HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Bachelorarbeit, Abteilung Informatik

HiPeR
High Performance RADIUS Server

Hochschule für Technik Rapperswil

Frühlingssemester 2015
12. Juni 2015

Autoren: Filippo Pitrella & Michael Schefer
Betreuer: Prof. Beat Stettler
Projektpartner: CloudGuard Software AG
Arbeitsperiode: 16.02.2015 - 12.06.2015

Aufgabenstellung

Network Access Control, kurz NAC, wird in vielen Firmen immer mehr zum zentralen Thema. Wie schon beim Wireless-LAN wird die Zugriffskontrolle am Netzwerk mit dem Standard IEEE 802.1x gelöst. Für die Authentisierung wird fast ausschliesslich auf RADIUS mit EAP-TLS oder PEAP mit MSCHAPv2 gesetzt. Die RADIUS Server spielen deshalb eine zentrale Rolle in jeder gesicherten Netzwerkarchitektur und müssen höchste Anforderungen in den Bereichen Zuverlässigkeit sowie Geschwindigkeit erfüllen.

Heute allgemein verfügbare Lösungen wie FreeRADIUS genügen leider diesen Performance Anforderungen nicht. Ziel dieser Arbeit ist es daher, einen hochperformanten RADIUS Server basierend auf Netty (http://netty.io/) in Java zu entwickeln. Im Fokus stehen dabei die drei RADIUS Funktionsgruppen “Authentication“, “Authorization“ sowie “Accounting“.

Die Arbeit wird eng von der Firma CloudGuard Software AG begleitet, welche viel Erfahrung in asynchroner Programmierung sowie verteilten Systemen mitbringt.

Abstract

Um Netzwerke gegen unbefugte Zugriffe zu schützen, müssen Benutzer authentifiziert werden. Zu diesem Zweck wird häufig ein RADIUS Server eingesetzt. Die vom Projektpartner CloudGuard Software AG verwendete Lösung, setzt sich aus den Produkten Macman und FreeRADIUS zusammen. Diese Kombination beinträchtigt die Leistung, weil zusätzliche Kommunikation entsteht. Um die Performance zu steigern, soll deshalb die Grundlage für einen RADIUS Server geschaffen werden, welcher später mit Macman zu einem Produkt vereint wird.

Die Basis für die Implementation bildet das nicht-blockierende Client-Server-Framework Netty. Für die Bedienung der Applikation wird die Shell des Spring Frameworks eingesetzt. Unter Linux kann durch die Verwendung des epoll-Systemaufrufs ein UDP-Port mehrfach belegt und der Parallelisierungsgrad der Software gesteigert werden. Dadurch erhöht sich die Performance.

Die entstandene Applikation kann bis zu 99'000 Anfragen pro Sekunde beantworten. Sie ermöglicht die Authentifizierung über MAC Authentication Bypass (MAB), welche für Geräte gedacht ist die keinen IEEE 802.1X-Supplicant besitzen. Ebenso unterstützt wird die Authentifizierung mit PEAPv0/MSCHAPv2 von Linux-, Android-, iOS- und OS X-Clients. Bei Windows-Clients wird der Authentifizierungsvorgang nicht erfolgreich abgeschlossen. Aus zeitlichen Gründen konnte die Ursache dieses Problems nicht mehr ermittelt werden.

Accounting Daten werden erfolgreich in eine Datei geloggt und offene Sessions können mithilfe der Shell einzeln getrennt werden. Auch Change-of-Authorization (CoA) Requests können versendet werden. Allerdings ist es nicht möglich, wie vom Projektpartner gewünscht, den Namen einer Access Control List (ACL) mitzusenden. Das entsprechende Attribut wird in diesem Kontext nicht unterstützt. Deshalb beschränkt sich dieser Use Case auf das Versenden von CoA-Requests, welche lediglich die Attribute zur Identifizierung der Benutzersession beinhalten.

Management Summary

Ausgangslage

Sicherheit ist ein zentrales Thema in Netzwerken. Um sich gegen unerlaubte Zugriffe zu schützen, wird von Firmen häufig ein RADIUS Server eingesetzt. Benutzer müssen sich damit vor dem Zugriff auf das Netzwerk authentisieren. Die zurzeit von CloudGuard eingesetzte Lösung setzt sich aus den zwei Produkten Macman und FreeRADIUS zusammen. Diese Kombination zweier Lösungen verursacht zusätzliche Kommunikation, was die Performance beeinträchtigt. Deshalb soll mit dieser Arbeit die Grundlage für einen RADIUS Server geschaffen werden, welcher die Funktionalität beider Produkte in sich vereint und die Performance steigert.

Vorgehen/Technologien

Die Entwicklung wird in drei Phasen aufgeteilt, welche sich an den primären Aufgaben des Servers orientieren. In einem ersten Schritt müssen Benutzer in der Lage sein, sich am Netzwerk anzumelden. Anschliessend muss die Nutzungsstatistik eines Benutzers erfasst werden können. Die letzte Phase betrifft schliesslich die Trennung des Benutzer vom Netzwerk. Damit HiPeR auf allen Betriebssystemen eingesetzt werden kann, wird als Programmiersprache Java (Version 8) verwendet. Die Grundlage für die Implementation bildet das Netzwerk Framework Netty. Dieses erlaubt die asynchrone Bearbeitung von Anfragen und ermöglicht eine hohe Performance. Die Bedienung der Applikation erfolgt über eine Eingabeaufforderung, welche auf der Shell des Spring Frameworks basiert.

Ergebnis

Die entstandene Applikation ist in der Lage bis zu 99'000 Anfragen pro Sekunde zu bearbeiten. Sie erlaubt die Authentifizierung von Linux- und Mac-Clients sowie von Android-Smartphones und iPhones durch das Protected Extensible Authentication Protokoll (PEAP). Geräte, welche dieses Protokoll nicht unterstützen, können anhand ihrer MAC-Adresse authentifiziert werden. Sobald ein Gerät verbunden ist, werden im Hintergrund laufend Nutzungsstatistiken erfasst, um zum Beispiel Verrechnungen von bezogenen Leistungen zu ermöglichen. Sollte ein verbundener Benutzer nicht mehr für das Netzwerk zugelassen sein, besteht die Möglichkeit, diesen über die Eingabeaufforderung vom Netzwerk zu trennen.

Danksagung

Wir bedanken uns ganz herzlich bei folgenden Personen für Ihre Unterstützung während der Bachelorarbeit:

- Michael Schneider für die angenehme und konstruktive Zusammenarbeit
- Prof. Beat Stettler für die kompetente Betreuung der Bachelorarbeit
- Fabian Beck und Remo Grüebler für die technische Unterstützung und das Einrichten der Testgeräte im Labor
- Unseren Familienangehörigen und Freunden für ihr Verständnis und ihre Geduld

Inhaltsverzeichnis

Teil I.

Technischer Bericht

1. Technischer Bericht

1.1. Einführung

Dieses Dokument setzt voraus, dass der Leser mit dem RADIUS Protokoll vertraut ist und die gängigen Begriffe und Abläufe kennt. Im Anhang C ist ein Theorieteil enthalten, welcher die Grundlagen des Protokolls beschreibt.

1.2. Ausgangslage & Beschreibung

Sicherheit ist in der IT ein "cross-cutting concern". Ein offenes Netzwerk birgt nicht nur ein erhöhtes Angriffspotenzial, es erhöht auch das Risiko dass sensible Daten in fremde Hände gelangen. Deshalb wollen immer mehr Firmen ihr Netzwerk schützen, sodass Benutzer sich für den Zugriff authentifizieren müssen. Häufig wird dafür ein RADIUS Server eingesetzt. Die bestehende Lösung unseres Projektpartners besteht aus einem FreeRADIUS[1] Server und dem hauseigenen Produkt Macman[2]. Dabei empfängt der FreeRADIUS Server AAA-Requests (Schritt 1), wandelt diese in eine für Macman verständliche Form um und leitet sie anschliessend an Macman weiter (Schritt 2). Dort werden die Requests beantwortet und wieder an den FreeRADIUS Server zurückgeschickt (Schritt 3). Schlussendlich schickt der FreeRADIUS Server die Antwort an seinen Kommunikationspartner weiter (Schritt 4). In der Abbildung 1.1 wird das an einem Beispiel illustriert:

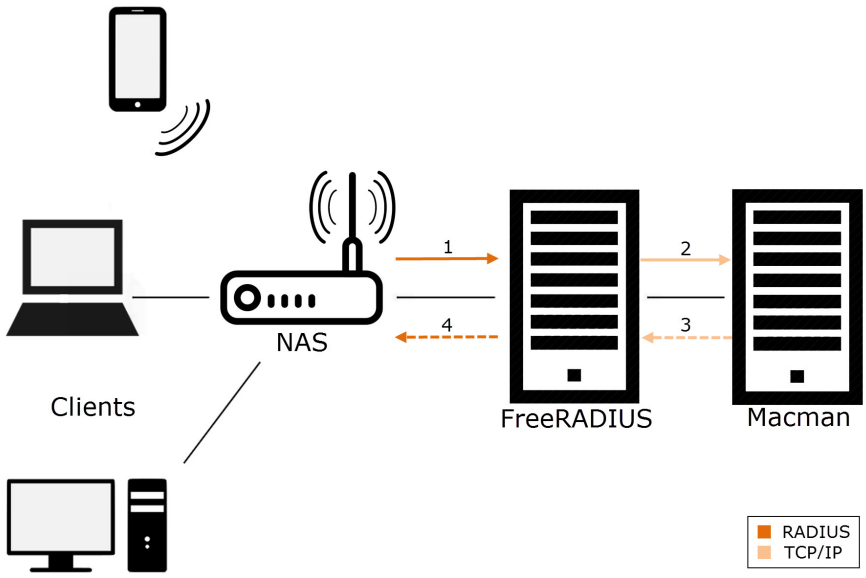


Abbildung 1.1.: Systemübersicht der bestehenden Lösung

[1] `http://freeradius.org`, letzter Zugriff 20.03.2015
[2] `http://www.cloudguard.ch/Platforms.html`, letzter Zugriff 20.03.2015

Es ist sofort ersichtlich, dass dadurch zusätzliche Kommunikation entsteht, die sich negativ auf die Performance auswirken kann. Dies wäre vermeidbar, wenn der FreeRADIUS Server die AAA-Requests selber beantworten könnte. Prinzipiell ist das natürlich möglich. Wenn aber für die Beantwortung der Requests Businesslogik nötig ist, ist es nicht mehr so trivial, denn diese lässt sich in FreeRADIUS nur schwer abbilden. Deshalb wird die Businesslogik in der bestehenden Lösung in Macman umgesetzt.

Ziel der Bachelorarbeit HiPeR - *High Performance RADIUS Server* ist es, eine Grundlage für einen robusten und performanten RADIUS Server zu bauen, welche die Möglichkeit bietet Businesslogik abzubilden. Der Projektpartner stellt die Bedingung, dass HiPeR auf Netty[3] basieren muss. Andere Vorgaben gibt es nicht.

1.3. Lösungskonzept

1.3.1. Grundidee

HiPeR soll in drei unterschiedliche Teile aufgeteilt werden. Diese Aufteilung basiert auf der Tatsache, dass es folgende drei unterschiedliche Hauptverantwortlichkeiten gibt:

1. Authentifizierung und Autorisierung
2. Accounting
3. Disconnect Message (DM) und Change-of-Authorization (CoA)

Im ersten Teil agiert HiPeR als Server. Hier werden Access-Requests von vertrauenswürdigen Kommunikationspartnern beantwortet. Dieser Server besitzt seinen eigenen Channel, der standardmässig an den Port 1812 gebunden ist. Ein Channel ist definiert als eine offene Verbindung zu einem Gerät, einer Datei, einem Netzwerk Socket, oder einem I/O fähigen Programm [8, S. 7]. Einem Channel können mehrere Handler hinzugefügt werden. Somit kann jeder Channel seine eigene, individuelle Handler-Chain haben. Dabei kann man sich einen Handler als eine Art Callback vorstellen, der aufgrund eines bestimmten Events aufgerufen wird [8, S. 14]. Events können wiederum von einkommenden Requests ausgelöst werden.

Auch im zweiten Teil gibt es einen separaten Channel, welcher wiederum seine eigene Handler-Chain besitzt. Der Channel ist allerdings an den Port 1813 gebunden, da Accounting-Requests standardmässig über diesen Port eintreffen.

Im dritten Teil agiert HiPeR als Client. In diesem Fall ist der Channel standardmässig an den Port 1700 gebunden. Mit dem Client lassen sich Disconnect- und CoA-Anfragen mithilfe der Shell versenden.

Alle drei Teile laufen nach dem Start von HiPeR gleichzeitig nebeneinander.

[3] `http://netty.io/`, letzter Zugriff 14.03.2015

1.3.2. Funktionen (Use Cases)

Es ist vorgesehen, dass HiPeR folgende Funktionen bieten wird:

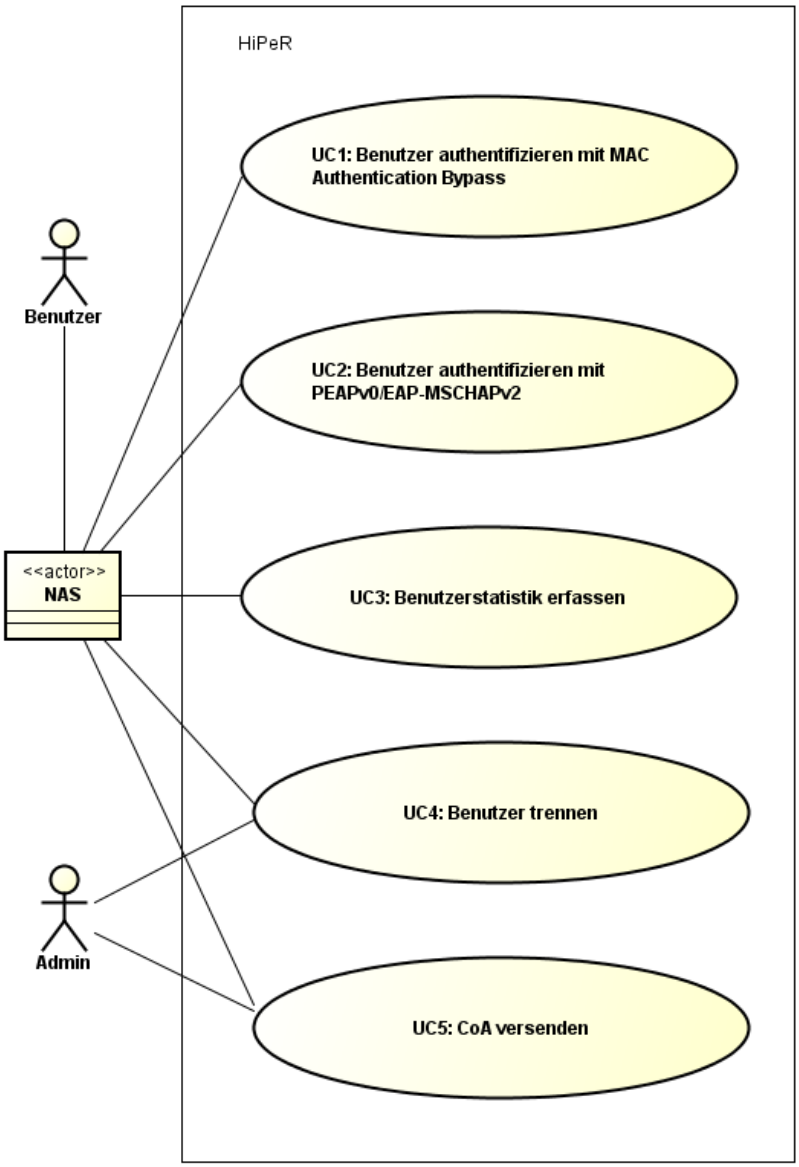


Abbildung 1.2.: Use Case Diagramm

Ein möglicher Ablauf eines Anwendungsszenarios, dass mit den definierten Use Cases umsetzbar wäre, ist folgender:

1. Der Benutzer befindet sich in einer Starbucks Filiale und möchte sich mit dem verfügbaren WLAN verbinden. Auf der Quittung seiner Bestellung stehen die benötigten Login-Daten, mit dem Vermerk, dass man damit 30 Minuten lang gratis ins Internet kann. Er wählt auf dem Handy das richtige WLAN aus, gibt die Login-Daten ein und meldet sich an (UC2).
2. Da der Benutzer nun angemeldet ist werden im Hintergrund sofort Nutzungsdaten erfasst. Dazu gehören Informationen wie: Session-ID, Anzahl eingehender/ausgehender Bytes, Anzahl eingehender/ausgehender Pakete, etc. (UC3).
3. Die Zeit ist abgelaufen. Der Benutzer muss nun sofort vom WLAN getrennt werden. HiPeR versendet deshalb einen Disconnect Request um den Benutzer zu trennen (UC4).

4. Der Benutzer wurde erfolgreich getrennt. Zum Abschluss erhält HiPeR nochmals die letzten Nutzungsdaten um diese ins Log schreiben zu können (UC3).

1.3.3. Nicht-funktionale Anforderungen (informal)

In diesem Abschnitt werden lediglich die wichtigsten nicht-funktionalen Anforderungen informal beschrieben. Eine detailliertere Beschreibung befindet sich im Kapitel 2.3.5.

- HiPeR muss in der Lage sein, mit verschiedenen Kommunikationspartnern zu kommunizieren die als NAS fungieren. Dabei müssen die Attribute, welche im Kapitel 2.3.5 aufgeführt sind, korrekt interpretiert werden können. Diese Anforderung wird mittels einer PoC Box, Unit-Tests und RADIUS Client-Simulatoren überprüft.
- HiPeR darf Anmeldungen mit ungültigen Zugangsdaten nicht zulassen.
- Um Brute-Force-Angriffe zu verhindern darf sich ein Benutzer nur eine bestimmte Anzahl an fehlgeschlagener Anmeldungen leisten, danach wird er geblockt. Anzahl und Dauer soll konfigurierbar sein.
- Beim Empfang ungültiger Datenpakete muss HiPeR so fehlertolerant sein, dass diese kein Fehlverhalten provozieren können. Datenpakete werden je nach Situation entweder ignoriert oder mit einem Access-Reject quittiert.
- Duplicate Requests sollen geloggt und beantwortet werden.
- Die geforderte Leistung beträgt 10'000 Authentifizierungen pro Sekunde mit MAB und 50 Authentifizierungen pro Sekunde mit PEAPv0/EAP-MSCHAPv2.
- Der Arbeitsspeicherverbrauch darf 300 MByte nicht übersteigen. Trotz dieser Obergrenze muss HiPeR dennoch in der Lage sein die definierte Maximalleistung zu erbringen.
- Die Architektur von HiPeR muss klar und gut strukturiert sein, damit Änderungen mit geringem Aufwand möglich sind. Interfaces und eine API-Dokumentation sollen dies unterstützen.

1.4. Umsetzung

1.4.1. Layering

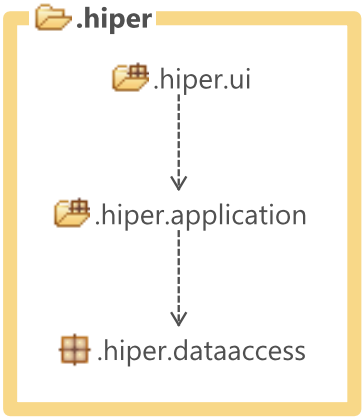


Abbildung 1.3.: Layer Übersicht

HiPeR ist in die 3 Layer UI, Application und Data Access aufgeteilt. Zugriffe erfolgen jeweils nur auf den direkt darunterliegenden Layer.

Im UI Layer befindet sich die Implementation der Shell. Sie wird für die Bedienung der Applikation verwendet. Unter anderem kann damit das Logging Level angepasst werden oder Benutzer vom Netzwerk getrennt werden.

Ein Grossteil der Logik wird vom Application Layer übernommen. Alle Klassen welche die verschiedenen Pakettypen und Attribute repräsentieren befinden sich in diesem Layer. Ebenso sind hier sämtliche Handler Klassen enthalten. Diese bearbeiten eingehende Pakete und implementieren das RADIUS Protokoll. Alle Handler erfüllen eine bestimmte Aufgabe und werden von Anfragen schrittweise durchlaufen. Zwischen den Handlern bestehen keine direkten Abhängigkeiten, wodurch sie einfach ausgetauscht oder erweitert werden können.

Der Data Access Layer enthält Interfaces und Klassen welche den Zugriff auf alle benötigten Daten ermöglichen. Dies sind vor allem Benutzer- und Accounting Daten, aber auch Informationen zu den bekannten Network Access Servern.

1.4.2. Design Prinzipien

Bei der Umsetzung wurde darauf geachtet, dass verschiedene Design Prinzipien eingehalten werden um die gestellten Qualitätsansprüche zu erfüllen. Die Aufteilung der Logik auf verschiedene Handler trägt dazu bei, dass das Prinzip „Separation of Concerns“ eingehalten wird. Mit der gewählten Schichtenarchitektur, der top-down Kommunikation und dem strictly-layered Ansatz wird zusätzlich auch das Design Prinzip der losen Kopplung erfüllt. Ein weiteres Design Prinzip welches man befolgt, ist das Prinzip der „Extensibility“. Neue Attribute können durch das Implementieren der passenden Attributsklasse und durch das Hinzufügen der Attributinformationen in die vorgesehene Dictionary Datei, unterstützt werden. Die Einhaltung aller o.g. Design Prinzipien, begünstigen schlussendlich auch die Wartbarkeit von HiPeR.

1.4.3. Tests

Für die Sicherstellung der funktionalen Anforderungen wurden verschiedene Unit-Tests geschrieben und Integrationstests durchgeführt. Es wurde darauf geachtet, dass vor allem das Handler Package im Application-Layer, wo die eigentliche Businesslogik angesiedelt ist, getestet wird. Die Code Coverage des Handler Packages beträgt zum Zeitpunkt der Abgabe:

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
ch.cloudguard.hiper.application.handler	M: 685 C: 3349 83%	M: 103 C: 188 65%	M: 88 C: 151 63%	M: 150 C: 725 83%	M: 5 C: 81 94%	M: 1 C: 20 95%

Abbildung 1.4.: Code Coverage Handler Package

Die Code Coverage des Application-Layers und des Data Access-Layers beträgt hingegen:

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
ch.cloudguard.hiper.application	M: 944 C: 888 48%	M: 36 C: 10 22%	M: 48 C: 22 31%	M: 187 C: 128 41%	M: 28 C: 19 40%	M: 5 C: 6 55%

Abbildung 1.5.: Code Coverage Application Layer

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
ch.cloudguard.hiper.dataaccess	M: 476 C: 621 57%	M: 45 C: 11 20%	M: 46 C: 32 41%	M: 106 C: 67 39%	M: 21 C: 29 58%	M: 2 C: 18 90%

Abbildung 1.6.: Code Coverage Data Access Layer

Der UI-Layer konnte aufgrund der tieferen Priorität und der fehlenden Zeit leider nicht mehr durch Unit-Tests abgedeckt werden.

1.5. Ergebnisdiskussion & Ausblick

1.5.1. Ergebnisdiskussion

Die entstandene Applikation erfüllt die Anforderungen an die Performance vollständig. Bei Messungen wurden bis zu 99'000 Authentifizierungen pro Sekunde mit MAB und 87 Authentifizierungen pro Sekunde mit PEAPv0/EAP-MSCHAPv2 erreicht.

Von den funktionalen Anforderungen konnten die Use Cases "Benutzer authentifizieren mit MAB", "Benutzerstatistik erfassen" (Accounting) und "Benutzer trennen" (Disconnect Message) wie geplant umgesetzt werden. Die Use Cases "Benutzer authentifizieren mit PEAPv0/EAP-MSCHAPv2" und "CoA versenden" wurden jedoch nur teilweise umgesetzt.

Für die Authentifizierug mit PEAPv0/EAP-MSCHAPv2 waren ursprünglich zwei Wochen eingeplant. Tatsächlich aufgewendet wurden allerdings 6 Wochen. Dennoch konnte der Use Case nicht vollständig implementiert werden. Windows Clients können nicht authentifiziert werden. Die erste Phase der Authentifizierung verläuft noch erfolgreich. Der Handshake wird abgeschlossen und der TLS Tunnel aufgebaut. Sobald aber das erste Paket durch den Tunnel verschickt wird, gibt der Client keine Antwort mehr und die Authentifizierung bleibt stehen. Es wurde versucht anhand der Windows Ereignisanzeige und verschiedenen Traces welche mit dem Tool Netsh erstellt wurden die Ursache dafür zu finden. Dies ist leider nicht gelungen und aus zeitlichen Gründen wurde die weitere Fehlersuche eingestellt.

Für den Use Case "CoA versenden" war ursprünglich vorgesehen beim Versenden eines Requests den Namen einer ACL mitzuschicken. Dies wurde mit dem herstellerspezifischen Attribut "Airespace ACL-Name" versucht. Der NAS akzeptiert dieses Attribut in einem CoA-Request jedoch nicht. Gemäss der Dokumentation von Cisco kann es allerdings in einem Access-Request verwendet werden. Eine Möglichkeit wäre also durch den CoA-Request eine Reauthentifizierung auszulösen und anschliessend den ACL-Namen beim Access-Request mitzuschicken. In der verbleibenden Zeit, hätte diese Lösung aber nicht mehr unseren Ansprüchen entsprechend implementiert werden können. Aus diesem Grund haben wir uns gegen diese schnelle Lösung entschieden und die Umsetzung des Use Cases abgebrochen.

Obwohl die Applikation die gestellten Anforderungen nicht vollständig erfüllt, ist dennoch eine solide Basis für einen RADIUS Server entstanden. Die anfänglich ausgearbeitete Architektur hat sich über die ganze Projektlaufzeit bewährt und musste nie angepasst werden. Alle Arbeitsschritte und Attribute die im Verlauf der Entwicklung erstellt wurden, konnten unseren ursprünglichen Vorstellungen entsprechend integriert werden.

1.5.2. Ausblick

Da die Implementation der Use Cases deutlich mehr Zeit in Anspruch genommen hat als angenommen, konnten nicht so viele Unit-Tests geschrieben werden wie geplant. Deshalb wäre es empfehlenswert, in einem ersten Schritt die Testabdeckung zu erhöhen.

Desweiteren gibt es noch einige Stellen im Source Code welche verbessert werden können. Diese sind im Code mit "TODO" markiert.

Anschliessend wäre es natürlich sinnvoll, die unvollständigen Use Cases abzuschliessen. Bei PEAP muss dabei auch noch die Möglichkeit zu Fragmentierung von Paketen erarbeitet werden. Falls für den TLS Tunnel ein grosser Schlüssel verwendet wird, kann es sein dass dieser nicht in einem einzelnen RADIUS Paket übertragen werden kann.

Teil II.

Software Engineering Dokumente

2. Anforderungsspezifikation

2.1. Ist-Szenario

Hans Keller ist am Flughafen und möchte vor dem Abflug noch seine E-Mails abrufen. Deshalb verbindet er sein Smartphone mit dem WLAN. Er wird dabei aufgefordert, seinen Benutzernamen und sein Passwort einzugeben. Diese Daten werden vom Wireless Access Point an einen FreeRADIUS Server weitergeleitet, welcher die Anfrage umwandelt und sie weiter zu Macman sendet. Dort werden Benutzername und Password überprüft, sowie ermittelt welche Rechte der Benutzer hat. Die Antwort wird via FreeRADIUS Server zurück an den Access Point gesendet. Hans Keller hat nun Zugriff auf das Netzwerk. Allerdings musste er kurz warten, bis er autorisiert war, weil die Pakete nicht direkt vom Access Point zu Macman gesendet werden können, sondern den Umweg über einen FreeRADIUS Server nehmen müssen. Dies wirkt sich negativ auf die Performance aus und hat vor allem dann einen Einfluss, wenn sehr viele Anfragen gleichzeitig kommen und Verschlüsselungsprotokolle verwendet werden, welche ein Handshake Verfahren anwenden.

