Selbstschutz in Organic- und Ubiquitous-Middleware-Systemen unter Verwendung von Computer-Immunologie

Dissertation

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Angewandte Informatik

der Universität Augsburg

eingereicht von

Dipl. Inf. Andreas Pietzowski

Erster Gutachter:	Prof. Dr. rer. nat. Theo Ungerer
Zweiter Gutachter:	Prof. Dr. rer. nat. Bernhard Bauer
Tag der mündlichen Prüfung:	17. Oktober 2008

Zusammenfassung

Die immer komplexer werdenden Computersysteme und das Einbeziehen von immer mehr elektronischen Sensoren, sowie mobilen und fest installierten Rechenknoten stellt die Forschung in diesem Gebiet vor immer neue Herausforderungen. Die Kommunikation der einzelnen Geräte untereinander erfordert eine skalierbare und intelligente Netzwerkarchitektur. Aus diesem Grund wurden Middlewaresysteme entwickelt, die von tieferen Kommunikationsschichten abstrahieren und eine vereinfachte Kommunikation zwischen den beteiligten Prozessen ermöglichen. Eine besondere Ausprägung dieser Middlewaresysteme stellen die Organic- und Ubiquitous-Middleware-Systeme dar. Die in dieser Arbeit hauptsächlich angesprochene Middleware trägt den Namen OCμ und stellt einer der Grundlagen der Organic-Computing-Forschung an der Universität Augsburg dar. Diese Middleware verfügt beispielsweise über Mechanismen zur Selbstoptimierung, Selbstkonfiguration oder Selbstheilung und soll die Middleware und die Kommunikation der einzelnen Prozesse robuster machen als konventionelle Middlewaresysteme.

Jede neue Architektur bringt allerdings auch automatisch neue Schwachstellen mit sich, die von Angreifern ausgenutzt werden können. Selbstverständlich kann ein System perfekt gegen Eindringlinge geschützt werden, in dem es komplett von der Außenwelt abgeschottet wird. Dies steht aber gleichzeitig im enormen Gegensatz zur Dynamik und Interaktion einer solchen Middleware. Gerade die Offenheit und Flexibilität zeichnen eine ubiquitäre Middleware aus, die den Anforderungen der heutigen Zeit gerecht wird. Mobile Knoten sollen der Middleware jederzeit beitreten aber auch genauso einfach die Kommunikation mit der Middleware wieder einstellen können. Eine gänzlich offene Netzwerkinfrastruktur öffnet jedoch andererseits die Tür für jede Art von Eindringling, was ebenso wenig erwünscht ist.

Das Ziel dieser Arbeit ist es, ein Sicherheitskonzept zu entwickeln, das die Middleware gegen unbekannte Anfragen schützt, aber gleichzeitig auch die Offenheit der Middleware berücksichtigt und erhält. Generell lassen sich diese beiden Ziele nur schwer vereinbaren. In der Natur hingegen hat sich genau diese Art von Schutzsystem über Millionen von Jahren etabliert: Unser Immunsystem. Es schützt uns sowohl gegen bekannte als auch gegen unbekannte und noch nie zuvor aufgetretene Bedrohungen. Dennoch kann sich unser Körper frei bewegen und mit der Welt in einem sehr hohen Freiheitsgrad interagieren.

Die Kernkomponenten und Schlüsselmechanismen des biologischen Vorbilds wurden in dieser Arbeit herangezogen und auf ein künstliches Immunsystem für den Selbstschutz in OCμ übertragen. Dabei wurde die Architektur so gewählt, dass sowohl die Offenheit des Systems als auch der Schutz der Middlewareknoten in gleichem Maße berücksichtigt werden. Durch verschiedene Evaluierungen wird der Zusammenhang der unterschiedlichen Parameter des Schutzsystems ersichtlich. Es werden optimale Konfigurationen aufge-

zeigt, um individuelle und unterschiedliche Systeme mit einem erfolgreichen Immunsystem auszustatten. Durch spezielle Optimierungen wird eine sehr gute Erkennungsrate von Eindringlingen bei einem gleichzeitig geringen Ressourcenverbrauch erzielt. Bei der Integration in die Middleware wurden insbesondere leistungsschwache Knoten, wie mobile Endgeräte oder Sensoren, berücksichtigt und es wird gezeigt, wie diese durch geschickte Gruppierung und intelligenter Kommunikation innerhalb der Gruppen ebenso in den Genuss eines effektiven Schutzsystems kommen können, wie leistungsstarke Knoten.

Vorwort

Die vorliegende Arbeit entstand in den Jahren 2005 bis 2008 während meiner Tätigkeit als wissenschaftlicher Angestellter am Institut für Informatik der Fakultät für Angewandte Informatik der Universität Augsburg. Die ursprüngliche und zu diesem Zeitpunkt bereits existierende Middleware „Amun“ wurde im Jahr 2005 Bestandteil der Forschungen im „DFG-Schwerpunktprogramm 1183 Organic Computing“ und durch die Ideen des Forschungsantrags geleitet. Der Forschungsantrag geht dabei intensiv auf die Anforderungen an die Selbst-X-Eigenschaften der Middleware ein und beschreibt Lösungsansätze. Im selben Zug wurde die Middleware in OCμ umbenannt, um dem Organic-Aspekt der Middleware mehr Bedeutung zu verleihen. In dieser Arbeit wurde der Bereich der Selbstschutzmechanismen aufgegriffen und an die Anforderungen der Middleware adaptiert.

Ich bedanke mich bei Herrn Prof. Dr. Theo Ungerer für die vielen interessanten Diskussionen und seine hervorragende Unterstützung. Außerdem danke ich Herrn Prof. Dr. Bernhard Bauer für die Übernahme des Korreferats. Mein Dank gilt weiterhin Wolfgang Trumler und Benjamin Satzger aus dem Organic-Computing-Bereich und allen übrigen Mitarbeitern und Kollegen in der Fakultät für Informatik, die in konstruktiven Diskussionen manche Sachverhalte in einem anderen Licht erschienen ließen und somit auch zum Gelingen der Arbeit beigetragen haben. Thomas Franke danke ich für das Korrekturlesen und seine konstruktive Kritik. Meiner Familie und meinen Freunden möchte ich auf diesem Wege für die ganzen gemeinsamen Stunden danken, die für den entsprechenden Ausgleich gesorgt haben, der ebenso – wenn auch indirekt – für das Gelingen einer Dissertation notwendig ist.

Augsburg, 28. Oktober 2008 *Andreas Pietzowski*

Inhaltsverzeichnis

1 Einleitung

Unser Körper stellt einen komplexen Organismus dar, in dem unzählige Komponenten zusammenarbeiten und nur dadurch unser Leben in dieser Welt ermöglichen. Täglich sind wir aber auch lebensbedrohlichen Gefahren, wie Bakterien und Viren ausgesetzt und müssen daher gegen diese Bedrohungen geschützt werden. Unser Immunsystem hat sich über Millionen von Jahren zu einem leistungsfähigen Schutzmechanismus entwickelt. Ohne dieses funktionierende Schutzsystem wäre der Mensch selbst kleinsten Bedrohungen hilflos ausgesetzt und könnte sich nicht in seiner Umwelt behaupten. Das Immunsystem passt sich ab der Geburt individuell an einen Menschen und seine Umgebung an, arbeitet sein Leben lang völlig selbständig und schützt uns in der Regel hervorragend gegen bekannte und unbekannte Eindringlinge und Gefahren.

In der heutigen Zeit stellen Lebewesen zwar immer noch die komplexesten Systeme auf diesem Planeten dar. Jedoch beginnt auch die Computerwelt mit stetig wachsendem Funktionsumfang permanent komplexer zu werden. Immer mehr Einzelkomponenten vernetzen sich zu komplexen Einheiten und die Kommunikation in Netzwerken steigt an. Genau wegen dieser verzahnten Zusammenarbeit von Einheiten wurden Middlewares eingeführt. Sie erleichtern die Kommunikation zwischen einzelnen Prozessen und abstrahieren von der darunterliegenden Infrastruktur.

Aber diese neuen Strukturen und Mechanismen einer Middleware bringen nicht nur Vorteile mit sich. Sie bieten auch neue Angriffsflächen, die ausgenutzt werden können, um ein System absichtlich zu schädigen oder gar außer Gefecht zu setzen. Es bestehen zahlreiche Forschungsarbeiten und Produkte, die ein Computersystem vor Eindringlingen, wie Trojanischen Pferden oder Viren schützen sollen. Diese Schutzsysteme beziehen sich in der Regel auf heutige Desktop-Computer-Systeme oder setzen auf tieferen Ebenen an, wie beispielsweise der TCP/IP-Kommunikation bei Firewalls. Gerade Ubiquitous- oder Organic-Middleware-Systeme sind darauf ausgelegt, viele kleine Komponenten untereinander zu vernetzen und eine einfache Kommunikation zwischen den einzelnen Teilnehmern zu ermöglichen. Die Funktion der Komponenten soll dabei in den Vordergrund und die Komplexität der Kommunikation in den Hintergrund rücken. Aber gerade die Dynamik einer solchen Middleware, das Vorhandensein von eventuell mobilen Komponenten und die sich ständig ändernden Interaktionen zwischen diesen Einzelkomponenten bergen Gefahren und potenzielle Angriffsziele. Falschinformationen in Nachrichten oder bewusste Manipulationen der Nachrichtenpakete können einzelne Teilnehmer oder

ganze Teilsysteme außer Kraft setzen und die Middleware und ihre Funktionalität unter Umständen enorm beeinträchtigen. Kann etwas gegen diese Art der Bedrohung unternommen werden und wie können solche Angriffsversuche überhaupt erkannt werden?

1.1 Ziele der Arbeit

Das menschliche Immunsystem liefert ein gutes Beispiel, wie sich ein Organismus relativ erfolgreich gegen unbekannte Eindringlinge oder Bedrohungen schützen kann. Dieses System hat sich allerdings im Laufe der Evolution immer mehr an die Bedürfnisse der Lebewesen in ihrer Umgebung angepasst und sich auf die besonderen Anforderungen spezialisiert. Das Zusammenwirken der einzelnen Komponenten in einem biologischen Immunsystem ist sehr komplex und nicht ohne weiteres auf die Computerwelt übertragbar.

Aber die Kernkomponenten und die prinzipielle Herangehensweise des menschlichen Immunsystems zeigen interessante Aspekte auf. In dieser Arbeit wird das Immunsystem des Körpers als Vorbild für ein künstliches Immunsystem einer Middleware herangezogen. Das Ziel ist dabei, ein flexibles und anpassungsfähiges Schutzsystem für die Teilnehmer einer Middleware zu entwerfen. Dabei wird besonderer Wert darauf gelegt, dass das Schutzsystem wenig Ressourcen benötigt, Flexibilität in der Kommunikation ermöglicht, aber gleichzeitig auch ein höchstes Maß an Sicherheit bietet.

