	-------	-----	
2	7	0	0	0	0	0	------	------	---	------	-------	-----	
3	7	0	0	0	0	0	------	------	---	------	-------	-----	
4	9	0	0	0	0	0	------	------	---	------	-------	-----	
5	45	0	0	6	0	0	------	------	---	------	-------	-----	
6	21	0	0	47	0	0	------	------	---	------	-------	-----	
7	6	0	0	32	0	0	------	------	---	------	-------	-----	
8	18	0	0	0	0	0	------	------	---	------	-------	-----	
9	5	0	100	0	0	0	------	------	---	------	-------	-----	
10	28	0	0	36	0	0	------	------	---	------	-------	-----	
11	23	0	0	0	0	0	------	------	---	------	-------	-----	
12	21	0	0	0	0	0	------	------	---	------	-------	-----	
13	3	0	0	30	0	0	------	------	---	------	-------	-----	
14	14	0	0	0	0	0	------	------	---	------	-------	-----	
15	4	0	100	0	0	0	------	------	---	------	-------	-----	
16	34	0	100	16	0	0	------	------	---	------	-------	-----	
17	66	0	100	16	0	0	------	------	---	------	-------	-----	Jammer1detected3within3sensors
18	93	0	100	0	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
19	65	0	100	0	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
20	65	0	100	0	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
21	80	0	100	0	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
22	92	0	100	0	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
23	87	0	100	18	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
24	67	0	100	18	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
25	68	0	100	18	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
26	69	0	100	71	0	100	MaxHold	------	EVM	------	-------	Switch	Jammer1detected3within3sensors
27	66	0	100	1	0	100	MaxHold	------	EVM	------	-------	Switch	Jammer1detected3within3sensors
28	65	0	100	0	0	100	MaxHold	------	EVM	------	-------	Switch	Jammer1detected3within3sensors
29	65	0	100	14	0	100	MaxHold	------	EVM	------	-------	Switch	Jammer1detected3within3sensors
30	81	0	100	57	0	100	MaxHold	------	EVM	------	-------	Switch	Jammer1detected3within3sensors
31	82	0	100	0	0	100	MaxHold	------	EVM	------	-------	Switch	Jammer1detected3within3sensors
32	71	0	100	3	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
33	67	0	100	8	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
34	66	0	100	10	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
35	71	0	100	21	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
36	67	0	100	21	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
37	66	0	100	21	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
38	74	0	100	25	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
39	66	0	100	7	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
40	66	0	100	2	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors
41	81	0	100	44	0	0	MaxHold	------	EVM	------	-------	-----	Jammer1detected3within3sensors

5 Implementation of the Health Attack Manager Subsystem

In this section, the components that implement the CHAM/HAM are described.

5.1 Overview of the HAM

The CHAM/HAM has been implemented under Eclipse (Kepler) under Debian. Initially the UML modeller Papyrus has been used for modelling the different components of the CHAM/HAM. Then the skeleton of these components have been automatically generated thanks to the EMF project of Eclipse.

The figure below shows the different components of HAM. Only active classes are displayed below. In other words, classes (like HeartBeatData – figure 3) that contains only data with no code are not displayed.

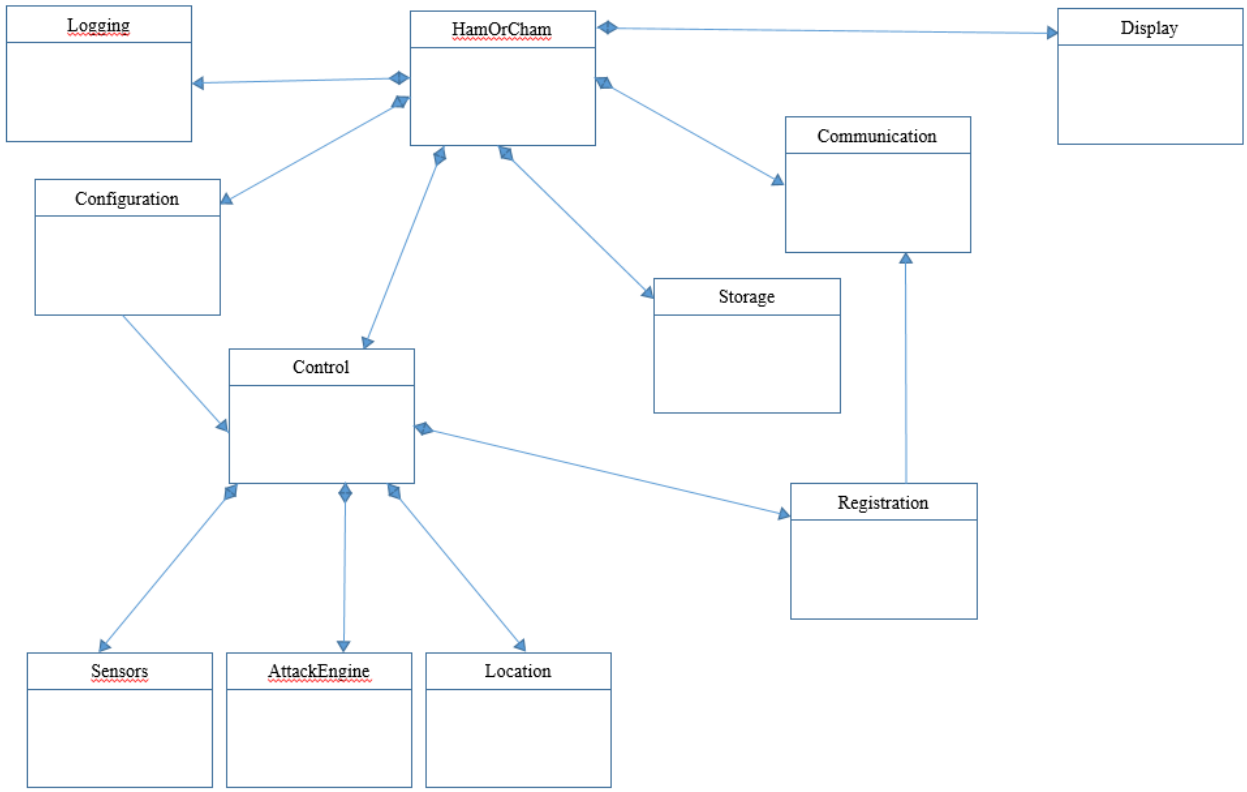


Figure 2: The different classes that compose the HAM

The interfaces that correspond to this description has been automatically generated during this generation process. They are:

- Display
- AttackEngine
- Communication
- Control
- HamOrCham

- Location
- Logging
- Registration
- Sensors
- Storage

See figure below for the detailed list

Address bar: Linux Ext Volume 2\michel\git\rca\ds\src\main\java\eu\secret\project\rca\ds\model\

File name	Type	Size	Modified	File ID
impl	File folder	DIR	14/04/2015	3097185
util	File folder	DIR	17/04/2015	3097205
Display.java	Fichier JAVA	341	13/04/2015	3097188
HamAttackEngine.java	Fichier JAVA	628	15/04/2015	3097196
HamCommunication.java	Fichier JAVA	1 550	13/04/2015	3097223
HamConfiguration.java	Fichier JAVA	737	17/04/2015	3097202
HamControl.java	Fichier JAVA	4 671	15/04/2015	3097184
HamLocation.java	Fichier JAVA	446	13/04/2015	3097221
HamOrCham.java	Fichier JAVA	6 010	13/04/2015	3097210
HamRegistration.java	Fichier JAVA	462	13/04/2015	3097187
HamSensors.java	Fichier JAVA	2 070	16/04/2015	3097225
HamStorage.java	Fichier JAVA	774	16/04/2015	3097201
HeartbeatData.java	Fichier JAVA	1 199	13/04/2015	3097211
InterfaceStatusConf.java	Fichier JAVA	2 725	16/04/2015	3096628
InterfaceStatusList.java	Fichier JAVA	1 810	17/04/2015	3097200
Logging.java	Fichier JAVA	126	15/04/2015	3096626
LogWindow.java	Fichier JAVA	687	17/04/2015	3096629
ModelFactory.java	Fichier JAVA	3 149	13/04/2015	3097216
ModelPackage.java	Fichier JAVA	22 954	13/04/2015	3097219
RegisterData.java	Fichier JAVA	1 923	13/04/2015	3097226

Figure 3: list of the interfaces of the HAM

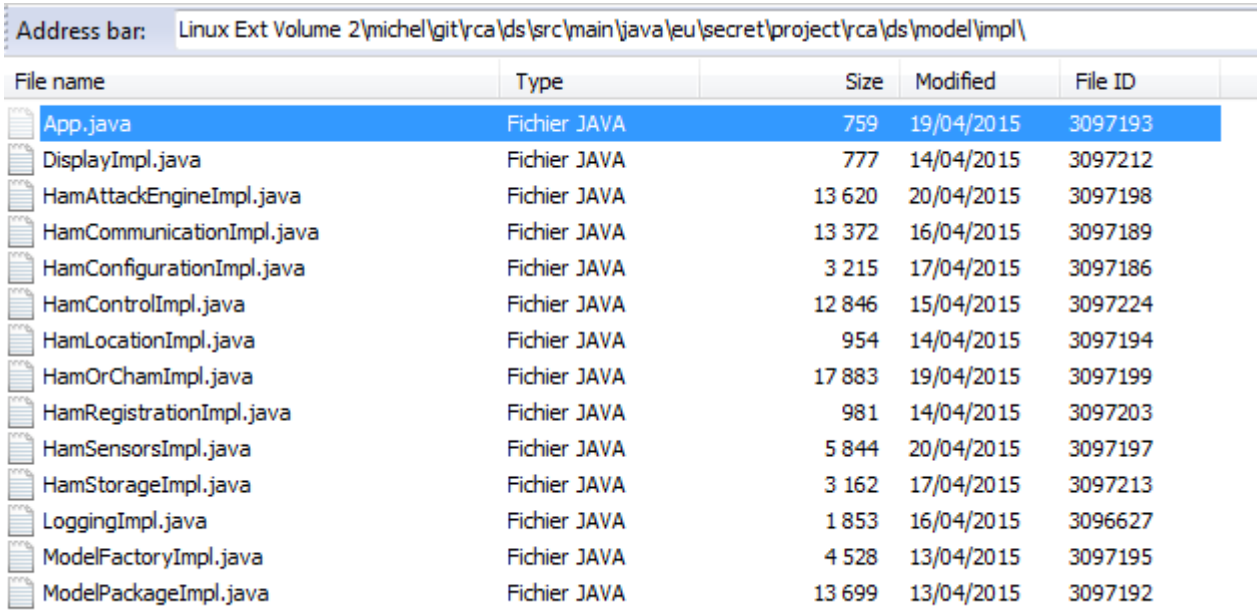
The skeleton of the corresponding implementation classes have been generated. These classes will be discussed in the following sub sections.

Some other utility classes have also been created. They only contain data with no really active code (except instantiation, getters and setters) and have been added during the implementation that's to say after the use of Papyrus, so they have not been displayed on the figure above. They are:

- HeartbeatData
- InterfaceStatusConf
- InterfaceStatusList
- RegisterData

5.2 Description of the classes of the HAM

The figure below shows the different implementation classes of the HAM



File name	Type	Size	Modified	File ID
App.java	Fichier JAVA	759	19/04/2015	3097193
DisplayImpl.java	Fichier JAVA	777	14/04/2015	3097212
HamAttackEngineImpl.java	Fichier JAVA	13 620	20/04/2015	3097198
HamCommunicationImpl.java	Fichier JAVA	13 372	16/04/2015	3097189
HamConfigurationImpl.java	Fichier JAVA	3 215	17/04/2015	3097186
HamControlImpl.java	Fichier JAVA	12 846	15/04/2015	3097224
HamLocationImpl.java	Fichier JAVA	954	14/04/2015	3097194
HamOrChamImpl.java	Fichier JAVA	17 883	19/04/2015	3097199
HamRegistrationImpl.java	Fichier JAVA	981	14/04/2015	3097203
HamSensorsImpl.java	Fichier JAVA	5 844	20/04/2015	3097197
HamStorageImpl.java	Fichier JAVA	3 162	17/04/2015	3097213
LoggingImpl.java	Fichier JAVA	1 853	16/04/2015	3096627
ModelFactoryImpl.java	Fichier JAVA	4 528	13/04/2015	3097195
ModelPackageImpl.java	Fichier JAVA	13 699	13/04/2015	3097192

Figure 4: Implementation classes of the HAM

5.2.1 The HamOrCham class

As its name suggests, this class is common to the HAM and the CHAM. Whether the instance is a HAM or a CHAM is indicated in its configuration file. Dependent of its role, this class will initialize differently the classes it will work with.