2.2. Soll-Szenario

Hans Keller ist am Flughafen und möchte online einen Last Minute Flug buchen. Dazu versucht er seinen Laptop mit dem WLAN zu verbinden und wird dabei aufgefordert, Benutzername und Passwort einzugeben. Der Wireless Access Point leitet diese Daten direkt an HiPeR weiter. Dort wird überprüft, ob der Benutzername und das Passwort zu einem gültigen Account gehören und welche Zugriffsrechte damit verbunden sind. Die Antwort wird zurück an den Access Point gesendet. Hans Keller hat nun ohne Verzögerung Zugriff auf das Netzwerk.

2.3. Use Cases

2.3.1. Use Case Diagramm

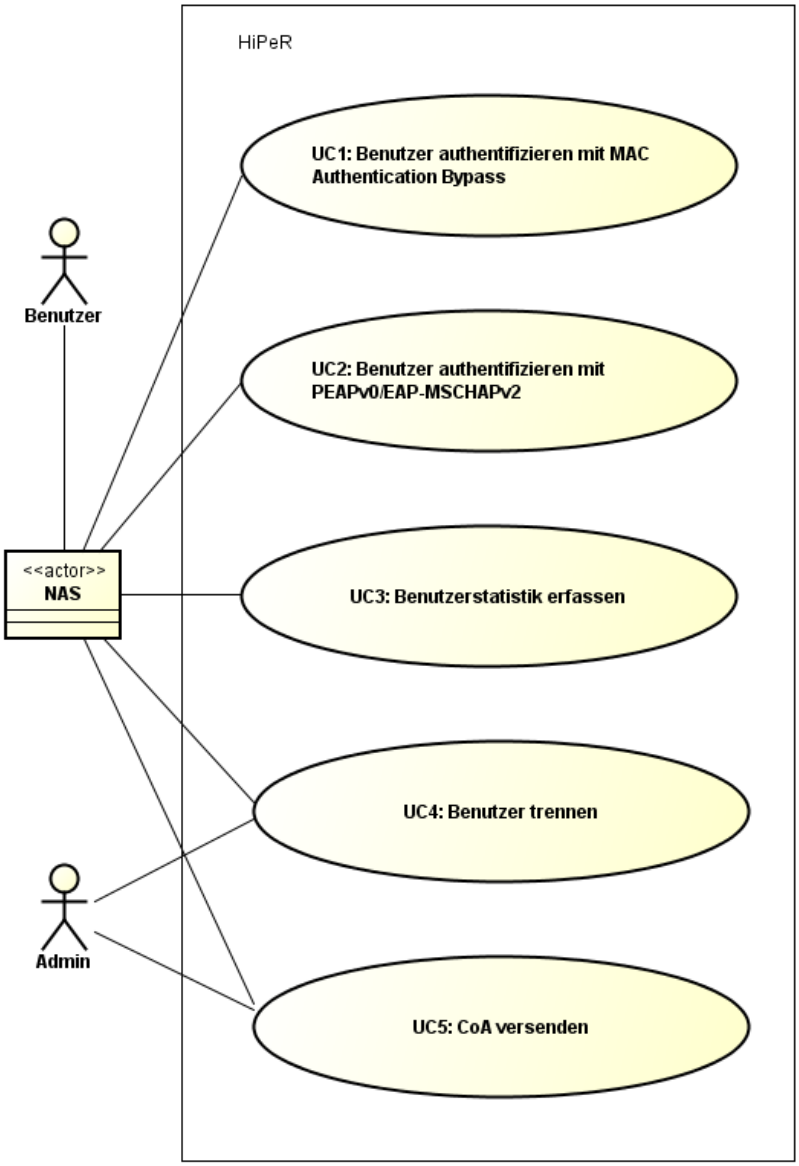


Abbildung 2.1.: Use Case Diagramm

2.3.2. Aktoren

Benutzer (Aktor)

Der Benutzer möchte in der Lage sein, sich mit dem Netzwerk zu verbinden und verschiedene Dienste in Anspruch zu nehmen. Er interagiert nicht direkt mit HiPeR, sondern mit dem Network Access Server (NAS). Der Benutzer muss nicht zwingend wissen, dass HiPeR überhaupt existiert.

Admin (Aktor)

Der Admin ist für die Konfiguration von HiPeR zuständig. Er möchte Benutzer abmelden und deren Rechte bearbeiten können.

NAS (technischer Aktor)

Der Network Access Server (NAS) ist der wichtigste Kommunikationspartner von HiPeR. Wenn sich ein Benutzer mit dem Netzwerk verbinden möchte, sendet er Anfragen an HiPeR, welche für Authentifizierung, Authorisierung und Accounting verwendet werden.

2.3.3. Beschreibungen (Brief)

UC1 Benutzer authentifizieren mit MAC Authentication Bypass

Der Benutzer versucht sein Gerät mit dem Netzwerk zu verbinden. Der Network Access Server sendet darauf eine Anfrage an HiPeR um zu überprüfen ob die MAC-Adresse des Gerätes bekannt ist. HiPeR bearbeitet die Anfrage und teilt dem NAS mit, ob sich der Benutzer verbinden darf.

UC2 Benutzer authentifizieren mit PEAPv0/EAP-MSCHAPv2

Der Benutzer versucht sich mit dem Netzwerk zu verbinden und wird vom NAS aufgefordert, Benutzername und Passwort einzugeben. Diese Daten werden vom NAS an HiPeR weitergeleitet. HiPeR bearbeitet die Anfrage und teilt dem NAS mit, ob sich der Benutzer verbinden darf.

UC3 Benutzerstatistik erfassen

Der NAS sendet Nutzungsdaten an HiPeR, womit unter anderem überprüft werden kann, wie lange der Benutzer verbunden war und wie viele Daten er übertragen hat. Dies geschieht einerseits, wenn sich der Benutzer mit dem Netzwerk verbindet oder sich davon trennt, andererseits auch zwischendurch, z. B. beim Roaming wenn der NAS gewechselt wird.

UC4 Benutzer trennen

Eine bestehende Verbindung eines Benutzer mit dem Netzwerk kann von HiPeR durch das senden eines Befehls an den NAS getrennt werden.

UC5 CoA versenden

HiPeR kann durch das Senden eines CoA-Requests an den NAS eine Benutzersession bearbeiten. Dabei wird dem Benutzer eine Access Control List (ACL) zugeteilt, welche seine Rechte im Netz bestimmt.

Es wäre auch möglich weitere Einstellungen anzupassen wie z. B. das Quality of Service Level oder die Bandbreite welche dem Benutzer zur Verfügung gestellt wird. Im Rahmen dieser Arbeit wird jedoch nur der Name einer ACL übertragen, welche auf den Benutzer angewandt werden soll.

2.3.4. Rolle von HiPeR

HiPeR kann zwei verschiedene Rollen übernehmen. In den Use Cases UC1-UC3 agiert HiPeR als RADIUS-Server. In dieser Rolle beantwortet er Requests bezüglich der Authentifizierung, Authorisierung und des Accountings eines Benutzers. In den Use Cases UC4 und UC5 hingegen, fungiert HiPeR als Client. In dieser Rolle sendet HiPeR Disconnect- und CoA-Requests an ein NAS um eine bestehende Verbindung eines bestimmten Benutzers zu trennen oder dessen Rechte zu verändern.

2.3.5. RADIUS Attribute

Für die Implementierung der oben aufgeführten Use Cases müssen folgende RADIUS Attribute unterstützt werden:

Typ:	1
Name:	User-Name
Beschreibung:	Name des Benutzers / MAC-Adresse
Datentyp:	String
RFC:	2865
Use Case:	1, 2, 3, 4, 5
Typ:	2
Name:	User-Password
Beschreibung:	Passwort des Benutzers / MAC-Adresse
Datentyp:	String
RFC:	2865
Use Case:	1
Typ:	4
Name:	NAS-IP-Address
Beschreibung:	IP Adresse des NAS
Datentyp:	Adresse
RFC:	2865
Use Case:	1, 2, 3
Typ:	24
Name:	State
Beschreibung:	Status einer Access-Challenge oder Access-Accept mit Termination-Action Attribut
Datentyp:	Oktette
RFC:	2865
Use Case:	2
Typ:	30
Name:	Called-Station-Id
Beschreibung:	IP-Adresse des NAS
Datentyp:	String
RFC:	2865
Use Case:	1, 2, 3
Typ:	31
Name:	Calling-Station-Id
Beschreibung:	MAC-Adresse des Benutzers
Datentyp:	String
RFC:	2865
Use Case:	1, 2, 3, 4, 5

Typ:	40
Name:	Acct-Status-Type
Beschreibung:	Unterscheidung Start / Stop Accounting
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	41
Name:	Acct-Delay-Time
Beschreibung:	Zeit in Sekunden in welcher der Client versucht hat das Paket zu senden
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	42
Name:	Acct-Input-Octets
Beschreibung:	Anzahl empfangene Oktette (Byte)
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	43
Name:	Acct-Output-Octets
Beschreibung:	Anzahl gesendete Oktette (Byte)
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	44
Name:	Acct-Session-Id
Beschreibung:	Eindeutige Accounting ID
Datentyp:	Text
RFC:	2866
Use Case:	3, 4, 5

Typ:	45
Name:	Acct-Authentic
Beschreibung:	Art der Authentifizierung
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	46
Name:	Acct-Session-Time
Beschreibung:	Dauer der Session in Sekunden
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	47
Name:	Acct-Input-Packets
Beschreibung:	Anzahl empfangener Pakete
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	48
Name:	Acct-Output-Packets
Beschreibung:	Anzahl gesendeter Pakete
Datentyp:	Integer
RFC:	2866
Use Case:	3

Typ:	64
Name:	Tunnel-Type
Beschreibung:	Zu verwendendes Tunneling Protokoll
RFC:	2868
Datentyp:	Integer (1 Byte), Integer (3 Byte)
Use Case:	1, 2, 3

Typ:	65
Name:	Tunnel-Medium-Type
Beschreibung:	Transportmedium für den Tunnel
Datentyp:	Integer (1 Byte), Integer (3 Byte)
RFC:	2868
Use Case:	1, 2, 3

Typ:	79
Name:	EAP-Message
Beschreibung:	EAP Paket
RFC:	2869, 2759
Datentyp:	Oktette
Use Case:	2

Typ:	80
Name:	Message-Authenticator
Beschreibung:	Signiert Pakete
Datentyp:	Oktette
RFC:	2869
Use Case:	2

Typ:	81
Name:	Tunnel-Private-Group-Id
Beschreibung:	Group ID einer Tunnel-Session (z. B. VLAN ID)
Datentyp:	Integer (1 Byte), String
RFC:	2868
Use Case:	1, 2, 3

2.4. Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen legen fest, welche Eigenschaften ein Produkt hat. Diese Eigenschaften tragen entscheidend zur Anwendbarkeit eines Systems bei. Folgende nicht-funktionalen Anforderungen sollen erfüllt werden:

2.4.1. Funktionalität

Interoperabilität

HiPeR ist fähig mit Kommunikationspartnern zu operieren die die Rolle eines NAS übernehmen. Die Interoperabilität von HiPeR beschränkt sich dabei auf die im Kapitel 2.3.5 aufgeführten Attribute und RFCs.

Die Interoperabilität von HiPeR wird mittels einer PoC Box, Unit-Tests und RADIUS Client Simulatoren überprüft.

Sicherheit

Bei der Authentifizierung über MAB ist die Kommunikation zwischen HiPeR und seinen Kommunikationspartnern grundsätzlich unverschlüsselt. Die MAC-
Adresse im Feld User-Name ist somit für alle ersichtlich. Das Passwort hingegen, wird entsprechend der im RFC 2865 unter Kapitel 5.2. beschriebenen Methode versteckt. Obwohl diese Methode nicht mehr als sicher betrachtet wird, ist die Authentifizierung über MAB dennoch notwendig um Geräte wie Drucker, Scanner, etc. in eine IEEE 802.1X Umgebung zu integrieren. Es bleibt aber u.a. das Risiko, dass ein Angreifer eine authorisierte MAC-Adresse annimmt und sich so mit dem zuvor errechneten Secret, unerlaubten Zugriff ins Zielnetz gewährt.

Bei der Authentifizierung über PEAPv0/EAP-MSCHAPv2 ist der Benutzername ebenfalls für alle ersichtlich. Allerdings wird hier das Passwort in einer ersten Phase durch ein TLSv1-Tunnel verschlüsselt übertragen. Die Authentifizierung findet dabei zusätzlich mit MSCHAPv2 statt - ein Protokoll zur gegenseitigen Authentifizierung mithilfe unidirektional verschlüsselter Passwörter.

Folgende sicherheitsrelevanten Punkte sollen durch Unit-Tests abgedeckt werden:

- Anmeldung mit ungültigen Zugangsdaten darf nicht möglich sein.
- Zur Verhinderung von Brute-Force-Angriffen sollen Benutzer welche mehrmals versucht haben sich mit falschem Passwort anzumelden, vorüberehend gesperrt werden. Die Dauer und Anzahl Versuche soll konfigurierbar sein.
- Alle in Kapitel 2.4.2 aufgeführten Punkte bezüglich Fehlertoleranz sollen überprüft werden.

Konformität

HiPeR ist im Bezug auf den Implementationsumfang nicht vollständig konform mit den im Kapitel 2.3.5 aufgeführten RFCs. Die für HiPeR irrelevanten Aspekte werden demzufolge nicht implementiert. Attribute die allerdings von HiPeR unterstützt werden sollen, entsprechen vollumfänglich den o.g. RFCs. Dies wird durch Unit-Tests, Integrations-Tests, sowie RADIUS Client Simulatoren überprüft.

2.4.2. Zuverlässigkeit

Fehlertoleranz

HiPeR ist soweit fehlertolerant, dass ungültige Datenpakete kein Fehlverhalten provozieren können. Folgende Requests werden von HiPeR ignoriert (silently discarded):

- Requests welche von einem nicht vertrauenswürdigen Absender kommen.
- Requests welche ein falsches Secret verwenden.
- Requests mit einem ungültigen Code-Feld.
- Requests mit einer kleineren Paketlänge, als das jeweilige Längenfeld propagiert.
- Ein Access-Request, mit einem EAP-Message Attribut, aber ohne Message-Authenticator Attribut.
- Ein Access-Request, mit einem EAP-Message Attribut und einem ungültigen Message-Authenticator Attribut.
- Ein Accounting-Request, das eine ungültige Längenangabe für ein bestimmtes Attribut hat.

In folgenden Fällen werden Requests von HiPeR mit einem Access-Reject quittiert:

- Nach einem Access-Request, das einen fehlenden oder ungültigen Attributwert hat.
- Nach einem Access-Request, das eine ungültige Längenangabe für ein bestimmtes Attribut hat.
- Nach einem Access-Request, das in seinem Tunnel-Type Attribut ein für die Nutzung unautorisiertes Tunneling-Protokoll aufführt.
- Nach einem Access-Request, welches ein Authentifizierungsprotokoll verlangt, dass nicht unterstützt wird.
- Nach mehrfach fehlgeschlagenem Access-Challenge. Die Anzahl ist konfigurierbar.
- Nach einem Access-Request an HiPeR, welches ein EAP-Message Attribute enthält, dass HiPeR nicht korrekt interpretieren kann.

Identische Requests welche vom NAS mehrfach gesendet werden (Duplicate Requests) werden geloggt und beantwortet.

Wiederherstellbarkeit

Durch einen Absturz können Accounting-Daten von Benutzern welche zu diesem Zeitpunkt verbunden sind, sowie Pakete welche gerade verarbeitet werden verloren gehen. Die verbundenen Geräte bleiben zwar autorisiert und werden nicht vom Netz getrennt, es ist aber nicht mehr möglich einen Disconnect- oder CoA-Request an sie zu versenden.

Im Rahmen dieses Projektes ist es nicht vorgesehen, verlorene Daten wiederherstellen zu können.

2.4.3. Benutzbarkeit

Verständlichkeit

HiPeR richtet sich nicht an gewöhnliche Endbenutzer, sondern an Fachpersonen, wie zum Beispiel Netzwerkadministratoren. Folglich wird ein gewisses Mass an Fachwissen über RADIUS für die Bedienung und Konfiguration von HiPeR, vorausgesetzt. Dennoch soll die Interaktion mit HiPeR aus Sicht einer Fachperson möglichst einfach sein. Die Dokumentation der einzelnen Konfigurations- und Bedienmöglichkeiten soll dies sicherstellen.

Bedienbarkeit

Da die Entwicklung eines GUIs nicht im Vordergrund steht, wird HiPeR lediglich mittels CLI bedient und mithilfe von Konfigurationsdateien individuell angepasst. Der Aufwand ist somit leicht höher als bei einer Bedienung mittels GUI, hat allerdings den Vorteil, dass gewisse Arbeitsschritte automatisiert werden können.

Attraktivität

Der Fokus bei der Entwicklung von HiPeR liegt nicht auf der Attraktivität, sondern auf der Performance und Robustheit. Die Bedienungsart, welche in der nicht-funktionalen Anforderung "Bedienbarkeit" beschrieben ist, hat aus Sicht eines Netzwerkadministrators keine hohe Attraktivität. Die Bedienungsart erfüllt aber ihren Zweck und ist somit für die Arbeit vollkommen ausreichend.

2.4.4. Effizienz

Zeitverhalten

Die Übertragung des Passworts bei PEAPv0/EAP-MSCHAPv2 erfordert die Erstellung eines sicheren Tunnels wozu mehrere Pakete zwischen dem NAS und HiPeR ausgetauscht werden müssen. Zur Effizienzsteigerung soll deshalb zuerst überprüft werden, ob der Benutzername überhaupt bekannt ist, bevor der Tunnel aufgebaut wird.

Im Rahmen dieser Arbeit werden keine Anforderungen an die Speicherung von Benutzerdaten gestellt. Falls diese aus einer Datenbank oder einem LDAP-Server gelesen werden müssen, würde dies den grössten Bottleneck bezüglich Performance darstellen. In-Memory Caching wäre als effizienzsteigernde Massnahme denkbar.

HiPeR ist in der Lage die folgende Durchsatzraten zu erbringen:

- MAB: 10'000 Authentifizierungen pro Sekunde
- PEAPv0/EAP-MSCHAPv2: 50 Authentifizierungen pro Sekunde

Verbrauchsverhalten

Der Arbeitsspeicherverbrauch darf 300 MByte nicht übersteigen. Trotz dieser Obergrenze muss HiPeR dennoch in der Lage sein die im Abschnitt Verbrauchsverhalten definierten Durchsatzraten zu erbringen.

2.4.5. Modifizierbarkeit

HiPeR muss eine klare, gut strukturierte Architektur aufweisen, damit Änderungen, wie zum Beispiel das Implementieren eines neuen RADIUS-Attributes, mit geringem Aufwand möglich sind. Unterstützend wirken dabei vor allem Interfaces die implementiert werden müssen in Kombination mit einer guten API-Dokumentation.

2.4.6. Übertragbarkeit

Anpassbarkeit

HiPeR basiert auf Netty[1] und somit auch auf Java. Es kann folglich auf jeder Plattform (Windows, Linux, etc.) ausgeführt werden, für die es eine Java SE Runtime Environment 8 (JRE) gibt. Damit HiPeR unter Linux epoll verwendet, müssen vor der Ausführung minimale Anpassungen gemäss folgender Anleitung durchgeführt werden:
`http://netty.io/wiki/native-transports.html`

Installierbarkeit

HiPeR wird als ZIP-Datei ausgeliefert und muss nicht installiert, sondern lediglich entpackt werden. Zur Ausführung benötigt man aber eine Java SE Runtime Environment 8 (JRE).

Koexistenz

HiPeR ist fähig neben anderen Systemen mit ähnlichen oder gleichen Funktionen zu arbeiten. Einzig die Ausführung mehrerer RADIUS-Server auf der gleichen Maschine ist nicht möglich, sofern sie nicht auf unterschiedlichen Ports erreichbar sind.

Austauschbarkeit

HiPeR kann anstelle eines anderen, bereits im Einsatz befindlichen, RADIUS-Server verwendet werden. Es muss aber berücksichtigt werden, dass HiPeR den zu ersetzenden RADIUS-Server nur hinsichtlich der im Kapitel 2.3.5 aufgeführten Attribute und RFCs, ersetzen kann. Der für den Austausch benötigte Aufwand beinhaltet folgende Arbeitsschritte:

- Die Abschaltung des zu ersetzenden RADIUS-Server.
- Die Übertragung der HiPeR JAR-Datei auf den gewünschten Rechner.
- Das Eintragen der IP-Adresse und allfälligen Portnummern in die Konfigurationsdatei von HiPeR, unter welcher HiPeR für die Umgebung erreichbar sein soll.
- Das Eintragen der Daten unter denen HiPeR erreichbar ist, bei Kommunikationspartnern von HiPeR.
- Das Aufstarten von HiPeR.

Neben den o.g. Arbeitsschritten, kann evtl. noch eine zusätzliche, individuelle Anpassung an eine spezielle Umgebung vonnöten sein.

[1] `http://netty.io/`, letzter Zugriff 14.03.2015

3. Domainanalyse

3.1. Domain Modell

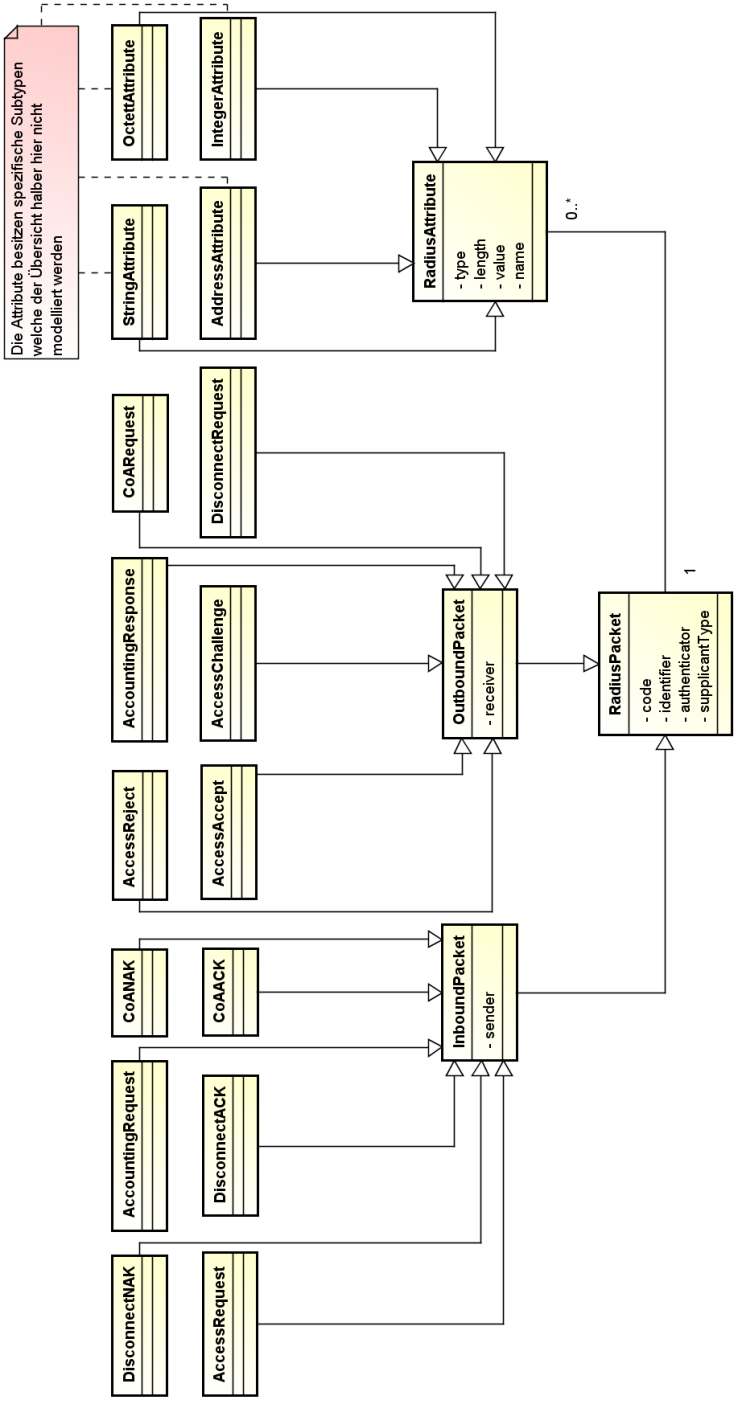


Abbildung 3.1.: Domain Modell

3.1.1. Wichtige Konzepte

Einteilung der Pakettypen

Im Domain Modell werden die Pakettypen in zwei Hauptkategorien unterteilt: InboundPacket und Outboundpacket. Dies geschieht, weil es für die Bearbeitung der Pakete von zentraler Bedeutung ist, ob sie empfangen oder versendet werden.

Grundsätzlich wäre eine Einteilung in RequestPacket und ReplyPacket auch denkbar. Allerdings kann HiPeR sowohl als Client wie auch als Server auftreten. Ein AccessRequest beispielsweise ist ein eingehendes Paket, da HiPeR in diesem Fall als Server agiert. Ein CoARequest ist jedoch ein ausgehendes Paket da HiPeR als Client auftritt. Um Verwechslungen wegen dieser Doppeldeutigkeit zu vermeiden, haben wir uns für die Unterteilung in Inbound- und OutboundPacket entschieden.

Attributtypen

Die Attribute werden (abhängig vom Datentyp den sie enthalten) in verschiedene Subtypen unterteilt (StringAttribute, IntegerAttribute, etc.). Diese besitzen jeweils wieder spezifische Subtypen. Allerdings gibt es eine grosse Anzahl dieser Typen, weshalb sie der Übersicht halber im Domain Modell nicht dargestellt werden. Zwei Beispiele sind UserName und UserPassword, welche Subtypen von StringAttribute sind.

Lebensdauer von Domainobjekten

Die Lebensdauer der Domainobjekte ist beschränkt. Sie existieren lediglich für die Dauer der Bearbeitung einer Anfrage.