Um all diesen Anforderungen gerecht zu werden, kann das biologische Immunsystem nicht einfach in die digitale Welt überführt werden. Es würden viel zu wenige Überschneidungspunkte vorhanden sein und die Effektivität des Immunsystems wäre fragwürdig. Vielmehr werden die Schlüsselkomponenten des biologischen Vorbilds analysiert und die Grundkonzepte in die Computerimmunologie überführt. Dabei werden in dieser Arbeit neben leistungsstarken Desktopcomputern insbesondere die leistungsschwachen Knoten wie mobile Endgeräte berücksichtigt, da diese eine immer wichtigere Rolle in einer ubiquitären Umgebung spielen werden. Diese Kleingeräte verfügen in der Regel nur über wenig Speicher und Rechenkapazität, sollen aber dennoch in einem ausreichendem Maße vor Bedrohungen geschützt werden.

1.2 Aufbau der Arbeit

Im folgenden Kapitel werden bestehende Organic- und Ubiquitous-Middleware-Systeme aufgeführt und kurz beschrieben. Sofern diese Systeme über Sicherheitsmechanismen verfügen, wird auf diese Aspekte ein besonderes Augenmerk gelegt und auf die Vor- und Nachteile der einzelnen Systeme näher eingegangen. Abschließend werden die genannten

Systeme mit der Organic Middleware OCμ[1] in Vergleich gestellt.

Das Kapitel 3 befasst sich mit den Angriffspunkten und Gefahren, die einer Middleware drohen können. Es wird sowohl auf potenzielle Angreifer eingegangen, als auch auf Szenarien, die eine Bedrohung für die Middleware darstellen können. Eine eventuelle Lösungsstrategie stellt oft die Verschlüsselung von Nachrichten dar. In diesem Kapitel werden allerdings auch die Nachteile eines solchen Ansatzes behandelt.

Im darauf folgenden Kapitel wird das Gesamtsystem der Computerimmunologie in OCμ vorgestellt. Dabei wird das System zuerst von anderen Techniken abgegrenzt und dann Schritt für Schritt das gestufte Sicherheitskonzept von der Administration der Teilnehmer, über die Erkennung von eventuellen Gefahren bis hin zur Beseitigung der Bedrohungen erläutert.

Im Kapitel 5 wird das biologische Immunsystem schematisch dargestellt und aufgezeigt, wie die Schlüsselkomponenten in die digitale Welt überführt und genutzt werden können. Im weiteren Verlauf werden verschiedene Forschungsarbeiten skizziert, die ebenfalls die Computerimmunologie in ihre Systeme integriert haben. Abschließend werden diese Herangehensweisen mit der Implementierung der Computerimmunologie in OCμ verglichen.

Die Architektur des künstlichen Immunsystems wird in Kapitel 6 ausführlich erklärt und mathematisch fundiert. Es werden in diesem Kapitel viele Aspekte aufgezeigt, die sich bei der Konstruktion eines künstlichen Immunsystems ergeben. Es wird unter anderem auf die negative und positive Selektion eingegangen, sowie die optimale Rezeptorlänge der künstlichen Antikörper bestimmt. Ferner wird gezeigt, welche Abhängigkeiten – und auch Unabhängigkeiten – zwischen verschiedenen Parametern bestehen. Durch spezielle Optimierungen kann ein effizientes und speicherarmes Immunsystem erreicht werden, das durch sehr gute Trefferraten besticht.

Im Kapitel 7 wird gezeigt, wie sich die Architektur des künstlichen Immunsystems in die bestehende Infrastruktur von OCμ einbetten lässt. Dabei werden die einzelnen Komponenten in der Middleware als Dienste und Monitore realisiert und benötigen daher keine Entwicklung von neuen Komponenten im Grundaufbau der Middleware. In diesem Kapitel wird auch auf die Problematik der Knoten eingegangen, die nur über wenig Ressourcen verfügen. Speziell für diese Teilnehmer wird aufgezeigt, wie sie sich im selben Maße schützen können, wie leistungsstarke Desktop-Systeme.

Die Reaktion auf eventuelle Bedrohungen und die Beseitigung dieser Gefahren wird im Kapitel 8 abgehandelt. Dabei werden Lösungsstrategien erörtert, die es der Middleware ermöglichen, sich gegen bösartige Dienste oder Knoten abzuschotten.

Im letzten Kapitel wird die Arbeit zusammengefasst und nochmals auf die wichtigsten

[1] Bei OCμ handelt es sich um ein Akronym, bei dem die Abkürzung OC für Organic Computing steht und das μ den Schwerpunkt „**M**iddleware for **U**biquitous **E**nvironments“ symbolisiert.

Ergebnisse eingegangen. Mit einem abschließenden Ausblick auf eventuell zukünftige Forschungsaufgaben wird die Arbeit abgeschlossen.

2 Organic- und Ubiquitous-Middleware-Systeme

Ubiquitäre Systeme erhalten immer mehr Einzug in unseren Alltag. Jeder möchte überall und zu jeder Zeit Informationen entweder konsumieren, oder auch anderen Teilnehmern anbieten können. Monolithische PC-Systeme in Arbeitszimmern treten dabei immer mehr in den Hintergrund und spielen bei der Vernetzung eine zunehmend geringere Rolle. Kabellose Geräte, wie Laptops, Handhelds, Mobiltelefone oder allgemein eingebettete Systeme stellen im Gegensatz dazu immer größere Ansprüche an eine flexible und zuverlässige Vernetzung zwischen allen Komponenten in ihrer Reichweite oder gar eine Verbindung mit dem Internet. Eine Middleware spielt bei diesem Vorhaben eine Schlüsselrolle. Sie kümmert sich um die ordnungsgemäße Verteilung von Ressourcen, den ungehinderten Netzwerkfluss, die nahtlose Integration von Geräten und ebenso deren Ausgliederung bei bewusstem oder ungewolltem Verlassen des Netzwerks [AB05]. In ihren Aufgabenbereich fallen aber ebenso Sicherheitsaspekte wie beispielsweise Verschlüsselung, Authentifizierung oder der oft vernachlässigte Datenschutz. Gerade die steigende Komplexität, die Dynamik solcher Netzwerke und der damit verbundene immer höher werdende Konfigurationsaufwand, stellt die Middleware vor eine große Herausforderung. Die Middleware soll bei allen ihren Diensten bewusst in den Hintergrund treten, damit der Benutzer die reine Funktionalität der ubiquitären Geräte oder der darauf laufenden Dienste nutzen kann.

Im Folgenden werden fünf unterschiedliche Middlewaresysteme aus der aktuellen Forschung vorgestellt. Dabei wird besonderer Wert auf die darin vorhandenen oder eventuell fehlenden Schutzmechanismen gelegt. Die unterschiedlichen Systeme werden abschließend anhand bestimmter Parameter verglichen, die für ein Sicherheitskonzept von Relevanz sind.

2.1 BASE/PCOM

An der Universität Stuttgart wurde das Middlewaresystem PCOM/BASE entwickelt. Bei BASE handelt es sich um eine Architektur für eine dienstbasierte Middleware, wogegen PCOM darauf aufsetzt und ein Komponentensystem zur dynamischen Anpassung für

ubiquitäre Anwendungen realisiert [BSGR03, BHSR04].

BASE

Gerade in ubiquitären Umgebungen verbinden sich die unterschiedlichsten Geräte dynamisch miteinander, nehmen gegenseitig Dienste in Anspruch und greifen auf verschiedene Ressourcen zu. Die Middleware BASE wurde genau für diesen Einsatz konzipiert, denn sie ermöglicht eine einheitliche Schnittstelle, um entfernte Dienste über unterschiedliche Kommunikationswege zu nutzen. Durch ihre gute Skalierbarkeit läuft sie sowohl auf eingebetteten Systemen, als auch auf leistungsfähigen Desktop-Systemen.

Bei der Entwicklung von BASE wurde auf drei Aspekte großer Wert gelegt:

1. *Einheitliche Programmierschnittstelle*
   Nicht nur die Kommunikationsschnittstellen, sondern auch der Zugriff auf Geräte soll einheitlich gestaltet sein. So wird beispielsweise der Zugriff bei unterschiedlichen Plattformen durch Proxy-Objekte vereinheitlicht.
2. *Flexible Protokolle*
   Sowohl die Wahl des Kommunikationsmediums als auch das verwendete Kommunikationsprotokoll sollen unabhängig vom verwendeten Servicemodell sein. Durch eine Plugin-Technologie können der Kommunikationskanal und das Nachrichtenprotokoll zwischen unterschiedlichen Diensten dynamisch je nach vorhandenen Ressourcen anders gewählt werden. Sogar Hin- und Rückkanal können bei einer Kommunikation zwischen zwei Knoten unterschiedlich sein, falls ein Kommunikationskanal während der Laufzeit nicht mehr verfügbar sein sollte.
3. *Anpassungsfähigkeit*
   Zukünftige Geräte sollen von der Middleware genauso unterstützt werden, wie bereits bestehende Endgeräte und der Einsatz auf eingebetteten Systemen sollte ebenso gewährleistet sein, wie auf leistungsstarken Desktop-Computern oder gar Servern. Dazu sollten die Kernfunktionen klein genug sein, um beispielsweise auf Sensorknoten zu laufen, aber gleichzeitig leicht erweiterbar sein, um die Ressourcen von leistungsstarken Knoten nutzen zu können.

Das Kernstück von BASE stellt der sogenannte Micro-Broker dar, der sich an dem Paradigma eines Micro-Kernels orientiert. Dieser Micro-Broker kümmert sich nur um die grundlegende Kommunikationsabwicklung und kann mit Plugins erweitert werden. Somit ist es möglich, den Kern der Middleware sowohl auf leistungsschwachen Sensorknoten, als auch auf Servern auszuführen.

Die Kommunikation der teilnehmenden Knoten erfolgt in BASE durch einen entfernten Aufruf von Funktionen und dem bekannten Stub-Skeleton-Prinzip. Dabei wird für jeden Dienst auf dem lokalen Knoten ein Stub angelegt, das die entfernte Schnittstelle im-

plementiert und als Platzhalter für den Aufruf dient. Der Stub kommuniziert über den gewählten Kommunikationskanal mit dem Skeleton auf der Serverseite. Das Skeleton entspricht dem Gegenstück zum Stub, hat allerdings direkten Zugriff auf die angeforderte Funktion.

Jeder Knoten in BASE besteht aus einem Invocation-Broker, einem Service-Registry und einem Device-Registry. Die beiden Registries beinhalten alle Informationen über die erreichbaren Dienste und Geräte. Der Invokation-Broker ist die zentrale Instanz eines Knotens und für die Kommunikationsabwicklung zuständig. Dabei wird für jede eingehende und ausgehende Nachricht ein eigener Thread verwendet, der aus einem Pool von wiederverwendbaren Threads gewählt wird. Ein Thread ist so lange aktiv, bis seine Nachricht bearbeitet wurde, beziehungsweise eine Antwort auf seine Anfrage eingegangen ist. Der Vorteil an dieser Technik liegt darin, dass die Dienste selber nicht als Thread implementiert werden müssen, sondern einen Thread vom Invokation-Broker verwenden können.

