Moduldokumentation

(MOD)

(TIT10AID, SWE I Praxisprojekt 2011/2012)

Projekt:	Multicastor 2.0
Auftraggeber:	*Andreas Stuckert und Markus Rentschler*
Auftragnehmer:	***Team 3*** *Michael Kern* *Pascal Schuhmann* *Roman Scharton* *Manuel Eisenhofer* *Tobias Michelchen*
Autor:	*Roman Scharton*

Modul

MMRP-Paketdekodierung und -erzeugung

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1. History

Version	Datum	Autor(en)	Kommentare
1.0	19.04.2012	Scharton	Dokument angelegt, Anforderungen und Voruntersuchungen ausgefüllt
1.1	22.04.2012	Scharton	Begonnen mit der Systemanalyse und Design
1.2	27.04.2012	Scharton	Systemanalyse Design sind vollständig. Begonnnen mit Risiken
1.3	29.04.2012	Scharton	Begonnen mit den Kapitlen Komponententest und Implementierung
1.4	11.05.2012	Scharton	Paketaufbau und Beschreibung hinzugefügt
1.5	17.05.2012	Scharton	Beschreibung: JRegistry, Quellen, Verweise

2. Scope

The Module Documentation (MOD) describes the architecture, the interfaces and the main features of the module. It also describes the module/component test including the results. It can also serve as a programming or integration manual for the module. If there are some risks related to the module itself, they shall be noted and commented within this document.

Die Moduldokumentation beschreibt die Architektur, die Schnittstellen und die Hauptmerkmale des Moduls. Außerdem werden die Modul bzw. Komponententests einschließlich der Ergebnisse beschrieben und dokumentiert. Die MOD dient bei Bedarf auch als Programmier- oder Integrationshandbuch für das Modul. Wenn bestimmte Risiken direkt mit der Verwendung des Moduls verknüpft sind, so sind sie in diesem Dokument zu benennen und zu kommentieren.

3. Definitionen

3.1. Abkürzungen

JVM – Java Virtual Machine
MMRP – Multiple MAC Registration Protocol
API – Application Programming Interface
IEEE – Institute Of Electrical and Electronics Engineers

3.2. Definitionen

Protocol Header - ein Teil des Datenpakets, welcher alle notwendigen Informationen für das jeweilige Protokoll enthält.
Jnet Pcap API - eine Code-Bibliothek mit einem C- und Java-Anteil, welche unterschiedliche Ressourcen für Arbeit mit Netzwerkprotokollen zur Verfügung stellt. Diese Ressourcen ermöglichen das Versenden, Empfangen und Dekodieren von Paketen der unterschiedlichen Netzwerkprotokolle.
Java Virtual Machine - ein Teil der Laufzeitumgebung, welcher für die Ausführung der Java Programme verantwortlich ist und die Betriebssystemunabhängigkeit gewährleistet.
Java Garbage Collector - ein Mechanismus zum Freigeben vom ungenutzten Speicher zur Laufzeit.
Object Pool - ein Designpattern zum Vermeiden der teueren Objekterzeugung. Es führt eine Liste bestimmter Objekte. Sobald eine Anfrage nach einem neuen Objekt ankommt, wird zunächst geprüft, ob es ungenutzte Objekte des angeforderten Typs vorliegen um sie anschließend zu übergeben und die Neuerzeugung zu vermeiden.
Ethernet-2-Frame - ein durch IEEE 802.3 festgelegtes Paketformat.

4. Anforderungen

4.1. Benutzersicht

Das Modul PacketBuilder and Analyzer MMRP beschäftigt sich mit dem Erstellen und Interpretieren von MMRPD-Units. Die MMRPD-Units regeln die Registrierung der Teilnehmer für die Multicastströme.

4.2. Kontext der Anwendung

Dieses Modul stellt eine zentrale Komponente für die Funktionsweise des MMRP-Protokolls dar.
Dieses Modul unterstützt die Klasse PcapListener [1] die eingehenden MMRPD-Units aus dem gesamten Paketstrom auszufiltern und die Steuerungsinformationen aus den MMRPD-Units an die Instanzen der Klasse MMRPLocalController [3] für die weitere Auswertung weiterzuleiten.
Die Erstellung der MMRP-Pakete erfolgt in den Instanzen der Klasse MMRPPacketBuilder.

4.3. Anforderungen

/LF10/ MultiCastor 2.0 muss das Multicast-registrierungsprotokoll nach IEEE802.1ak (MMRP) unterstützen. Es muss sowohl ein Betrieb zwischen Sender und Empfänger mit einem zwischengeshalteten MMRP-fähigen Netzwerk als auch direkt Back-to-Back möglich sein.