3.2. Systemsequenzdiagramme (SSD) & Contracts

Die Use Cases werden nachfolgend als Systemsequenzdiagramme illustriert und die jeweiligen Systemoperationen in Form eines Contracts beschrieben. Es ist wichtig anzumerken, dass die Kommunikation aus Sicht von HiPeR nur zwischen dem NAS und HiPeR stattfindet. Der Benutzer ist in einigen SSDs nur deshalb eingezeichnet, weil er dort als Trigger fungiert

3.2.1. MAB

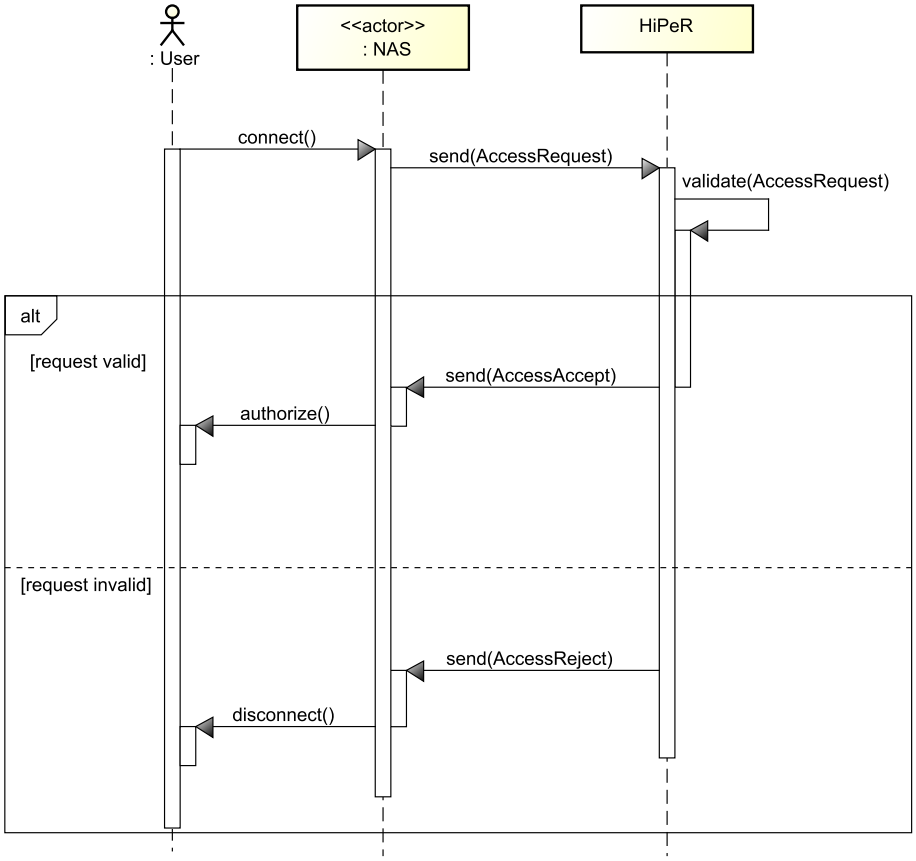


Abbildung 3.2.: Sequenzdiagramm MAB

Beschreibung: Der Benutzer versucht sich mit dem NAS zu verbinden. Darauf sendet der NAS ein AcessRequest Paket welches die MAC-Adresse des Benutzers enthält. HiPeR überprüft die Adresse und sendet ein AccessAccept oder AccessReject.

Querverweis: UC1 Benutzer authentifizieren mit MAC Authentication Bypass

Vorbedingung:

- NAS ist in der Liste der vertrauenswürdigen Absender

Nachbedingung:

- AccessAccept oder AccessReject versendet
- Benutzer ist authentifiziert (nach einem AccessAccept)

3.2.2. PEAPv0/EAP-MSCHAPv2

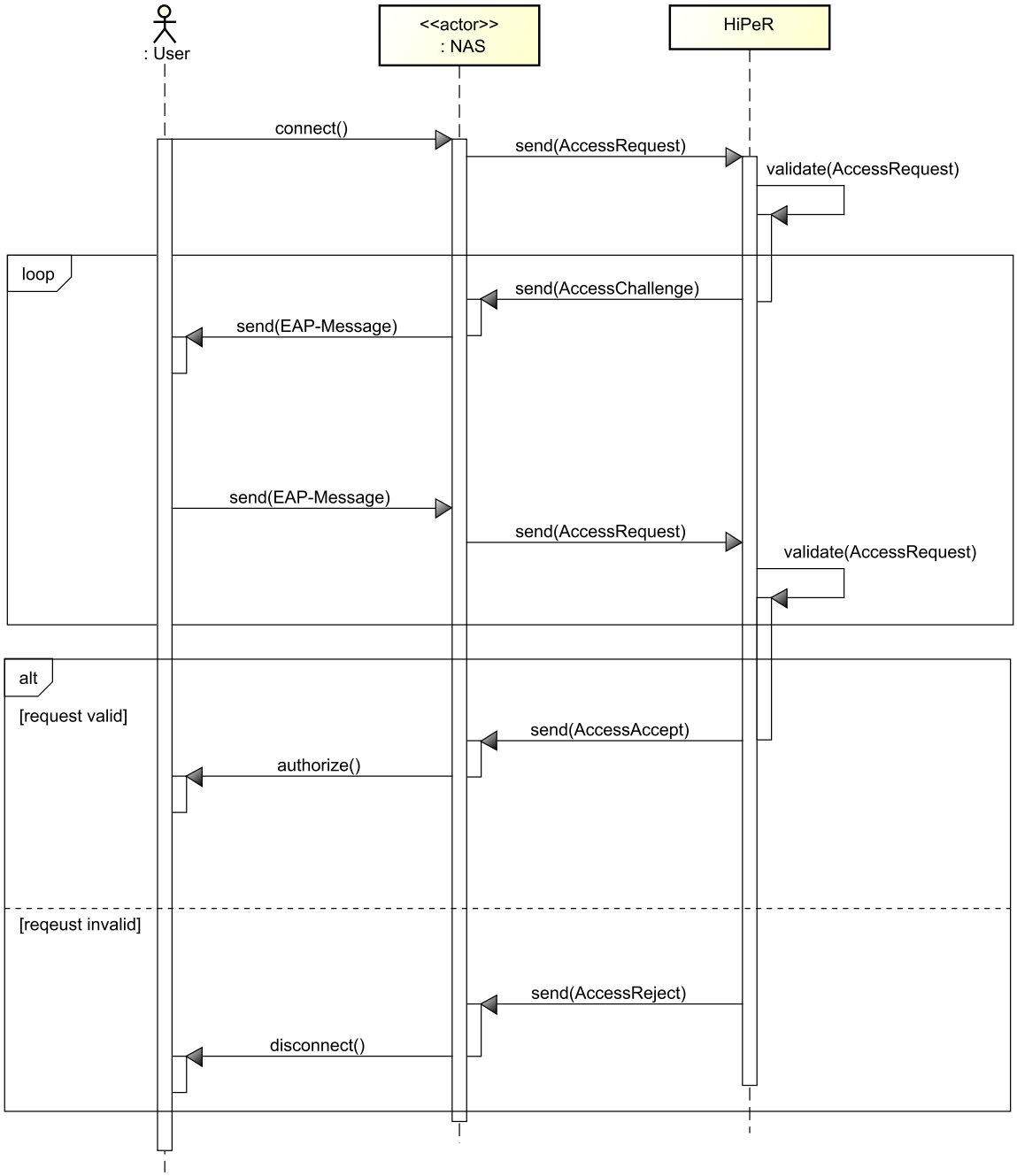


Abbildung 3.3.: Sequenzdiagramm PEAPv0/EAP-MSCHAPv2

Beschreibung: Der Benutzer versucht sich mit dem NAS zu verbinden. Darauf werden eine Reihe von AccessRequest und AccessChallenge Paketen zwischen dem NAS und HiPeR ausgetauscht. Diese Pakete enthalten jeweils eine EAP-Message welche bis zum Benutzer weitergeleitet wird. Darin wird ein TLSv1-Tunnel aufgebaut und das Passwort mit MSCHAPv2 übertragen. Wenn die EAP Übertragung abgeschlossen ist und Benutzername und Passwort überprüft wurden, sendet HiPeR entweder ein AccessAccept oder ein AccessReject.

Querverweis: UC2 Benutzer authentifizieren mit PEAPv0/EAP-MSCHAPv2

Vorbedingung:

- NAS ist in der Liste der vertrauenswürdigen Absender

Nachbedingung:

- AccessAccept oder AccessReject versendet
- Benutzer ist authentifiziert (nach einem AccessAccept)

3.2.3. Accounting

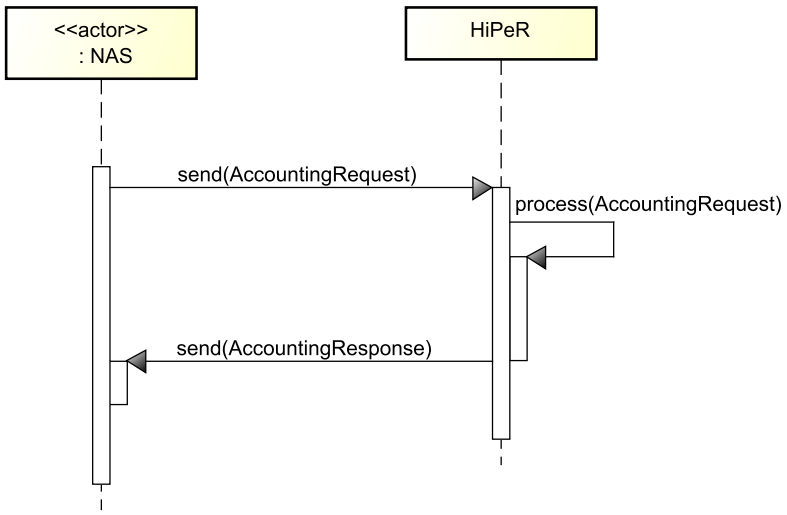


Abbildung 3.4.: Sequenzdiagramm Accounting

Beschreibung: Damit überprüft werden kann, wie lange ein Benutzer verbunden war und wie viele Daten er übertragen hat, schickt der NAS jeweils ein AccountingRequest an HiPeR. HiPeR verarbeitet den AccountingRequest, sofern die Anfrage von einem vertrauenswürdigen NAS kommt. Ist dies der Fall, sendet HiPeR ein AccountingResponse an den NAS zurück. Diese Prozedur geschieht am Anfang, wenn sich der Benutzer neu am Netz anmeldet, zwischendurch bspw. bei einem Roaming und am Ende, wenn sich ein Benutzer vom Netz wieder trennt.

Querverweis: UC3 Benutzerstatistik erfassen

Vorbedingung:

- NAS ist in der Liste der vertrauenswürdigen Absender
- Benutzer ist authentifiziert

Nachbedingung:

- AccountingResponse versendet

3.2.4. DM

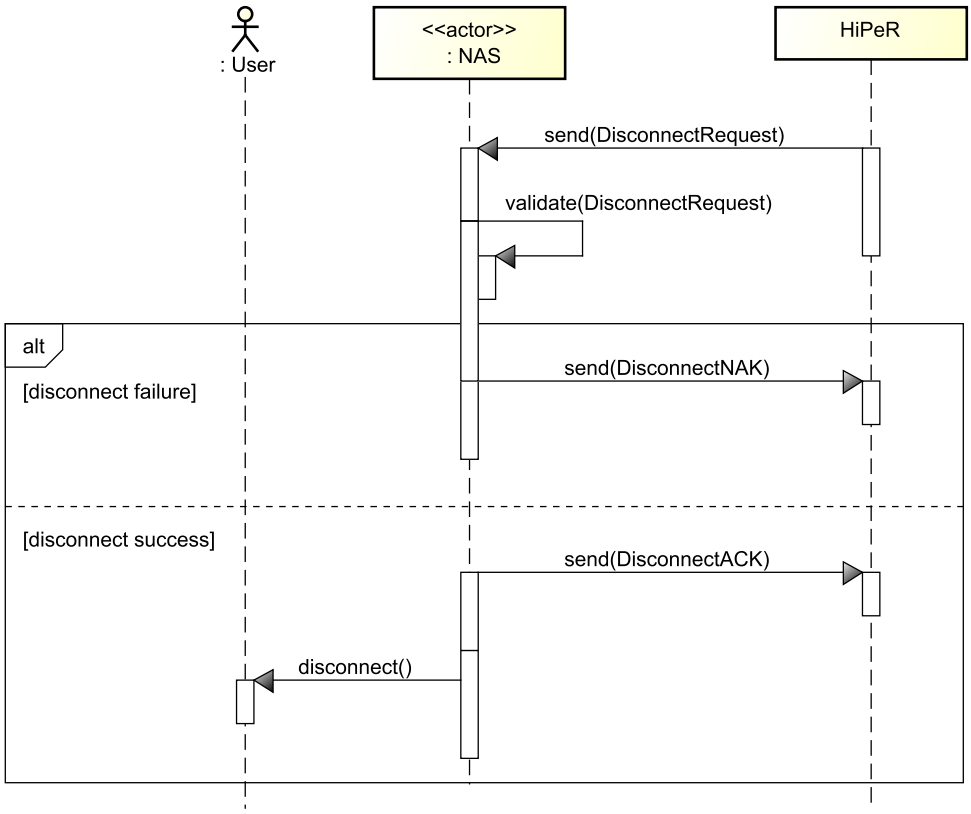


Abbildung 3.5.: Sequenzdiagramm DM

Beschreibung: Der angemeldete Benutzer soll vom Netz getrennt werden. Um dies zu erreichen, schickt HiPeR ein DisconnectRequest an den NAS, bei welchem der Benutzer angemeldet ist. Der NAS überprüft den DisconnectRequest auf seine Gültigkeit und schickt daraufhin entweder ein DisconnectNAK oder DisconnectACK. Bei Letzterem trennt der NAS anschliessend auch noch den Benutzer vom Netz.

Querverweis: UC4 Benutzer trennen

Vorbedingung:

- Benutzer ist authentifiziert

Nachbedingung:

- DisconnectACK oder DisconnectNAK erhalten
- Benutzer ist vom Netz getrennt (nach einem DisconnectACK)

3.2.5. CoA

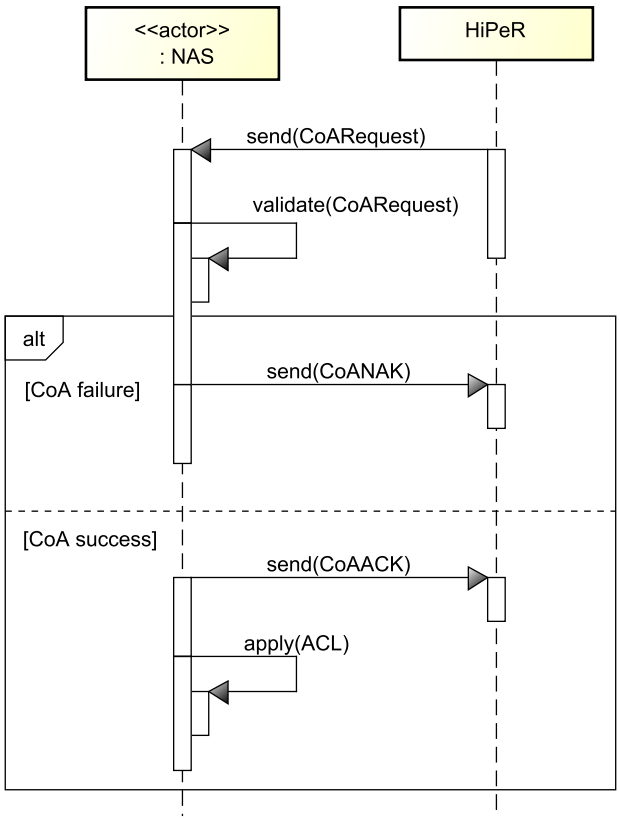


Abbildung 3.6.: Sequenzdiagramm CoA

Beschreibung: Dem angemeldeten Benutzer sollen andere Rechte zugeteilt werden. HiPeR versendet deshalb ein CoARequest mit einer neuen ACL an den NAS. Dieser überprüft den CoARequest auf seine Gültigkeit. Daraufhin sendet der NAS entweder ein CoANAK oder ein CoAACK an HiPeR zurück. Bei Letzterem wendet der NAS zusätzlich die neue ACL auf den Benutzer an. Damit werden die alten Rechte des Benutzers mit den Neuen ersetzt.

Querverweis: UC5 CoA versenden

Vorbedingung:

- Benutzer ist authentifiziert

Nachbedingung:

- CoAACK oder CoANAK erhalten
- ACL auf Benutzer angewandt (nach einem CoAACK)

4. Architektur & Design

4.1. System & näheres Umfeld

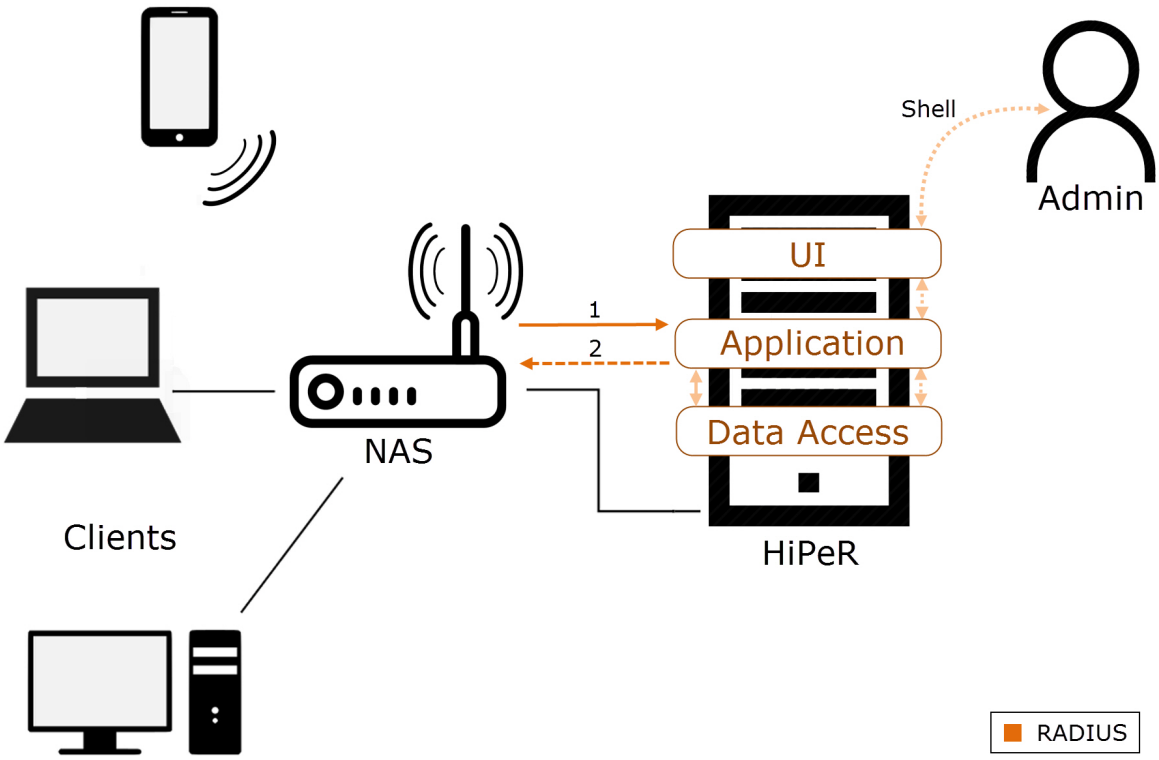


Abbildung 4.1.: HiPeR & das nähere Umfeld

In Abbildung 4.1 wird HiPeR in einen möglichen Kontext gesetzt. Beim Verbindungsversuch eines Clients, sendet der NAS ein Access-Request Paket, welches im Datenfeld eines UDP-Pakets eingeschlossen ist. Der Application-Layer von HiPeR empfängt dieses UDP-Paket und überprüft durch einen Aufruf in den Data Access-Layer zuerst ob die Anfrage von einem vertrauenswürdigen Sender kommt. Nur wenn dies der Fall ist, wird das Paket weiterverarbeitet, ansonsten wird es ignoriert. Der Kommunikationsweg dieses Beispiels wird durch die dunkelorangen Pfeile illustriert (Schritt 1 und 2).

Während HiPeR weiter solche oder andere RADIUS Anfragen beantwortet, kann sich ein Administrator gleichzeitig über den aktuellen Stand von HiPeR informieren (cremefarbene Pfeile) oder auch Disconnect- und CoA-Requests für beliebige Clients verschicken. Hierfür steht ihm eine Shell zur Verfügung, die sich im UI-Layer von HiPeR befindet.

4.1.1. Architektonische Ziele

Die Architektur soll so aufgebaut sein, dass sie auf der einen Seite die Erreichung der Hauptziele "Zuverlässigkeit" und "Performance" positiv beeinflusst. Auf der anderen Seite soll sie aber auch einen gut strukturierten, modularen Aufbau bieten, um eine einfache Erweiterung der Funktionalität zu ermöglichen.

Eine Schichtenarchitektur mit zu wenigen Schichten, würde das Potential für Wiederverwendbarkeit, Änderbarkeit und Portabilität nicht voll ausschöpfen [5, S. 50]. Womöglich könnte sie aber eine höhere Performance erlauben. Im Umkehrschluss sind aber zu viele Schichten kontraproduktiv. Sie erhöhen die Komplexität und den Overhaed bei der Trennung der Schichten und bei der Transformation von Argumenten und Rückgabewerten [5, S. 50].

Die gewählte Architektur die nachfolgend beschrieben wird, stellt einen Mittelweg dar, der für die Erreichung der o.g. Ziele nötig ist.

4.1.2. Architektonische Einschränkungen

In der Wahl des Netzwerk-Frameworks sind wir insofern eingeschränkt, dass Netty von unserem Projektpartner festgelegt wurde.

Netty ist so konzipiert, dass jeweils nur ein einzelner Thread für die Bearbeitung eines Channels zuständig ist [8, S. 11]. Da UDP ein verbindungsloses Netzwerkprotokoll ist, werden alle Anfragen über denselben Channel abgewickelt und somit von einem einzelnen Thread bearbeitet. Dies stellt eine Einschränkung bezüglich der Performance dar. Auf einem Linux System kann diese Einschränkung umgangen werden. Unter Verwendung von epoll kann ein Port mehrfach belegt und somit von mehreren Channels verwendet werden. Unter Windows ist dies nicht möglich.

4.2. Logische Architektur

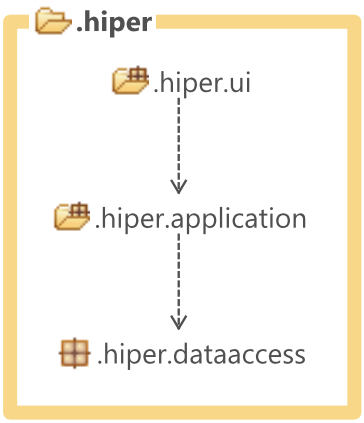


Abbildung 4.2.: Layer Übersicht

HiPeR ist in die 3 Layer UI, Application und Data Access aufgeteilt. Die Packages sind so konzipiert, dass eine Top-Down Architektur gewährleistet wird. Zugriffe erfolgen jeweils nur auf den direkt darunterliegenden Layer. Dies ermöglicht eine klare Strukturierung der Applikation.

4.2.1. UI Layer

Abbildung 4.3.: UI Layer

Der UI Layer beinhalted das Shell Package. Es wird für die Bedienung von HiPeR verwendet. Die Implementation basiert auf der Shell des Spring Frameworks[1]. Die Klasse HiPeRCommands enthält die Befehle welche verarbeitet werden können. Die restlichen Klassen im Package werden für die Initialisierung der Shell benötigt.

[1] `https://github.com/spring-projects/spring-shell`, letzter Zugriff 10.04.2015

4.2.2. Application Layer

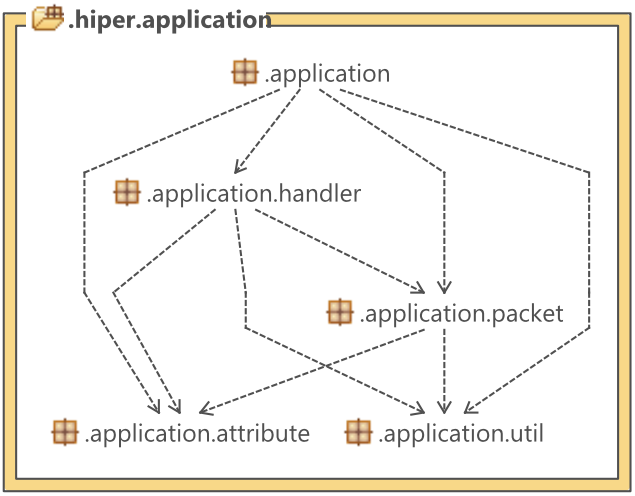


Abbildung 4.4.: Application Layer

Der Application Layer ist das Herzstück von HiPeR und enthält die Domain Objekte, die Business Logik sowie den Netzwerk Code. Der Layer ist in verschiedene Packages unterteilt welche nachfolgend genauer betrachtet werden:

Application Package

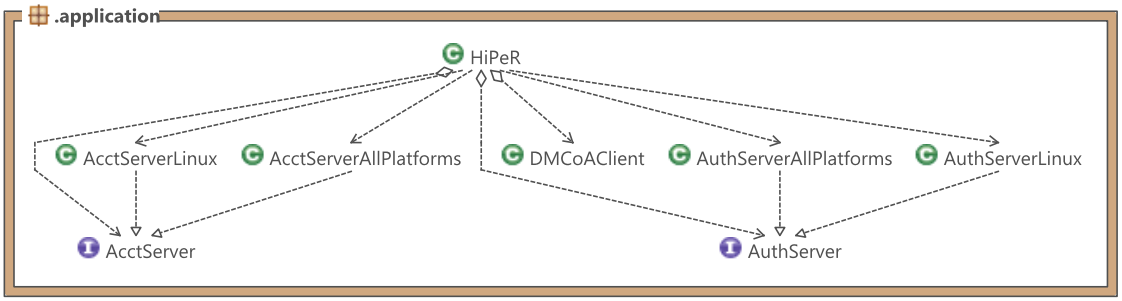


Abbildung 4.5.: Application Package

Im Application Package befindet sich der Einstiegspunkt in die Applikation. Die Klasse HiPeR startet die Applikation, initialisiert die Shell und erstellt Instanzen von der Klasse DMCoAClient und den beiden Interfaces AcctServer (siehe Listing 4.2) und AuthServer (siehe Listing 4.1) von welchen es jeweils eine Implementation für Linux und eine für alle anderen Plattformen gibt. Die Beiden Funktionen Authentifizierung und Accounting können unabhängig voneinander betrieben werden.

Die Klasse DMCoAClient ist zuständig für das Versenden von Disconnect Messages und Change of Authorization Requests. Weil dazu Informationen zu den laufenden Sessions benötigt werden, kann der DMCoAClient nicht ohne Accounting Server betrieben werden.