PCOM

Bei PCOM handelt es sich um ein adaptives Komponentensystem für ubiquitäre Anwendungen. Es setzt direkt auf BASE auf und nutzt dessen Kommunikationsschnittstellen. Eine High-Level-Abstraktion erleichtert dem Programmierer das Erstellen von Komponenten und kapselt die Abhängigkeiten zwischen den verschiedenen Komponenten intern in sogenannten *Contracts* in einer Baumstruktur. Eine Anwendung kann erst dann gestartet werden, wenn alle ihre Komponenten gestartet werden können. Dabei müssen sich die Komponenten einer Applikation nicht zwingend auf dem selben Knoten befinden, sondern können über mehrere Knoten im Netzwerk verteilt sein. Komponenten können sowohl redundant im Netzwerk vorhanden sein als auch Alternativen besitzen. Wenn sich – gerade bei mobilen Endgeräten – die Netzwerkstruktur ändert oder Knoten ihren Status der Erreichbarkeit ändern, kann sich die Applikation durch die Verwendung unterschiedlicher Komponenten zur Laufzeit an die neue Situation anpassen.

Auf den Bereich der Sicherheitsaspekte wird bei der Forschung an BASE/PCOM leider wenig eingegangen. Durch das Plugin-Verfahren in BASE kann jedoch bei den Kommunikationskanälen dynamisch ein Verschlüsselungsmechanismus auf den Knoten hinzugeschaltet werden [Rau04]. Dadurch können Komponenten oder Appliakationen innerhalb der Middleware sicher kommunizieren.

2.2 Plan B

Bei Plan B handelt es sich streng genommen nicht um eine Middleware, sondern um ein Betriebssystem für ubiquitäre Umgebungen, das auf der Entwicklung des Betriebssystems „Plan 9“ der Bell Labs aufbaut [Lab07, BSLG06]. Da die nach außen hin sichtbare Funktionalität jedoch ähnlich der einer Middleware ist, wird die Architektur von Plan B ebenso in diesem Kapitel aufgeführt. Die Besonderheit von Plan B besteht darin, dass Systemressourcen ausschließlich über ein virtuelles Dateisystem exportiert werden, die dadurch von anderen Netzwerkteilnehmern genutzt werden können. Durch diese Reduzierung auf rudimentäre Dateizugriffsmethoden entfällt die Notwendigkeit einer Middleware.

Die Idee der einheitlichen Betrachtung aller Ressourcen als Dateien wurde bereits in der UNIX-Welt aufgeriffen und vereinfacht ebenso bei Plan B die Interoperabilität zwischen den kommunizierenden Einheiten. Sowohl Eingabegeräte wie Mäuse oder Tastaturen, als auch Sensoren, grafische Oberflächen oder sonstige Ausgabegeräte sind im Netzwerk als Dateien vorhanden, worauf je nach Art der Ressource lesend und/oder schreibend zugegriffen werden kann. Dabei besitzt jeder Knoten sein eigenes lokales Dateisystem. Durch diese vereinfachte Dateistruktur, müssen die Clients über keine spezielle Sprache der Middleware verfügen, sondern jede Komponente, die Dateien verwalten kann, kann sich an dieser ubiquitären Umgebung beteiligen.

Durch den speziellen Netzwerk-Dateisystem-Protokoll „9P“ werden die unterschiedlichen Systemressourcen über das Netzwerk miteinander verbunden. Die ubiquitäre Umgebung ergibt sich dabei lediglich durch das Exportieren und Importieren von Ressourcen von entfernten Netzwerkknoten. Dabei wird in der Dokumentation jedoch nicht näher auf einen Discovery- oder Broker-Service eingegangen, die zum Auffinden eines Dienstes notwendig sind. Vielmehr ist die starke Verwandschaft zu normalen TCP/IP-Verbindungen und dem Dateimanagement aus der UNIX-Welt ersichtlich.

Spezielle Bedingungen (*engl. Constraints*) ermöglichen die Auswahl von bestimmten Ressourcen im Netzwerk. Diese Bedingungen beinhalten Einschränkungen nach Benutzer- oder Knotenzugehörigkeit, Typisierung oder örtliche Angaben. Diese Bedingungen beinhalten dabei stets Paare von Attribut und Wert, die durch Ausrufezeichen getrennt sind. Der String `!Hmonkey!Uguybrush!Tcamera` beschreibt demnach beispielsweise die Bedingung für eine Ressource vom Typ `Type=camera` des Benutzers `User=guybrush` die auf dem Knoten `Host=monkey` zu finden ist. Diese Ressourcen werden von Peers als sogenannte Volumes im lokalen Dateisystem gemountet und können anschließend wie normale Dateien angesprochen werden.

Bei der Architektur von Plan B entfällt das aufwändige Stub-Skeleton-Prinzip von anderen bekannten Middleware-Architekturen, die mit RMI kommunizieren. Ein gravierender

Nachteil an dieser Architektur besteht jedoch in der Sicherheit und dem Schutz vor unbefugtem Zugriff. Da beim zugrunde liegenden Plan 9 die Authentifizierung mit einem zentralen Authentifizierungs-Server mittels *Access Control Lists (ACL)* realisiert wurde, stellt diese Instanz den Flaschenhals des gesamten Sicherheitskonzepts dar. Einerseits muss dieser Server zentral von einem Administrator gepflegt werden und andererseits zu jeder Tageszeit von jedem Knoten aus erreichbar sein. Gerade für eine ubiquitäre Umgebung stellt eine zentrale und unverzichtbare Instanz ein besonders großes Hindernis dar, um in der realen Welt zum Einsatz zu kommen. Plan B erwähnt in eigenen Veröffentlichungen die Problematik, dass sich teilnehmende Knoten an bestimmte Konventionen halten müssen, wenn sie Dienste nutzen möchten [BSLG06]. Dies erschwert folglich die Bekämpfung böswilliger Teilnehmer. Ein Vorteil der bei Plan B verwendeten Technologie ist hingegen das langjährige Wissen über die Funktionsweise von verteilten Dateisystemen und dass bestehende Anwendungen ohne Änderungen die Möglichkeiten der ubiquitären Umgebung dieses Betriebssystems nutzen können.

2.3 GaiaOS

Bei GaiaOS handelt es sich um ein Middlewaresystem zur Verwaltung von ubiquitären Umgebungen. Dabei reichert das System die physikalische Welt um Rechenleistung an und vernetzt die dadurch entstehenden Active Spaces. Die Virtualisierung der physikalischen Objekte und deren Ressourcen ermöglicht dem Benutzer die Interaktion mit den Active Spaces durch das GaiaOS [RHC+02a]. Die Middleware kümmert sich dabei um die Verteilung der physikalischen und virtuellen Ressourcen an die teilnehmenden Knoten. Um dies zu ermöglichen, bietet GaiaOS einen Unified Object Bus (UOB) an, auf den die einzelnen Knoten zugreifen können. Dieser Bus verfügt zudem über Schnittstellen, um auch mit CORBA, Java Beans oder anderen verbreiteten Technologien kommunizieren zu können.

Die Middleware ist ein komponentenbasiertes, verteiltes Meta-Betriebssystem *(Component Management Core)*, das auf einem bestehenden Betriebssystem aufsetzt. Der eigentliche Kernel ist dabei um fünf Basisdienste erweitert, die zusammen die notwendige Funktionalität von GaiaOS zur Verfügung stellen [RHC+02b]:

1. *Event Manager Service*
   Der Event Manager Service ist verantwortlich für die Ereignisverteilung im Active Space und für die Bereitstellung von Kommunikationskanälen zwischen Erzeugern und Verbrauchern zuständig.
2. *Presence Service*
   Der Presence Service dient der Erkennung von Teilnehmern wie beispielsweise Geräte oder Menschen im Active Space.

3. *Context Service*
   Der Context Service bietet Informationen über den realen Kontext im Active Space an, um Diensten die Möglichkeit zu bieten, sich an ihre Umgebung anpassen zu können.
4. *Space Repository Service*
   Der Space Repository Service speichert alle Informationen über Hard- und Software, die im Active Space vorhanden sind.
5. *Context File System*
   Das Context File System erweitert das traditionelle Dateisystem um Funktionen für mobile Teilnehmer, heterogene Geräte und Kontextinformationen.

Diese Grundausstattung von GaiaOS geht von einem vertrauenswürdigem Umfeld aus, weshalb bei den Basisdiensten auch keine sicherheitsrelevanten Aspekte betrachtet werden. Weitere Forschungsarbeiten [SNC02, KSC04] an der Middleware haben diese Thematik jedoch aufgegriffen und ein Sicherheitskonzept für GaiaOS ausgearbeitet. Dabei wurde verstärkt das Augenmerk auf die Verwendung von Zugriffskontrolllisten (Access Control Lists, ACLs) gelegt. Benutzer müssen sich gegenüber einem Authentfizierungs-Dienst identifizieren, bevor sie das System oder damit verbundene Ressourcen in Anspruch nehmen können. Der verwendete Authentifizierungsmechanismus wurde dabei als Plugin realisiert. Dadurch ist es möglich, verschiedene Mechanismen, wie beispielsweise die Eingabe eines Passworts, eine Chipkartenerkennung oder gar die Erkennung von biometrischen Daten zur Authentifizierung zu verwenden. Benutzer erhalten vom System nach einer erfolgreichen Authentifizierung einen Berechtigungsnachweis, der an eine bestimmte Rolle gebunden ist. Rollen wiederum sind im System durch Zugriffskontrolllisten mit bestimmten Berechtigungen verknüpft. Um der Anforderung der Rekonfiguration gerecht zu werden, wurde zusätzlich noch das Konzept der *Sessions* eingeführt, die die Gültigkeit von Berechtigungen auf ein bestimmtes Zeitfenster beschränkt.

2.4 Die Middleware-Architektur Aura/AIPIS

An der Carnegie Mellon Universität in Pittsburgh (USA) wurde das Middlewaresystem Aura [SG02] entwickelt, das den Mensch in den zentralen Mittelpunkt der ubiquitären Umgebung stellt. Unabhängig von dessen Aufenthaltsort soll ihm das System Informationsdienste anbieten und die angeforderten Aufgaben transparent für den Benutzer erledigen. Zur Integration von Sensoren oder allgemeinen Ein- und Ausgabegeräten in die Middleware dient die eigens dafür entwickelte Schnittstelle AIPIS [TS03]. Das Akronym AIPIS steht dabei für „Architecture for the Integration of Physical and Informational Spaces“.

Die in dieser Middleware verwendeten *Auras* stehen dabei symbolhaft für den Benutzer

und sollen die Umgebung proaktiv auf die angeforderten Aufgaben (Tasks) des Benutzers anpassen. Die Aufgaben werden dabei als eine Ansammlung von Dokumenten aufgefasst, die vom System ortssensitiv aufbereitet werden müssen. Dabei werden aber nicht nur die am jeweiligen Ort vorhandenen Ein- und Ausgabegeräte betrachtet und automatisch ausgewählt, sondern auch anwesende Personen und deren Beziehung zur aktuellen Aufgabe werden in diese Entscheidung mit einbezogen. In [TS03] wird beispielsweise ein Szenario beschrieben, in dem ein Benutzer auf dem Weg in die Arbeit seine Dokumente bearbeitet. Bevor er das Flugzeug verlässt, werden die Daten im aktuellen Zustand auf dem Server gesichert und beim Eintreffen im Büro werden die Dokumente wieder nahtlos am Arbeitsplatz für den Benutzer dargestellt. Falls sich andere Personen im Raum befinden und eventuell unbefugten Einblick auf die Dokumente erhalten könnten, werden diese Dokumente vom System ohne Eingreifen des Benutzers automatisch (z.B. durch Minimieren des Fensters) versteckt.