5. Analyse

5.1. Voruntersuchungen

Aufgrund der Widersprüchlichkeit der vorliegenden Dokumentation über den Aufbau eines MMRPD-Units müssen weitere Quellen erschlossen werden. Dazu wird die bestehende Implementierung eines Dissectors des Tools WireShark verwendet.
Durch die Analyse der vorliegenden Quellen kann in der Abbildung... dargestellte Paketaufbau ermittelt werden.
Um die Richtigkeit des ermittelten Aufbaus zu prüfen, wird eine Testumgebung erstellt. Diese Testumgebung besteht aus einem einfachen Programm, welches Byte-Arrays nach dem beschriebenen Aufbau generiert und mit Hilfe vom Framework jnetPcap versendet.
Die Richtigkeit des Pakets wird beim Erfassen der gesendeten Pakete mit dem Tool Wireshark festgestellt.
Nach der erfolgreichen Durchführung von diesem Test kann der zweite Schritt eingeleitet werden. Dabei werden die MMRP-Pakete an einen MMRP-fähigen Switch weitergeleitet und das als Antowrt versendete Paket wird als endgültige Vorlage genommen.
Ein MMRPD-Unit wird durch einen Header repräsentiert, welcher sich direkt im Datenteil des Ethernetpakets befindet. Die Abbildung 16 demonstriert den ermittelten Aufbau eines MMRPDUs im Detail.
Ein MMRP-Header weist eine stark variierende Struktur auf:

1. Der Header kann mehrere Message-Felder besitzen
2. Ein Message-Feld kann mehrere VectorAttribute-Felder besitzen
3. Ein VectorAttribute-Feld kann eins bis mehrere Event-Felder besitzen

Ein Message Feld kann entweder Events für eine oder mehrere MAC-Group-Adressen enthalten oder Event(s), welches als Service Requirement dient. Der letzte Punkt steuert das allgemeine Verhalten vom Switch und wird im Rahmen dieser Arbeit nicht behandelt.
Enthält ein MMRPD-Unit Events für mehrere nicht aufeinanderfolgende MAC-Group-Adressen, so werden diese in unterschiedlichen VectorAttribute-Feldern aufgeführt.
Enthält ein MMRPD-Unit Events für mehrere aufeinanderfolgende MAC-Group-Adressen, so werden diese Events in einem VectorAttribute-Feld aufgeführt. Dabei ist jedes zusätzliche Event für die nachfolgende MAC-Adresse beginnend mit der Adresse im Feld FirstValue. Beinhaltet das Feld die Adresse „01:20:30:00:00:00" und zehn Events, so sind diese Events für die Adressen von „01:20:30:00:00:00" bis „01:20:30:00:00:09" bestimmt. Die nachfolgende Abbildung demonstriert ein Beispielaufbau von einem Message-Feld:

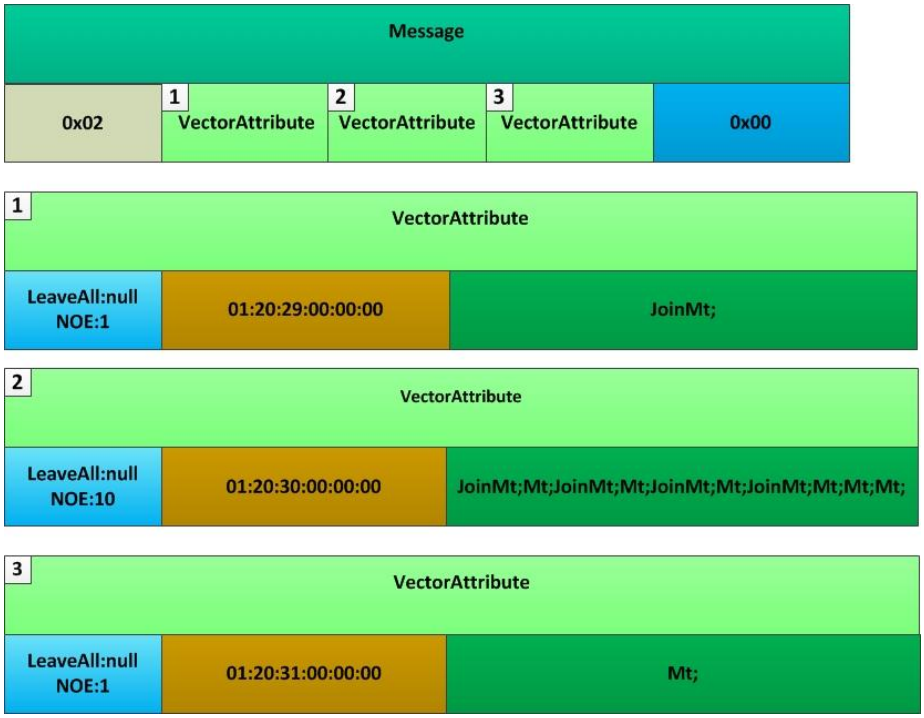

Abbildung 1: Beispiel Message-Feld[1]

5.2. Systemanalyse

5.2.1. Analyse der Jnet-Pcap API

Beim Ananlysieren von MMRPD-Units kommt die jnet PCAP-API zum Einsatz. Bei der Analyse der MMRPD-Units kommt die Klassen der API zum Einsatz, welche die native C-Struktur eines Datenpakets mit bestimmten Java Objekten verknüpfen. Diese Objekte enthalten eine Adresse und Größe des zugehörigen Pakets im nativen Speicher. Die Abbildung 2 demonstriert die Verknüpfung:

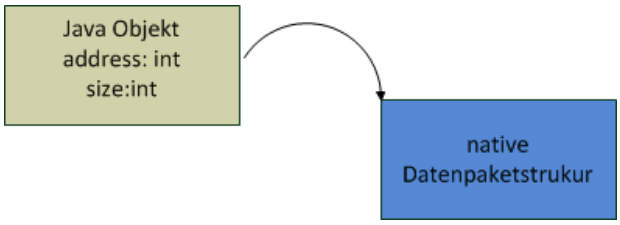

Abbildung 2: Verknüpfung der nativen Datanstruktur

5.2.2. Besonderheit beim Empfang der Pakete

Der Empfang von MMRPD-Units wird in der „MOD_MMRP-Paketanalyse _und_Steuerung_der_Multicastströme"[3] in Detail beschrieben. Hier wird nur auf eine Besonderheit der umgesetzten Architektur des Tools Multicastor 2.0 eingegangen, welche einen direkten Einfluss auf die Analyse der MMRPD-Units ausübt.
Der Weg einer Informationseinheit aus einem MMRPD-Unit bis zur Auswertung wird in der folgenden Abbildung dargestellt:

[1] vereinfachte Darstellung

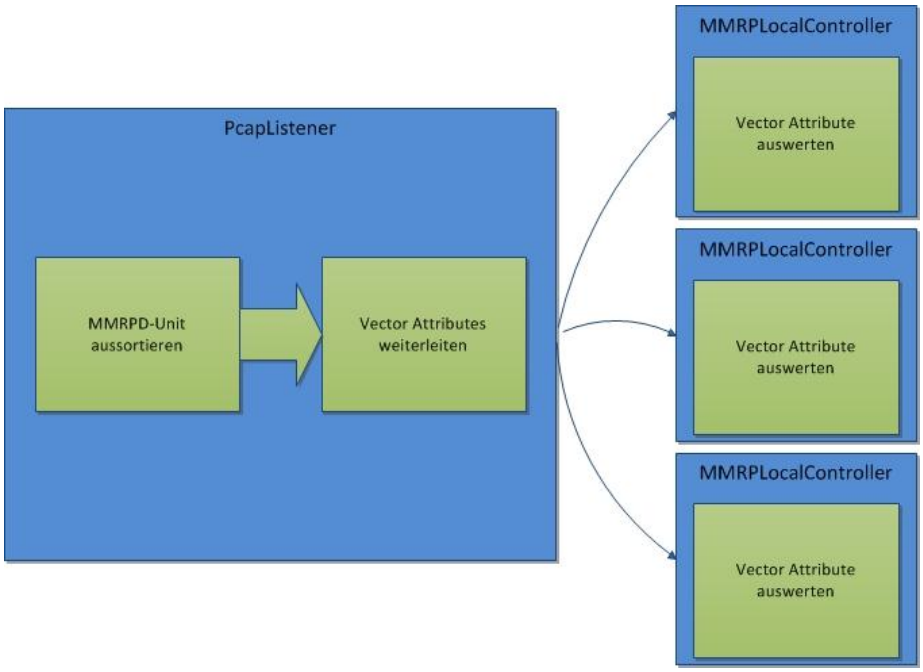

Abbildung 3: Informationsfluss aus einem MMRPD-Unit

Dabei erfolgt die Auswertung der Steuerungsnachrichten (Vector Attributes, siehe Abbildung 16) in nebenläufigen Threads. Sobald alle Vector Attributes aus einem MMRPD-Unit weitergeletiet wurden, wird das nächste Paket in der Instanz von PcapListener geladen. Beim Laden eines neuen Packets wird das vorherige Paket im nativen C-Speicher überschrieben. Da die Verarbeitung der Informationen aus dem vorigen Paket noch in einer der MMRPLocalController Instanzen noch nicht abgeschlossen sein kann, müssen diese Informationen zwischengespeichert werden.

5.2.3. JRegistry-Klasse

Diese Klasse enthält Informationen über die Netzwerkprotokolle, welche bei der Dekodierung der Pakete zum Einsatz kommen. Diese Klasse ermöglicht die Bindung zwischen den einzelnen Protokoll-Header-Klassen und der nativen Datenstruktur, enthält eine Liste aller registrierten Protokoll-Header und die zugehörige Bindungen.
Die Protokoll-Bindung schreibt vor, dass jedes Protokoll-Header ein Feld enthält, welches das nachfolgende Header (falls überhaupt vorhanden) identifiziert. So dient diesem Zweck das Feld Ether Type im Ethernet-Header. Da das MMRP-Header ans Ethernet-Header gebunden werden muss, wird in jedem MMRP-Paket der Wert 0x88F6 eingetragen.
Um die Verwendung eines neuen Protokoll-Headers zu ermöglichen, wird die zugehörige Java Klasse über die JRegistry registriert. Dabei muss das neue Header die Informationen zur Protokollbindung enthalten (z.B. EtherType der Ethernet-Header muss 0x88F6 sein). Nach der Registrierung erhält das neue Protokoll-Header eine ID, welche für eine schnelle Identifizierung des Headers in der nativen Datenstruktur sorgt. Die Registrierung muss vor der ersten Verwendung der Pcap durchgeführt werden.