Beim Start der Applikation erfolgt das Bootstrapping. Unter Linux können dabei mehrere Channels auf demselben Port geöffnet werden, was die Performance erhöht. Unter Windows kann nur ein Channel geöffnet werden. Für jeden Channel wird eine Handler-Chain instanziert. Diese ist für die Verarbeitung von eingehenden und ausgehenden Paketen zuständig.

```
1 public interface AuthServer {
2   public void start() throws InterruptedException;
3   public void stop() throws InterruptedException;
4 }
```

Listing 4.1: AuthServer Interface

```
1 public interface AcctServer {
2   public void start() throws InterruptedException;
3   public void stop() throws InterruptedException;
4 }
```

Listing 4.2: AcctServer Interface

Handler Package

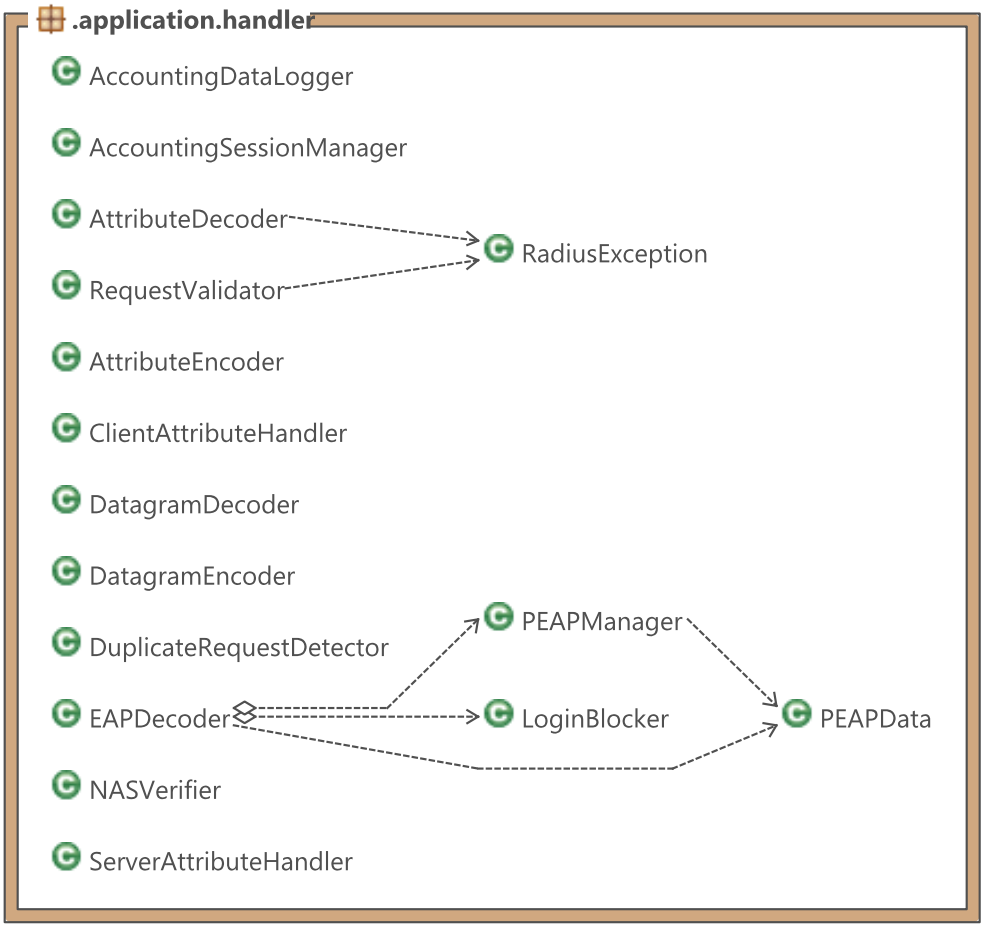


Abbildung 4.6.: Handler Package

Das Handler Package beinhaltet den Grossteil der Logik. Die Handler-Chain ist für die Bearbeitung der Pakete Zuständig und implementiert das RADIUS Protokoll. Nachfolgend wird auf die

einzelnen Klassen genauer eingegangen, in der Reihenfolge in welcher sie von einer eingehenden Anfrage durchlaufen werden:

NASVerifier	Überprüft anhand der Source-Adresse ob das Paket von einem vertrauenswürdigen Absender stammt.
DatagramDecoder	Liest die Header Informationen des RADIUS Pakets, bestimmt den SupplicantType des Absenders und erstellt eine Instanz des entsprechenden Domain Objekts (eine Subklasse von RadiusPacket).
DuplicateRequestDetector	Bestimmt anhand der IP-Adresse und der Portnummer des Absenders, sowie dem Identifier des RADIUS Pakets, ob es sich um einen Duplicate Request handelt.
AttributeDecoder	Liest die Attribute des Pakets und erstellt Instanzen der entsprechenden Domain Objekte. Falls ein User-Password Attribut vorhanden ist, wird dieses entschlüsselt. Ein Message-Authenticator Attribut wird überprüft.
EAPDecoder	Ist zuständig für die Authentifizierung mit PEAPv0/EAP-MSCHAPv2. Alle EAP-Pakete werden hier gelesen und verarbeitet. Die einzelnen Sessions werden anhand einer UUID identifiziert welche in einem State Attribut übertragen wird. Die TLS Daten werden in der Klasse **PEAPData** gespeichert und von der Klasse **PEAPManager** verwaltet. Eine Instanz der Klasse **LoginBlocker** wird verwendet um Benutzer zu blockieren, welche in kurzer Zeit merfach versucht haben, sich mit einem falschen Passwort anzumelden.
RequestValidator	Überprüft Benutzername und Passwort bei der Authentifizierung mit PAP oder MAB und erstellt eine Instanz von AccessAccept oder AccessReject. Bei DM und CoA Nachrichten wird überprüft ob die Anfrage erfolgreich ausgeführt wurde.
AccountingDataLogger	Übergibt Accounting Daten an eine Instanz des AccountingDataWriter Interfaces (dataacess Pakage). Die bestehende Implementierung schreibt diese Daten anschliessend in eine Datei.
AccountingSessionManager	Übergibt alle für DM und CoA Requests relevanten Accounting Daten an eine Instanz des AccountingSessionAccessor Interfaces (dataacess Pakage).
ServerAttributeHandler	Fügt einem ausgehenden Paket Attribute wie z. B. TunnelType hinzu.
ClientAttributeHandler	Fügt einem DM oder CoA Request Attribute hinzu.
AttributeEncoder	Wandelt alle Attribute eines ausgehenden Pakets in Bytes um und schreibt diese in einen ByteBuf. Falls ein MessageAuthenticator Attribute versenden wird, wird zusätzlich dessen Inhalt berechnet.
DatagramEncoder	Berechnet die Header Informationen des RADIUS Pakets und schickt es gekapselt in ein UDP-Paket an den Empfänger.

Die verschiedenen Handler arbeiten unabhängig voneinander. Somit kann durch Hinzufügen eines weiteren Handlers problemlos ein weiterer Arbeitschritt durchgeführt werden.

Packet Package

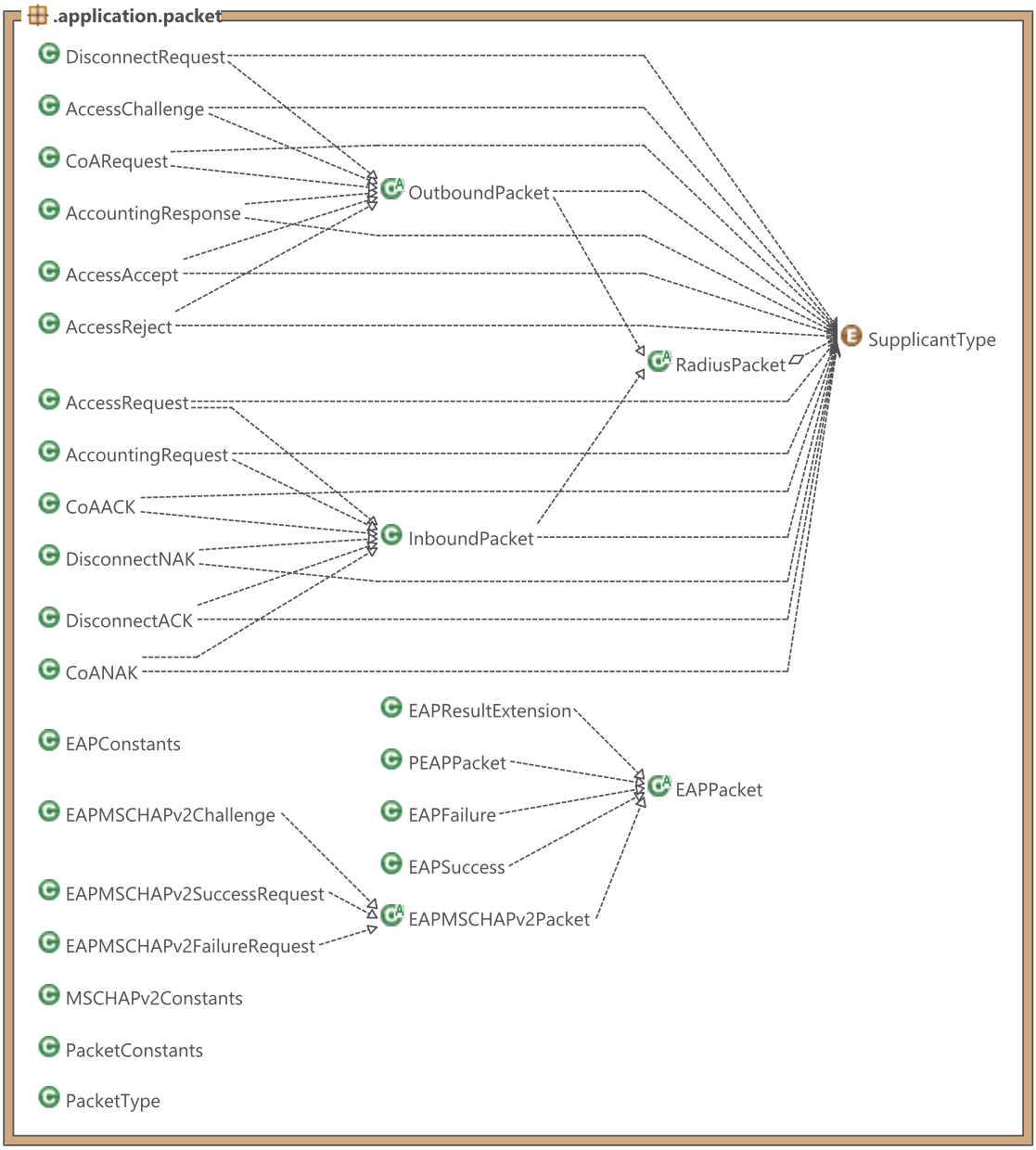


Abbildung 4.7.: Packet Package

Im Packet Package befinden sich die Domain Klassen welche die RADIUS Pakete modellieren. Die abstrakte Basisklasse heisst RadiusPacket. Davon abgeleitet werden die beiden (ebenfalls abstrakten) Klassen InboundPacket und OutboundPacket. Dadurch kann zwischen eingehenden und ausgehenden Paketen unterschieden werden, was bei der Traversierung der Handler-Chain wichtig ist. Eine Stufe weiter unten in der Vererbungshierarchie sind die konkreten Paket-Klassen wie z. B. AccessRequest und AccessAccept.

Das Enum SupplicantType wird verwendet, um den Hersteller des NAS, von welchem ein Paket verschickt wurde, zu identifizieren.

Die Klassen PacketConstants und enthält Konstanten welche für alle RADIUS Pakete gelten. Beispiele dafür sind die maximale Länge eines Pakets oder der Index und die Anzahl Byte des Authenticator Feldes.

Auch die Klasse PacketType enthält Konstanten, welche für ein Mapping zwischen dem Code Feld des RADIUS Pakets und der dazugehörenden Java Klasse verwendet wird.

Neben den RADIUS Paketen befinden sich in diesem Package auch die EAP Pakete. Bei der Authentifizierung mit PEAPv0/EAP-MSCHAPv2 werden diese gekapselt in einem oder mehreren EAP-Message Attributen übertragen.

Die Klassen EAPContants und MSCHAPv2Constants enthalten weitere Konstanten welche für die EAP Pakete relevant sind.

Attribute Package

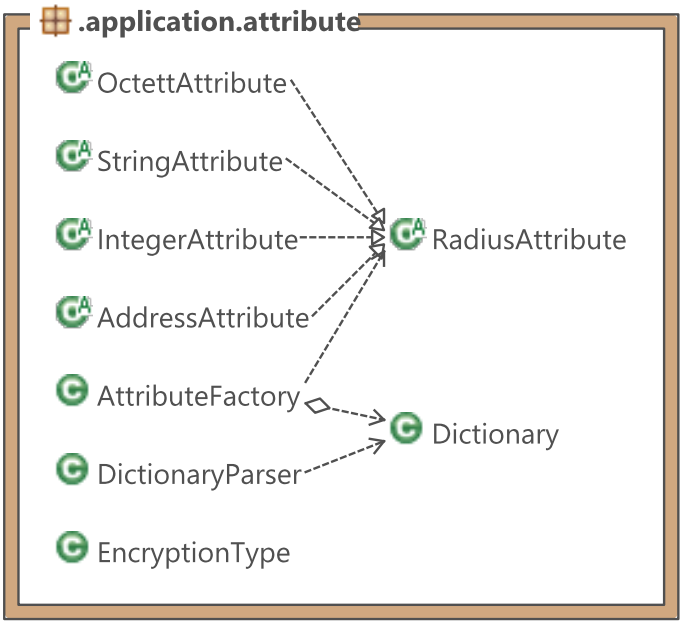


Abbildung 4.8.: Attribute Package

Das Attribute Package beinhaltet eine Klasse für jedes unterstützte RADIUS Attribut. In der Anforderungsspezifikation im Kapitel 2.3.5 ist eine komplette Liste dieser Attribute aufgeführt. Der Übersicht halber werden diese in Abbildung 4.2.2 jedoch nicht abgebildet.

Alle Attributsklassen erben von einer der vier Attributstypklassen OctettAttribute, StringAttribute, IntegerAttribute oder AddressAttribute. Diese wiederum erben von der abstrakten Basisklasse RadiusAttribute.

Der DictionaryParser liest die RFC spezifischen Dictionary Dateien und baut im Arbeitsspeicher ein entsprechendes Dictionary auf, der intern zwei HashMaps hält. Eines bildet das Typenfeld eines Attributs auf die passende Attributsklasse ab und das andere bildet das Typenfeld auf den Verschlüsselungstyp ab, falls das Attribut verschlüsselt ist. Dies ist z. B. bei der Attributsklasse AttrUserPassword der Fall.

Die AttributeFactory, als weitere Klasse in diesem Package, verwendet das Dictionary, welches vom DictionaryParser aufgebaut wird. Sie ist dafür verantwortlich Instanzen von den existierenden Attributsklassen zu erstellen. Hierfür übergibt man ihr den Typ und den Bytestrom des Attributs, welches instanziert werden soll. Die AttributeFactory wird z.B. beim AttributeDecoder gebraucht, wenn aus dem ankommenden Bytestrom eine passende Objektstrukur aufgebaut werden soll.

Util Package

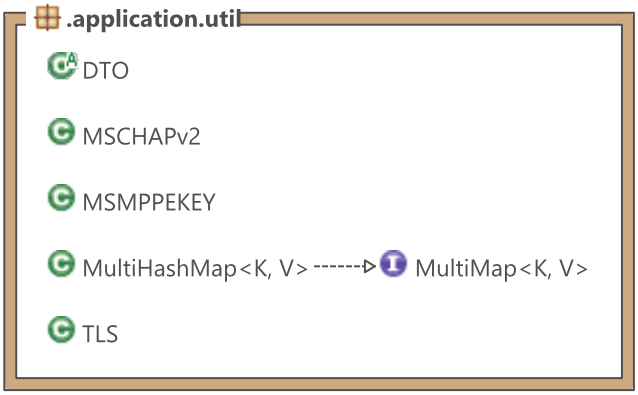


Abbildung 4.9.: Util Package

Im Util Package befindet sich unter anderem das Interface MultiMap (siehe Listing 4.3) und eine Implementation davon: MultiHashMap. Diese wird in der Klasse RadiusPacket verwendet um die Attribute zu speichern, wobei die Typ Nummer als Schlüssel verwendet wird. In einem RADIUS Paket können gewisse Attribute mehrfach vorkommen. Die meisten Attribute kommen allerdings höchstens einmal vor. Deshalb verwendet die MultiHashMap intern zwei HashMaps. Eine enthält die einzelnen Attribute und die andere enthält mehrfach vorkommende Attribute jeweils in einer Liste pro Attribut Typ. Dadurch können Attribute effizient ausgelesen werden und der Overhead welcher entstehen würde wenn jedes Attribut automatisch in einer Liste gespeichert werden würde, entfällt.

Mit der forEach Methode kann eine Funktion auf alle Attribute angewendet werden. Hierbei kann der Polymorphismus von Java gut eingesetzt werden indem die accept Methode der Consumer Instanz für jedes gewünschte Attribut überladen wird. Auf diese Weise kann die Business Logik einfach erweitert werden.

```java
public interface MultiMap<K, V> {

  public boolean contains(K key);

  public boolean containsSingle(K key);

  public boolean containsMulti(K key);

  public void put(K key, V value)
      throws IllegalArgumentException;

  public V getSingle(K key);

  public List<V> getMulti(K key);

  public void remove(K key);

  public void forEach(Consumer<? super V> action)
      throws NullPointerException;

}
```

Listing 4.3: MultiMap Interface

Die Klasse DTO ist eine Datenstruktur welche verwendet wird, um Instanzen von verschiedenen Klassen typsicher in einem einzelnen Objekt zu transportieren. Eine Liste oder ein Array kann dazu nicht verwendet werden, weil dort nur Objekte vom selben Typ enthalten sein können.

Die Klasse MSCHAPv2 enthält verschiedene statische Methoden welche zur Verifizierung einer MSCHAPv2 Response und zur Generierung eines MSCHAPv2 Success Requests benötigt werden. Die Implementierung folgt dabei dem Pseudocode aus RFC 2759 [11].

Für die Berechnung der beiden Attribute MS-MPPE-Recv-Key und MS-MPPE-Send-Key kann die pseudorandom Funktion (PRF) in der Klasse TLS verwendet werden. Diese Funktion wird in RFC 2246 [4, S. 10] beschrieben. Der Inhalt der Attribute muss zusätzlich noch verschlüsselt werden. Dazu kann die Methode encryptKey der Klasse MSMPPEKEY verwendet werden. Die Attribute müssen bei der Authentifizierung mit PEAPv0/EAP-MSCHAPv2 dem AccessAccept Paket angehängt werden.

4.2.3. Data Access Layer

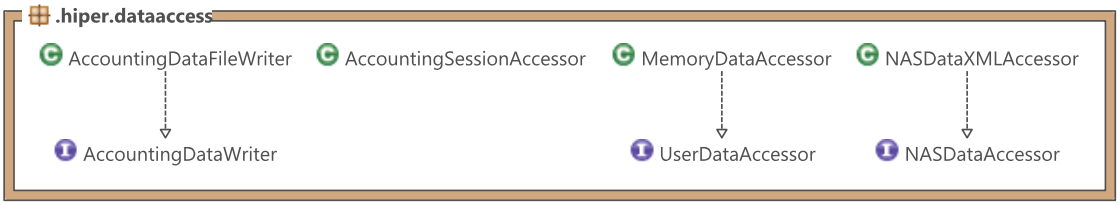


Abbildung 4.10.: Data Access Layer

Der Data Access Layer kapselt den Zugriff auf flüchtige (Arbeitsspeicher) und persistente (Dateisystem) Daten. Es beinhaltet verschiedene Interfaces und jeweils eine Implementation davon.

Die Klasse MemoryDataAccessor implementiert das Interface UserDataAccessor (siehe Listing 4.4). Es bietet Zugriff auf Benutzerdaten, wie Username, Passwort und TunnelAttribute (u.a. VLAN-ID). Die Methode getPassword, gibt das Passwort im Klartext zurück. Gewisse Authentifizierungsprotokolle wie z. B. MSCHAPv2 benötigen das Passwort in dieser Form. Falls das Passwort nicht im Klartext vorliegt, wird eine UnsupportedOperationException geworfen.

```
1  public interface UserDataAccessor {
2
3    public void initialize();
4
5    public void terminate();
6
7    public boolean usernameExists(String username);
8
9    public boolean verifyPassword(String username,
10       String password);
11
12   public String getPassword(String username)
13       throws UnsupportedOperationException;
14
15   public boolean hasTunnelAttributes(String username);
16
17   public boolean hasTunnelAttributeSet(String username,
18       String tunnelPrivateGroupId, long tunnelType,
19       long tunnelMediumType);
20
21   public Map<Integer, String> getMapOfPredefinedTunnel
22       Attributes(String username);
23
24 }
```

Listing 4.4: UserDataAccessor Interface

Der NASDataXMLAccessor implementiert das Interface NASDataAccessor (siehe Listing 4.5). Es liest NAS Daten, die in einer XML-Datei abgelegt sind und stellt sie im Arbeitsspeicher zur Verfügung. Daten die zu einem NAS gehören sind das Shared Secret, der Supplicant Type und die IP-Adresse. Der NASDataXMLAccessor stellt eine Methode zur Verfügung um zu überprüfen ob ein NAS vertrauenswürdig ist. Des Weiteren gibt es Methoden um das Shared Secret und den Supplicant Type anzufordern.

```
1  public interface NASDataAccessor {
2
3    public void initialize();
4
5    public void terminate();
6
7    public Optional<byte[]> getSecret(String ip);
8
9    public Optional<String> getSupplicantType(String ip);
10
11   public boolean isIpPermitted(String ip);
12 }
```

Listing 4.5: NASDataAccessor Interface

Für das Erfassen von Accounting Daten bietet das Interface AccountingDataWriter (siehe Listing 4.6) verschiedene Methoden an. Die Klasse Accounting DataFileWriter ist eine Implementation davon welche alle Daten in eine Datei schreibt.

```
1  public interface AccountingDataWriter {
2
3    public void initialize();
4
5    public void terminate();
6
7    public void writeAndFlush() throws IOException;
8
9    public void writeToBuffer(String attrName,
10       long attrValue);
11
12   public void writeToBuffer(String attrName,
13       String attrValue);
14 }
```

Listing 4.6: AccountingDataWriter Interface

Die Klasse AccountingSessionAccessor speichert die Attribute welche bei DM- und CoA-Requests für die Identifizierung einer Benutzersession benötigt werden. Konkret sind das die Attribute Acct-Session-Id, User-Name, NAS-IP-Address und Calling-Station-Id.

4.3. Sequenzdiagramme

Die in diesem Kapitel aufgeführten Sequenzdiagramme sollen einen Überblick über einen typischen Ablauf bei der Bearbeitung einer MAB-Authentifizierung geben. Zu beachten ist, dass nur die wichtigsten Funktionsaufrufe und Klassen dargestellt werden. Alternative Abläufe wurden weggelassen. Ebenfalls anzumerken ist, dass die Decode- und Encode-Aufrufe nicht wie in den Diagrammen dargestellt, direkt von den Handlern ausgeführt werden, sondern von Klassen aus dem Netty Framework.

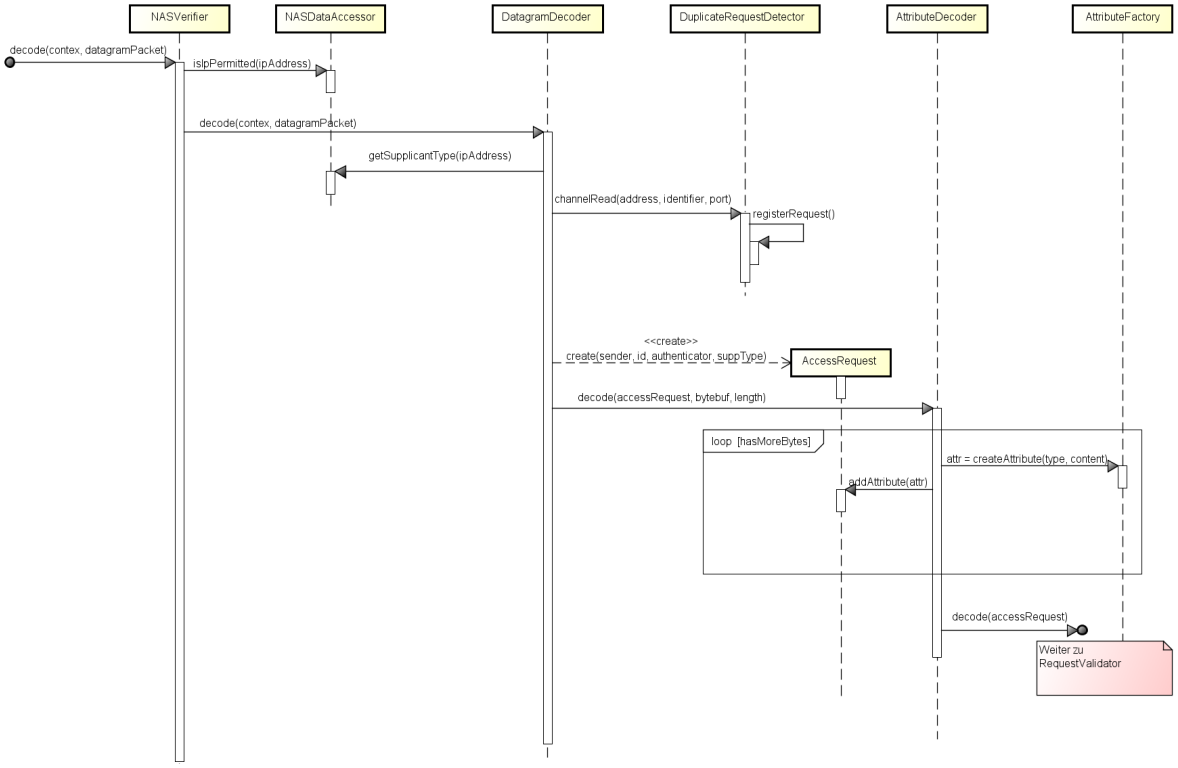


Abbildung 4.11.: Sequenzdiagramm MAB (Teil 1)

Abbildung 4.11 zeigt, wie durch einen Request an HiPeR, Netty die decode-Methode des ersten Handlers (NASVerifier) aufruft. Da in diesem Fall der Request von einem vertrauenswürdigen NAS kommt, wird dieser an den DatagramDecoder weitergeleitet. Dort wird der SupplicantType des Absenders bestimmt und eine Instanz von AccessRequest erstellt. Zusätzlich werden Daten an den DuplicateRequestDetector weitergeleitet welcher überprüft ob eine identische Anfrage bereits bearbeitet wurde. Anschliessend werden im Loop-Block des AttributeDecoders die Attribute im Bytestrom sukzessive ausgelesen und für jedes unterstütze Attribut, eine entsprechende Attributklasse instanziert. Diese Instanzen werden dann ans AccessRequest-Paket angehängt, damit es an den RequestValidator weitergeleitet werden kann.