Das Aura-System bietet eine Vielfalt an Informationen an, die von Anwendungen und Diensten abgefragt werden können. Darunter zählen insbesondere die Daten von Netzwerk- und Knotenmonitoren, die es den Anwendungen ermöglichen, sich automatisch an die jeweils vorhandenen Ressourcen anzupassen.

Die AIPIS-Schnittstelle erweitert die Middleware Aura um einige Bestandteile, um die realen Objekte der Umgebung mit dem virtuellen System oder allgemeinen Informationen koordinieren zu können. Darunter ist hauptsächlich eine gemeinsame Notation zu verstehen, die eine Kommunikation zwischen den unterschiedlichen Geräten und Anwendungen ermöglicht. Dabei müssen sich die beteiligten Kommunikationspartner auch nicht explizit gegenseitig kennen.

Der Sicherheitsaspekt in Aura/AIPIS wird von den Entwicklern aufgegriffen, jedoch beschränkt er sich bei Benutzerberechtigungen auf Ressourcen wie beispielsweise Räume, Geräte, Anwendungen und Daten. Das Sicherheitskonzept besteht aus folgenden drei Teilen:

1. *Ressource Monitor*
   Der Ressource Monitor ist zuständig für die Zugangskontrolle von physikalischen Räumen, Geräten, und Anwendungen. In AIPIS werden Zugriffe anhand einer Benutzerauthentifizierung (Passwort, Fingerabdruck, ...) gewährt.
2. *Information Monitor*
   Der Information Monitor bietet den gleichen Schutzmechanismus für gespeicherte Daten wie der Ressource Monitor für physikalische Ressourcen oder Objekte.
3. *Context Monitor*
   Der Context Monitor in AIPIS berechnet die minimale Berechtigung aus dem aktuellen Kontext, der anwesenden Personen und der Gruppenberechtigungen. Somit können beispielsweise nur die Informationen angezeigt werden, die keine Schutz-

verletzung gegenüber anderen anwesenden Personen verursachen würden.

Der Schutzmechanismus entspricht dem von bekannten Mehrbenutzerbetriebssystemen, ist jedoch an manchen Stellen an die Anforderungen einer ubiquitären Umgebung erweitert worden. Es werden in der Literatur leider keine weiteren Details zu der tatsächlichen Implementierung des Rechtesystems genannt, weshalb ein Vergleich in bestimmten Bereichen nur mit Annahmen getroffen werden kann.

2.5 Die Middleware OCμ

Bei der Middleware OCμ handelt es sich um eine verteilte, nachrichtenbasierte Middleware. Dies bedeutet, dass ein Informationsaustausch zwischen den teilnehmenden Knoten ausschließlich durch Nachrichten im Textformat geschieht, und eine Nachricht zwischen zwei Teilnehmern kein Wissen über einen eventuell vorherigen Nachrichtenaustausch besitzt.

OCμ ist aus der ursprünglichen Middleware AMUN [TBPU05, TPBU06] im Rahmen des DFG Schwerpunktprogramms 1183 entstanden. Dabei wurde die Middleware vor allem im Hinblick auf unterschiedliche Self-X-Eigenschaften untersucht und dahingehend erweitert. Bei OCμ handelt es sich um ein Akronym, bei dem die Abkürzung OC für Organic Computing steht und das μ den Schwerpunkt „**M**iddleware for **U**biquitous **E**nvironments“ symbolisiert.

Die Architektur von OCμ teilt sich in mehrere Schichten auf (siehe Abbildung 2.1). Diese Aufteilung in Transportschicht, Vermittlungsschicht und Diensteschicht ist identisch zu vielen herkömmlichen Middlewaresystemen, jedoch besitzt OCμ einige neue Elemente, wie beispielsweise einen typisierten Nachrichtenaustausch und Monitoring-Einheiten auf mehreren Ebenen. Bei der Zustellung der Nachrichten bietet die Typisierung der Nachrichten mehr Freiheiten als Middlewaresysteme, die auf dem Stub-/Skeleton-Prinzip aufsetzen. Ändert sich bei herkömmlichen Systemen die Schnittstelle oder Version eines Dienstes, so ist eine neue Übersetzung aller abhängigen Komponenten erforderlich, die diese Funktionalität nutzen. Durch die Typisierung kann ein Dienst jederzeit um zusätzliche Parameter erweitert werden, ohne dass andere Dienste davon betroffen sind oder gar neu übersetzt werden müssen.

Derzeit ist die Middleware in der Programmiersprache Java implementiert. Wenn OCμ jedoch in Zukunft auch auf eingebettenen Systemen mit minimalem Speicher laufen soll, muss eventuell aus Performanzgründen eine hardwarenähere Sprache verwendet werden. Die Transportschicht in OCμ ist zur Zeit mittels JXTA realisiert, kann jedoch jederzeit durch ein alternatives Kommunikationsprotokoll ausgetauscht werden, ohne Veränderungen an darüberliegenden Schichten vornehmen zu müssen.

Nachrichten werden in OCμ stets mit einem Typ versehen und asynchron zwischen den teilnehmenden Knoten ausgetauscht. Lokale Dienste können sich bei der Middleware für verschiedene Nachrichtentypen registrieren und diese sozusagen „abonnieren“. Jeder Knoten übermittelt die eintreffenden und typisierten Nachrichten dann automatisch an alle für diesen Typ registrierten Dienste.

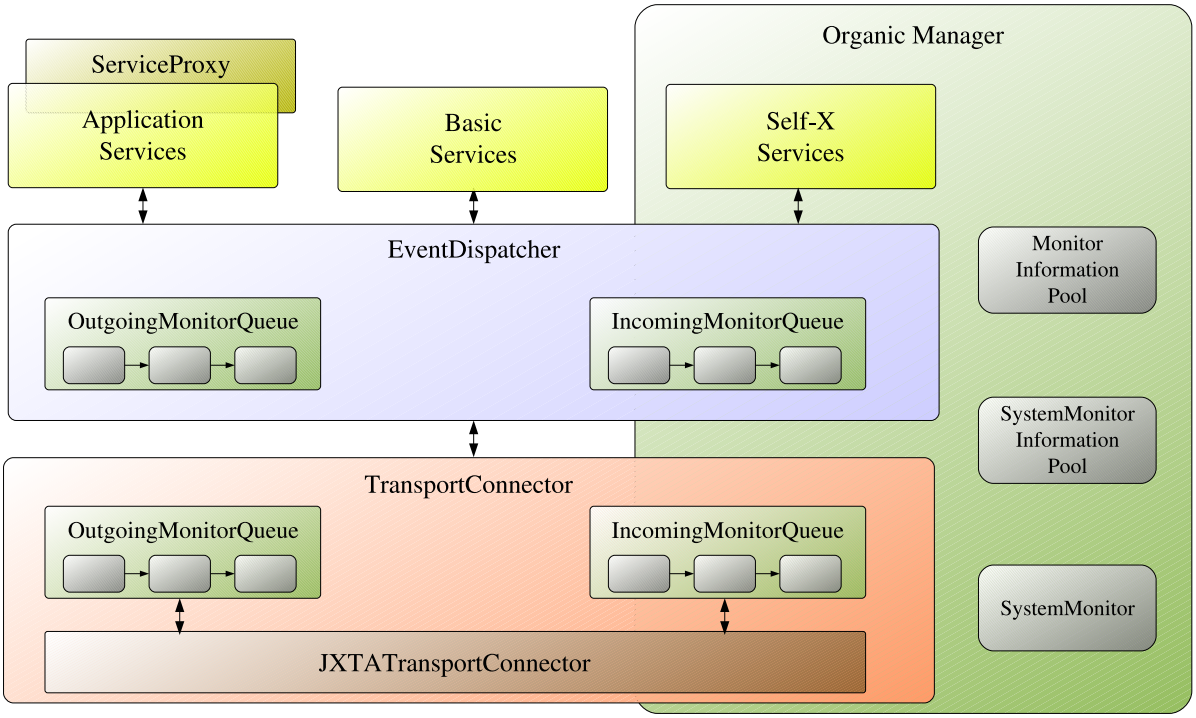


Abbildung 2.1: Aufbau der Middleware OCμ

Bevor eine eintreffende Nachricht jedoch von anderen Diensten verarbeitet wird, durchläuft sie zuerst die Incoming-Monitor-Queue der Transportschicht, dann die Incoming-Monitor-Queue der Vermittlungsschicht und wird erst dann den entsprechenden Diensten zur weiteren Verarbeitung zugestellt. Eine ausgehende Nachricht von einem Dienst durchläuft auf einem Knoten zuerst die Outgoing-Monitor-Queue der Vermittlungsschicht und die Outgoing-Monitor-Queue der Transportschicht, bis sie schließlich an das darunterliegende Netzwerk weitergereicht wird. Dort wird die Nachricht je nach darunterliegenden Transportschicht mit bestimmten Routingverfahren zum entsprechenden Zielknoten vermittelt.

Jede dieser vier Monitor-Queues (zwei pro Kommunikationsrichtung) kann beliebig viele Monitore beinhalten und jeder dieser Monitore kann sowohl lesend, als auch schreibend auf die durchlaufenden Nachrichten zugreifen. Somit ist es möglich, die Nachrichten zu analysieren, um eventuell vorzeitig Entscheidungen treffen zu können, oder den Nachrichten zusätzliche Information aufzuprägen, die für andere Knoten von Relevanz sein könnten [TBPU05]. Speziell das Prinzip der Monitore wird im späteren Verlauf die-

ser Arbeit eine wichtige Rolle spielen, da mit deren Hilfe eine vorzeitige Analyse der Nachrichten durchgeführt werden kann, bevor diese eventuelle Schäden am System anrichten können. Grundsätzlich kann mit Hilfe der Monitore auf einfache Art und Weise ein Observer-Controller-System realisiert werden, das bei der Entwicklung der Selbst-X-Eigenschaften oft von großer Bedeutung ist.

Die Entwicklung von Applikationen im Bereich des Organic Computing [MS04], wie beispielsweise für die Middleware OCμ, erfordert ein generelles Umdenken beim Design der Anwendungen. Die Aufteilung einer Applikation in Dienste wird zwar bereits von den Webservices gefordert, die freie Verteilung der Dienste auf die Knoten des Netzes ist jedoch eine zusätzliche Eigenschaft der Middleware, die zur Umsetzung der Selbst-X-Eigenschaften benötigt wird [Tru06].

2.6 Vergleich der unterschiedlichen Middlewaresysteme

Die vorgestellten Middleware-Systeme unterscheiden sich in ihrer Architektur sehr deutlich voneinander, was jedoch in dieser Arbeit keine so große Rolle spielen soll. Vielmehr werden die verschiedenen Middleware-Systeme in Tabelle 2.1 im Bezug auf das vorhandene Sicherheits-System verglichen[1].