6. Design

6.1. Paket-Dekodierung

6.1.1. Auswahl einer geeigneten Klasse

Der Aufbau eines MRPDUs wird mit Hife der Klasse MMRP implementiert. Die nachfolgende Abbildung demonstriert die Klassenhierarchie:

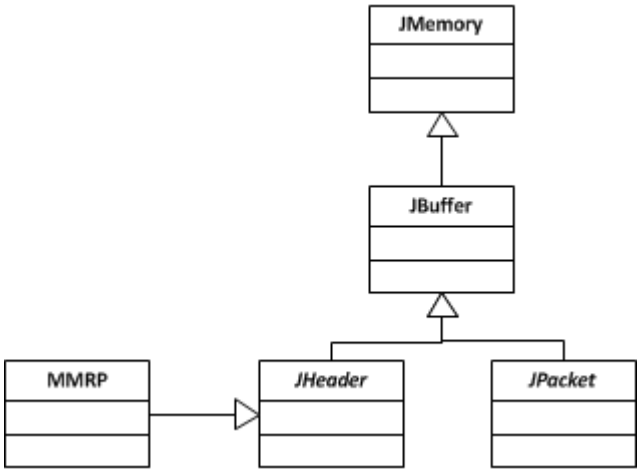

Abbildung 4: MMRP Klassenhierarchie

Die Klasse JMemory enthält die Adresse der zugehörigen nativen C-Struktur, deren Größe und einige Methode um zusätzlichen Speicher zu allokieren oder den verknüpften Speicher zu kopieren. Beim Erstellen der Instanzen dieser Klasse werden diese mit nativen Speicherbereichen verknüpft. Ein Konstruktor dieser Klasse als Beispiel: JMemory(int size). Der Parameter size gibt die Größe den zu allokierenden Speicherbereichs.
Die Klasse JBuffer stellt zusätzliche Methoden zum Erreichen der bestimmten Speicherbereichen zur Verfügung. So z.B. die Methode getByteArray(int index, int size) liefert ein byte-Array zurück. Dieser Array enthält Werte aus dem verknüpften Speicher ab der Position index und der Länge size.
Die abstrakte Klasse JHeader stellt Methoden zur Verfügung um die protokollspezifische Felder zu erreichen. Diese Klasse enthält auch eine Referenz auf die native Datenstruktur eines Pakets. Diese Datenstruktur enthält Informationen über die vorhandenen Protokoll-Header in einem Netzwerkpaket und ihre Reihenfolge. Diese Reihenfolge wird durch Protokollbindung realisiert. Dieses Verfahren wird im Kapitel 5.2.3 näher beschrieben.
Die Klasse MMRP definiert einen konkreten MMRPD-Unit.
Die Instanzen der abstrakten Klasse JPacket werden mit einem ganzen Datenpaket im nativen Speicher verknüpft. Diese Klasse stellt Methoden zur Verfügung um die einzelnen Protokollheader und den Datenteil eines Pakets zu erreichen. So z.B. die Methode getHeader(JHeader header) verknüpft die übergebene Instanz der Klasse JHeader mit dem passenden nativen Speicher des Datenpakets.

6.1.2. Aufbau der MMRP-Klasse

Für die Abblidung eines MMRPD-Units wird folgender Design gewählt:

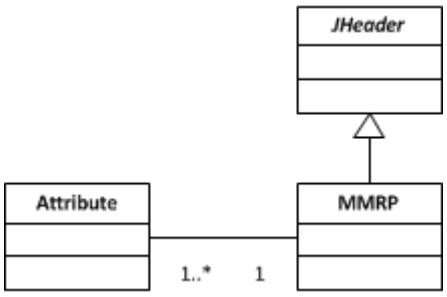

Abbildung 5: MMRP Header Aufbau

Der MMRP-Header bildet einen MMRPD-Unit ohne den Ethernet-Teil (siehe Kapitel 5.1) ab. Die meisten Felder eines MMRPD-Units werden in die Klasse Attribute ausgelagert. Diese Architekturentscheidung hat die folgenden zwei Hintergründe:

1. Die einzelnen Felder eines Protokoll-Headers müssen einen eindeutigen und eindeutigen Namen haben. Da ein MMRPD-Unit eine beliebige Anzahl von Feldern mit dem gleichen Namen haben kann, z.B. Message-Felder, wird es schwer diese Struktur allein anhand der JHeader Klasse umzusetzen.
2. Die Informationen (Vector Attributes), welche die Instanzen der Klasse Attribute halten, müssen nach der Auflösung der MMRP-Instanz vorhanden bleiben (siehe Kapitel 5.2.2).