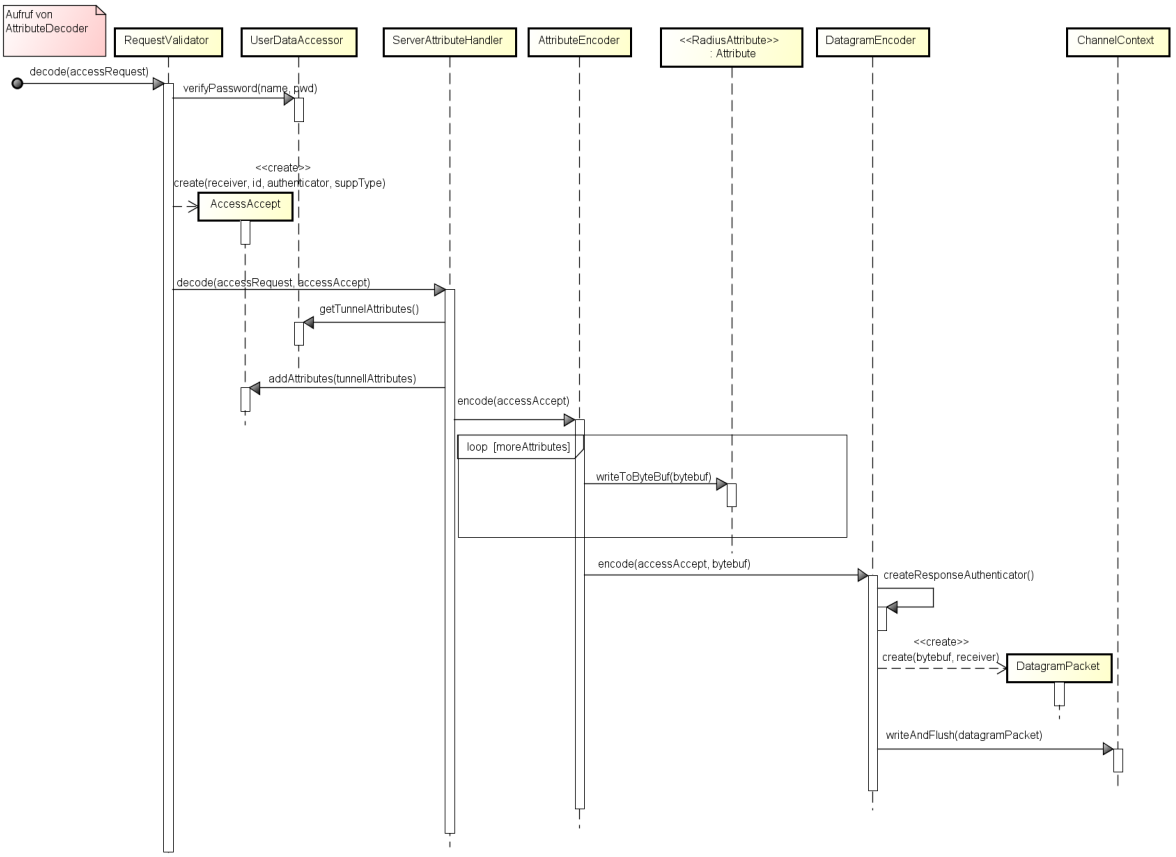


Abbildung 4.12.: Sequenzdiagramm MAB (Teil 2)

Die Bearbeitung des AccessRequests wird in der Abbildung 4.12 fortgesetzt. Der RequestValidator überprüft nun die Anmeldedaten des Benutzers und erstellt ein AccessAccept-Paket. Dieses wird im ServerAttributeHandler mit Tunnel-Attributen (u. a. VLAN-ID) ergänzt. Anschliessend iteriert der AttributeEncoder im Loop-Block durch alle Attribute die im AccessAccept-Paket vorhanden sind und wandelt so jedes benötigte Attribut in einen Bytestrom um. Der DatagramEncoder berechnet danach den ResponseAuthenticator und erstellt das UDP-Paket, welches an den Empfänger zurückgeschickt wird.

Die Sequenzdiagramme der anderen Use Cases werden in diesem Kapitel nicht explizit aufgeführt, weil sie zum Grossteil miteinander identisch sind. Bei der Authentifizierung mit PEAPv0/EAP-MSCHAPv2 kommt zusätzlich die Klasse EAPDecoder zum Einsatz und die Klasse ServerAttributeHandler wird nicht verwendet. Beim Accounting werden die Klassen AccountingDataLogger und AccountingSessionManager eingesetzt. Bei CoA und DM wird der ServerAttributeHandler ebenfalls nicht verwendet, dafür die Klasse ClientAttributeHandler. Ansonsten ist der Ablauf der Gleiche.

4.4. Prozesse & Threads

Die Verwaltung der verschiedenen Threads wird hauptsächlich vom Netty Framework übernommen. Wie bereits in Kapitel 4.1.2 erwähnt, ist der gleiche Thread für die gesamte Bearbeitung eines Channels inklusive Handler-Chain zuständig. Unter Linux können mehrere Channels auf demselben Port geöffnet werden und somit mehrere Threads dafür eingesetzt werden. Die kon-

krete Anzahl ist konfigurierbar. Dadurch entsteht die Notwendigkeit zur Synchronisation gewisser Daten welche von allen Channels benötigt werden. Dies betrifft vor allem Session Daten und Duplicate Request Detection. Benutzerdaten und NAS-Daten müssen nicht zwingend synchronisiert werden, sofern sie nur gelesen und nicht geschrieben werden.

Die Threads werden von einer EventLoopGroup (eine Klasse des Netty Frameworks) verwaltet. Falls ein Handler eine blockierende Operation durchführt, wie z. B. das Lesen von Daten aus einer Datenbank, kann er einer anderen EventLoopGroup zugewiesen werden, und läuft dann in einem anderen Thread. Somit wird der Thread des Channels nicht blockiert.

4.5. Qualitätsansprüche

Grundsätzliche Ansprüche an die Architektur sind Prinzipien wie: „Separation of Concerns“, „low Coupling“, „Extensibility“ und „Maintainability“. Nachfolgend sind einige Anmerkungen zu den einzelnen Prinzipien und deren Umsetzung im System aufgeführt:

4.5.1. Separation of concerns

Dieses Prinzip ist auf mehrere Arten umgesetzt. Zum einen sind verschiedene Verantwortlichkeiten auf unterschiedliche Klassen aufgeteilt. So besteht die Handler-Chain aus mehreren Handlern, die alle eine andere Aufgabe haben. Ein Beispiel hierfür ist der NASVerifier und der DatagramEncoder. Während der NASVerifier als erste Handler Instanz anhand der Source Adresse des UDP-Pakets entscheidet, ob die Anfrage von einem vertrauenswürdigen Absender stammt, ist der DatagramEncoder als letzte Handler Instanz dafür verantwortlich, die Java Objekthierarchie wieder in einen Bytestrom umzuwandeln und das UDP-Paket zu verschicken.

Zum anderen sind alle Klassen, wie bspw. die Attribut-, Handler- und die Paketklassen in unterschiedlichen Packages zusammengefasst. Auf diese Weise werden Klassen mit ähnlichem Verhalten miteinander gruppiert.

4.5.2. Low coupling

In der gewählten Schichtenarchitektur geht der Hauptkommunikationsweg von oben zu der direkt darunterliegenden Schicht (top-down). Da keine Schichten ausgelassen werden sondern immer nur innerhalb der eigenen oder mit der direkt darunterliegenden Schicht kommuniziert wird, entspricht dies den strukturellen Charkteristiken des Layers Pattern [5, S. 34]. Die Umsetzung dieses Patterns begünstigt u.a. das Design-Prinzip der losen Kopplung.

Netty trägt ebenfalls dazu bei, die Kopplung der einzelnen Komponenten im System möglichst tief zu halten. Es erlaubt uns eine Handler-Chain aufzubauen, dessen Handler sich gegenseitig nicht kennen und somit grundsätzlich unabhängig voneinander sind. Eine gewisse Abhängigkeit ist nur durch die von uns festgelegte Verarbeitungsreihenfolge gegeben. Dies ist allerdings vernachlässigbar, das es bspw. keinen Sinn ergibt eine Anfrage zu bearbeiten bevor geprüft wurde, ob sie von einer vertrauenswürdigen Quelle kommt, da sie allenfalls gar nicht bearbeitet werden dürfte.

4.5.3. Extensibility

Mit zunehmender Funktionalität steigt die Wahrscheinlichkeit zusätzliche Attributtypen unterstützen zu müssen. Deshalb ist es wichtig, das System auf eine einfache Art und Weise erweitern zu können. HiPeR kann denkbar einfach um die Unterstützung weiterer Attribute erweitert werden.

Für ein neues Attribut wird zuerst eine Klasse erstellt, welche eine der Attributstypklassen (StringAttribute, OctettAttribute, etc.) erweitert. Der Name dieser Klasse muss sich aus dem Präfix "Attr" und dem Attributnamen zusammensetzen. Anschliessend muss das Attribut noch in die entsprechende Dictionary Datei eingetragen werden, welche beim Systemstart gelesen wird und den Link zwischen der Typ-Information aus dem Bytestrom und der Attributsklasse herstellt.

Da HiPeR ab diesem Zeitpunkt das neu hinzugefügte Attribut kennt, kann nun noch die gewünschte Behandlung dieses Attributs implementiert werden. Auf diese Weise ist HiPeR sehr einfach erweiterbar, was die Unterstützung neuer Attribute betrifft.

4.5.4. Maintainability

Die Wartbarkeit wird durch die Umsetzung der oben genannten Prinzipien per Definition begünstigt. Vor allem die Aufteilung der Logik auf die verschiedenen, voneinander unabhängigen Handler Klassen fördert die Wartbarkeit. Des Weiteren wirken auch Unit- und Integrationstests unterstützend.

4.6. Verworfene Lösungsoptionen

4.6.1. Request und Reply über unterschiedliche Ports

Eine Lösungsoption welche verworfen wurde ist die Verwendung von unterschiedlichen Ports zum Empfangen und Versenden von Paketen. Die Idee war, dass HiPeR Requests auf dem standard Port 1812 empfängt, Replies jedoch über einen anderen Port zurückschickt. Dadurch könnten zwei Channels aufgemacht werden und die Last auf zwei Threads verteilt werden, was vor allem unter Windows interessant wäre. Pakete welche nicht vom erwarteten Port stammen, werden allerdings durch gewisse NAS ignoriert, wodurch diese Idee unbrauchbar wird.

4.6.2. Eigene Implementation SSL-Handler

Für die Erstellung des TLS-Tunnels wurde zu Beginn der Implementation von Use Case 2 (PEAPv0/EAP-MSCHAPv2) damit begonnen, ein SSL-Handler zu schreiben, welcher intern eine SSLEngine verwendet. Es hat sich aber herausgestellt dass die vom Netty Framework bereitgestellte Klasse SSLHandler auf einfache Art und Weise in einem EmbeddedChannel gekapselt für denselben Zweck verwendet werden kann. Deshalb wurde diese Idee wieder verworfen.

4.7. Third Party Libraries

HiPeR verwendet mehrere Libraries von Drittherstellern. Diese werden mit Maven verwaltet und automatisch importiert. Abbildung 4.13 zeigt eine Übersicht aller Libraries welche von HiPeR

direkt verwendet werden. Diese wiederum können weitere Abhängigkeiten haben welche hier nicht dargestellt werden. Eine vollständige Liste aller Abhängigkeiten befindet sich im Anhang G.

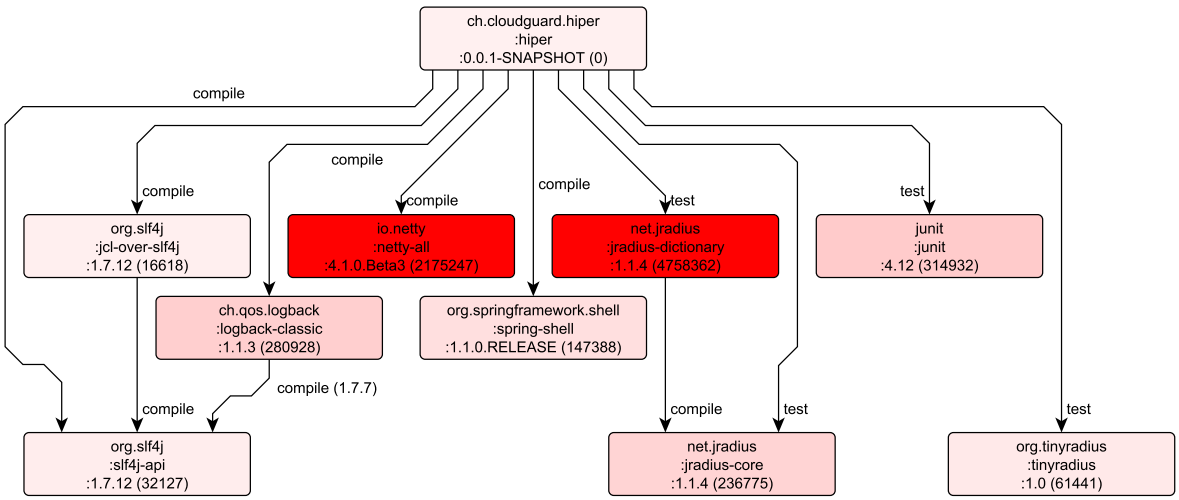


Abbildung 4.13.: Maven Dependencies

4.7.1. Verwendung

Gewisse Libraries werden nur für Unit-Tests eingesetzt und sind nicht Teil des fertigen Produkts. Diese erkennt man in Abbildung 4.13 an der Bezeichnung “test” neben den Pfeilen.

Name	Verwendung
netty-all	Empfangen, Bearbeiten und Versenden von RADIUS Paketen
spring-shell	Kommandozeile für die Bedienung von HiPeR
logback-classic	Erstellen von Log-Einträgen
slf4j-api	Ermöglicht den Einsatz von verschiedenen Logging Frameworks über eine einheitliche API
jcl-over-slf4j	Leitet Aufrufe auf Apache Commons Logging zu SLF4J weiter
junit	Erstellen von Unit-Tests
jradius-core	Erstellen und Überprüfen von RADIUS Paketen um HiPeR zu testen
jradius-dictionary	Enthält RADIUS Attribute welche von jradius-core verwendet werden
tinyradius	Erstellen und Überprüfen von RADIUS Paketen um HiPeR zu testen

4.7.2. Lizenzen

Name:	netty-all
Lizenz:	Apache License V2.0
URL:	`https://github.com/netty/netty/blob/master/LICENSE.txt`, letzter Zugriff 10.04.2015
Name:	spring-shell
Lizenz:	Apache License V2.0
URL:	`http://www.springframework.net/license.html`, letzter Zugriff 10.04.2015
Name:	logback-classic
Lizenz:	EPL V1.0 oder LGPL V2.1
URL:	`http://logback.qos.ch/license.html`, letzter Zugriff 10.04.2015
Name:	slf4j-api, jcl-over-slf4j
Lizenz:	MIT License
URL:	`http://www.slf4j.org/license.html`, letzter Zugriff 10.04.2015
Name:	junit
Lizenz:	EPL V1.0
URL:	`http://junit.org/license.html`, letzter Zugriff 10.04.2015
Name:	jradius-core, jradius-dictionary
Lizenz:	LGPL
URL:	`http://www.gnu.org/licenses/lgpl.html`, letzter Zugriff 10.04.2015
Name:	tinyradius
Lizenz:	LGPL V2.1
URL:	`https://www.gnu.org/licenses/lgpl-2.1.html`, letzter Zugriff 10.04.2015

Alle verwendeten Libraries unterliegen Open Source Lizenzen und dürfen in proprietärer Software verwendet und verbreitet werden sofern sie nicht verändert werden. Anpassungen müssen unter der entsprechenden Lizenz öffentlich zugänglich gemacht werden.

5. Installationsanleitung

5.1. Ausführen von HiPeR

Für die Ausführung von HiPeR ist keine Installation notwendig. Man muss lediglich die mitgelieferte ZIP-Datei an einem beliebigen Ort entpacken. Anschliessend können in der Datei "config/permitted_nas_list.xml" die IP-Adressen von allen Network Access Servern eingetragen werden. Die Applikation kann danach von der Kommandozeile mit folgendem Befehl gestartet werden:

```
java -jar hiper-<version>.jar
```

Die einzige Voraussetzung ist eine Java Runtime Environment (JRE), welche unter der Adresse `https://java.com/de/download/` bezogen werden kann.

5.2. Erweitern von HiPeR mit Eclipse

HiPeR wurde in der Entwicklungsumgebung Eclipse geschrieben. Diese kann unter der Adresse `https://www.eclipse.org/downloads/` heruntergeladen werden. Natürlich kann das Projekt auch mit einer anderen IDE weiterentwickelt werden, diese Anleitung beschränkt sich jedoch auf Eclipse (Version 4.4.0).

Das Projekt kann unter "File > Import > Maven > Existing Maven Projects" in den Workspace importiert werden. Anschliessend kann das Projekt im Package Explorer mit der rechten Maustaste angeklickt werden. Im Kontextmenü unter "Maven > Update Project" können alle Dependencies importiert werden.

Damit die Spring Shell in der Eclipse Konsole korrekt funktioniert, müssen beim Ausführen von HiPeR folgende VM Parameter verwendet werden:

Windows:
-Djline.WindowsTerminal.directConsole=false
-Djline.terminal=jline.UnsupportedTerminal

Linux:
-Djline.terminal=org.springframework.shell.core.IdeTerminal

Falls HiPeR auf einem 32-bit Linux System ausgeführt werden soll, muss die Dependency "netty-transport-native-epoll" in der Datei pom.xml angepasst werden.

6. Testbericht

6.1. Funktionale Anforderungen

6.1.1. Unit- und Integrationstests

Um die Erfüllung der funktionalen Anforderungen sicherzustellen, wurden Unit-Tests und Integrationstests durchgeführt. Sie befinden sich im Testverzeichnis der Source-Code Abgabe. Die folgenden Abbildungen zeigen die durchgeführten Tests:

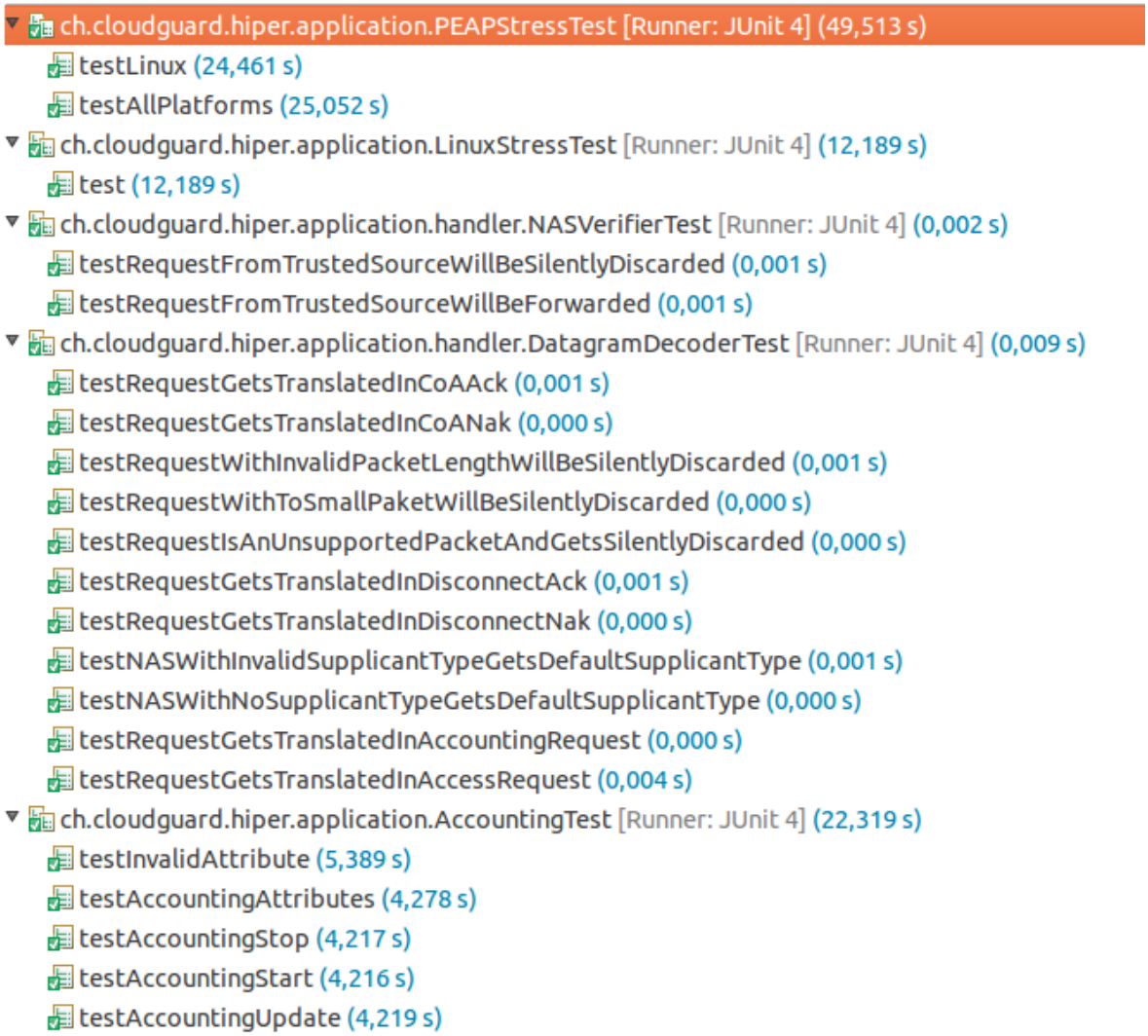


Abbildung 6.1.: Durchgeführte Unit- und Integrationstests (Teil 1)

Abbildung 6.2.: Durchgeführte Unit- und Integrationstests (Teil 2)

6.1.2. Code-Coverage

In der Abbildung 6.3 ist die Code Coverage ersichtlich. Da sich die eigentliche Businesslogik im Handler Package befindet, wurde darauf geachtet, dass dieser Teil gründlicher getestet wird. Die Shell konnte aufgrund der tieferen Priorität und der fehlenden Zeit leider nicht mehr durch Unit-Tests abgedeckt werden.

Overall Coverage Summary

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
all classes	72% M: 3410 C: 8594	54% M: 262 C: 309	58% M: 337 C: 468	69% M: 731 C: 1606	70% M: 154 C: 358	84% M: 22 C: 114

Coverage Breakdown by Package

Name	instruction	branch	complexity	Zeilen	Methoden	Klassen
ch.cloudguard.hiper.application	M: 942 C: 886 48%	M: 36 C: 10 22%	M: 48 C: 22 31%	M: 187 C: 128 41%	M: 28 C: 19 40%	M: 5 C: 6 55%
ch.cloudguard.hiper.application.attribute	M: 885 C: 1857 68%	M: 25 C: 41 62%	M: 90 C: 125 58%	M: 187 C: 321 63%	M: 70 C: 112 62%	M: 5 C: 39 89%
ch.cloudguard.hiper.application.handler	M: 685 C: 3349 83%	M: 103 C: 188 65%	M: 88 C: 151 63%	M: 150 C: 725 83%	M: 5 C: 81 94%	M: 1 C: 20 95%
ch.cloudguard.hiper.application.packet	M: 91 C: 698 88%	M: 16 C: 26 62%	M: 23 C: 88 79%	M: 27 C: 186 87%	M: 8 C: 82 91%	M: 4 C: 25 86%
ch.cloudguard.hiper.application.util	M: 35 C: 1183 97%	M: 3 C: 33 92%	M: 9 C: 50 85%	M: 12 C: 179 94%	M: 6 C: 35 85%	M: 1 C: 6 86%
ch.cloudguard.hiper.dataaccess	M: 476 C: 621 57%	M: 45 C: 11 20%	M: 46 C: 32 41%	M: 106 C: 67 39%	M: 21 C: 29 58%	M: 2 C: 18 90%
ch.cloudguard.hiper.ui.shell	M: 296 C: 0 0%	M: 34 C: 0 0%	M: 33 C: 0 0%	M: 62 C: 0 0%	M: 16 C: 0 0%	M: 4 C: 0 0%

Abbildung 6.3.: Code Coverage von HiPeR

6.2. Nicht-funktionale Anforderungen

Nachfolgend sind die Testergebnisse der einzelnen nicht-funktionalen Anforderungen aufgeführt. Die Überschriften wurden zur besseren Übersicht gleich gehalten wie im Kapitel 2.3.5.

6.2.1. Funktionalität

Interoperabilität

Die Interoperabilität von HiPeR ist mit folgenden Geräten erfolgreich getestet worden:

- Cisco Aironet 2700 Series 802.11ac Dual Band Access Point
- Cisco Aironet 3500 Series 802.11n Dual Band Access Point
- Cisco 2500 Series Wireless Controller
- Cisco Catalyst 3560-CG Series PoE Switch
- D-Link DIR-635 Wireless N IP Router
- TP-Link AC750 Wireless Dual Band Gigabit Router
- Asus RT-N12 SuperSpeedN Router

Sicherheit

Folgende sicherheitsrelevante Punkte wurden durch Unit- resp. Integrationstests überprüft:

- Ignorieren von Requests welche nicht von vertrauenswürden Quellen stammen.
- Ignorieren von Requests welche eine ungültige Paketlängenangebe haben.
- Ignorieren von Requests welche eine zu kurze Paketlänge haben.
- Ignorieren von Requests welche ein ungültiges Code-Feld haben.
- Ignorieren von Requests welche ein falsches Secret verwenden.
- Senden von Access-Rejects bei ungültigem Benutzernamen.
- Senden von Access-Rejects bei ungültigem Passwort.
- Blockieren von Benutzern welche sich mehrmals mit falschem Passwort versuchen anzumelden.

Konformität

Die implementierten RADIUS Attribute werden von allen verwendeten Testgeräten und Client-Simulatoren korrekt interpretiert. Das Verhalten der Geräte und Simulatoren entspricht ebenfalls den Erwartungen. Es ist deshalb anzunehmen, dass die RFC-Konformität eingehalten wird.

6.2.2. Zuverlässigkeit

Fehlertoleranz

Siehe Abschnitt Sicherheit im Kapitel 6.2.1

6.2.3. Effizienz

Zeitverhalten

Belastungstests auf einem Linux System zeigen, dass HiPeR die geforderte Leistung sowohl bei MAB als auch bei PEAP übertrifft. Abbildung 6.4 zeigt einen Durchsatz von ca. 99'950 Requests pro Sekunde bei MAB, wobei ein Request jeweils einer Authentifizierung entspricht.

```
Starting stress test with 80 threads.
Total time: 16008 ms
Total number of requests: 1600000
Time per Request: 0.010005 ms
Requests per second: 99950.02498750624
Number of sucessful requests: 1600000
Number of failed requests: 0
Number of exceptions: 0
```

Abbildung 6.4.: MAB Belastungstest

Mit PEAP sind stattdessen nur ca. 87 Authentifizierungen pro Sekunde möglich. Bei 8 Requests pro PEAP-Authentifizierung ergibt das einen Durchsatz von 696 Requests pro Sekunde.

```
Number of sucessful authentications: 2000
Number of failed authentications: 0
Total time: 22917 ms
Total number of authentications: 2000
Time per authentication: 11.4585 ms
Authentications per second: 87.27145786970371
```

Abbildung 6.5.: PEAP Belastungstest

Verbrauchsverhalten

Abbildung 6.6 zeigt, dass die Auslastung des Arbeitspeichers während des Belastungstests ca. 115 MByte beträgt. Damit wird die geforderte Limite von 300 MByte deutlich unterschritten.

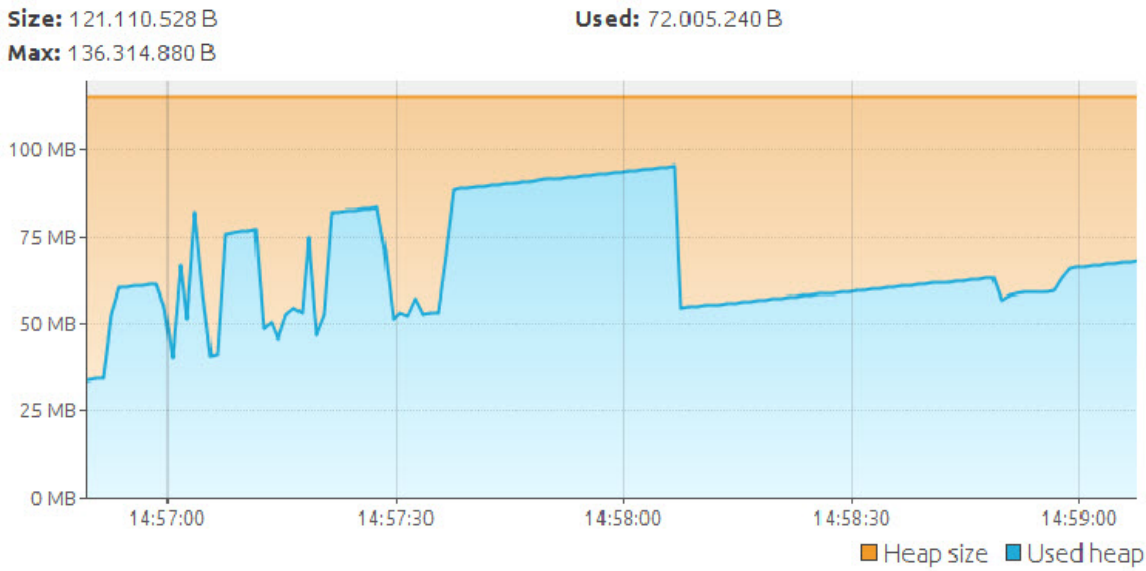


Abbildung 6.6.: Beanspruchung des Arbeitsspeichers (Linux)

Es muss allerdings erwähnt werden, dass der maximal zur Verfügung stehende Arbeitsspeicher eingeschränkt werden muss. Ansonsten beansprucht die Applikation mehr Speicher, wenn dieser frei zur Verfügung steht. Dies ist auch dann der Fall, wenn der Speicher gar nicht zwingend benötigt wird. Der Grund liegt darin, dass der Garbage Collector seltener aufräumt, wenn genügend Speicher vorhanden ist.