Einige der aufgeführten Merkmale werden erst in späteren Kapiteln erläutert. Aus diesem Grund werden die nicht unbedingt offensichtlichen Schlagwörter kurz stichpunktartig erklärt:

Zentrale Instanz: Eine zentrale Instanz beschreibt eine unverzichtbare Einheit in einem System. Der Nachteil einer zentralen Instanz in einer verteilten Architektur ist, dass manche Aufgaben nicht mehr erledigt werden können, falls kein Kommunikationsweg mehr zu dieser Instanz vorhanden sein sollte. Der Vorteil ist eine leichtere Administration.

Rekonfiguration: Eine Middleware, die ständig wechselnde Knoten und Dienste zu verwalten hat, wie beispielsweise dynamisches Hinzukommen oder Entfernen von mobilen Endgeräten oder neuen Diensten, muss sich an die neue Umgebung anpassen. Dies geschieht durch eine Rekonfiguration der Middleware, bei der auch eventuell das Sicherheitssystem angepasst werden muss.

Verteilte Kontrolle: Eine Kontrolle der Zugriffsberechtigungen kann entweder bei einer zentralen Instanz erfolgen oder verteilt im System hinterlegt werden. Der Vorteil

[1] Die Dokumentation der Middleware Aura/AIPIS ist nicht sehr umfangreich. Ebenso wird auf manche Aspekte in der Literatur über BASE/PCOM nicht ins Detail eingegangen. Aus diesem Grund stellen die eingeklammerten Eigenschaften größtenteils nur logische Annahmen dar.

Middleware	BASE/PCOM	Plan B	GaiaOS	Aura/AIPIS	OCμ
Basiskonzept	Verschlüsselung	Dateirechte	Rollen, ACLs	Benutzerrechte	Rezeptoren
Kommunikation	asynchron	synchron	synchron	(synchron)	asynchron
Zentrale Instanz zur Verwaltung des Sicherheitskonzepts	nein	ja	ja	(ja)	nein
Automatische Rekonfiguration des Sicherheitskonzepts nach Rekonfiguration des Systems	(nein)	nein	ja	(ja)	ja
Verteilte Kontrolle der Zugriffsberechtigungen	(nein)	ja	nein	(nein)	ja
Automatische Anpassung nach Migration von Diensten	(ja)	ja	nein	(ja)	ja

Tabelle 2.1: Vergleich einiger Middleware-Systeme im Hinblick auf relevante Einflussfaktoren für das Sicherheitskonzept

einer verteilten Kontrolle besteht in der Unabhängigkeit einer zentralen Instanz, die eventuell nicht erreichbar sein kann.

Migration von Diensten: Dienste können in der Middleware auf unterschiedlichen Knoten ausgeführt werden. Wenn solche Dienste den Knoten wechseln, dann muss auch das Sicherheitssystem dahingehend angepasst werden.

Bei den Middlewaresystemen Plan B und GaiaOS wird jeweils eine zentrale Instanz zur Verwaltung des Sicherheitssystems eingesetzt. Ein solches Konzept sollte jedoch im Zusammenhang mit Sicherheitseigenschaften stets vermieden werden, da eine zentrale Instanz immer den kritischen Ausfallpunkt eines Systems darstellt (Single point of failure) und für Angreifer ein sowohl gut geeignetes als auch einfaches Angriffsziel bietet. Bei Plan B ist allerdings nur die Administration des Sicherheitssystems zentralisiert, nicht jedoch die eigentliche Kontrolle der Zugriffsberechtigungen zur Laufzeit. Die Sicherheitssysteme in BASE/PCOM und das in dieser Arbeit vorgestellte Konzept in OCμ setzen explizit auf eine dezentrale Architektur. Die Rekonfiguration der Middleware und der damit zwingend zusammenhängenden Rekonfiguration des Sicherheitssystems wird nur bei GaiaOS und OCμ behandelt und bei den anderen vorgestellten Architekturen – zumindest in der vorhandenen Literatur – komplett ausgelassen. Die Migration von Diensten zwischen unterschiedlichen Knoten wird zwar von den meisten Middlewaresystemen in der Liste der unterstützen Funktionen aufgegriffen. Allerdings wird nur bei Plan B und OCμ explizit auf die Migration im Zusammenhang mit dem Sicherheitskonzept auf

nähere Details eingegangen.

2.7 Fazit

Anhand der unterschiedlichen Middlewaresysteme wird deutlich, dass intensive Forschung in diesem Bereich betrieben wird. Die unterschiedlichen Projekte weisen viele interessante Aspekte in ihrer Architektur und ihrem Einsatzgebiet auf. Oft wird jedoch der Aspekt der Sicherheit nur am Rande oder überhaupt nicht berücksichtigt. Im Folgenden werden daher unterschiedliche Bedrohungen in Middlewaresystemen aufgezeigt und speziell auf die Problematik in OCμ eingegangen. Der hier verwendete Ansatz der Computer-Immunologie soll zeigen, in wie weit es möglich ist, eine Middleware mit einem innovativen Ansatz vor diesen Bedrohungen zu schützen. Dabei wird ein besonderes Augenmerk auf einen geringen Speicherverbrauch gerichtet, was gerade bei mobilen Endgeräten mit geringen Kapazitäten von Bedeutung ist.

3 Bedrohungen in Middlewaresystemen

Eine Middleware soll zum einen ein offenes System darstellen, das von vielen Teilnehmern genutzt werden kann. Andererseits sollte sie gleichzeitig aber auch eine robuste und sichere Infrastruktur aufweisen, um die teilnehmenden Knoten vor eventuellen Gefahren zu schützen. Diese beiden Ziele schließen sich im Allgemeinen jedoch gegenseitig aus. Trotzdem soll versucht werden, diese beiden Ziele zu einem höchst möglichen Grad zu erreichen.

3.1 Potenzielle Angreifer

Jede beteiligte Instanz der Middleware kann als potenzieller Angreifer betrachtet werden. Im Folgenden werden die einzelnen Teilnehmer und Komponenten genauer betrachtet. Es wird jeweils auf die Aufgabe der Instanz sowie die Gefahr als potenzielle Bedrohung näher eingegangen.

Administrator

Ein Administrator installiert die Middleware auf den teilnehmenden Knoten. Er kümmert sich auch um die Installation einzelner Dienste oder die Integration von Ressourcen auf bestimmten Knoten. Der Administrator kann ein System daher zu jedem Zeitpunkt verändern und somit auch manipulieren. Gegen diesen Eingriff besteht keine Möglichkeit der Erkennung oder gar der Abwehr. Aus diesem Grund wird der Administrator als potenzieller Angreifer nicht näher betrachtet. Er muss immer als vertrauenswürdige Person anerkannt werden.

Benutzer

Ein Benutzer kann stets verteilte Denial-of-Service-Attacken (DDoS-Attacken) durchführen, die in heutigen Netzwerken allseits bekannt sind [MR04]. Dabei reicht es meistens schon aus, wenn mehrere Teilnehmer im Netzwerk unter Verwendung einer falschen Absenderadresse einen Knoten ständig mit (sinnlosen) Anfragen überhäufen. Dem Rechner bleibt dann vor lauter Anfragen keine Zeit mehr zum Beantworten dieser

Anfragen und kann seine eigentliche Funktionalität somit nicht mehr aufrecht erhalten. Es existieren jedoch bereits Maßnahmen gegen derartige Angriffe in tieferen Netzwerkschichten, wie beispielsweise das temporäre Ändern der IP-Adresse, um die Angriffe ins Leere laufen zu lassen. Die Bedrohung durch DDoS-Attacken und deren Bekämpfung wird daher in dieser Arbeit nicht weiter behandelt.

Ein Benutzer der Middleware kann jedoch auch – absichtlich oder unabsichtlich – unbekannte Dienste in die Middleware einschleusen und diese auf bestimmte Knoten verlagern auf denen dann beispielsweise schädlicher Code ausgeführt werden kann. Das stellt eine potenzielle Bedrohung für das System dar. Dieses Szenario wird später noch genauer beschrieben.

Netzwerk

Das Netzwerk an sich stellt zwar die Grundlage der Kommunikation der Middleware dar, jedoch geht von ihm keine direkte Bedrohung aus. Eine Bedrohung ist nur dann vorstellbar, wenn unberechtigte Teilnehmer ins Netzwerk eindringen und dort die Kommunikation abhören oder bekannte Netzwerkattacken durchführen. Diese Bedrohungen fallen jedoch nicht in den Bereich der Middleware da sie in tieferen Schichten anzutreffen sind und bereits ausgereifte andere Sicherheitsmechanismen dafür existieren [ZCC00].

Rechner (Knoten)

Ein Rechner in der Middleware wird als Knoten bezeichnet, auf dem Dienste und Monitore gestartet oder Nachrichten weitergeleitet werden. Somit hat jeder Knoten Zugriff auf durchfließende Nachrichten oder den Code, den dieser ausführt. Nachrichten können analysiert oder verfälscht werden und ausgeführter Code kann zur Laufzeit manipuliert oder sein Verhalten analysiert werden. Beispielsweise könnte durch eine Analyse des Nachrichtenaustausches oder des Codes darauf geschlossen werden, wie bestimmte vertrauenswürdige Dienste miteinander kommunizieren, um somit einen schädlichen Dienst nach diesem Vorbild zu entwickeln.

Dienste und Monitore

Dienste und Monitore sollten normalerweise vor ihrer Aktivierung von einem vertrauenswürdigen Administrator authentifiziert werden, da unbekannter Code immer potenziell Schaden am System anrichten kann. Unbekannte Dienste, die zur Laufzeit auf einem Knoten aktiviert werden, stellen immer eine mögliche Bedrohung sowohl für den Knoten als auch für die restlichen Middlewareteilnehmer dar. Die Dienste können beispielsweise Ressourcen und lokale Daten eines Knotens ausspionieren, ein- und ausgehende Nachrichten des Knotens manipulieren oder den Knoten durch hohe Rechenauslastung handlungsunfähig machen. Ebenso können Dienste anderen Knoten in der Middleware

fehlerhafte oder absichtlich manipulierte Nachrichtenpakete zukommen lassen, um diese in einer bestimmten Art und Weise zu kompromittieren oder diese handlungsunfähig zu machen.

3.2 Bedrohungsszenarien in OCμ

Gerade im Hinblick auf die immer mehr in den Vordergrund tretenden ubiquitären Umgebungen ist eine starke Fluktuation an hinzukommenden oder spontan nicht mehr vorhandenen Knoten in OCμ unvermeidbar – und auch gewollt. Da in der Middleware nur Nachrichten zwischen den Knoten anhand von Zeichenketten ausgetauscht werden, stellt dies den einzigen direkten Angriffspunkt von OCμ dar. Alle anderen Netzwerkangriffe auf tieferen Ebenen werden hier nicht betrachtet, da hierfür die Konfigurationen der Firewall und des Betriebssystemkerns verantwortlich ist.

Nachrichten bestehen in OCμ immer aus einem Nachrichtentyp und einem Nachrichteninhalt. Anhand des Nachrichtentyps entscheidet ein Knoten, ob eine Nachricht zu einem Dienst weiter geleitet werden soll oder nicht. Der Nachrichteninhalt enthält dagegen die eigentliche Information, die entweder eine Anfrage an einen Dienst darstellt oder eine Antwort auf eine solche Anfrage repräsentiert.