The common part instantiates the following classes:

- Communication
- Display, for the debugging phase, this class
- Configuration which reads the configuration file associated to the current instance of HamOrCham class. In particular, it will determine its role.
- Storage
- Control
- Logging

When done, the execution defers depending of the role.

5.2.1.1 HAM instance

When this class plays the role of a HAM. There are two types of HAM. The first one (OHAM) is located on the trains and the second one (THAM) is along the tracks. The only difference between the two is on the Location component (see below). The type is indicated ed in the configuration file of the HAM. The HAM creates threads for sending periodically heartbeat to the CHAM and to the MCM. Then it executes the initialisation of the control class.

5.2.1.2 CHAM instance

Currently it just waits for messages from the HAMs, in particular the heartbeats. It displays all the messages it received and notifies the administrator about problems detected on the overall system.

5.2.2 The Communication class

When this class is associated with a HAM, it executes the following actions:

- Subscribe to the MCM in order to be noticed of every change in the network. When a notification of a change is receive, the Storage class is informed of this change. Then a thread is created. It waits for notifications from MCM.
- Connect to the MCM for being able to send requests to the MCM when needed
- Connect to the CHAM for being able to send notifications to the CHAM when needed

The Communication class uses the following API

- Message.Type.GET_MCM_CONF
- Message.Type.SET_MCM_CONF, this message is mainly used for sending the mode and the heartbeat period to the MCM.
- Message.Type.GET_INTERFACE_LIST, this message is used for obtaining from the MCM the list of the available network interfaces
- Message.Type.GET_INTERFACE_CONF,
- Message.Type.SET_INTERFACE_CONF, this message is mainly used by the AttackEngine class for modifying the characteristics of the network interface that is currently used or for requesting the MCM to use a specified network interface
- Message.Type.GET_INTERFACE_STATUS, this message is used for obtaining the status of a specific network interface
- Message.Type.HEARTBEAT, this message is sent periodically to the MCM for informing it that the HAM is still alive.

More information on this API can be found in the section 6.

5.2.3 The Control class

It instantiates the classes it controls:

- Registration
- Sensors
- AttackEngine
- Location

And initialises them.

It serves mainly as a gateway between the two classes Sensors and AttackEngine.

5.2.4 The Location class

For the OHAM (on board a train), it's role is to keep the location of the train up-to-date. But as we do not have a GPS sensor in the project, the code is empty. A solution would have been to use an open source GPSD server like we have done in another project.

For the THAM (along a track), the location is specified in its configuration file.

5.2.5 The Sensors class

It is responsible of the interaction with the ASA subsystem. For this, it instantiates the class AsSink of ASA. Then it creates a thread that periodically interrogates this instance in order to know the current results of the different algorithms implemented that detect the presence of a jammer. Based

on these current results and results from previous iterations, it calls the Control class which in turn calls the AttackEngine class that will take the appropriate decision.

5.2.6 The AttackEngine class

This class uses the data from the Sensors class for determining which action to execute when an attack has been detected. The decision depends of the policies in place. During the project a certain number of policies have been proposed. At runtime, the policy to use is specified in the configuration file. The different policies are presented below.

5.2.6.1 Traffic Policies for the HAM

In this section traffic policies are described. For each of these policies, the corresponding configuration is presented, followed by the expected behaviour of the system.

Five traffic policies for the HAM have been defined:

- one-interface
- multiple-interfaces
- multiple-interfaces-dynamic
- multiple-interfaces-dynamic-MPTCP-replication
- multiple-interfaces-dynamic-MPTCP-load-balancing

a) Policy one-interface

Initial configuration

Desired interface	Not-desired interfaces
interface_mode=enabled interface_metric= 100 mptcp_mode=disabled	interface_mode=disabled interface_metric=N/A mptcp_mode=N/A

Behaviour

No reaction against attacks. When there is an attack to the selected interface, the communication is damaged or lost.

b) Policy multiple-interfaces

Initial configuration

Desired interface	Not-desired interfaces
interface_mode=enabled interface_metric= 100 mptcp_mode=disabled	interface_mode=enabled interface_metric=higher_metrics_values mptcp_mode=disabled

Behaviour

No reaction against attacks. When there is an attack if the attack is so powerful that tears down the interface, then the next preferred interface is used. This is a hard-handover and affects TCP, UDP and ICMP (every IP) traffic. However, if the attack doesn't tear down the interface, the packet losses are going to produce a bad quality on communications.

c) Policy Multiple-interfaces-dynamic

Initial configuration

Desired interface	Not-desired interfaces
interface_mode=enabled interface_metric= 100 mptcp_mode=disabled	interface_mode=enabled interface_metric=higher_metrics_values mptcp_mode=disabled

Behaviour

Change the metric to the less "attackable" interface:

- hard handover: ICMP and UDP traffic

The system is able to overcome to attacks by changing the preferred interface but the established communication are lost and new communications are required to establish.

d) Policy Multiple-interfaces-dynamic-MPTCP-replication

Initial configuration

Desired interface	Not-desired interfaces
interface_mode=enabled interface_metric= 100 mptcp_mode= enabled	interface_mode=enabled interface_metric=higher_metrics_values mptcp_mode= enabled

Behaviour

Change the metric to the less "attackable" interface:

- hard handover: ICMP and UDP traffic
- soft handover: TCP/MPTCP traffic

TCP/MPTCP communications are not lost when we change the preferred interface due to attacks. Futhermore, the TCP/MPTCP is sent replicated by all interfaces to ensure a quick, low-latency as-soon-as-possible delivery.

e) Policy Multiple-interfaces-dynamic-MPTCP-load-balancing

Initial configuration

Desired interface	Not-desired interfaces
interface_mode=enabled	interface_mode=enabled

interface_metric= 100 mptcp_mode= backup	interface_metric=higher_metrics_values mptcp_mode= backup

Behaviour

Change the metric to the less "attackable" interface. If there is a very powerful attack set mptcp_mode to disabled.