Obwohl HiPeR darauf ausgelegt ist, unter Linux zu laufen, ist das Verbrauchsverhalten auch unter Windows getestet worden (siehe Abbildung 6.7).

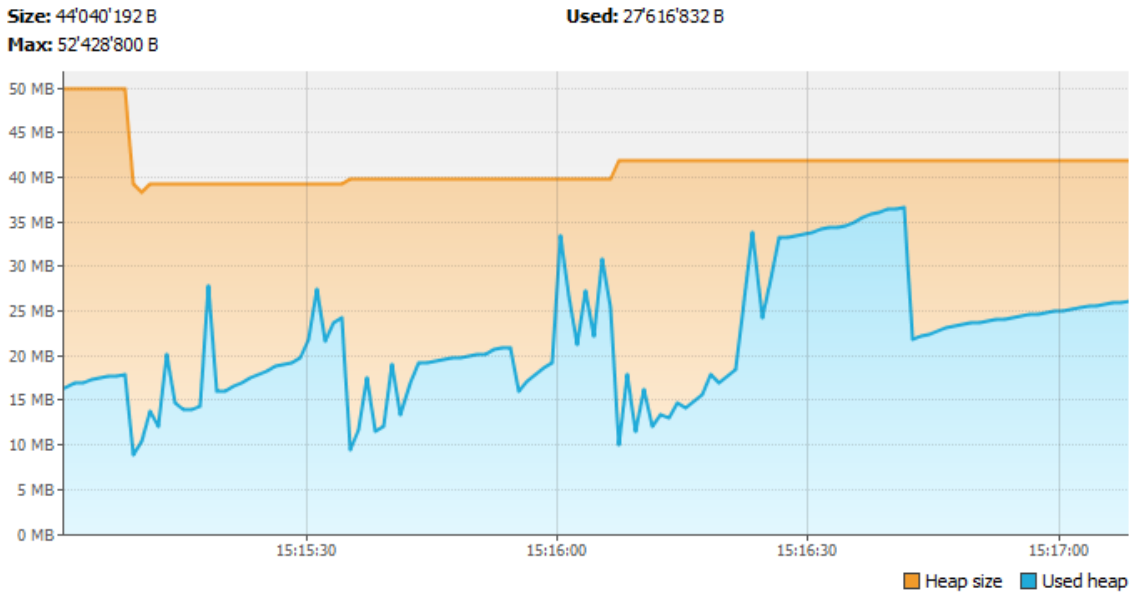


Abbildung 6.7.: Beanspruchung des Arbeitsspeichers (Windows)

Der Arbeitsspeicherverbrauch ist mit maximal 50 MByte kleiner als bei Linux. Das liegt daran, dass unter Windows nur ein einziger Channel eingesetzt werden kann und somit weniger Ressourcen benötigt werden. Allerdings sinkt dadurch auch der maximale Durchsatz.

6.2.4. Übertragbarkeit

Anpassbarkeit

HiPeR läuft unter Windows und Linux. Die Ausfühung sollte auch auf anderen Betriebssysstemen möglich sein, dies ist allerdings nicht getestet worden.

6.3. Vergleich des Durchssatzes von HiPeR und FreeRADIUS

6.3.1. Testsystem

HiPeR und FreeRADUS sind auf folgendem Testsystem ausgeführt worden:

- Ubuntu 15.04 64-Bit
- Intel Core i7-3720QM 2.6GHz (4 Cores)
- 16,0 GB RAM

6.3.2. Messergebnisse

Die Abbildungen 6.8 und 6.9 zeigen, dass die Anzahl Authentifizierungen pro Sekunde von HiPeR und FreeRADIUS in etwa gleich sind. Die Ergebnisse von PEAP sind allerdings mit Vorsicht zu geniessen, da die CPU des Client jedesmal zu 100% ausgelastet war. Die Ergebnisse könnten also beim Test mit mehreren Clients höher ausfallen.

MAB

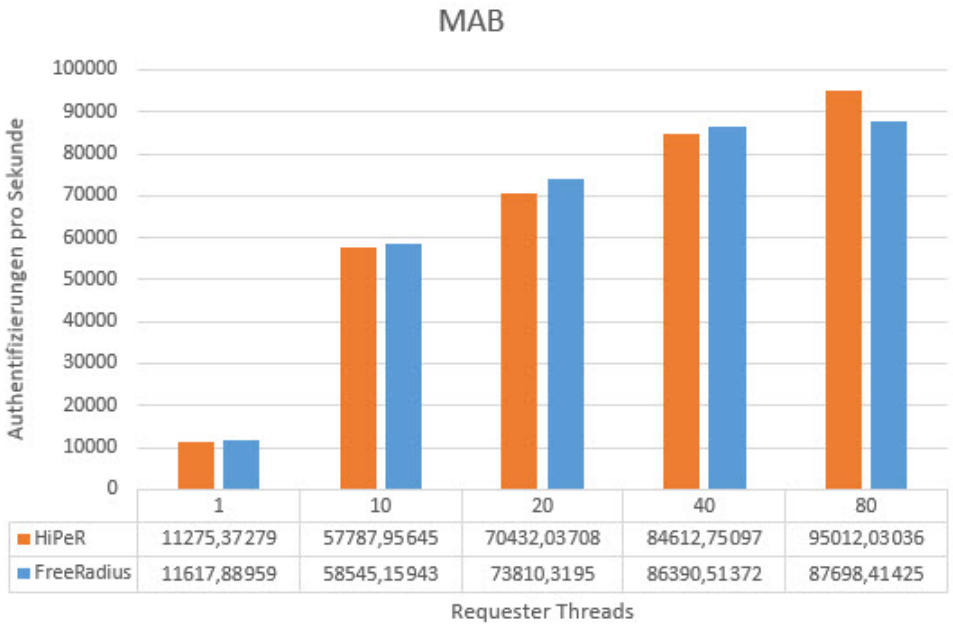


Abbildung 6.8.: Durchsatz MAB

PEAP

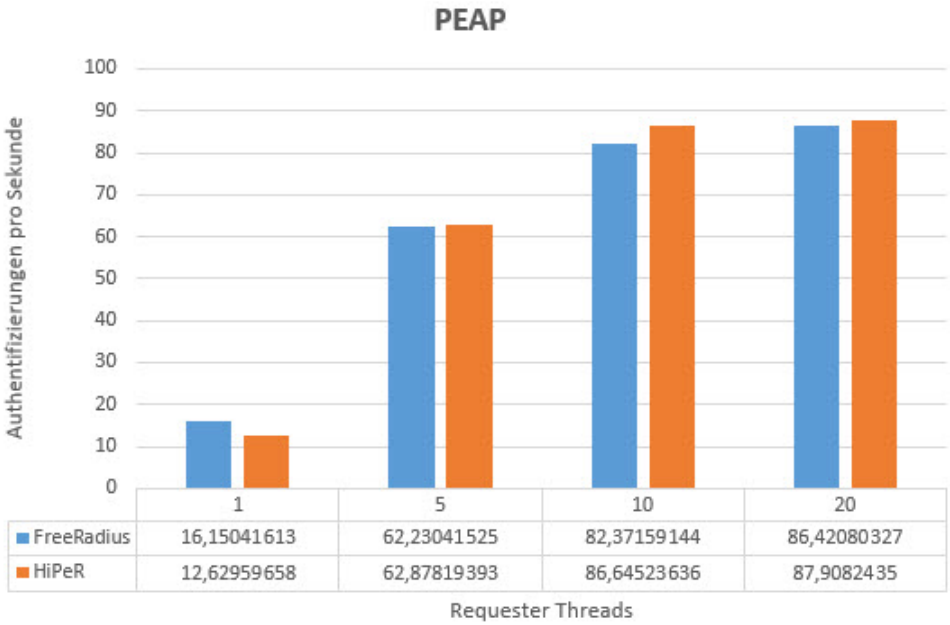


Abbildung 6.9.: Durchsatz PEAP

Es ist sofort ersichtlich, dass sich die Messergebnisse von MAB und PEAP in zwei komplett unterschiedlichen Leistungsbereichen aufhalten. Allerdings ist anzumerken, dass bei MAB eine Authentifizierung mit einem Request erledigt werden kann. Bei PEAP sind für eine Authentifizierung je nach Implementation ca. 8-10 Requests nötig. Das ist darauf zurückzuführen, dass bei PEAP in der ersten Phase ein Handshaking stattfindet und in der zweiten Phase eine EAP-MSCHAPv2-Kommunikation. Die dafür benötigten Berechnungen beanspruchen mehr Zeit und Leistung. Darum fällt die Anzahl Authentifizierungen pro Sekunde wesentlich tiefer aus, als bei MAB.

Teil III.

Anhang

A. Projektplan

A.1. Projekt Übersicht

A.1.1. Zweck und Ziel

Die aktuell von CloudGuard eingesetzte RADIUS Lösung setzt sich aus zwei verschiedenen Produkten zusammen - dem Open Source Server FreeRADIUS[1] und dem hauseigenen Produkt Macman[2] (RADIUS und LDAP Server). Weil in FreeRADIUS die Businesslogik nur schwer abbildbar ist, wird sie in Macman umgesetzt. Requests die sich auf die Authentifizierung, Authorisierung und das Accounting beziehen, werden an den FreeRADIUS Server geschickt. Dieser leitet die Requests allerdings weiter an Macman, wo sie schlussendlich beantwortet und zum FreeRADIUS Server zurückgeschickt werden. Diese zusätzliche Kommunikation zwischen FreeRADIUS und Macman wirkt sich negativ auf die Perfomance aus. Auch können Änderungen am Verhalten des FreeRADIUS Servers nicht zur Laufzeit vorgenommen werden, ohne dass dieser anschliessend neugestartet werden muss.

Ziel des Projektes HiPeR - *High Performance RADIUS Server* ist es, eine serverseitige Implementierung des RADIUS Protokolls zu erstellen, welche die Möglichkeit bietet, Businesslogik abzubilden. Dabei stehen vor allem Zuverlässigkeit und Performance im Vordergrund.

Zu einem späteren Zeitpunkt (ausserhalb dieses Projektes), soll HiPeR direkt in Macman integriert werden um das o.g. Setup zu ersetzen. Damit kann den Kunden eine performantere und dynamischere RADIUS-Lösung angeboten werden, welche die gesamte Logik in sich vereint.

A.1.2. Lieferumfang

Die Applikation wird in Form einer ZIP-Datei, sowie Source Code und Dokumentation ausgeliefert.

Funktionalität

Folgende Funktionen sollen implementiert werden:

1. MAB (MAC Authentication Bypass)
2. PEAPv0/EAP-MSCHAPv2 (Protected EAP)
3. Accounting
4. DM (Disconnect Message)
5. CoA (Change-of-Authorization)

[1] `http://freeradius.org`, letzter Zugriff 20.03.2015
[2] `http://www.cloudguard.ch/Platforms.html`, letzter Zugriff 20.03.2015

Die Implementierung erfolgt gemäss folgenden RFCs und Internet Drafts:

- RFC 2759 (Microsoft PPP CHAP Extensions, Version 2) [11]
- RFC 2865 (Remote Authentication Dial In User Service) [10]
- RFC 2866 (RADIUS Accounting) [9]
- RFC 2868 (RADIUS Attributes for Tunnel Protocol Support) [12]
- RFC 2869 (RADIUS Extensions) [11]
- RFC 3748 (PPP Extensible Authentication Protocol (EAP)) [1]
- RFC 5176 (Dynamic Authorization Extensions to RADIUS) [3]
- draft-josefsson-pppext-eap-tls-eap-02 (Protected EAP Protocol) [2]
- draft-kamath-pppext-peapv0-00 (Microsoft's PEAP version 0) [7]
- draft-kamath-pppext-eap-mschapv2-02 (Microsoft EAP CHAP Extensions) [6]

Für die Implementierung von PEAPv0/EAP-MSCHAPv2 wird zusätzlich die online Referenz[3] von Microsoft zu Hilfe gezogen.

Die aufgelisteten Dokumente werden nicht vollständig implementiert, da dies vom Projektpartner bewusst nicht gefordert wird. Die Applikation ist somit nicht konform mit diesen Standards. Im Mittelpunkt steht primär die Implementierung der o.g. Funktionen, die Erreichung einer hohen Performance und die Robustheit des Systems.

Optionale Funktionalität

Das Erreichen folgender Ziele wäre wünschenswert, steht aber nicht im Fokus:

1. Erstellen eines GUI für die Konfiguration

[3] `https://msdn.microsoft.com/en-us/library/cc238354.aspx`, letzter Zugriff 08.06.2015

A.2. Projektorganisation

Das Team besteht aus zwei Mitgliedern welche einander gleichgestellt sind.

A.2.1. Organisationsstruktur

Name: Filippo Pitrella
E-Mail: fpitrell@hsr.ch

Name: Michael Schefer
E-Mail: mschefer@hsr.ch

A.2.2. Externe Schnittstellen

Die Arbeit wird von Prof. Beat Stettler betreut. Die Ansprechsperson beim Projektpartner, CloudGuard Software AG, ist Herr Michael Schneider.

A.3. Management Abläufe

Wir haben uns dazu entschieden bei der Durchführung des Projektes prinzipiell nach Scrum vorzugehen. Aufgrund des kleinen Teams von nur zwei Personen und des daraus resultierenden kleineren Kommunikations- und Koordinationsaufwandes, werden allerdings einige Elemente von Scrum in einer vereinfachten Form durchgeführt. So werden das Daily Scrum Meeting und die Sprint-Retrospektive in die tägliche teaminterne Kommunikation eingebettet, anstatt sie in der von Scrum üblichen Form durchzuführen.

A.3.1. Kostenvoranschlag

Das Projekt beginnt am 16.02.2015 und hat einen Zeitrahmen von 17 Wochen. Der geschätzte Mindestaufwand pro Person und Woche beträgt 21 Stunden. Daraus ergeben sich bei 2 beteilig-

ten Personen 720 Stunden. Falls im Verlauf des Projektes festgestellt wird, dass die Soll-Ziele nicht erreicht werden, sind die Mitglieder bereit, ein tragbares Mass an Mehraufwand zu leisten. Wird dieses überschritten kommt es zu einer Kürzung der Feature-Liste.

A.3.2. Zeitliche Planung

Für unser Projekt haben wir eine Sprintlänge von 2 Woche gewählt. Die Ausnahme bildet hierbei der letzte Sprint, der nur 1 Woche dauert.

Sprints

Sprint	Start	Ende	Artefakte / Ziele
1	16.02.2015	27.02.2015	Projektplan
2	02.03.2015	13.03.2015	Anforderungsspezifikation
3	16.03.2015	27.03.2015	Domainanalyse, Prototyp
4	30.03.2015	10.04.2015	Architekturplanung, MAB implementiert
5	13.04.2015	24.04.2015	PEAPv0/EAP-MSCHAPv2 implementiert
6	27.04.2015	08.05.2015	Accounting implementiert
7	11.05.2015	22.05.2015	DM, CoA implementiert
8	25.05.2015	05.06.2015	Bugs gefixt, Abschlussdokumentation begonnen
9	08.06.2015	12.06.2015	Projekt abgeschlossen

A.3.3. Sprint Ablauf

Ein Sprint erfolgt im Normalfall gemäss folgendem Ablauf:

1. Sprint Planning
2. Tasks bearbeiten und abschliessen gemäss Definition of Done (siehe Kapitel A.3.4)
3. Sprint Review (erfolgt während dem wöchentlichen Meeting zu Beginn des nächsten Sprints)

A.3.4. Definition of Done

Ein Task gilt als abgeschlossen, wenn folgende Kriterien erfüllt sind:

1. Unit Tests geschrieben und erfolgreich ausgeführt
2. Code im Git-Repository eingecheckt
3. Jenkins Build und Tests erfolgreich durchgelaufen
4. Ticket auf Redmine als erledigt markiert

A.3.5. Besprechungen

Sitzungen finden wöchentlich, am Dienstag von 11:00 bis 12:00 Uhr statt. Teilnehmer sind jeweils: Prof. Beat Stettler (Betreuer), Herr Michael Schneider (Projektpartner), Michael Schefer und Filippo Pitrella. Allfällige Abwesenheiten werden in den Sitzungsprotokollen festgehalten, die der Betreuer und der Projektpartner jeweils nach einer Sitzung erhalten.

			Stand: 27.02.2015			Stand: 16.03.2015			Stand: 30.03.2015				
Nr	**Titel**	**Beschreibung**	**max. Schaden [h]**	**Eintrittsw'keit**	**Gewichteter Schaden**	**max. Schaden [h]**	**Eintrittsw'keit**	**Gewichteter Schaden**	**max. Schaden [h]**	**Eintrittsw'keit**	**Gewichteter Schaden**	**Vorbeugung**	**Verhalten beim Eintreten**
R1	Unvollständige Implementation	Das Radius Protokoll ist sehr umfangreich. Bei der Implementierung wurden Spezialfälle nicht berücksichtigt, oder falsch interpretiert.	20	10%	2	20	10%	2	20	10%	2	Requirements genau spezifizieren und vom Projektpartner abnehmen lassen. Zusätzlich sollen Unit-Tests die implementierte Funktionalität fortlaufend testen.	Fehlende Funktionalität implementieren.
R2	Ungenügende Performance	Die Applikation kann die Perfomanceanforderungen nicht erfüllen.	150	5%	7,5	150	5%	7,5	150	5%	7,5	Gut strukturierte Architektur	Thread-Model überarbeiten, Refactorings
R3	Fehlende Kenntnisse PEAP	Für die Implementation der Authentisierung mit PEAP wird mehr Zeit benötigt als geplant, weil die Komplexität aufgrund fehlender Kenntnisse des Protokolls falsch eingeschätzt wurde.	-	-	-	25	15%	3,75	25	15%	3,75	Studium der entsprechenden RFCs + Fachliteratur	Überstunden leisten / Planung anpassen
R4	Softwarefehler	Ein nicht reproduzierbarer Fehler in der Applikation tritt auf, dessen Ursache schwer zu finden ist.	40	5%	2	40	5%	2	40	5%	0	Unit Tests	Dozenten oder andere Mitstudenten um Rat bitten, Developer Foren
R5	Backup / Restore	Ausfall der VMs. Datenverlust in Redmine. Erhöhte Zeitinvestition beim Zurückspielen eines Backups + evtl. Rekonfiguration der Systeme.	4	1%	0,04	4	1%	0,04	4	0,75%	0,03	Git + regelmässige Backups auf externe Filesysteme	Aufarbeiten von letztem Backup
R6	Unvorhergesehene Risiken	Probleme die während der Entwicklung auftauchen und schwierig zu bewältigen sind.	20	10%	2	20	10%	2	2	10%	0	Gute, durchdachte Planung	Indivduelle Anpassung
R7	Krankheit	Ein Teammitglied fällt aus gesundheitlichen Gründen vorübergehend aus.	360	1%	3,6	300	1%	3	260	1%	2,6	Keine	Features streichen
Summe			**234**		**13,54**	**259**		**17,29**	**241**		**13,28**		

			Stand: 27.04.2015			Stand: 11.05.2015				
Nr	**Titel**	**Beschreibung**	**max. Schaden [h]**	**Eintrittsw'keit**	**Gewichteter Schaden**	**max. Schaden [h]**	**Eintrittsw'keit**	**Gewichteter Schaden**	**Vorbeugung**	**Verhalten beim Eintreten**
R1	Unvollständige Implementation	Das Radius Protokoll ist sehr umfangreich. Bei der Implementierung wurden Spezialfälle nicht berücksichtigt, oder falsch interpretiert.	20	10%	2	20	10%	2	Requirements genau spezifizieren und vom Projektpartner abnehmen lassen. Zusätzlich sollen Unit-Tests die implementierte Funktionalität fortlaufend testen.	Fehlende Funktionalität implementieren.
R2	Ungenügende Performance	Die Applikation kann die Perfomanceanforderungen nicht erfüllen.	150	5%	7,5	150	5%	7,5	Gut strukturierte Architektur	Thread-Model überarbeiten, Refactorings
R3	Fehlende Kenntnisse PEAP	Für die Implementation der Authentisierung mit PEAP wird mehr Zeit benötigt als geplant, weil die Komplexität aufgrund fehlender Kenntnisse des Protokolls falsch eingeschätzt wurde.	25	100%	25	20	100%	20	Studium der entsprechenden RFCs + Fachliteratur	Überstunden leisten / Planung anpassen
R4	Softwarefehler	Ein nicht reproduzierbarer Fehler in der Applikation tritt auf, dessen Ursache schwer zu finden ist.	40	5%	2	40	5%	2	Unit Tests	Dozenten oder andere Mitstudenten um Rat bitten, Developer Foren
R5	Backup / Restore	Ausfall der VMs. Datenverlust in Redmine. Erhöhte Zeitinvestition beim Zurückspielen eines Backups + evtl. Rekonfiguration der Systeme.	4	1%	0,04	4	1%	0,04	Git + regelmässige Backups auf externe Filesysteme	Aufarbeiten von letztem Backup
R6	Unvorhergesehene Risiken	Probleme die während der Entwicklung auftauchen und schwierig zu bewältigen sind.	20	10%	2	20	10%	2	Gute, durchdachte Planung	Indivduelle Anpassung
R7	Krankheit	Ein Teammitglied fällt aus gesundheitlichen Gründen vorübergehend aus.	200	1%	2	140	1%	1,4	Keine	Features streichen
R8	Fehlende Kenntnisse CoA	Für die Implementation von CoA wird mehr Zeit benötigt als geplant, weil die Komplexität aufgrund fehlender Kenntnisse falsch eingeschätzt wurde.	10	10%	1	8	25%	2	Studium der entsprechenden RFCs + Fachliteratur	Feature streichen
Summe			**259**		**38,54**	**254**		**33,54**		

A.5. Tasks

Im Folgenden sind alle Tasks (aufgeteilt in einzelne Sprints) aufgelistet.