Die Klasse Attribute beinhaltet alle Felder der Teilstruktur VectorAttribute eines MMRPD-Untis (siehe Kapitel 5.1). Alle Methoden und Attribute der Klassen MMRP und Attribute befinden sich im Anhang.
Nach dem Aufruf der getHeader()-Methode der Klasse JPacket, wird der MMRPD-Unit dekodiert. Die Dekodierung wird nach Verknüpfung des nativen Speichers mit Hilfe der Methode decodeHeader() durchgeführt. Diese Methode berechnet die Länge des MMRP-Headers und die Länge der beinhalteten Feldern mit der variablen Größe, und erstellt für jedes VectorAttribute-Feld eine Instanz der Klasse Attribute, welche gleichzeitig mit den notwendigen Werten gefüllt wird.

6.2. Paketgenerator

Für die Generierung der MMRPD-Units ist die Klasse PacketBuilderMMRP zuständig. Die Instanzen dieser Klasse werden von der Klasse MMRPLocalController erzeugt und gehalten. Als Übergabeparameter wird eine Instanz der zugehörigen MulticstSenderMMRP- oder MulticstReceiverMMRP-Klasse übergeben um die Adresse der Geräteschnittstelle und die Multicastadresse auszulesen. Um eine schnelle Generierung von Paketen zu gewährleisten, wird diese Klasse wie folgt aufgebaut:

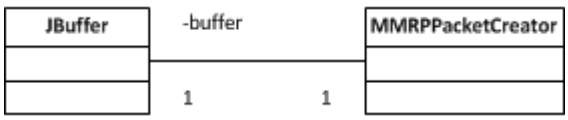

Abbildung 6: MMRPPacketCreator

Das Objekt buffer referenziert auf den nativen C-Speicher einer festen Größe (60 Byte). Diese Größe wird gewählt um die Anforderungen an die Mindestgröße eines Ethernet 2 Frames zu erfüllen.
Die direkte Auslagerung der Paketstruktur in den nativen C-Speicher ermöglicht schnelles Versenden von Paketen über die Jnet Pcap API [3], da die API Pakete aus dem nativen C-Speicher versendet werden.
Das Objekt buffer wird im Konstruktor der Klasse MMRPPacketCreator mit vordefinierten Werten gefüllt. Diese Werte bleiben bei jedem MMRPD-Unit mit einem einzelnen Event immer gleich (Tabelle 3: Beschreibung der Felder eines MMRPD-Units).

Bei jedem Erstellen von einem MMRPD-Unit werden nur die Felder leaveAll-Event und event entsprechend der übergebenen Parameter gesetzt (siehe Kapitel **Error! Reference source not found.**).

7. Risiken

Dieses Modul weist eine vollständige Abhängigkeit von der externen API jnet PCAP. So werden auch alle Stärken und Schwächen[2] der API beim Erstellen und Dekodieren von Netzwerkpaketen auf diese Komponente (Modul) übertragen.
Das Versagen dieser Komponente würde zum Versagen der gesamten Erweiterung des Tools Multicastor 2.0 um das MMR-Protokoll.
Eine mögliche Alternative ist die Verwendung einer anderen API. Aufgrund der starken Zeitbegrenzung des Projekts wird dennoch die Suche nach Alternativlösungen verworfen.
Die Komponente enthält ansonsten keine kritischen Teilkomponenten oder kritischen Systemprozesse.
Die Erweiterung des Tools Mutlicastor 2.0 wird hauptsächlich dazu benutzt um die MMRP-fähigen Switche zu testen. Dabei wird aufgrund der Performanceoptimierung darauf verzichtet eine fehlerhafte Struktur der Pakete zu erkennen.
Als ein Schutzmechanismus gegen den Absturz der Anwendung wird eine Behandlungsroutine der BufferUnderflowException in der anderen Komponente eingebaut [1].

[2] nachzusehen in der Beschreibung der entsprechenden jnet PCAP-Beschreibung(11.1)

8. Implementierung

8.1. Paket-Dekodierung

Die Implemenitierung dieser Teilkomponente ergab sich aus der Analyse des Systems und Paketaufbaus (siehe Kapitel 5). Alle Methoden und Felder mit der zugehöriger Beschreibung können im Anhang eingesehen werden.

8.2. Paketgenerator

Für die Generierung der Pakete wird eine umfangreichere Version der Klasse PacketBuilderMMRP erstellt. Diese Klasse ermöglicht es Pakete mit mehreren Events zu generieren. Diese Teilkomponente wird vor allem beim Testen der Teilkomponente zur Paket-Analyse eingesetzt.
Der Einsatz dieser erweiterten Klasse im Tool Multicastor 2.0 erscheint nicht sinnvoll, da dadurch zusätzliche Synchronisationsprobleme erscheinen können.
Alle Methoden und Felder mit der zugehörigen Beschreibung können im Anhang eingesehen werden.