- hard handover: ICMP and UDP traffic
- soft handover: TCP/MPTCP traffic

TCP/MPTCP communications are not lost when we change the preferred interface due to attacks. Furthermore, the TCP/MPTCP is sent load-balanced by all interfaces to ensure a right bandwidth usage. If there is problems (interferences) in one interface the packet sent by this interface will be lost.

5.3 Configuration of the HAM

The configuration of the HAM is described in D4.5 of WP4.

6 Implementation of the Multipath Communication Subsystem

The main and unique component of the Multipath Communication System (MCS) is the Multipath Communication Manager (MCM). These devices provide hardened communications between two MCMs thanks to the use of multiple communication paths that can be modified depending on the requirements or current conditions. The MCM can work autonomously or can receive instructions from the Detection System (DS), more precisely from the Health/Attack Manager (HAM), in order to modify the traffic policy.

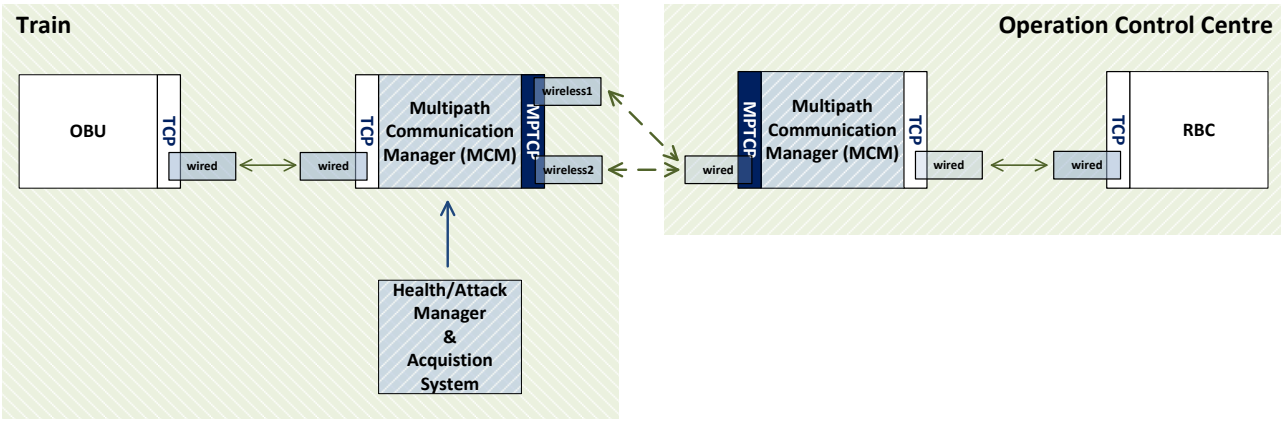


Figure 5: Architecture of the Multipath Communication System (MCS).

The MCM can provide soft-handovers for TCP traffic thanks to the use of Multipath TCP (MPTCP), whereas the rest of IP traffic such as UDP or ICMP will suffer hard-handovers. The MCM is more focused on TCP because ETCS protocol is being ported to TCP protocol.

There are detailed information about the specifications and design decisions for the MCM in deliverable D4.2 and D4.1 of the SECRET Project. Following, we are going to detail the implementation details of each component of the MCM. There are three main building blocks for the MCM detailed in the Figure 6.

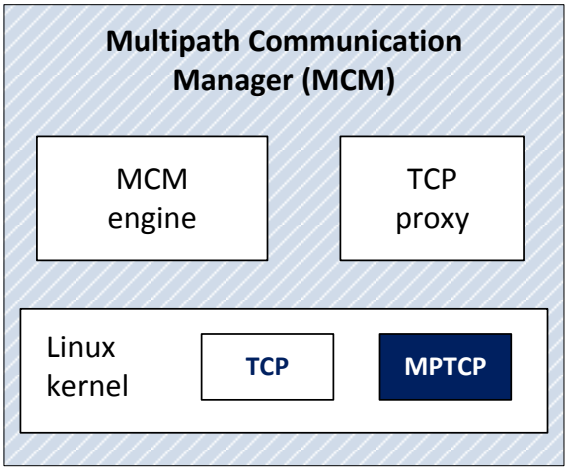


Figure 6: Components of the Multipath Communication Manager (MCM).

Firstly, it is required a Linux kernel with support of MPTCP. Secondly, there is a need for a TCP proxy in order to split the end-to-end TCP connection in multiple segments. One of them, the segment between MCMs, will make use of the MPTCP protocol instead of plain TCP (see Figure 5). Finally, the third component is the MCM engine that is going to govern the proxy in both modes: auto and managed mode.

6.1 Linux kernel with support of MPTCP

The development of the official MPTCP implementation for the Linux Kernel is being carried out in the github project https://github.com/multipath-tcp/mptcp and in the mailing list mptcp-dev@listes.uclouvain.be under the terms of the GPL2 free license. The work is being leaded by people of the *Université catholique de Louvain* (Belgium).

The current implementation of the MPTCP comes only with two schedulers. The scheduler is the component of the MPTCP code that decides through which subflow (also named path) the data is sent when there is more than one available subflow. The two schedulers included in the current implementation of the MPTPC for Linux are the default and the roundrobin schedulers.

The default scheduler estimates the Round Trip Time (RTT) per subflow. Furthermore, one subflow can be configured as active or backup. Thus, when there is a chunk of data that must be sent, the scheduler will first consider the active subflows and will select the "valid" active subflow with best RTT. In other words, the valid active subflow with best latency is selected. However, there could be no valid active subflow, for example, because all the subflows are backup ones or the active subflows are not fully established or the active subflows suffer congestion. In some of these conditions, the data chunk must wait to be sent through an active subflow but however in other conditions the backup subflows will be considered and the backup subflow with best RTT will be selected to send the data.