A.5.1. Sprint 1

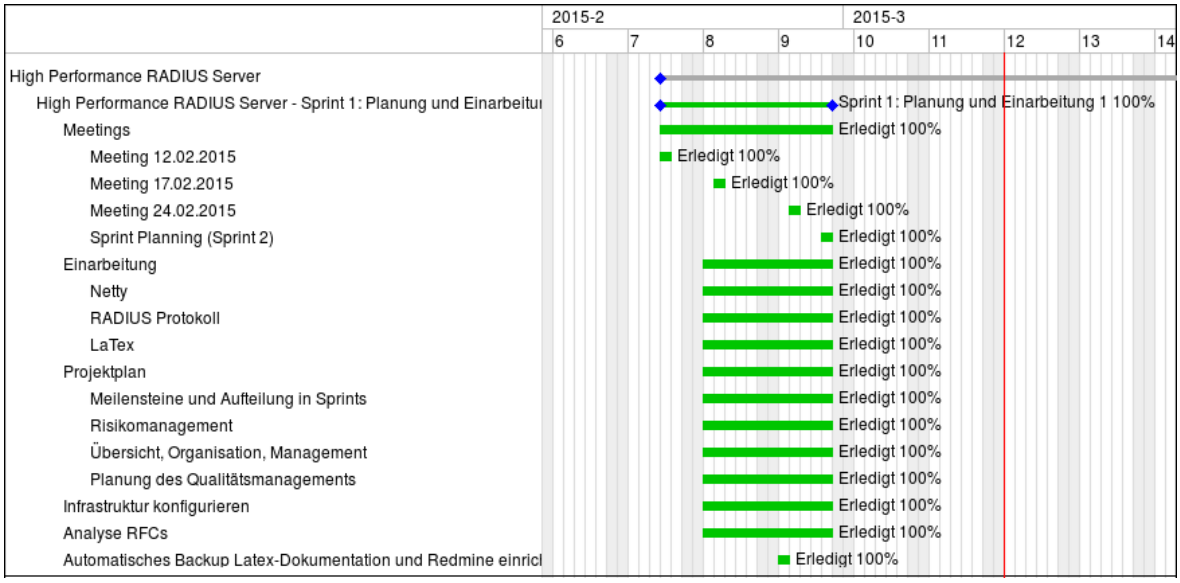


Abbildung A.1.: Sprint 1

A.5.2. Sprint 2

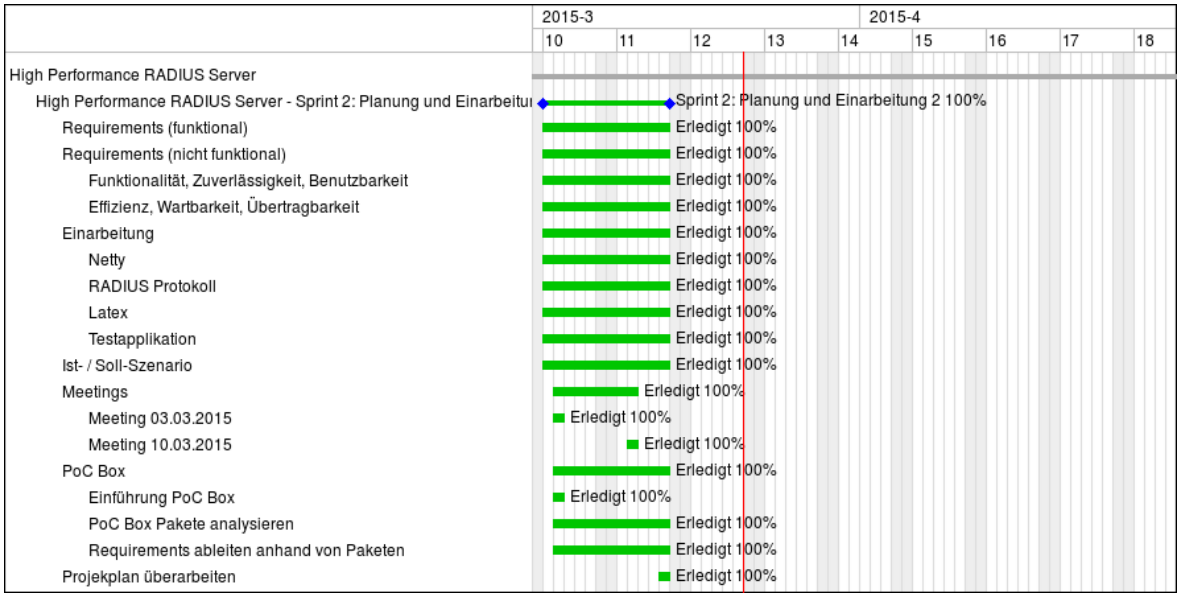


Abbildung A.2.: Sprint 2

A.5.3. Sprint 3

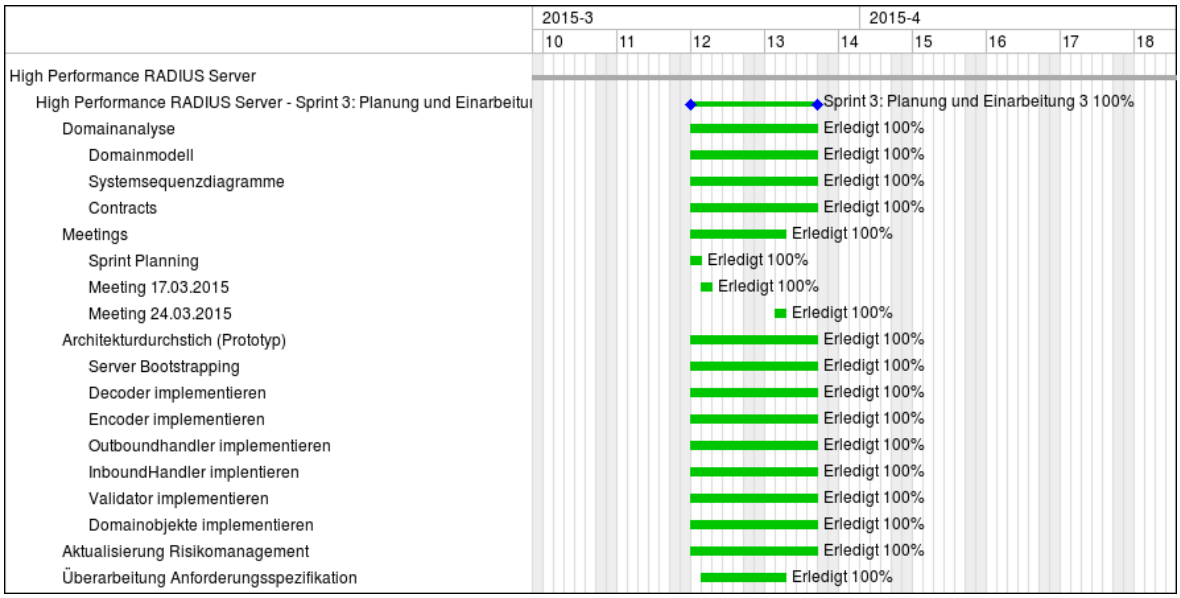


Abbildung A.3.: Sprint 3

A.5.4. Sprint 4

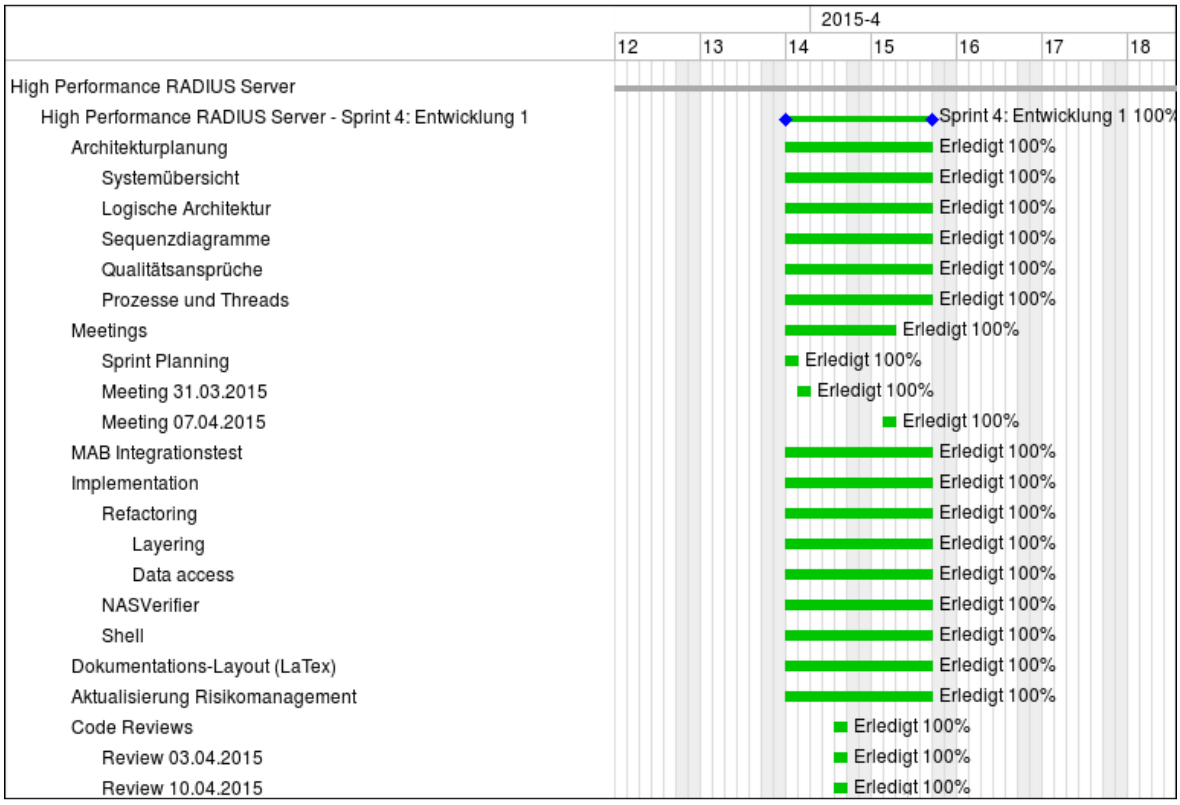


Abbildung A.4.: Sprint 4

A.5.5. Sprint 5

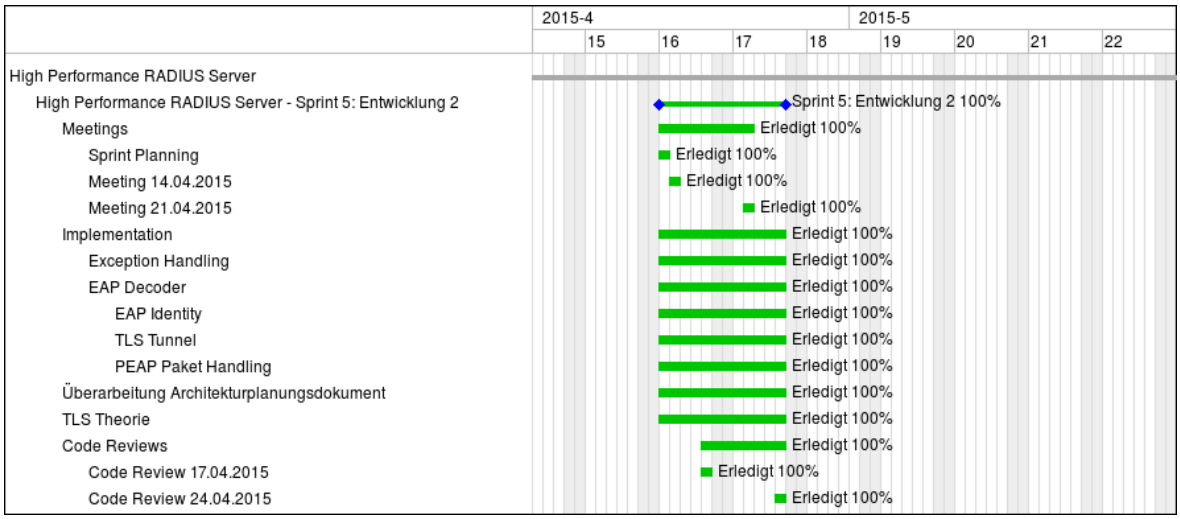


Abbildung A.5.: Sprint 5

A.5.6. Sprint 6

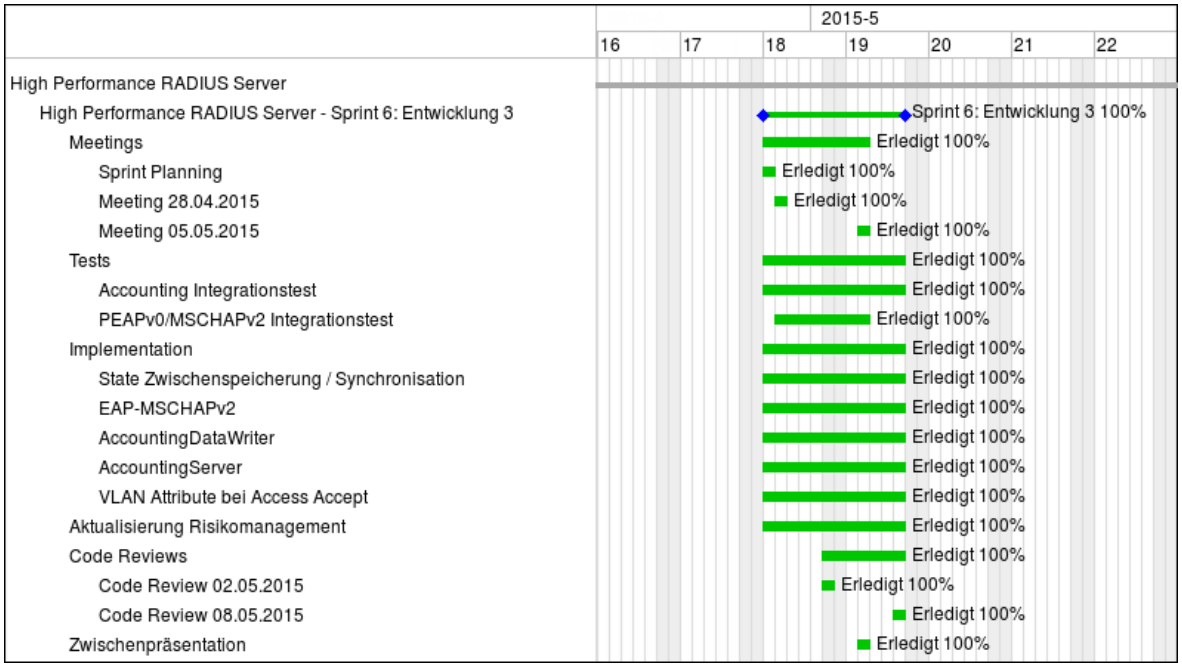


Abbildung A.6.: Sprint 6

A.5.7. Sprint 7

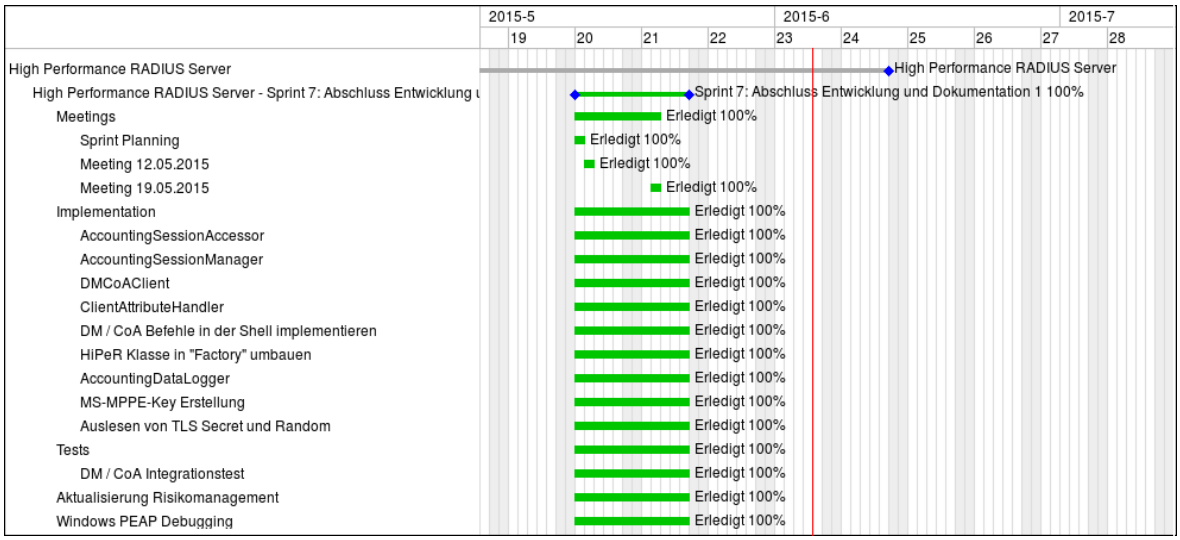


Abbildung A.7.: Sprint 7

A.5.8. Sprint 8

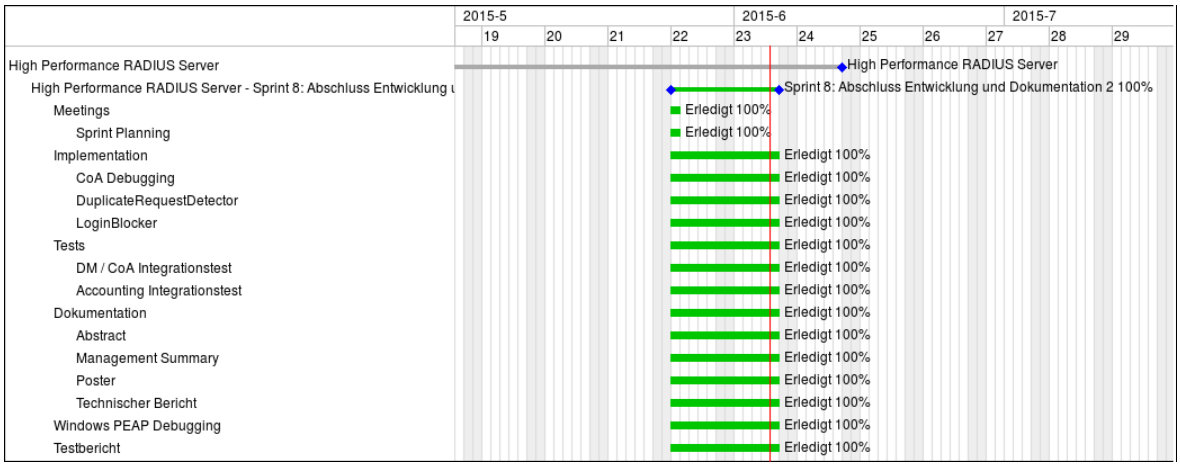


Abbildung A.8.: Sprint 8

A.5.9. Sprint 9

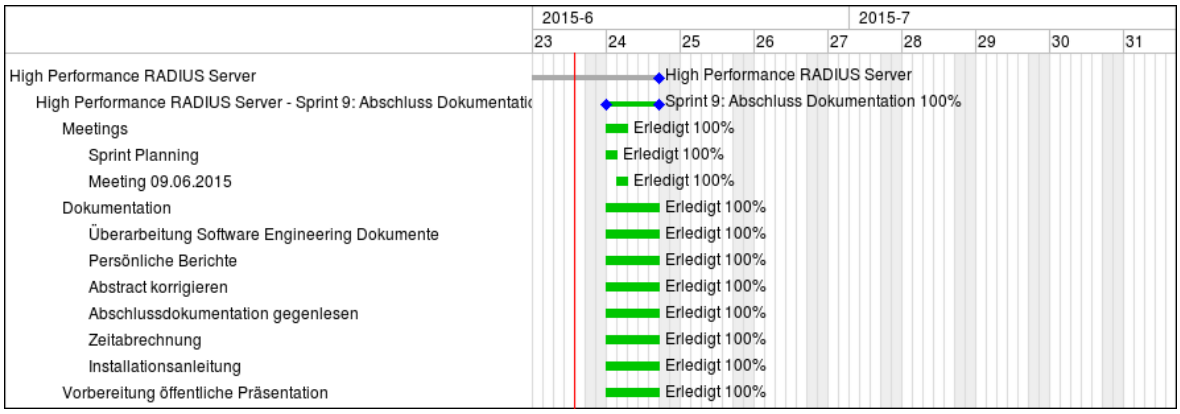


Abbildung A.9.: Sprint 9

A.6. Infrastruktur

A.6.1. Entwicklung

Die Entwickler arbeiten mit ihren persönlichen Notebooks (Windows 8.1). Als Entwicklungsumgebung wird Eclipse (Version Luna) und das Java SE Development Kit 8 eingesetzt.

A.6.2. Testgeräte

Für Tests steht eine PoC-Box zur Verfügung, welche verschiedene Netzwerkgeräte enthält. Damit kann ein reales System simuliert, und die Interaktion von HiPeR mit einem Network Access Server (NAS) getestet werden.

Folgendende Komponenten sind in der PoC-Box enthalten:

- Cisco Catalyst 3560-CG Series PoE Switch
- Cisco Aironet 2700 Series 802.11ac Dual Band Access Point
- Cisco 2500 Series Wireless Controller
- HP ProLiant DL320e Gen8 v2 Server

A.6.3. Tools

- Eclipse - Programmierumgebung
- Maven* - Build-Management
- Git* - Versionsverwaltung
- Astah Community - Diagramme
- ShareLatex - Dokumentation

- Redmine* - Projektverwaltung
- Jenkins* - Build- und Testserver

A.6.4. Server

Zur Projektverwaltung steht ein virtueller Server (Ubuntu 12.04 LTS) zur Verfügung. Die in Kapitel A.6.3 mit Stern * markierten Tools sind darauf installiert.

A.7. Qualitätsmassnahmen

A.7.1. Dokumentation

Die Dokumentation wird mit dem online LaTeX Editor ShareLatex[4] erstellt. Änderungen an den Dokumenten werden automatisch jede Stunde in ein Git-Repository gepusht (unter: `https://git.hsr.ch/git/BA/Sharelatex-Dokumentation` erreichbar). Falls ein Dokument nicht von beiden Teammitgliedern zusammen verfasst wird, wird es zur Qualitätskontrolle vom Anderen gegengelesen.

A.7.2. Projektmanagement

Für die Projektverwaltung und als Bug-Tracker verwenden wir Redmine, welches unter der Adresse `http://sinv-56011.edu.hsr.ch/redmine` erreichbar ist. Ein Gast Benutzer ist eingerichtet (Benutzername und Passwort: "guest").

A.7.3. Entwicklung

Für die Versionsverwaltung verwenden wir Git. Das Repository befindet sich auf dem Git-Server der HSR unter `https://git.hsr.ch/git/BA/HiPeR`. Features werden generell in Feature-Branches implementiert. Erst wenn ein Feature vollständig abgeschlossen und fehlerfrei ist, wird es in den Master-Branch eingefügt. Für die Sicherstellung der Codequalität setzen wir verschiedene Massnahmen ein (siehe nachfolgende Punkte).

Reviews

Code Reviews werden fortwährend durchgeführt. Sobald der Entwickler ein Task abgeschlossen hat, markiert er das Ticket als "gelöst" und weist es dem Reviewer zu. Falls dieser keine Mängel oder Fehler im Code findet, markiert er das Ticket als "erledigt". Ansonsten wird es nochmals dem Entwickler zur Korrektur zugewiesen.

Code Style Guidelines

Als Code Style Guidelines werden die Standard-Java Guidelines von Sun / Oracle verwendet[5].

[4] `http://www.sharelatex.com`

[5] `http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html`, letzter Zugriff 24.02.2015

A.7.4. Testen

Alle relevanten Aspekte des Projekts sollen durch Unit-Tests abgedeckt werden. Diese werden vom Entwickler fortlaufend geschrieben. Änderungen werden erst dann ins Git-Repository übertragen, wenn lokal alle Tests erfolgreich durchlaufen. Zusätzlich werden weitere funktionale Tests mit RADIUS Client Simulatoren durchgeführt, wo anfänglich die Grundfunktionalität und später Belastungsspitzen, langlebende Benutzer-Sessions, wechselnde Benutzerrechte, etc. getestet werden können.

B. Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- das wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil, 12.06.2015

Michael Schefer

Filippo Pitrella

C. RADIUS Grundlagen

C.1. Einführung

Das RADIUS Protokoll ermöglicht Authentifizierung, Autorisierung und Accounting (AAA) von Benutzern, welche sich mit einem Netzwerk verbinden wollen. RADIUS verwendet UDP als Transport-Protokoll. Wichtige Features von RADIUS sind folgende:

Client/Server Model
Ein Network Access Server (NAS) fungiert als Client eines RADIUS Servers. Der Client ist Verantwortlich für die Übertragung von Informationen an designierte RADIUS Server und muss auf die Antwort reagieren.

RADIUS Server sind verantwortlich für das Empfangen von Anfragen, die Authentifizierung von Benutzern und das Versenden von allen Informationen welche der NAS benötigt, um dem Benutzer den gewünschten Service zu ermöglichen.

Netzwerk Sicherheit
Transaktionen zwischen dem Client dem RADIUS Server werden durch ein Shared Secret authentifiziert. Das Secret wird nie über das Netzwerk versendet. Zusätzlich werden Passwörter immer verschlüsselt übertragen.

Flexible Authentifizierung
Ein RADIUS Server kann verschiedene Methoden zu Authentifizierung eines Benutzers unterstützen. Unter anderem PPP PAP oder CHAP, UNIX Login, EAP und weitere Mechanismen.

Erweiterbares Protokoll
Neue Attribute können hinzugefügt werden, ohne bestehende Implementierungen des Protokolls zu beinflussen.

C.2. Authentifizierung

C.2.1. Grundlegender Ablauf

Zu Beginn der Authentifizierung sendet der NAS einen Access Request an den RADIUS Server. Dieser Request enthält Zugangsdaten, typischerweise in der Form von Benutzernamen und Passwort. Zusatzlich kann der Request weitere Informationen enthalten, wie z. B. die MAC-Adresse des Benutzers oder der physikalische Ort an dem der Benutzer mit dem NAS verbunden ist. Der RADIUS Server überprüft die Anfrage und sendet eine der drei möglichen Antworten:

- Access Reject - Der Benutzer erhält keinen Zugriff auf alle angeforderten Netzwerk-Ressourcen.
- Access Challenge - Fordert zusätzliche Informationen vom Benutzer wie z. B. ein sekundäres Passwort. Access Challenge Pakete werden auch in komplexeren Authentifizierungsmethoden wie beispielsweise PEAPv0/EAP-MSCHAPv2 verwendet.

- Access Accept - Der Benutzer erhält Zugriff auf das Netzwerk. Ein Access Accept Paket kann verschiedene Attribute enthalten wie z. B. die IP Adresse des Benutzers oder VLAN Parameter.

C.2.2. MAB

Die Authentifizierung mit MAC Authentication Bypass ist für Geräte gedacht, welche keine komplexere Form der Authentifizierung unterstützen.

Bei einem Access Request werden dabei die drei Attribute User-Name, User-Password und Calling-Station-ID übermittelt. Alle drei Attribute beinhalten dabei die MAC-Adresse des Benutzers. Der RADIUS Server überprüft die Attribute und antwortet mit einem Access Accept oder Access Reject.

C.2.3. PEAPv0/EAP-MSCHAPv2

PEAPv0/EAP-MSCHAPv2 ist eine Form der Authentifizierung welche erhöhte Sicherheit durch die Verwendung einer mit TLS verschlüsselten Verbindung bietet. Dabei wird das Extensible Authentication Protocol (EAP) für die Übertragung verwendet. Dieses unterstützt verschiedene Authentifizierungsmechanismen und erlaubt die Kapselung von Protokollen.

Wie in Abbildung C.1 ersichtlich ist, kommuniziert der RADIUS Server über EAP direkt mit dem Gerät des Benutzers (nachfolgend als Peer bezeichnet). Der NAS leitet die EAP Pakete lediglich weiter und muss sie nicht interpretieren können.

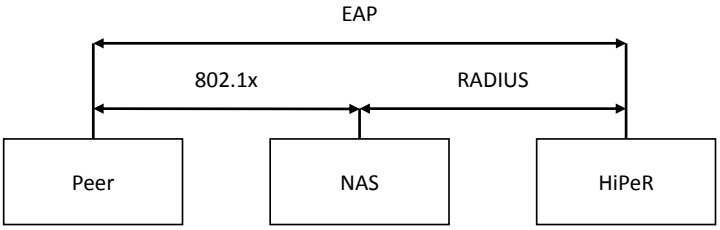


Abbildung C.1.: EAP Übersicht

EAP Pakete werden zwischen dem NAS und dem RADIUS Server in EAP-Message Attributen übertragen. Weil ein RADIUS Attribut maximal 255 Byte gross sein kann, ist es möglich, dass ein einzelnes EAP Paket auf mehrere EAP-Message Attribute aufgeteilt wird. Der RADIUS Server muss diese beim Empfang wieder zusammensetzen. Die Authentifizierung mit PEAPv0/EAP-MSCHAPv2 erfolgt in zwei Phasen:

Phase 1
In der ersten Phase wird innerhalb von EAP ein TLS Tunnel aufgebaut. Die Phase beginnt typischerweise damit, dass der Peer ein EAP Identity Paket an den RADIUS Server übermittelt. Dieser antwortet mit einem PEAP-Start Paket. Ein PEAP Paket ist ein EAP Paket mit EAP-Type=PEAP.

Anschliessend werden mehrere Pakete ausgetauscht, welche TLS Handshake Informationen enthalten. Die TLS Daten werden dabei innerhalb von PEAP Paketen übertragen.

Wenn der Handshake erfolgreich durchgeführt wurde, sendet der Peer ein leeres PEAP Paket an den RADIUS Server. Ansonsten wird ein EAP Failure Paket versendet und die Authentifizierung beendet. Damit ist Phase 1 abgeschlossen.

Phase 2
In Phase 2 erfolgt innerhalb des TLS Tunnels eine zweite komplette EAP Konversation. Die EAP Pakete werden dabei leicht komprimiert, indem die ersten 4 Byte nicht übertragen werden, weil diese von den äusseren EAP Paketen übernommen werden können.

Zu Beginn sendet der RADIUS Server einen EAP-Identity Request, welcher vom Peer mit einer EAP-Identity Response beantwortet wird. Die übertragene Identität kann eine andere sein als in Phase 1.

Danach erfolgt eine EAP-MSCHAPv2 Konversation. Diese setzt sich aus folgenden Pakete zusammen:

- EAP-MSCHAPv2 Challenge - Beginnt die MSCHAPv2 Konversation.
- EAP-MSCHAPv2 Response - Enthält das Passwort des Benutzers in verschlüsselter Form.
- EAP-MSCHAPv2 Success Request - Wird versendet, falls das Passwort korrekt ist und enthält einen Authenticator Response String welcher verifiziert werden muss.
- EAP-MSCHAPv2 Success Response - Wird versendet, wenn der Authenticator Response String aus dem Success Request erfolgreich verifiziert wurde.
- EAP-MSCHAPv2 Failure Request - Wird versendet, falls das Passwort falsch ist, oder der Benutzer keinen Zugriff erhalten soll.

Anschliessend wird Ein EAP Result Extension Paket vom RADIUS Server an den Peer versendet, welches das Resultat der EAP Konversation enthält (Success oder Failure). Der Peer antwortet darauf mit einer EAP Notification mit demselben Inhalt.

Zuletzt wird ein Access Reject oder Access Accept versendet. Bei einem Access Accept müssen die beiden Attribute MS-MPPE-Recv-Key und MS-MPPE-Send-Key versendet werden, welche aus den TLS Session Daten berechnet werden. Damit ist die Authentifizierung abgeschlossen.

Abbildung C.2 zeigt den kompletten Ablauf einer Authentifizierung mit PEAPv0/EAP-MSCHAPv2. Es werden alle Pakete dargestellt die für den RADIUS Server relevant sind. Zwischen dem NAS und dem Peer können weitere Pakete ausgetauscht werden, welche hier nicht gezeigt werden.