Gerade die Selbst-X-Eigenschaften zeichnen OCμ aus und heben sie von anderen Middlewaresystemen ab. Diese unterschiedlichen Selbst-X-Eigenschaften sind alle auf ihre eigene Art und Weise realisiert und arbeiten prinzipiell autonom. Auf Grund der verschiedenen Verfahren können jedoch Seiteneffekte auftreten, durch die eine Bedrohung oder ein Fremdeingriff ins Gesamtsystem stattfinden könnte. Genau diese Selbst-X-Eigenschaften, die die Middleware eigentlich robuster machen sollen, bringen gleichzeitig auch neue potenzielle Sicherheitslücken mit sich, die ein Angreifer zu seinen Gunsten ausnutzen kann.

3.2.1 Bedrohung durch Hinzukommen von neuen Knoten

Die Middleware OCμ stellt ein dynamisches System dar. Dies bedeutet einerseits enorme Flexibilität, da jederzeit Knoten und Dienste zum Netzwerk hinzukommen können und das Gesamtsystem somit um deren Fähigkeiten und Dienstleistungen erweitert wird. Andererseits kann gerade die einfache Integration von Knoten und Diensten ein Sicherheitsproblem für die Middleware darstellen. Vor allem mobile Endgeräte stellen bei dieser Integration ein erhöhtes Risiko dar. Beispielsweise kann ein neu hinzugekommener Knoten in einem ersten Schritt den Netzwerkverkehr analysieren und anschließend entsprechend angepasste Nachrichten generieren, um bestimmte Knoten oder Dienste von OCμ zu beeinflussen.

Auch wenn ein neuer Knoten im System noch nicht bekannt ist, kann er trotzdem mit einem teilnehmenden Knoten kommunizieren, in dem er sich für einen bestimmten Nachrichtentyp des Knotens registriert. Sobald dann Nachrichten empfangen werden, kann der neue Knoten auch eine Antwort an den Sender zurück senden. Somit findet eine Datenübertragung vom neuen Knoten in das bestehende Middlewaresystem statt. In Abhängigkeit des Dienstes, der die Antwort entgegennimmt oder der Art und Weise seiner Nachrichtenverarbeitung können keine bis gravierende Schäden an dem entsprechenden Knoten verursacht werden. Dies kann den Knoten entweder kommunikations- und handlungsunfähig machen, oder ihn dementsprechend manipulieren, dass er andere - und eventuell schädliche - Aufgaben übernimmt.

3.2.2 Bedrohung durch die Selbstkonfiguration der Middleware zum Verteilen von Diensten

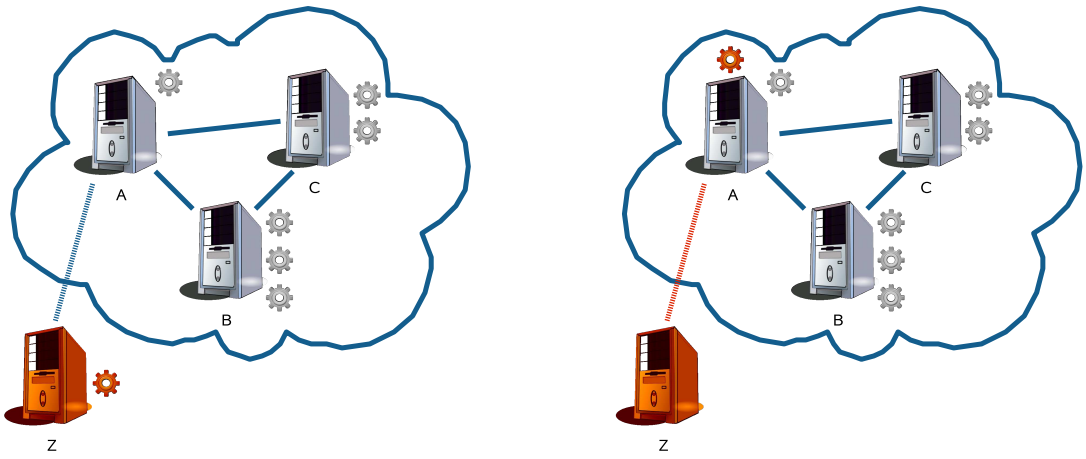
Wenn die Middleware OCμ gestartet wird, sorgt eine Konfigurationsdatei und ein kooperativer Algorithmus für die optimale Verteilung der notwendigen Dienste auf den beteiligten Knoten [TKU06]. Zu Beginn berechnet jeder Knoten anhand der Konfigurationsdatei, mit welcher Güte er einen angefragten Dienst erbringen kann. Die Güte eines Knotens berechnet sich dabei aus den vorhandenen Ressourcen für einen Dienst, wie beispielsweise benötigter Speicherplatz, erforderliche Rechenleistung oder das Vorhandensein von speziellen Sensoren. Ein effizientes und dezentrales Wahlverfahren mit bewusst wenig Kommunikationsaufwand verteilt die Dienste anschließend sukzessive auf die beteiligten Knoten.

In der Regel erhalten nur vom Administrator als gutartig eingestufte Knoten die Konfigurationsdatei, um überhaupt am Wahlverfahren teilnehmen zu können. Aber dennoch ist es denkbar, dass sich ungewollte Knoten trotzdem an solch einem Wahlverfahren beteiligen. Dadurch könnten Dienste auf Knoten ausgeführt werden, die beispielsweise die erforderliche Güte überhaupt nicht erbringen können, aber sich für die Ausführung bereit erklärt haben. Dies kann die Gesamtfunktionalität der Middleware beeinträchtigen.

3.2.3 Bedrohung durch die automatische Selbstoptimierung der Middleware zur Lastverteilung

Die Selbstoptimierung in OCμ findet bereits in den Monitoren eines Knotens statt. Falls die eintreffenden Nachrichten Informationen bezüglich der Auslastung des Quellknotens aufgeprägt bekommen haben, werden diese im Zielknoten bereits in den Monitoren analysiert – also noch bevor entschieden wird, ob die Nachricht an einen Dienst weitergereicht werden soll [TTU06]. Dieses Monitoring kann die Knoten dazu veranlassen,

bestimmte Dienste zwischen den Knoten zu migrieren, wodurch auch ungewollte Dienste ins System eingeschleust werden können.



(a) Bedrohung durch Knoten Z (b) Infizierung durch Knoten Z

Abbildung 3.1: Bedrohungsszenario eines neu hinzukommenden Knotens und der automatischen Verlagerung eines bösartigen Dienstes durch die Selbstoptimierung von OCμ.

In Abbildung 3.1 ist ein solches Szenario beispielhaft dargestellt. Die blaue Wolke stellt ein in sich funktionierendes und nicht infiziertes OCμ-Netzwerk der Knoten A, B und C dar. Die Zahnräder stehen dabei beispielhaft für die Dienste, die auf den entsprechenden Knoten gestartet sind. Der Knoten Z gehört ursprünglich nicht zum Netzwerk und bei seinem orangefarbenen Dienst handelt es sich um einen bösartigen Dienst (siehe linkes Teilbild a). Falls der Knoten A nun Nachrichten von Knoten Z erhält, in denen vorgetäuscht wird, dass der Knoten Z zu einem hohen Grad ausgelastet ist und Knoten A zur selben Zeit keine große Auslastung aufweist, besteht die Möglichkeit einer Migration des bösartigen Dienstes von Knoten Z nach Knoten A (siehe rechtes Teilbild b). Dieser Dienst könnte dann anschließend beispielsweise ungewollte und unkontrollierbare Aktionen auf Knoten A ausführen, wie beispielsweise weitere Dienste auf diesem Knoten nachladen oder den Knoten durch hohe Rechenauslastung zur Handlungsunfähigkeit zwingen.

3.3 Verschlüsselung als Lösung?

Es besteht selbstverständlich die Möglichkeit einer synchronen oder asynchronen Verschlüsselung, um nur autorisierten Geräten und Diensten den Zugriff auf die Middleware zu gewähren. Durch den Einsatz derartiger Technologien kann die Teilnahme oder das Eindringen von bösartigen Diensten oder Knoten so gut wie ausgeschlossen werden. Es

existieren jedoch auch zwei entscheidende Nachteile, die mit der Einführung einer Verschlüsselung in OCμ einhergehen würden:

1. *Verschlüsselungsalgorithmen benötigen Ressourcen*
   Ausreichend gute Verschlüsselungsalgorithmen sind im Allgemeinen sehr rechenintensiv [EMS+02]. Gerade bei kleinen mobilen Endgeräten oder Sensoren, wie sie in ubiquitären Umgebungen in der Regel eingesetzt werden, stellt dies jedoch ein Problem dar. Meist verfügen diese kleinen Endgeräte nur über eine niedrige Prozessorleistung und auch nur geringe Speicherkapazität [WR05]. Verschlüsselungsalgorithmen können die Leistungsfähigkeit des Geräts schnell beeinträchtigen. Die Energieversorgung bei den Kleingeräten ist oft aus Platzgründen auch sehr knapp bemessen. Zur Berechnung der Verschlüsselung wird auch relativ viel Energie benötigt, da die Prozessoren bei derartigen Aufgaben sehr belastet werden. Die Verwendung von Verschlüsselungsalgorithmen kann dadurch schneller zu einer Ressourcenknappheit und letztendlich zur früheren Handlungsunfähigkeit eines ubiquitären Knotens führen.
2. *Verschlüsselung schließt Teilnehmer aus*
   Durch den Einsatz einer Verschlüsselung werden automatisch alle Teilnemer in der Middleware zur Verwendung dieser Verschlüsselung gezwungen. Ein Knoten, der seine Nachrichten ohne eine entsprechende Verschlüsselung versenden würde, würde von den anderen Teilnehmern als bösartig eingestuft und folglich ignoriert werden. Andererseits könnte ein Knoten ohne entsprechende Verschlüsselung aber auch keine verschlüsselten Nachrichten von anderen Knoten der Middleware entschlüsseln und verarbeiten.

In einer offenen Middleware wie OCμ, bei der sich dynamisch Geräte zur Laufzeit hinzufügen können sollen, bringt eine Verschlüsselung eher Nachteile als Vorteile mit sich. Selbstverständlich kann es allerdings für bestimmte Dienste von Vorteil sein, beispielsweise sensible Daten zu verschlüsseln. Dies ist jedoch Aufgabe der kommunizierenden Dienste und für die Middleware transparent. Die Thematik der Verschlüsselung wird in dieser Arbeit daher nicht näher betrachtet.