Figure 7 shows the behaviour of the default scheduler. For simplicity, the two subflows are considered to be active. Thus, according to the default scheduler, the data is sent only through the valid subflow that provides the best RTT. The RTT will vary depending on the congestion of the subflow, and the data chunks will load-balance between the two available subflows.

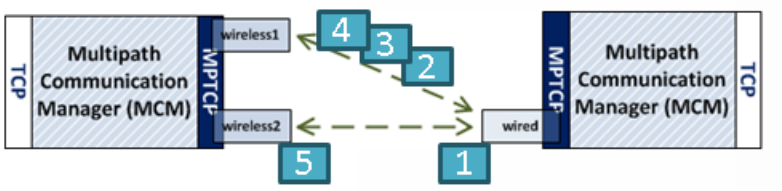


Figure 7: Behaviour of the default scheduler.

The second scheduler provided by the current implementation of the MPTCP is the roundrobin scheduler. This scheduler is simpler; it only sends data chunks alternatively through the available subflows without considering other aspects as it is shown in the Figure 8.

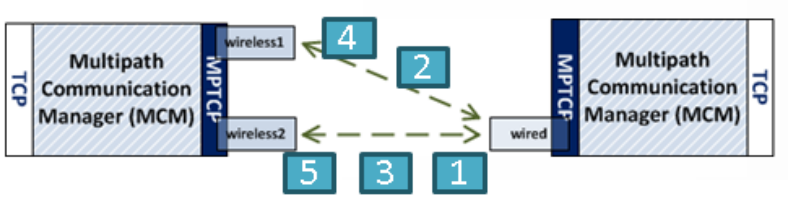


Figure 8: Behaviour of the roundrobin scheduler.

None of these subflows are used for the SECRET Project. The ETCS traffic over TCP presents some specific characteristics for which a new scheduler has been deployed. The ETCS traffic has very low bandwidth requirements but very high latency and jitter constraints. Thus, the load-balancing feature of the default or redundant schedulers is not interesting for this kind of TCP traffic and retransmissions should be avoided to limit the end-to-end delay and jitter in data delivery. The scheduler developed in the SECRET Project is the "redundant scheduler" or "redundant MPTCP (RMPTCP)". This new scheduler (see Figure 9) replicates data through all the valid active subflows with the aim of ensure data delivery with minimum delay and jitter. As soon as one data chunk is received through one subflow, the data is provided to the application and the duplicated data chunks are managed at the MPTCP layer without the application being aware of them.

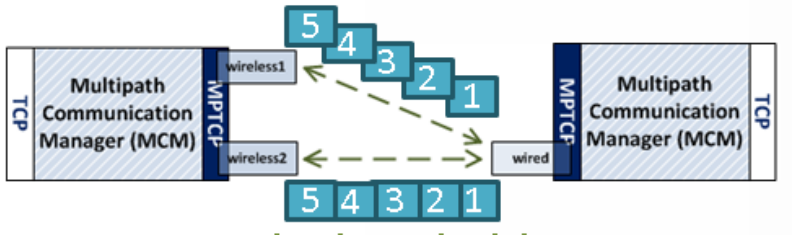


Figure 9: Behaviour of the new redundant scheduler implemented in the SECRET Project.

This new scheduler also support backup subflows in order to easily modify the traffic policy by modifying the active/backup state of the subflows:

- If there is a mix of active and backup subflows, backups subflows are only used in retransmissions when the replicated data sent by all the active subflows fails, so it is ideal for failover paths not used on normal conditions.
- If there are only backup subflows and no active subflows, the data chunks are load-balanced through all the available backup subflows based on the RTT and not replicated. This behaviour is similar to the behaviour of the default scheduler previously explained.

The behaviour of the scheduler can be summarized with the pseudocode provided in Table 1. When a packet is going to be sent, if it is a replicable packet and if it is sent through an active subflow, then it will be considered to be replicate through the rest of active subflows.

```
# redundant_scheduler

def select_best_subflow(packet):
  subflow_list = active_subflows_notused(subflows, packet)
  subflow_best = best_rtt_subflow_and_ready(subflow_list)
  if subflow_best is None:
```

```
    subflow_list = backup_subflows_notused(subflows, packet)
    subflow_best = best_rtt_subflow_and_ready(subflow_list)
  return subflow_best

for packet in packets_to_send:
  if packet is new:
    replicable = 1
  elif packet is reinjected:
    replicable = 1
  else packet is redundant:
    replicable = 0

  subflow_best = select_best_subflow(packet)
  send(packet, subflow_best)

  if replicable and subflow_best is active:
    for subflow in subflows:
      if subflow is active and subflow != subflow_best:
    if subflow.ready():
      # subflow is ready to send: established,
      # without congestion, ...
      send(packet, subflow)
```

Table 1: Pseudocode in Python of the new redundant scheduler for MPTCP.

The redundant scheduler has been developed using the official MPTCP code from commit `f9ca33d` and taking as reference the default scheduler. The new code is available in the private git repository of the SECRET Project https://gitprojects.i2t.ehu.eus/secret/mptcp and it is going to be publically released in the public git repository https://github.com/i2t/rmptcp.

In order to use the redundant scheduler, kernel provided in the private SECRET git repository must be compiled and the new redundant scheduler module must be selected for compilation inside the kernel binary or as a module.

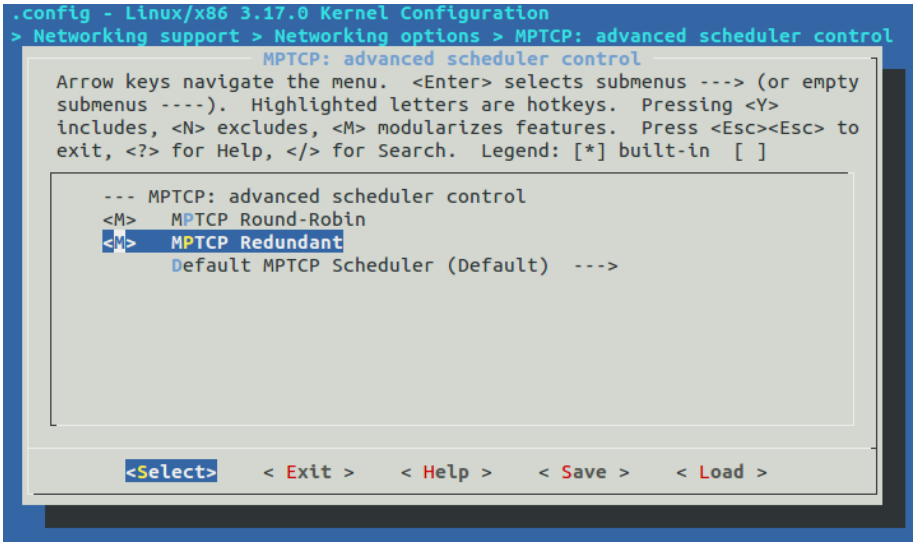


Figure 10: Selection of the redundant scheduler as a module for compilation.