9. Komponententest

Da die Teilkomponente zur Paketanalyse unterschiedliche Paketaufbauarten unterstützen muss, wird zum Testen dieser Komponente eine erweiterte Version der Klasse PacketBuilderMMRP eingestzt. Da sogar diese Klasse nicht alle möglichen Paketaufbau-Kombinationen abdecken kann, wird eine Klasse ArrayGenerator erstellt. Diese Klasse generiert Byte-Arrays mit der Struktur eines MMRPD-Units. Der Inhalt wird dabei manuell festgelgt.
Die beiden oben beschriebenen Klassen generieren passende Byte-Arrays. Der Inhalt dieser Byte-Arrays wird über einen Konstruktor der Klasse JMemoryPacket in die nativen C-Struktur eines Ethernet-Packets eingebunden. Anschließend kann der der MMRP-Header mit der Methode getHeader() dekodiert werden. Dadurch wird der Zugriff auf die Felder und Methoden der beteiligten Klassen MMRP und Attribute möglich.
Die Klasse PacketBuilderMMRP wird aufgrund ihres trivialen Aufbaus nur mithilfe vom Tool Wireshark getestet. Dabei werden generierte Pakete über die jnet Pcap API versendet und anschließend im Wireshark Trace beobachtet.

9.1. Komponententestplan

TestNo	Feature ID	Test Specification (Description or TCS)
1	Events	Erkennen von einem Paket mit mehreren Events in einem VectorAttribute
2	VectorAttributes	Erkennen von einem Paket mit mehreren VectorAttribute-Feldern
3	Messages	Erkennen von einem Paket mit mehreren Message-Feldern
4	ServiceRequirement Attribute Type	Erkennen von einem Paket mit einem Message-Feld vom Typ Service Requierement
5	Paketgenerierung	Generieren von Paketen mit einem Event

Tabelle 1: Tests

9.2. Komponententestreport

TestNo	Pass/Fail	If failed: Test Result	Date	Tester
1	Pass		29.04.2012	Scharton
2	Pass		29.04.2012	Scharton
3	Pass		29.04.2012	Scharton
4	Pass		29.04.2012	Scharton
5	Pass		29.04.2012	Scharton

Tabelle 2: Tests-Auswertung

10. Zusammenfassung

Die Komponente ist getestet und kann im Tool Multicastor 2.0 eingesetzt werden.

10.1. Beurteilung der Komponente

Nur Teile dieses Moduls, die Klasse MMRPPacketCreator, weisen eine schwache Bindung an das Tool Multicastor 2.0 auf und können ohne Weiteres außerhalb des Projekts verwendet werden. Da das Dekodieren der Netzwerkpakete stark an die Architektur des Projekts angepasst ist, kann diese Komponente nur in Projekten mit einer ähnlichen Architektur verwendet werden (siehe Kapitel 5.2.2).
Die Besonderheit dieser Komponente besteht darin, dass beim Dekodieren eines MMRPD-Units mehrere Objekte für bestimmte Felder eines MMRPD-Units erstellt werden können. Dieser Umstand führt dazu, dass nach einem unbestimmten Zeitintervall der Speicher der VM vollständig gefüllt wird und der Garbage Collector aufgerufen wird. Je nach der Konfiguration der aktuellen Ausführungsumgebung (VM) kann es zu einer kurzen Unterbrechung aller Programmprozesse führen. Dennoch wird dieser Zustand beim Testen nicht festgestellt.

10.2. Ausblick für die Weiterentwicklung

Eine Weiterentwicklungsmöglichkeit stellt die Beseitigung der ständigen Neuerstellung der Objekte beim Dekodieren eines MMRPD-Units. Dies kann durch das Anwenden des Patterns ObjectPoll beseitigt werden.
Eine weitere Weiterentwicklungsmöglichkeit stellt die Erweiterung der Klasse MMRPPaketCreator um die Unterstützung der MMRPD-Units einer beliebigen Größe. Diese Erweiterung wird die Netzwerkauslastung bei Verwendung vieler Multicastadressen deutlich reduzieren.

11. Anhang

11.1. JavaDoc

dhbw.multicastor.program.model.mmrp

Class MMRP

java.lang.Object
  org.jnetpcap.nio.JMemory
    org.jnetpcap.nio.JBuffer
      org.jnetpcap.packet.JHeader
        dhbw.multicastor.program.model.mmrp.MMRP

All Implemented Interfaces:

org.jnetpcap.packet.JPayloadAccessor

```
@Header(name="MMRP",
        osi=DATALINK,
        nicname="mmrp",
        characteristics=POINT_TO_MULTIPOINT)
public class MMRP
extends org.jnetpcap.packet.JHeader
```

Klasse zur Erkennung von MMRPD-Units. Registrierung bei der JRegistry vor der ersten Verwendung ist erforderlich. Der MMRP-Header ist in folgende drei Felder aufgeteilt:

- protocolVersion
- messages
- endMark

Die Unterfelder des Feldes messages werden in listMes gehalten. Dabei werden Felder mit irrelevanten Informationen aussortiert.