3.4 Fazit

Die Middleware OCμ stellt ein offenes System dar, wodurch potenziellen Angreifern die Möglichkeit gegeben wird, das System zu beeinträchtigen. Es existieren unterschiedliche Bedrohungsszenarien, die von einfachen Netzwerkangriffen, bis hin zu OCμ-spezifischen Angriffen gehen. OCμ wurde in den letzten Jahren der Forschung um einige Selbst-X-Eigenschaften erweitert. Diese Mechanismen verleihen der Middleware eine gewisse Eigenintelligenz und machen sie einerseits robuster als andere Middlewaresysteme. An-

dererseits bringen aber genau diese Mechanismen potenzielle Angriffsmöglichkeiten mit sich, die die Middleware in ihrer Funktionsweise beeinträchtigen können. Eine mögliche Lösung des Problems besteht immer in der Verwendung einer Verschlüsselung von Nachrichten. Der Nachteil daran ist jedoch, dass *jeder* teilnehmende Knoten die verwendete Verschlüsselung als Software implementiert haben muss. Zusätzlich muss er über die entsprechende Prozessorleistung zum Entschlüsseln verfügen, was bei mobilen Endgeräten oder gerade auch eingebetteten Systemen eine große Herausforderung an die verwendete Technik darstellt. In den folgenden Kapiteln wird beschrieben, wie die Computer-Immunologie daher zum Selbstschutz der Middleware beitragen kann. Dabei können sowohl Knoten *mit*, als auch *ohne* dieses Schutzsystem miteinander kommunizieren.

4 Gesamtarchitektur des Selbstschutzsystems

Eine Middleware bietet den Teilnehmern die Möglichkeit, jederzeit Ressourcen oder Dienste in Anspruch zu nehmen, ohne dass näheres Detailwissen über den Kommunikationspfad oder die genaue Bedienung bekannt sein muss. Die Teilnehmer können dabei sowohl reale Personen darstellen, aber auch durch Rechnerknoten, Dienste oder beispielsweise Sensoren einer ubiquitären Umgebung repräsentiert sein. Die Middleware erledigt dabei den Informationsaustausch zwischen den unterschiedlichen Instanzen. Eine offene Architektur bietet dabei viele Vorteile, da das Netzwerk von den Teilnehmern ohne großen Aufwand beliebig verwendet, erweitert oder gar umkonfiguriert werden kann. Je freier solch ein System jedoch veränderbar oder angepassbar ist, desto leichter lässt es sich in der Regel auch für Angriffe und Manipulationen missbrauchen. Es scheint, als ob das Ziel eines sicheren Systems nicht unmittelbar mit der Idee einer offenen Netzinfrastruktur vereinbar ist. In der Tat handelt es sich dabei um entgegengesetzte Zielvorstellungen. Aufgabe eines Selbstschutzsystems ist es daher, die Möglichkeiten der Angriffe oder des Missbrauchs einzuschränken, dabei aber dennoch eine größtmöglich freie und offene Kommunikation im Netzwerk zu ermöglichen.

Offene Systeme

Das beste Beispiel für ein offenes System sind die Anfänge der Vernetzung von Computersystemen. Ein TCP/IP-Netzwerk, wie das Internet, bietet in seiner Grundarchitektur jedem angeschlossenen Knoten die Möglichkeit, mit jedem anderen erreichbaren Knoten Kontakt aufzunehmen. Wenn aber jeder Teilnehmer in einem Netzwerk jeden anderem Teilnehmer wahllos Nachrichtenpakete zusenden kann, ist dies zwar die einfachste Art des Nachrichtenaustausches, aber zugleich auch die unsicherste Kommunikation überhaupt. Bei der Konzeption der TCP/IP-Netzwerkarchitektur in den 60er Jahren wurde das Augenmerk allerdings primär auf die Verfügbarkeit gelegt und Sicherheitsaspekte (absichtlich) vernachlässigt. Durch diese freie Architektur können unkooperative Teilnehmer heutzutage aber leicht Probleme verursachen, in dem sie beispielsweise Router kompromittieren, wichtige Server mit Anfragen überhäufen oder diese durch bekannt gewordene Sicherheitslücken unter ihre Kontrolle bringen. Aus diesem Grund wurden über die Jahre hinweg nachträglich ausgefeilte Sicherheitsmechanismen [MR04, ZCC00] entwickelt,

um die architekturbedingten Sicherheitsprobleme dieses ursprünglich offenen Netzwerks zu minimieren.

Geschlossene Systeme

In einem geschlossenen System – also in einem System, bei dem alle Teilnehmer fest vorgegeben sind, keine Knoten dynamisch zur Laufzeit hinzugefügt werden und in dem der Benutzer keine neuen Dienste starten darf – ist ein Selbstschutzsystem prinzipiell überflüssig und würde nur unnötigen Overhead erzeugen. Herkömmliche Automobile repräsentieren beispielsweise solch ein geschlossenes System. Die Sensoren, Aktuatoren und die übrigen elektronischen Bauteile kommunizieren im Fahrzeug über ein oder mehrere Bussysteme. Die Kommunikationspartner werden allerdings vom Hersteller kooperativ entworfen und implementiert, damit sich die Komponenten nicht gegenseitig Schaden zufügen können. Externe Geräte lassen sich bei diesen Fahrzeugen nicht dynamisch mit dem Automobil verbinden und können somit auch keinen Schaden anrichten. Alleinige Ausnahme stellt hierbei die Werkstatt dar, die sich zu Reparaturzwecken mit dem Kommunikationssystem des Fahrzeugs verbinden kann. Dieser externe Eingriff wird jedoch bereits im Entwicklungsstadium berücksichtigt und kann in diesem Beispiel als Teil des geschlossenen Systems angesehen werden. Falls der Kommunikationsbus des Fahrzeugs allerdings von außen über eine Funkverbindung erreichbar ist oder beispielsweise durch das Umklappen oder Abreißen eines Seitenspiegels erreicht werden kann, so trifft die Definition eines geschlossenen Systems nicht mehr zu.

Ein adaptives System

Es gibt eine Vielzahl von ubiquitären Umgebungen, in denen ein geschlossenes und statisches System unbrauchbar ist, aber im Gegenzug auch eine völlig offene und frei zugängliche Infrastruktur keine wirkliche Alternative darstellt. Als Beispiel hierfür sei ein Bürogebäude genannt, in dem eine integrierte Middleware den Aufenthaltsort aller derzeit anwesenden Personen kennt. Ein möglicher Mehrwert der intelligenten Büroumgebung wäre, dass die Besucher mit Hilfe ihres selbst mitgebrachten mobilen Endgeräts, wie beispielsweise einem Handy, automatisch zu dem Büro geleitet werden, in dem die gesuchte Person derzeit anzutreffen ist. Das Forschungsprojekt „Smart Doorplate“ [TBPU04] befasst sich mit diesem Thema und beschreibt ein solches Szenario ausführlich am Beispiel intelligenter Türschilder in einem Bürogebäude. Die Türschilder sind dabei fest im Flur montiert und können mittels Pfeilsymbolen als Wegweiser fungieren, in dem sie einem Besucher die zu laufende Richtung anzeigen. Die miteinander über die Middleware vernetzten Türschilder allein repräsentieren hierbei ein geschlossenes System. Ein mitgebrachtes Handy könnte aber beispielsweise die aktuelle Ortsbestimmung übernehmen und diese Informationen an die Middleware senden. Für diesen Zweck muss das Handy des Besuchers dynamisch in die Infrastruktur des Bürogebäudes

als neuer Knoten eingebunden werden können. In einer geschlossenen Middleware wäre dies nicht möglich und ein völlig offenes System würde zu viele potenzielle Gefahren für das Gesamtsystem in sich bergen, wenn das Handy alle Dienste und Ressourcen der Middleware frei nutzen könnte.

4.1 Konventionelle Techniken und deren Grenzen

Da bei der Kommunikation zwischen Applikationen über das Netzwerk die unterschiedlichsten Netzwerkschichten und Kommunikationsprotokolle beteiligt sind, kann auch auf jeder dieser Schichten ein potenzieller Angriff durchgeführt werden. Ein gestuftes Sicherheitssystem bietet Sicherheitsmechanismen auf diesen unterschiedlichen Kommunikationsebenen an.

So dient beispielsweise eine Firewall der Abwehr von Netzwerkangriffen auf tieferen Netzwerkebenen. Sie kann gezielt bestimmte Ports überwachen und den Datenverkehr zwischen Netzwerkkomponenten nach bestimmten Parametern filtern. Dadurch kann ein System beispielsweise vor SYN-Flooding oder Ping-of-Death geschützt werden.

Auf höheren Ebenen der Netzwerkschichten kann durch die Verwendung von Domänen und Zugriffsrechten für Netzwerkkomponenten und deren Benutzer gezielt der Zugriff auf Ressourcen erlaubt oder unterbunden werden. Diese Regelsätze sind jedoch heutzutage im Allgemeinen starr vorgegeben und eine Änderung erfordert stets den Eingriff eines Administrators.

Herkömmliche Virenscanner oder Intrusion-Detection-Systeme basieren in der Regel auf Signaturdatenbanken. Wenn Daten an einem System eintreffen, werden diese zuerst mit einer bereits vorhandenen Signaturdatenbank abgeglichen. Falls es keine Übereinstimmung gibt, scheinen die Daten gutartiger Natur zu sein. Andernfalls werden vom Abwehrsystem entsprechende Gegenmaßnahmen eingeleitet, um die eventuelle Bedrohung einzudämmen oder gänzlich zu eliminieren. Immer dann, wenn ein neuer Virus bekannt wird, wird ihm von den Herstellern der Abwehrsysteme eine spezifische Signatur zugeteilt. Diese neuen Signaturen müssen anschließend auf die einzelnen Abwehrsysteme verteilt werden, um die darunterliegenden Systeme gegen die neuen Bedrohungen zu schützen. Wenn ein Virus auch nur leicht in seinem Quelltext verändert und anschließend neu übersetzt wird, dann ergibt sich bei dieser neuen Ausprägung des Virus auch eine völlig neuartige Signatur. Dies stellt einen Nachteil dar, da ein neuer Eindringling oder schädlicher Code ohne die entsprechenden Signaturen vom System nicht identifiziert werden kann [HT05].

Vertrauensmodelle oder Vertrauensketten stellen ein Verfahren dar, um zur Laufzeit zu entscheiden, ob bestimmte Ressourcen einem Verbraucher zur Verfügung gestellt wer-

den sollen oder nicht [WV03]. Dabei lernen intelligente Vertrauensmodelle selbständig über die Zeit, wem sie vertrauen können und wem nicht und versuchen sich an ein Optimum anzunähern. Vertrauensketten nutzen das soziale Verhaltensmuster von Menschen als Vorbild, um zu entscheiden wem sie Vertrauen entgegenbringen und wem nicht. Die Einschätzung eines vertrauenswürdigen Menschen, ob eine dritte Person vertrauenswürdig ist, resultiert in der Regel in einer höheren Vertrauensstufe, als eine Empfehlung einer Person der man wenig vertraut. In dieser Arbeit wird auf Vertrauensmodelle und Vertrauensketten nicht weiter eingegangen. Sie stellen jedoch eventuell eine Möglichkeit dar, das hier vorgestellte Verfahren zu verfeinern und derzeit notwendige Eingriffe von Administratoren eventuell der Vergangenheit angehören zu lassen.