Once booted the new kernel, `sysctl` application can be used to select the redundant scheduler instead of the default scheduler.

```
root@linux:~# uname -a
```

```
Linux linux 3.17.0-20150223-rmptcp #2 SMP Mon Feb 23 13:24:20
CET 2015 x86 64 x86 64 x86 64 GNU/Linux
root@linux:~# sysctl net.mptcp.mptcp_scheduler=redundant
net.mptcp.mptcp_scheduler = redundant
root@linux:~# sysctl -a | grep "net.mptcp"
net.mptcp.mptcp_binder_gateways =
net.mptcp.mptcp_checksum = 1
net.mptcp.mptcp_debug = 0
net.mptcp.mptcp_enabled = 1
net.mptcp.mptcp_path_manager = fullmesh
net.mptcp.mptcp_scheduler = redundant
net.mptcp.mptcp_syn_retries = 3
```

Table 1: Ensure that correct kernel and MPTCP scheduler is loaded.

Moreover, it is recommended to install common Linux applications such as `iproute2` with support of MPTCP from the webpage http://www.multipath-tcp.org. These utilities do not need to be modified to support the new scheduler.

Finally, due to our work in MPTCP, it is also worth pointing out that we have also collaborated in the deployment of the official default scheduler by providing a patch to guarantee the selection of the best subflow (https://github.com/multipath-tcp/mptcp/commit/f9ca33df8007af2e31c887b6cee6a51b493801c7).

Figure 11: Contribution to the default scheduler of the official MPTCP implementation project.

6.2 TCP/MPTCP proxy

There are multiple options for selecting the kind of proxy and the specific software to use in the MCM. One alternative would be to use an application proxy; however, with an application proxy we are limiting the proxy to one specific application such as HTTP web browsing. Another alternative would be to use a SOCKS proxy valid for all kind of TCP and UDP traffic; however, it is not so easy to implement a transparent SOCKS proxy. Finally,

understanding that this is the less important component of the MCM and that can be replaced by other kind of proxy or implementation without any problem, we decided to use a simple TCP proxy.

The TCP proxy selected, named tcp-intercept, is a free AGPL licensed user-space proxy developed in C/C++ and it needs the collaboration of Linux iptables to forward the desire TCP traffic to it. Obviously, since it is a user-space implementation, it has not so good performance as kernel-space implementations but it is still valid and even more flexible for developing and testing purposes.

The code of the tcp-intercept proxy is available in github https://github.com/VRT-onderzoek-en-innovatie/tcp-intercept and it is also available in the private repository of the SECRET project https://gitprojects.i2t.ehu.eus/secret/tcp-intercept. Due to our work with this proxy we proposed some modifications to the creator of the software, which were accepted and included in the software (https://github.com/VRT-onderzoek-en-innovatie/tcp-intercept/commit/456bc091150aae2257bd045e839151d92e1b9fa3). This commit, commit number `456bc09`, is the version used in the implementation.

Figure 12: Contribution to the tcp-intercept project.

This proxy comes with an init script so it is automatically launched on every reboot of the MCM. Furthermore, now thanks to the patch provided the configuration of the proxy is quite straightforward and now manual configuration of iptables is not required because the init script can create the required iptables rules automatically. Only the `/etc/default/tcp-intercep` file must be configured with the options indicate in table below.

```
CONTROL_NETWORK=1
CONTROL_IPTABLES=1
BIND_LISTEN=[0.0.0.0]:5000
BIND_OUTGOING=[0.0.0.0]:0
LOGFILE=/var/log/tcp-intercept.log
```

Table 2: Configuration of the /etc/default/tcp-intercept file for the tcp-intercept proxy.

6.3 MCM engine

The MCM engine is the code responsible for managing the behaviour of the MCM and interacting with the HAM of the DS. The code for this component is located in the private git repository of the SECRET Project https://gitprojects.i2t.ehu.eus/secret/rca under the folder `mcs` (it means Multipath Communication System).

The MCM engine supports two operating modes: AUTO and MANAGED modes. In auto mode a default configuration is applied, while in managed mode the HAM of the DS configures external communications in the way it considers best.

To run the MCM engine, it must be provided with a configuration file with default values and with the list of external interfaces and parameters for these interfaces. Table 2 is an example of configuration file for the MCM engine.

```
# MCM default configuration file

# Server configuration
server.reply.port = 5001
server.publisher.port = 5002

# MCM related configuration
mcm.mode = auto
mcm.heartbeat_period = 60
mcm.heartbeat_timeout = 180

# log_levels = severe, warning, info, fine, finer, finest
mcm.log_file = /opt/rca/mcs/mcm.log
mcm.log_level = finest

# Interface list
mcm.interfaces = wlan0, p33p1

# Interface specific configuration
wlan0.type = WLAN2400
wlan0.network_mode = enabled
wlan0.network_metric = 100
wlan0.mptcp_mode = enabled

p33p1.type = WIMAX5000
p33p1.network_mode = enabled
p33p1.network_metric = 101
p33p1.mptcp_mode = enabled
```

Table 2: Example of configuration file for the MCM engine.

The most relevant part of the Table 2 is the section regarding external interfaces. The MCM has two external interfaces whose OS identifiers are wlan0 and p33p1. The first external interface is a wireless interface in the 2,4GHz frequency range, whereas the second one is a WiMAX interface in the 5GHz frequency range. Furthermore, it is provided the default configuration for these interfaces. The internal wired interfaces must not be listed in this configuration file.