Abbildung 7: MMRP-Class

Nested Class Summary

Nested classes/interfaces inherited from class org.jnetpcap.packet.JHeader
org.jnetpcap.packet.JHeader.State

Nested classes/interfaces inherited from class org.jnetpcap.nio.JMemory
org.jnetpcap.nio.JMemory.Type

Abbildung 8: MMRP-Innere Klassen

Constructor Summary

Constructors

Constructor and Description
MMRP ()

Abbildung 9: MMRP-Konstruktor

Field Summary

Fields

Modifier and Type	Field and Description
static byte	IN Event
static byte	JOIN_IN Event
static byte	JOIN_MT Event
private java.util.ArrayList<Attribute>	listMes Haelt alle Vector Attribute Felder des vorliegenden MMRPD-Units
static byte	LV Event
static int	mmrpEtherType Eintrag ins EtherType-Feld des Ethernet-Headers
static byte	MT Event
static byte	NEW Event

Fields inherited from class org.jnetpcap.packet.JHeader

annotatedHeader, BYTE, EMPTY_HEADER_ARRAY, isSubHeader, packet, state

Fields inherited from class org.jnetpcap.nio.JMemory

JNETPCAP_LIBRARY_NAME, MAX_DIRECT_MEMORY_DEFAULT, POINTER

Abbildung 10: MMRP-Felder

Method Summary

Methods

Modifier and Type	Method and Description
static boolean	bindToEthernet(org.jnetpcap.packet.JPacket packet, org.jnetpcap.protocol.lan.Ethernet eth)
private int	calculateVectorAttribute(int pos, int lengthAtt) Dekodiert ein AttributeVector-Feld eines MMRPD-Units.
protected void	decodeHeader()
private int	decodeMessage(int pos) Dekodiert ein Message-Feld eines MMRPD-Units
short	endMark()
int	endMarkOffset()
java.util.ArrayList<Attribute>	getListMes() Liefert die message-Felder des vorliegenden MMRPD-Units zurueck.
static int	headerLength(org.jnetpcap.nio.JBuffer buffer, int offset)
byte[]	messages()
int	messagesLength()
byte	protocolVersion()
java.lang.String	toString()

Methods inherited from class org.jnetpcap.packet.JHeader

decode, getAnnotatedHeader, getDescription, getFields, getGap, getGapLength, getGapOffset, getHeader, getHeaderLength, getHeaderOffset, getId, getIndex, getLength, getName, getNextHeaderId, getNextHeaderOffset, getNicname, getOffset, getPacket, getParent, getPayload, getPayloadLength, getPayloadOffset, getPostfix, getPostfixLength, getPostfixOffset, getPrefix, getPrefixLength, getPrefixOffset, getPreviousHeaderId, getPreviousHeaderOffset, getState, getSubHeaders, hasDescription, hasGap, hasNextHeader, hasPayload, hasPostfix, hasPrefix, hasPreviousHeader, hasSubHeaders, isFragmented, isGapTruncated, isHeaderTruncated, isPayloadTruncated, isPostfixTruncated, isPrefixTruncated, peer, peer, peerPayloadTo, setPacket, setSubHeaders, sizeof, transferPayloadTo, transferPayloadTo, transferPayloadTo, validateHeader

Methods inherited from class org.jnetpcap.nio.JBuffer

findUTF8String, getByte, getByteArray, getByteArray, getByteArray, getDouble, getFloat, getInt, getLong, getShort, getUByte, getUInt, getUShort, getUTF8Char, getUTF8String, getUTF8String, getUTF8String, getUTF8String, isReadonly, order, order, peer, peer, peer, peer, setByte, setByteArray, setByteBuffer, setDouble, setFloat, setInt, setLong, setShort, setShort0, setUByte, setUInt, setUShort, transferFrom, transferFrom, transferFrom, transferTo, transferTo, transferTo

Methods inherited from class org.jnetpcap.nio.JMemory

availableDirectMemory, check, cleanup, createReference, isInitialized, isJMemoryBasedOwner, isOwner, maxDirectMemory, peer, reservedDirectMemory, setSize, size, softDirectMemory, toDebugString, toHexdump, toHexdump, totalActiveAllocated, totalAllocateCalls, totalAllocated, totalAllocatedSegments0To255Bytes, totalAllocatedSegments256OrAbove, totalDeAllocateCalls, totalDeAllocated, transferFrom, transferFrom, transferFromDirect, transferOwnership, transferTo, transferTo, transferTo, transferTo, transferTo0

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Abbildung 11: MMRP-Methoden

dhbw.multicastor.program.model.mmrp

Class Attribute

java.lang.Object
    dhbw.multicastor.program.model.mmrp.Attribute

```
public class Attribute
extends java.lang.Object
```

Repraesentiert ein VectorAttribute-Feld eines MMRPD-Units. Stellt Methoden zur Dekodierung der einzelnen Events zur Verfuegung.