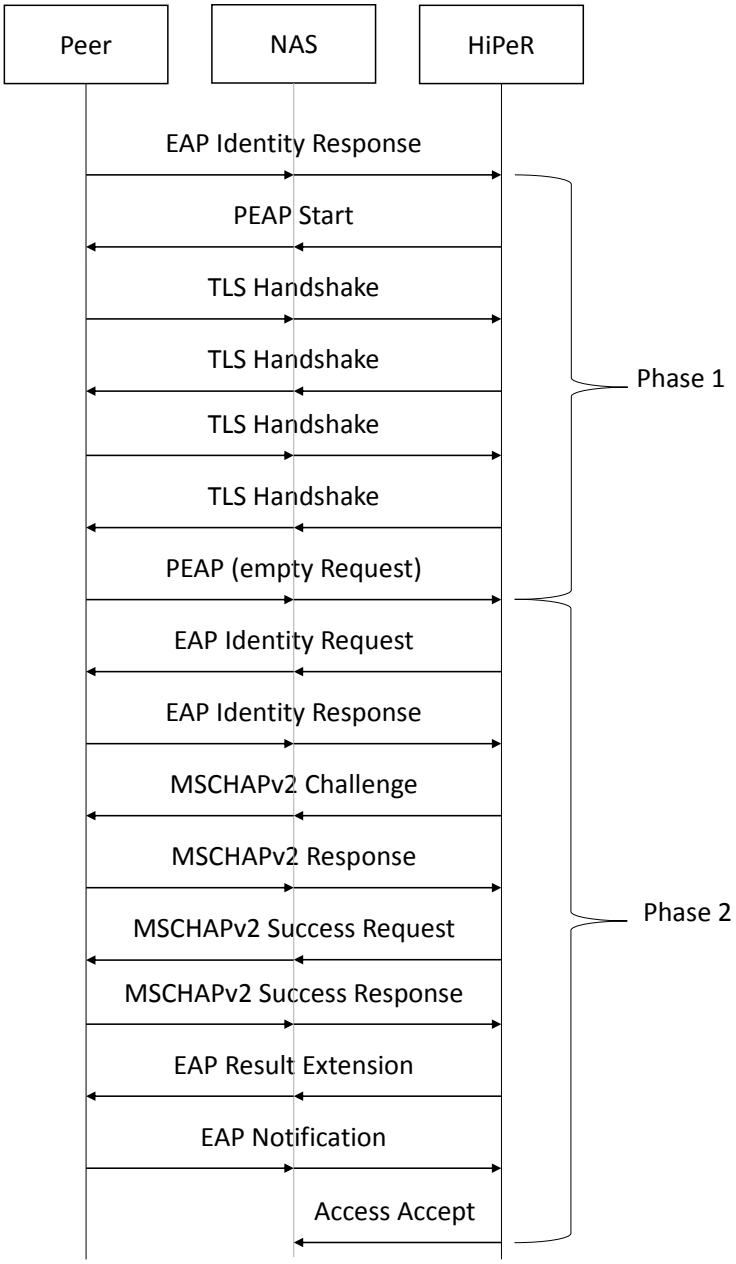


Abbildung C.2.: PEAPv0/EAP-MSCHAPv2 Ablauf

C.3. Accounting

Das Accounting wird verwendet um Statistiken zu erfassen. Der NAS sendet dazu Accounting Request Pakete an den RADIUS Server welche mit Accounting Response Paketen beantwortet werden.

Accounting Pakete werden versendet, wenn sich ein Benutzer verbindet (Accounting Start), wenn sich ein Benutzer trennt (Accounting Stop) oder zwischendurch (Interim Update).

C.4. DM / CoA

Disconnect Messages (DM) und Change-of-Authorization (CoA) Requests werden von einem Dynamic Authorization Client (DAC) verschickt. Der DAC kann zusammen mit dem RADIUS Authentication oder Accounting Server betrieben werden, muss aber nicht zwingend.

C.4.1. DM

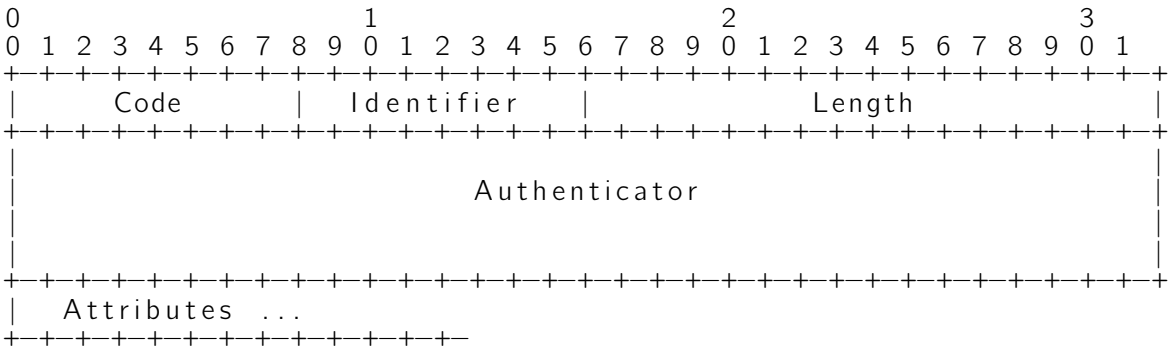
Disconnect Messages werden verschickt, um Benutzersessions zu terminieren und alle dazugehörigen Sessiondaten zu löschen. Disconnect Requests enthalten sowohl Client-Identifikations-Attribute als auch NAS-Identifikations-Attribute. Der NAS welcher einen Disconnect Request empfängt, antwortet entweder mit einem Disconnect-ACK oder mit einem Disconnect-NAK. Ein Disconnect-ACK wird nur dann versendet, wenn alle Benutzersessions identifiziert, terminiert und alle dazugehörigen Sessiondaten gelöscht werden konnten. In jedem anderen Fall versendet der NAS ein Disconnect-NAK.

C.4.2. CoA

CoA-Requests werden verschickt um, eine Autorisierungsänderung an einer oder mehreren Sessions vorzunehmen. CoA-Request Pakete enthalten neben Client-Identifikations-Attributen und NAS-Identifikations-Attributen, auch Attribute für die Autorisierungsänderung. Typischerweise sind das Filter Attribute, die den Namen eines Datenfilters enthalten, welcher auf eine oder mehrere Sessions angewendet werden soll.

C.5. Paket Aufbau

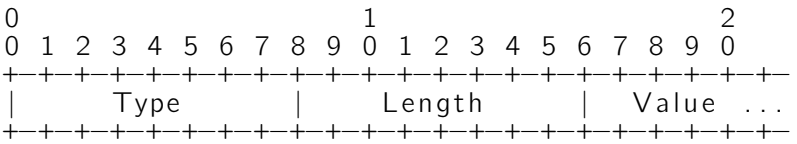
C.5.1. RADIUS Paket



Listing C.1: RADIUS Paket Aufbau

Code	1 Byte	Identifiziert den Typ des RADIUS Pakets
Identifier	1 Byte	Wird verwendet um zusammengehörende Request und Reply Pakete zu identifizieren
Length	2 Byte	Komplette Länge des RADIUS Pakets
Authenticator	16 Byte	Wird verwendet um die Antwort des RADIUS Servers zu verifizieren, sowie um Passwörter zu verschlüsseln

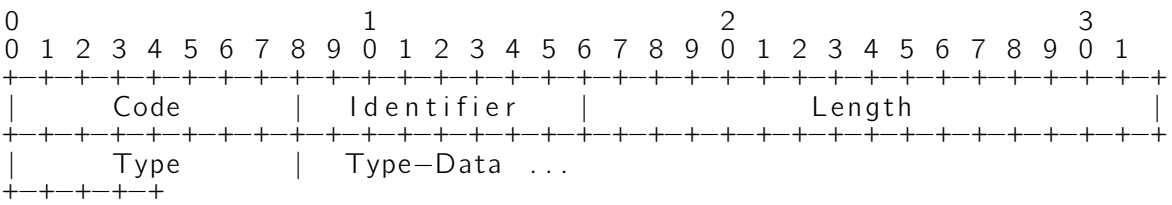
C.5.2. RADIUS Attribut



Listing C.2: RADIUS Attribut Aufbau

Type	1 Byte	Identifiziert den Typ des RADIUS Attributs
Length	1 Byte	Komplette Länge des RADIUS Attributs
Value	0-255 Byte	Inhalt des RADIUS Attributs

C.5.3. EAP Paket



Listing C.3: EAP Paket Aufbau

Code	1 Byte	Identifiziert den Typ des EAP Pakets (Request, Response, Success, Failure)
Identifier	1 Byte	Wird verwendet um zusammengehörende Request und Reply Pakete zu identifizieren
Length	2 Byte	Komplette Länge des EAP Pakets
Type	1 Byte	Identifiziert den Typ von Request oder Response
Type-Data	variabel	Enthält Daten abhängig vom Typ von Request oder Response

D. Persönlicher Bericht Michael Schefer

Die Realisierung der Bachelorarbeit war für mich eine interessante und herausfordernde Erfahrung. Da ich mich vorher weder mit dem RADIUS Protokoll, noch mit dem verwendeten Framework Netty auseinander gesetzt habe, war der Einarbeitungsaufwand zu Beginn zwar relativ hoch, ich konnte dadurch aber auch viel Neues lernen.

Die Implementation des Servers war sehr spannend und zeitweise auch ein wenig frustrierend. Bei der Authentifizierung mit PEAP ist nach jedem gelösten Problem sofort wieder ein neues aufgetaucht. Schrittweise konnten diese Probleme jedoch bis auf die Unterstützung von Windows Clients alle gelöst werden.

Die Zusammenarbeit mit Filippo empfand ich als sehr angenehm. Da wir bereits die Semesterarbeit zusammen geschrieben haben, sind wir ein eingespieltes Team. Wir konnten uns gegenseitig viele Tipps und Vorschläge geben und uns auf Fehler aufmerksam machen. Die Kommunikation in einem kleinen Team war sehr einfach und hat die Planung und Organisation erleichtert.

Wieder einmal musste ich im späteren Verlauf des Projektes feststellen, dass der Dokumentationsaufwand nicht unterschätzt werden darf. Da die Implementation von PEAP deutlich länger gedauert hat als geplant, haben wir zwischenzeitlich die Dokumentation etwas vernachlässigt. Dies mussten wir durch einen erhöhten zeitlichen Aufwand in den letzten zwei Wochen kompensieren.

Ich hatte sehr viel Spass bei der Umsetzung des Projektes und fühle mich bei der Wahl meiner Studienrichtung bestätigt.

E. Persönlicher Bericht Filippo Pitrella

Die Vorfreude auf die Arbeit war gross, da Michael und ich glücklicherweise unser Wunschthema zugeteilt bekamen. Am Anfang hatte ich allerdings Mühe einzuschätzen, was uns erwarten wird. Das lag daran, dass diese Bachelorabeit klar im Netzwerksegment angesiedelt ist, aber dennoch keine vertieften Netzwerkkenntnisse vorausgesetzt wurden. Im Nachhinein lässt sich das wohl nur mit dem eingesetzten Netzwerk Framework Netty erklären, welches die Netzwerkkommunikation in Java stark abstrahiert.

Die Zusammenarbeit mit unserem Projektpartner war interessant und hat mir viel Freude bereitet. Die unkomplizierte und hilfsbereite Art von Herrn Schneider schätze ich sehr. Seine nützlichen Inputs während den wöchentlichen Sitzungen haben uns Wege aufgezeigt, auf die wir vermutlich nicht gekommen wären. Bei der Wahl von Technologien waren wir - bis auf die Verwendung von Netty - grundsätzlich frei. Das kam uns sehr entgegen.

Auch die Zusammenarbeit mit Prof. Beat Stettler empfand ich als sehr angenehm. Seine Hilfestellungen während des Semesters, haben uns geholfen unsere Abschlussarbeit zu verbessern.

Ich bin froh darüber, dass ich auch diese Arbeit wieder mit Michael durchführen konnte. Man merkt, dass wir mittlerweile ein eingespieltes Team sind. Wir helfen uns gegenseitig wo wir können. Nur beim Dokumentieren werden wir vermutlich keine Weltmeister.

Die Einarbeitungsphase war sehr stark vom Studium RADIUS-spezifischer RFCs geprägt. Diese sind bekanntlich trocken geschrieben und deshalb mühsam zu lesen. Zusätzlich mussten wir uns auch in Netty einarbeiten, was allerdings wesentlich einfacher war. Da Michael und ich, den Wunsch hatten die Dokumentation zum ersten Mal in LaTex zu verfassen, hatten wir uns auch damit stark auseinandergesetzt. Die gleichzeitige Einarbeitung in so vielen neuen Themengebieten führte zu einer steilen Lernkurve und war für mich zeitweise Fluch und Segen zugleich.

Die Entwicklungsphase fand ich mit Abstand am interessantesten. Manchmal ist es sinnlos hunderte von Seiten zu lesen, ohne das Gelesene einmal anzuwenden. Viele Fragen die beim Lesen der Literatur aufkamen, konnten durch das nachträgliche Implementieren beantwortet werden. Mein heutiger Kenntnisstand über das Netty Framework und das RADIUS Protokoll, habe ich dieser Arbeit zu verdanken. Vorher waren meine Kenntnisse in diesen Bereichen inexistent bzw. nur marginal vorhanden.

Mit dem Ergebnis der Arbeit bin ich grundsätzlich zufrieden. Da mir das Projekt mittlerweile am Herzen liegt, gibt es natürlich auch Punkte die ich am liebsten verbessern würde. Worauf ich gar nicht stolz bin, ist die schwache Testabdeckung. Wir hatten geplant von Anfang an regelmässig Tests zu schreiben. Mit dem wachsenden Druck die fehlenden Use Cases zu implementieren, gelang uns das aber immer weniger. Ich bin allerdings dennoch der Meinung, dass wir mit HiPeR eine solide Basis für einen RADIUS Server gebaut haben, die offen ist für zukünftige Erweiterungen.

F. Zeiterfassung

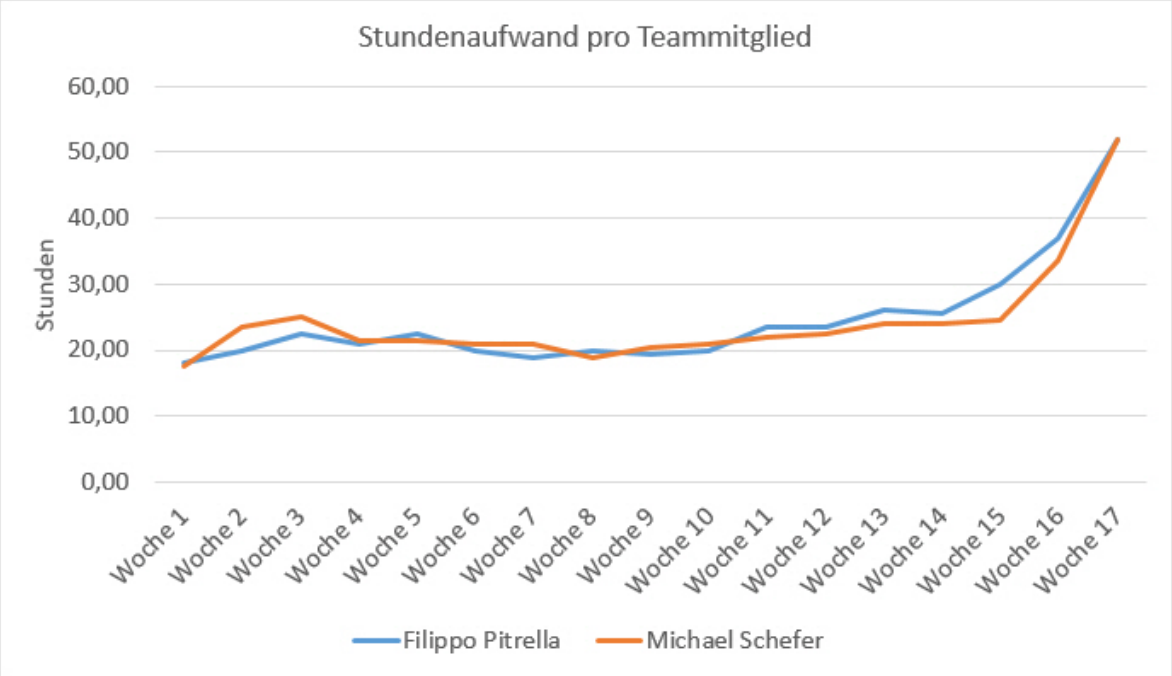


Abbildung F.1.: Stundenaufwand pro Teammitglied in Stunden

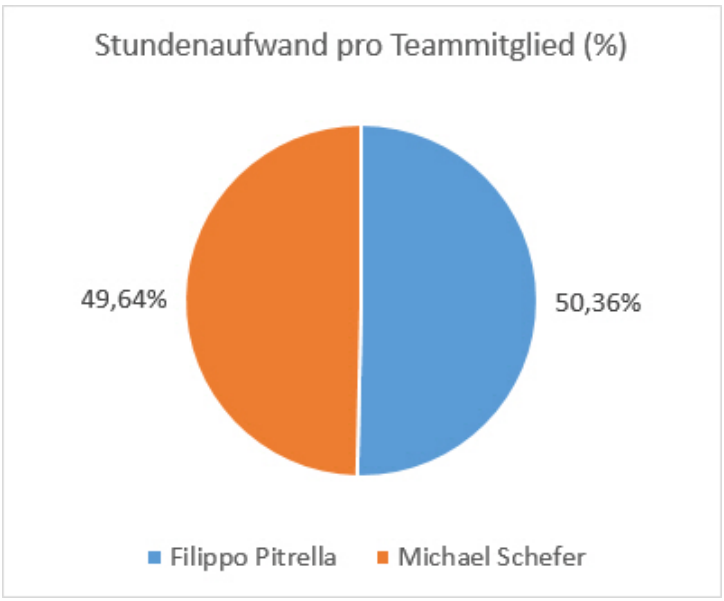


Abbildung F.2.: Stundenaufwand pro Teammitglied in Prozent

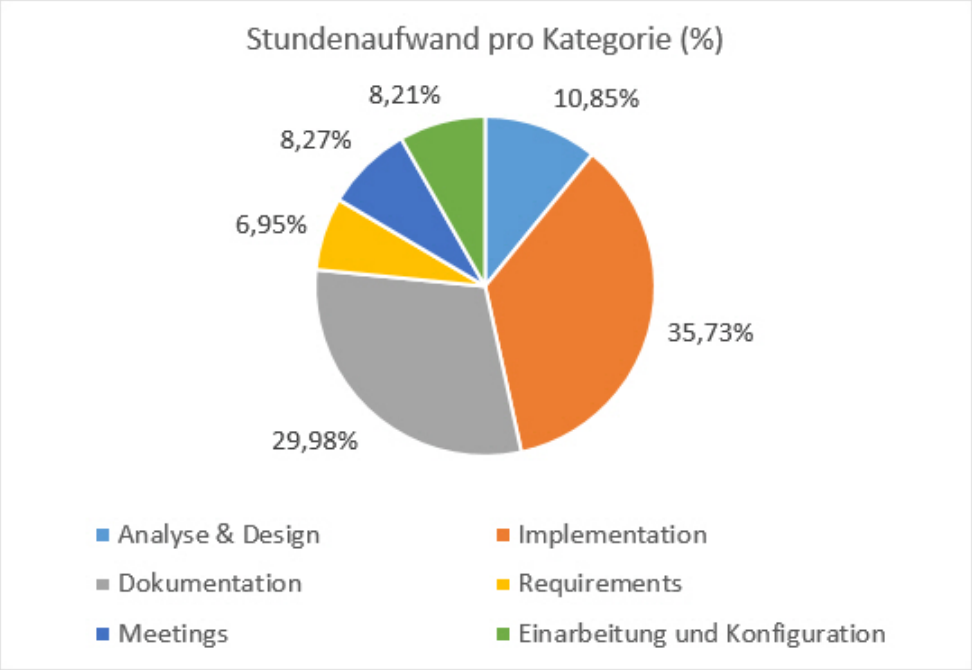


Abbildung F.3.: Stundenaufwand pro Kategorie in Prozent

G. Dependencies

In Kapitel 4.7 wurde bereits eine Abbildung mit allen Libraries welche von HiPeR direkt verwendet werden dargestellt. Diese können weitere Abhängigkeiten haben. Listing G.1 zeigt eine vollständige Übersicht aller Libraries welche von HiPeR direkt oder indirekt benötigt werden.

```
--- maven-dependency-plugin:2.10:tree (default-cli) @ hiper
ch.cloudguard.hiper:hiper:jar:0.0.1-SNAPSHOT
+- junit:junit:jar:4.12:test
|  \- org.hamcrest:hamcrest-core:jar:1.3:test
+- ch.qos.logback:logback-classic:jar:1.1.3:compile
|  \- ch.qos.logback:logback-core:jar:1.1.3:compile
+- io.netty:netty-all:jar:4.1.0.Beta3:compile
+- org.springframework.shell:spring-shell:jar:1.1.0.
|  RELEASE:compile
|  +- commons-io:commons-io:jar:2.3:compile
|  +- jline:jline:jar:2.11:compile
|  +- com.google.guava:guava:jar:15.0:compile
|  +- cglib:cglib:jar:2.2.2:compile
|  |  \- asm:asm:jar:3.3.1:compile
|  +- org.springframework:spring-context-support:jar:4.0.3.
|  |  RELEASE:compile
|  |  \- org.springframework:spring-beans:jar:4.0.3.
|  |     RELEASE:compile
|  \- org.springframework:spring-core:jar:4.0.3.RELEASE:
|     compile
|     \- commons-logging:commons-logging:jar:1.1.3:compile
+- org.slf4j:slf4j-api:jar:1.7.12:compile
+- org.slf4j:jcl-over-slf4j:jar:1.7.12:compile
+- org.tinyradius:tinyradius:jar:1.0:test
+- net.jradius:jradius-core:jar:1.1.4:test
|  +- org.springframework:spring-context:jar:3.0.5.RELEASE:
|  |  compile
|  |  +- org.springframework:spring-aop:jar:3.0.5.RELEASE:
|  |  |  compile
|  |  |  \- aopalliance:aopalliance:jar:1.0:compile
|  |  +- org.springframework:spring-expression:jar:3.0.5.
|  |  |  RELEASE:compile
|  |  \- org.springframework:spring-asm:jar:3.0.5.RELEASE:
|  |     compile
|  +- commons-configuration:commons-configuration:jar:1.5:
|  |  test
|  |  +- commons-collections:commons-collections:jar:3.2:
|  |  |  test
|  |  +- commons-lang:commons-lang:jar:2.3:test
|  |  +- commons-digester:commons-digester:jar:1.8:test
|  |  |  \- commons-beanutils:commons-beanutils:jar:1.7.0:
|  |  |     test
|  |  \- commons-beanutils:commons-beanutils-core:jar:
|  |     1.7.0:test
|  +- commons-chain:commons-chain:jar:1.2:test
|  +- commons-pool:commons-pool:jar:1.5.4:test
|  +- net.sf.ehcache:ehcache:pom:2.2.0:test
|  |  +- net.sf.ehcache:ehcache-core:jar:2.2.0:test
|  |  +- net.sf.ehcache:ehcache-terracotta:jar:2.2.0:test
|  |  \- org.terracotta:terracotta-toolkit-1.0-runtime:
```

```
51 |  |  |     jar:1.0.0:test
52 |  |  +- org.gnu:java-getopt:jar:1.0.13:test
53 |  |  \- org.gnu:gnu-crypto:jar:2.0.1:test
54 |  +- net.jradius:jradius-dictionary:jar:1.1.4:test
55 |  +- net.jradius:jradius-extended:jar:1.1.4:test
56 |  \- org.bouncycastle:bcprov-jdk15:jar:1.44:test
```

Listing G.1: Maven Dependencies

Literaturverzeichnis

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004. Updated by RFCs 5247, 7057. http://www.ietf.org/rfc/rfc3748.txt.

[2] H. Andersson, S. Josefsson, Glen Zorn, and Dan Simon. Protected EAP Protocol (PEAP). Working Draft, February 2002. http://tools.ietf.org/pdf/draft-josefsson-pppext-eap-tls-eap-02.txt.

[3] M. Chiba, G. Dommety, M. Eklund, D. Mitton, and B. Aboba. Dynamic Authorization Extensions to Remote Authentication Dial In User Service (RADIUS). RFC 5176 (Informational), January 2008. http://www.ietf.org/rfc/rfc5176.txt.

[4] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507. http://www.ietf.org/rfc/rfc2246.txt.

[5] Buschmann Frank, Meunier Regine, Rohnert Hans, Sommerlad Peter, and Stal Michael. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. John Wiley & Sons Ltd., 1996.

[6] Ryan Hurst and Ashwin Palekar. Microsoft EAP CHAP Extensions. Working Draft, June 2007. http://tools.ietf.org/pdf/draft-kamath-pppext-eap-mschapv2-02.txt.

[7] Vivek Kamath, Ashwin Palekar, and Mark Wodrich. Microsoft's PEAP version 0 (Implementation in Windows XP SP1). Working Draft, October 2002. http://tools.ietf.org/pdf/draft-kamath-pppext-peapv0-00.txt.

[8] Maurer Norman and Wolfthal Marvin Allen. *Netty in Action MEAP Edition V10*. Manning Publications Co., 2014.

[9] C. Rigney. RADIUS Accounting. RFC 2866 (Informational), June 2000. Updated by RFCs 2867, 5080, 5997. http://www.ietf.org/rfc/rfc2866.txt.

[10] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), June 2000. Updated by RFCs 2868, 3575, 5080, 6929. http://www.ietf.org/rfc/rfc2865.txt.

[11] G. Zorn. Microsoft PPP CHAP Extensions, Version 2. RFC 2759 (Informational), January 2000. http://www.ietf.org/rfc/rfc2759.txt.

[12] G. Zorn, D. Leifer, A. Rubens, J. Shriver, M. Holdrege, and I. Goyret. RADIUS Attributes for Tunnel Protocol Support. RFC 2868 (Informational), June 2000. Updated by RFC 3575. http://www.ietf.org/rfc/rfc2868.txt.

H. Abbildungsverzeichnis

I. Glossar

ACL

Access Control List - Liste von Berechtigungen welche festlegt welchen Zugriff ein Benutzer auf bestimmte Objekte hat. 3, 22, 37

CoA

Change-of-Authorization - RADIUS Paket zum Ändern von Benutzersessions. 3, 8, 12, 16, 17, 22, 83

epoll

Linux Systemaufruf für I/O-Event Notifizierungsmechanismus. 29, 39

Git

Software zur Versionsverwaltung von Dateien. 75, 76

LaTeX

Ein Softwarepaket welches das Textsatzsystem TeX zum erstellen von Dokumenten verwendet. 76

LDAP

Lightweight Directory Access Protocol - Netzwerkprotokoll welches die Abfrage und die Modifikation von Informationen eines Verzeichnisdienstes ermöglicht. 28

MAC-Adresse

Media-Access-Control-Adresse - Hardware-Adresse eines Netzwerkadapters. 21

Maven

Build-Management-Tool zum erstellen und verwalten von Java Programmen. 53

NAS

Network Access Server - Server welcher Zugang zu einem Netzwerk ermöglicht. 21–23, 26–28, 38, 44, 53, 75, 79

RFC

Request for Comments - Ein technisches Dokument des RFC-Editors. 26, 29, 45, 47, 66

SSD

Systemsequenzdiagramm - Diagramm welches den Ablauf von einem Szenario oder Use Case darstellt. 32