6.3.1 AUTO mode

In auto mode, the MCM works autonomously without the HAM. The list of external interfaces and the configuration for those interfaces is provided in the configuration file of the MCM engine. For example, according to the Table 2, in AUTO mode both interfaces (wlan0 and p33p1) would be enabled, MPTCP would be enabled and active in both interfaces and the minor metric value of the wireless interfaces makes it the preferred interface to establish new connections.

6.3.2 MANAGED mode

In managed mode, the DS (in fact the HAM) can customize the configuration of external interfaces in real-time. To allow that, one HAM-MCM interface and the message exchange format have been defined and implemented.

6.3.2.1 HAM-MCM Interface

The HAM-MCM interface has been defined by using the ZeroMQ framework. In fact, the JeroMQ java library version 0.3.4 is used for this purpose.

The MCM engine provides two ZeroMQ sockets to the HAM:

- One Reply socket that should by connected by the HAM using a ZeroMQ Request socket. This is the Req-Rep Interface.
- One Publisher socket that should be connected by the HAM using a ZeroMQ Subscriber socket. This is the Pub-Sub Interface.

The main interface is the Req-Rep Interface. Through this interface the HAM can send request/commands to the MCM and the MCM will answer successfully or with one error.

The second interface is the Pub-Sub Interface. This interface allows the HAM to connect to the publisher interface with a subscriber socket and to receive notifications when events occur in the MCM. This is the interface to send asynchronous notification from MCM to HAM. Then, he HAM might react to this notifications by sending commands through the Req-Rep Interface.

6.3.2.2 HAM-MCM Messages

The messages between the HAM and MCM are text-based. However, a Java library has been built to easily work with Java classes and objects inside the MCM and HAM applications, and to also easily serialize or de-serialize JSON strings. This library is shared between the HAM and MCM implementations, and it is located in the folder `lib` of the private git repository of the SECRET Project https://gitprojects.i2t.ehu.eus/secret/rca.

For example, to build a new message it is only needed the `eu.secretproject.rca.lib.mcm.Message` class. In the new Message object, there are two parts that are closely related. Firstly, the type of message must be set by using the `eu.secretproject.rca.lib.mcm.Message.Type` enum. Secondly, the data associated to the message type must be provided. For that, the data object can be set to any object derived from the `eu.secretproject.rca.lib.mcm.Data` abstract class:

- `eu.secretproject.rca.lib.mcm.McmConfData`
- `eu.secretproject.rca.lib.mcm.InterfaceData`
- `eu.secretproject.rca.lib.mcm.InterfaceListData`
- `eu.secretproject.rca.lib.mcm.InterfaceConfData`
- `eu.secretproject.rca.lib.mcm.InterfaceStatusData`
- `eu.secretproject.rca.lib.mcm.ErrorData`

Each data class provides different information:

- `eu.secretproject.rca.lib.mcm.McmConfData`
  - `mode`: MCM can be in auto or managed mode.
  - `heartbeatPeriod`: in managed mode hearbeats should be sent every heartbeatPeriod (seconds).
  - `hearbeatTimeout`: in managed mode the lack of heartbeats during hearbeatTimeout (seconds) cause a failover to auto mode.
- `eu.secretproject.rca.lib.mcm.InterfaceData`
  - `id`: ID of the interface
  - `type`: Type of the interface (WLAN2400, WLAN5000, GSM900, …)
- `eu.secretproject.rca.lib.mcm.InterfaceListData`
  - `interfaces`: List of InterfaceData objects.
- `eu.secretproject.rca.lib.mcm.InterfaceConfData`
  - `id`: ID of the interface
  - `type`: Type of the interface (WLAN2400, WLAN5000, GSM900, …)
  - `networkMode`: enable/disable the interface
  - `networkMetric`: set the preference for the default route in this interface
  - `mptcpMode`: active/backup/disable MPTCP and thus TCP/MPTCP soft-handovers in this interface.
- `eu.secretproject.rca.lib.mcm.InterfaceStatusData`
  - `id`: ID of the interface
  - `type`: Type of the interface (WLAN2400, WLAN5000, GSM900, …)
  - `networkStatus`: up/down
- `eu.secretproject.rca.lib.mcm.ErrorData`
  - `code`: code of error
  - `message`: text providing some explanation about the error.

The type and data of the message are closely related and only some combinations are allowed. Even, some messages are only allowed on one interface type, for example in the Req-Rep interface, and even in one specific mode, for example only in managed mode. These combinations are explained in the following subsections.

One example of using this library to obtain the JSON string to send through ZeroMQ interfaces is shown below in Table 3.

```
Message message = new Message();
Message.setMesage(Message.Type.GET_MCM_CONF, null);
String jsonString = getJsonString(message);
```

Table 3: Example of using the message library to get the JSON string of the message.

Furthermore, when a JSON string is read from the ZeroMQ interface it is also very easy to convert it to a Message object (see Table 4).

```
Message message = New Message();
Message = Message.setFromJsonString(receivedJsonString);
```

Table 4: Example of using the message library to get a Message object from a JSON string.

a) Req-Rep Messages

The next Table 5 shows the requests and replies messages allowed in Req-Rep Interface. Moreover, it shows the request messages allowed per operating mode (auto or managed) and the reply messages allowed per request message.

When the MCM is initialized, it boots in auto mode. Under this mode, the HAM cannot set or change the communication interfaces (SET_INTERFACE_CONF). To do that, HAM needs to change the MCM mode from auto to managed by using the (SET_MCM_CONF). Once done, the HAM needs to periodically send heartbeats (HEARBEAT) so that the MCM considers the HAM is still alive and ruling the MCM. If there is a lack of heartbeats (HEARTBEAT) during a specific period of time, the MCM will enter again automatically in auto mode and will apply the default configuration for the interfaces. Thus, in the managed mode, the HAM can instruct the MCM how to configure the network interfaces (SET_INTERFACE_CONF) and must provide periodically heartbeats (HEARTBEAT).

The MCM provides the HAM with all the possibilities to configure the external interfaces. The HAM might disable or enable one interface, set MPTCP disabled, active or backup in one interface, and set the interface as preferred or not by modifying the metric value. These options per interface produce a high number of available combinations. However, the HAM will use a smaller group of alternatives called traffic policies. These traffic policies of the HAM, defined in the page **Erreur ! Signet non défini.** of this document, relay on the use of SET_INTERFACE_CONF messages.