Abbildung 12: Attribute-Klasse

Constructor Summary

Constructors

Constructor and Description
`Attribute(byte[] pFirstValue, byte[] pEvents, int pLeaveAll, int pNumOfEvents)`

Abbildung 13: Attribute-Konstruktor

Field Summary

Fields

Modifier and Type	Field and Description
`private byte[]`	`firstValue` FirstValue-Feld eines MMRPD-Units
`private long`	`fV` FirstValue-Feld eines MMRPD-Units (als long)
`private int`	`leaveAll` LeaveAll-Feld eines MMRPD-Units
`private int`	`numOfEvents` NumberOfEvents-Feld eines MMRPD-Units
`private byte[]`	`unpackedEvents` Kodierte Events eines MMRPD-Units

Abbildung 14: Attribute-Felder

Method Summary

Methods

Modifier and Type	Method and Description
`int`	`getEventN(long l)` Holt einen Event unter der angegebenen Position.
`byte[]`	`getEvents()`
`byte[]`	`getFirstValue()`
`long`	`getfV()`
`int`	`getLeaveAll()`
`int`	`getMyEvent(long address)` Holt einen Event fuer die angegebene Adresse.
`int`	`getNumOfEvents()`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Abbildung 15: Attribute-Methoden

11.2. Diagramme

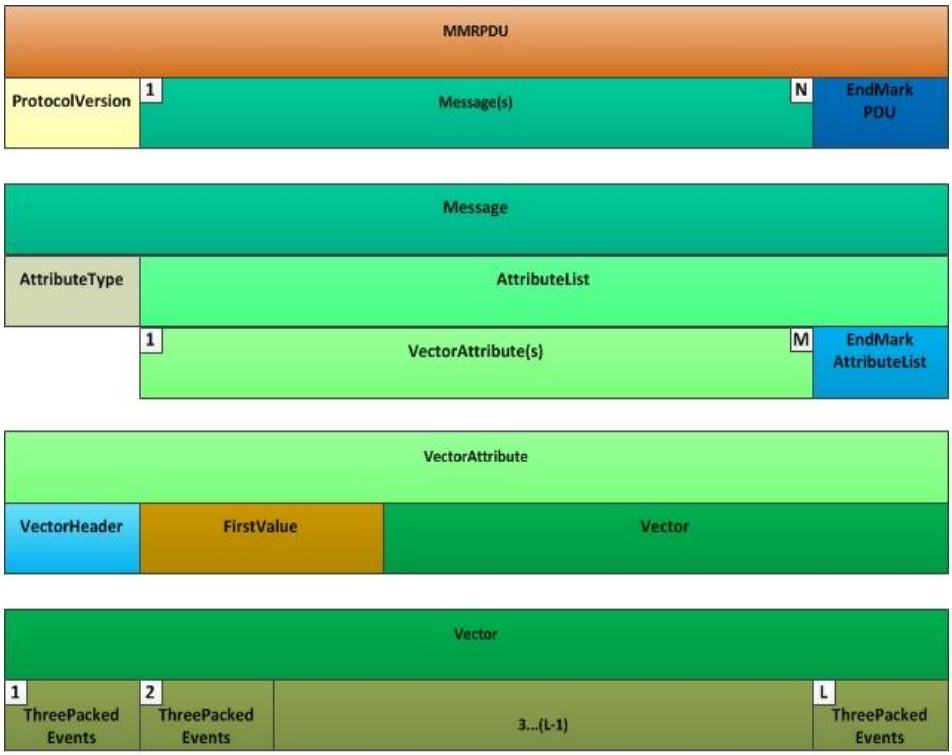

Abbildung 16: MMRPD-Unit

MMRP Komponente (Diagramm)	**Länge**	**Wert**
ProtocolVersion	1 Byte	0x00/0x01
AttributeType	1 Byte	1: Service Requirement Vector Attribute Type 2: MAC Vector Attribute Type
VectorHeader	1 Short	(**LeaveAllEvent** * 8192) + **Number-OfValues**
LeaveAllEvent	3 Bits	0: NullLeaveAllEvent Operator 1: LeaveAll Operator
NumberOfValues	13 Bits	Anzahl der kodierten Ereignisse im Vektor
FirstValue	6 Bytes/ 1 Byte	MAC Address/Value(0 oder 1)
ThreePackedEvents	1 Byte	(((((**AttributeEvent**) *6) + **AttributeEvent**) *6) + **AttributeEvent**)
AttributeEvent	3x in 1 Byte	0: New Operator 1: JoinIn Operator 2: In Operator 3: JoinMt Operator 4: Mt Operator 5: Lv Operator
EndMark	2 Bytes	0x00

Tabelle 3: Beschreibung der Felder eines MMRPD-Units

11.3. Quellen

Jochen, Jochannes (2011): Bachelorthesis: *Analyse des MMRP-Protokolls nach IEEE 802.3ak einschließlich Erstellung einer zustandsorientierten Simulation des Protokolls auf Basis von OMNeT++ sowie weiterer Tools für Testzwecke*
Sly Technologies (Stand: Mai 2012): *JnetPcap*. URL: jnetpcap.com
Elektronik-Kompendium (Stand: Mai 2012): *Ethernet-Frame*
URL: elektronik-kompendium.de/sites/net/1406191.htm
IEEE (2007): *IEEE Standard for Local and metropolitan area networks— Virtual Bridged Local Area Networks, Amendment 7: Multiple Registration Protocol*

11.4. Standards

IEEE Std 802.1ak™-2007

11.5. Literaturverzeichnis

[1] Michelchen, Tobias: MOD_PcapLitener
[2] TIT10AID Team3: STP
[3] Scharton, Roman: MOD_MMRP-Paketanalyse_und_Steuerung_der_Multicastströme