In the auto mode, the MCM uses the configuration per interface provided in the configuration file for the MCM engine. So, The MCM does not need the HAM and the HAM cannot make use of the SET_INTERFACE_CONF and HEARTBEAT requests in this state of the MCM (Any request of these types during the auto mode will result in ERROR_SET_INTERFACE_CONF and ERROR_HEARTBEAT reply messages).

MCM Mode		Request Message			Reply Message		
Auto	**Managed**	**Type**	**Data**	**Description**	**Type**	**Data**	**Description**
X	X	GET_MCM_CONF	Null	Get current MCM configuration	OK_GET_MCM_CONF	McmConfData	Provides information
					ERROR_GET_MCM_CONF	Null or ErrorData	Error
X	X	SET_MCM_CONF	McmConfData	Set a new MCM configuration	OK_SET_MCM_CONF	Null	Configuration applied
					ERROR_GET_MCM_CONF	Null or ErrorData	Error
X	X	GET_INTERFACE_LIST	Null	Get current list of WAN interfaces	OK_ GET_INTERFACE_LIST	InterfaceListData	Provides information
					ERROR_ GET_INTERFACE_LIST	Null or ErrorData	Error

X	X	GET_INTERFACE_CONF	InterfaceData	Get current configuration of the interface	OK_ GET_INTERFACE_CONF	InterfaceConfData	Provides information
					ERROR_ GET_INTERFACE_CONF	Null or ErrorData	Error
	X	SET_INTERFACE_CONF	InterfaceConfData	Set a new configuration for the interface	OK_ SET_INTERFACE_CONF	Null	Configuration applied
					ERROR_ SET_INTERFACE_CONF	Null or ErrorData	Error
X	X	GET_INTERFACE_STATUS	InterfaceData	Get current status of the interface	OK_ GET_INTERFACE_STATUS	InterfaceStatusData	Provides Information
					ERROR_ GET_INTERFACE_STATUS	Null or ErrorData	Error
	X	HEARTBEAT	Null	Heartbeat	OK_HEARTBEAT	Null	Heartbeat received
					ERROR_HEARTBEAT	Null or ErrorData	Error not in managed mode

Table 5: Request and Reply Messages for the Req-Rep Interface.

a) Pub-Sub Messages

Nowadays, only one notification has been defined in order to notify about changes in the status of one interface (for example, the interface goes down or goes up). This might be an important notification because the HAM might require to change the configuration of the MCM depending if one interface losses or recovers the connection.

MCM Mode		Notification Message		
Auto	**Managed**	**Type**	**Data**	**Description**
X	X	NOTI_INTERFACE_STATUS	InterfaceStatusData	Provides information about a change in the status of one interface

Table 6: Notification Messages for the Pub-Sub interface.

The HAM should subscribe to the Pub-Sub Interface of the MCM as soon as possible and it will receive notifications either in the auto mode or in the managed mode.

6.4 Additional considerations

Apart from these three main components of the MCM, other minor functions must be implemented by the Operating System so that the MCM can run successfully.

Interface configuration

The interface configuration must be performed as the usual way in any Linux OS for both, external and internal interfaces. This includes the specific technology settings (for example, WiFi security or PPP authentication) and the IP addressing related configuration. The external interfaces will make use of MPTCP, whereas internal interfaces do not need it. Furthermore, external interfaces might require the use of NAT or the application of firewall rules.

In Debian-like Linux OS, this configuration is performed in the `/etc/network/interfaces` file.

```
# WiFi MPTCP interface, WAN MCM interface
auto wlan0
iface wlan0 inet dhcp
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
    post-up ip link set dev wlan0 multipath on
    post-up iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
    pre-down iptables -t nat -D POSTROUTING -o wlan0 -j MASQUERADE

# internal NO-MPTCP interface
auto p34p1
iface p34p1 inet static
    address 10.98.99.1
    netmask 255.255.255.0
    post-up ip link set dev p34p1 multipath off
```

Table 7: Example of configuration for internal and external interfaces in the /etc/network/interfaces file.

Advanced IP routing configuration for MPTCP interfaces

The use of MPTCP requires a more complex routing configuration for the Linux device. Police Based Routing (PBR) is required and multiple routing tables must be configured, one for each MPTCP interface, because different routing tables are used depending on the source IP address (and thus the source interface) of the packet to be sent.

The configuration of these routing tables depends on the way interfaces are configured. If the configuration of the interface is static (static IP address), static configuration for PBR may also be provided. However, if interfaces are configured dynamically, for example, via DHCP, scripts must be developed so that the DHCP client software can configure correctly routing tables whenever a DHCP lease is provided or finished. Thus, it depends on the kind and configuration way of the interfaces.

Monitoring of the external interfaces

In order to monitor in real-time the usage of the external interfaces, a simple web page has been created, see Figure 13. This webpage allows verifying that the changes in the external interfaces are correctly applied and are generating changes in the traffic bandwidth.

Launching of required daemons

The tcp-intercept proxy already includes an init script.

However, the MCM engine does not have it and the simplest way to run the MCM engine on every reboot is to include the following command in the `/etc/rc.local` file.

Figure 13: Monitoring web page for external interfaces.

```
java -jar target/mcs-0.0.1-jar-with-dependencies.jar -i
config/mcm.properties -o /tmp/mcm.running.properties
```

Table 8: Command to run the MCM engine.

The "-i" option sets the file with the default configuration for the MCM engine (one example of configuration is provided in Table 2) and the "-o" option sets the output file to dump the current configuration applied to each external interface.

For the monitoring of the external interfaces, it is also required to launch the HTTP web server and run the monitoring script from `/etc/rc.local` after a reboot.

7 Conclusion

In this document how the different components of the Resilient Architecture have been implemented is described. In particular, the algorithms of the Acquisition System that allow to detect that an attack is ongoing are fully described. Inside the Health Attack Manager, some policies have been implemented. They indicate how the system must react in case of an attack. And finally, how some details about MPTCP on which the Multipath Communication System is based are given. They allow to understand how the system can react effectively in case of an EM attack.