4.2 Ansatz der Computer-Immunologie

Die Computer-Immunologie versucht eine neue Herangehensweise an die Erkennung und Bekämpfung von möglichen Bedrohungen oder Eindringlingen und nimmt sich die Funktionsweise der Immunsysteme von biologischen Organismen zum Vorbild [AT05]. Der biologische Organismus hat es anscheinend über einen Entwicklungszyklus von einigen Millionen Jahren geschafft, ein ausgeklügeltes und erstaunlich gut funktionierendes Immunsystem zu entwickeln. Dabei ist dieses Immunsystem mit Sicherheit nicht perfekt, aber es hat dennoch eine sehr gute Trefferwahrscheinlichkeit beim Erkennen von Eindringlingen und kann auch unbekannte und noch nie zuvor aufgetretene Viren als solche identifizieren. Die Computer-Immunologie greift die Herangehensweise der Natur auf und versucht zu verstehen, was unser Immunsystem permanent im Hintergrund für uns leistet und wie es funktioniert.

Dabei wird allerdings nicht versucht, das Immunsystem des Körpers zu kopieren. Dies wäre auch nicht sehr sinnvoll, da in der digitalen Welt im Gegensatz zu biologischen Organismen keine Moleküle, Zellen oder Erbinformationen vorhanden sind. Vielmehr arbeitet ein Computer nach festen Regeln. Auch die Kommunikationskanäle sind in einem Computernetzwerk fest definiert und nicht etwa evolutionär ohne Designprozess entstanden. In einem Organismus können beispielsweise ähnliche Moleküle eine ähnliche Wirkungsweise auf den Organismus haben, wohingegen ein ähnliches Bitmuster in einem Computersystem eine grundlegend andere Bedeutung haben kann.

Obwohl durchaus Forschungsarbeiten existieren, die versuchen, das menschliche Immunsystem komplett nachzubilden [GJMV+05], wird von dieser Methodik bei der Computer-Immunologie eher Abstand gehalten [Ber04]. Beim Ansatz der Computer-Immunologie wird vielmehr versucht, die grundlegenden Mechanismen und Herangehensweisen der Natur zu verstehen und ihre Schlüsselkomponenten und -konzepte anschließend sinnvoll auf bestimmte Bereiche der Computerwelt zu adaptieren. Ein offensichtlicher Unterschied

zwischen den beiden Welten ist der dreidimensionale Arbeitsraum des biologischen Immunsystems, wohingegen die computerbasierte Immunologie auf einem eindimensionalen Speicher agieren muss.

4.3 Der Selbstschutz in OCμ

Die nachrichtenbasierte Middleware OCμ bietet den teilnehmenden Knoten die Möglichkeit miteinander zu kommunizieren, ohne viel Wissen über das darunterliegende System besitzen zu müssen. Im Idealfall verhalten sich die Knoten und die darauf laufenden Dienste kooperativ. Dies bedeutet, dass sich die Dienste gegenseitig vertrauen, sinnvolle Daten austauschen und versuchen, sich gegenseitig nicht zu beeinträchtigen. Wie in jedem System kann es jedoch vorkommen, dass sich ungewollte Kommunikationpartner in das System einschleusen. Wie in Kapitel 3 beschrieben, kann dies auf verschiedene Art und Weise geschehen und in bestimmten Fällen zu unangenehmen Beeinträchtigungen oder sogar zum Ausfall des gesamten Systems führen.

Eine nachrichtenbasierte Middleware ist der Bedrohung von anderen Teilnehmern in erster Linie schutzlos ausgesetzt, da sie im Prinzip nur die Nachrichten zwischen den Knoten vermittelt. Die Nachrichtenpakete werden auf den Knoten keiner weiteren syntaktischen oder gar semantischen Prüfung unterzogen. Um das System dennoch vor den möglichen Bedrohungen zu schützen, wird in OCμ ein gestuftes Sicherheitskonzept eingeführt.

Stufe 1: Autorisierung von Knoten und Diensten

In einer ersten Stufe können Knoten und Dienste in der Middleware von einem Administrator autorisiert werden. Diese erhalten dann die Möglichkeit, in der Middleware Ressourcen zu nutzen, in dem ihre Nachrichten von anderen Knoten akzeptiert und verarbeitet werden. Der Unterschied zwischen der Autorisierung eines Knoten und der Autorisierung eines Dienstes besteht darin, dass ein autorisierter Knoten neue Dienste starten kann und diese automatisch autorisiert sind. Autorisierte Dienste hingegen ermöglichen es einem Knoten aber im Gegenzug noch lange nicht, neue Dienste zu starten. Gerade bei der Selbstoptimierung oder der Selbstheilung in OCμ werden aber auch Dienste zwischen den Knoten verschoben. Um diese Funktionalität der Migration von Diensten aufrecht zu erhalten, behält ein Dienst dabei seinen Status als autorisierter Dienst.

Stufe 2: Rollenvergabe an Knoten

Die zweite Stufe des Sicherheitskonzepts besteht in OCμ aus einer Zugriffsverwaltung zwischen Diensten und Knoten. Um diese zu realisieren, könnte jedem Dienst für jeden

Knoten ein spezielles Recht definiert werden. Somit könnte exakt festgelegt werden, welcher Dienst auf welchen Knoten gestartet und ausgeführt werden darf und welcher nicht. Diese feingranulare Rechtevergabe hätte aber den Nachteil, dass der administrative Aufwand exponentiell ansteigt, je mehr Knoten und Dienste in der Middleware vorhanden sind. Aus diesem Grund wurde in OCμ ein Rollensystem eingeführt, um eine skalierbare Zugriffsverwaltung zu ermöglichen. Dabei wird jedem Knoten eine spezifische Rolle zugewiesen. Die Dienste dürfen somit nur auf den Knoten ausgeführt werden, die über eine entsprechende Rolle verfügen.

Stufe 3: Erkennen und Beseitigen von potenziell bösartigen Diensten

In der dritten Stufe kommt das eigentliche Erkennungssystem der hier verwendeten Computer-Immunologie zum tragen. Es verwendet Konzepte der negativen und positiven Selektion, um mit geringem Rechen- und Speicheraufwand herauszufinden, ob es sich bei eintreffenden Nachrichten um gut- oder bösartige Dienste handelt[1].

Die Bedeutung des Wortes „Selbstschutz“ impliziert, dass die Middleware sich selbständig gegen Angriffe schützt – also ohne äußere Hilfe oder sonstige Einwirkungen. Zwar kann auf den Eingriff eines Administrators in manchen Fällen, wie beispielsweise der Autorisierung, nicht verzichtet werden. Aber beim Design des Schutzsystems wurde stets ein Augenmerk darauf gelegt, dass sich das Immunsystem der Middleware selbständig auf dem aktuellen Stand hält und auch Entscheidungen in Eigenregie treffen kann. Der Administrationsaufwand wurde beim Entwurf absichtlich gering gehalten. Zentrale Knoten oder Dienste können immer den Flaschenhals einer Architektur darstellen, da das System beim Ausfall einer dieser Instanzen handlungsunfähig werden könnte. Bei der Gesamtarchitektur des Selbstschutzsystems wurde deshalb großer Wert auf eine verteilte Funktionsfähigkeit gelegt.

Der Schutzmechanismus der dritten Stufe besteht in OCμ im Grunde aus Aktionen und eventuell eintretenden Reaktionen. Eine Aktion wird hierbei durch die Erkennung von unbekannten und eventuell bösartigen Nachrichten repräsentiert, wohingegen die Reaktion aus der Beseitigung der Bedrohung besteht. Im Folgenden wird im Zusammenhang mit anderen wichtigen Aspekten näher darauf eingegangen.

4.3.1 Wer darf welche Nachrichten versenden?

In einem adaptiven Systemumfeld ist diese Frage nicht so leicht zu beantworten, wie etwa in einem offenen oder geschlossenen System. In einem offenen System können sich

[1] Da die Middleware OCμ von tieferen Netzwerkschichten abstrahiert, werden die Sicherheitsaspekte, wie beispielsweise auf Ebene der TCP/IP-Kommunikation in dieser Sicherheitsarchitektur nicht betrachtet.

alle Knoten beliebig Nachrichten zusenden, wohingegen ein geschlossenes System seine Teilnehmer fest definiert hat und dadurch keine Kommunikation mit neuen oder unbekannten Knoten außerhalb des bestehenden Systems möglich ist.

Bei OCμ handelt es sich allerdings um ein adaptives System. Am Beispiel der „Smart Doorplates“ [TBPU04] lässt sich ein schönes Szenario von festen Knoten (intelligenten Türschilder) und dynamischen Knoten (z.B. Handys der Besucher) demonstrieren. Im Folgenden wird das Szenario mit Blick auf den Selbstschutz und der dafür verwendeten Architektur genauer beschrieben.

Ein Bürogebäude sei mit intelligenten Türschildern ausgestattet, das einem Besucher einen Mehrwert an Informationen über die Büroinsassen bieten kann. Damit der Besucher sein mobiles Endgerät im Bürogebäude mit der Middleware verwenden kann, erhält der Besucher an der Rezeption einen speziell für den Zweck ausgelegten Dienst mit einer vorgegebenen Identifikationsnummer auf sein Handy aufgespielt. Das Handy wiederum besitzt auch eine eindeutige Identifikationsnummer, um mit der Middleware kommunizieren zu können. Dieser neue teilnehmende Knoten ist der Middleware vorerst noch nicht bekannt und der darauf laufende Dienst würde von der Middleware als unbekannt und somit als bösartig eingestuft werden. Ein Administrator muss folglich der Middleware irgendwie mitteilen, dass es sich bei der Knoten-ID oder der Service-ID um einen gutartigen Teilnehmer handelt.

Da *die Middleware* als solches kein physikalisch fassbarer Begriff ist, sondern erst durch die Wechselwirkung zwischen den Knoten „sichtbar“ wird, kann man einen Knoten rein technisch gesehen auch nicht bei *der Middleware* autorisieren. OCμ akzeptiert schlichtweg alle eingehenden Nachrichten und verarbeitet diese anschließend. Für den Selbstschutz der Middleware existieren jedoch dedizierte Knoten, die sich um diese Autorisierung kümmern. Diese Knoten stellen die Grundpfeiler des Sicherheitssystems dar. Ihre Aufgabe ist es, die Informationen über neu gestartete, laufende und beendete Dienste von den autorisierten Knoten zu sammeln und mit den anderen Autorisierungsknoten auszutauschen. Anschließend wird die Datenbasis zur Überprüfung der Nachrichten angepasst und das Ergebnis an alle autorisierten Knoten versendet. Die genaue Funktionsweise der Autorisierungsknoten, deren Verteilung im Netzwerk und ihr Zusammenspiel wird in Kapitel 7 detailliert erläutert.

Die Autorisierung von Teilnehmern der Middleware ist derzeit noch Aufgabe des Administrators der Middleware oder anderen Benutzern mit den entsprechenden Berechtigungen. Eine Automatisierung dieser Autorisierung kann eventuell durch den Einsatz von Vertrauensmechanismen in zukünftigen Forschungsarbeiten erreicht werden.

4.3.1.1 Autorisierung von Knoten

Um neu hinzukommende Knoten in OCμ komplett zu integrieren, müssen diese von einem Administrator bei der Middleware autorisiert werden. Nach einer solchen Autorisierung genießt der Knoten den selben Status wie alle anderen autorisierten Knoten. Er wird von den anderen Knoten als gutartig angesehen und in seinem Handeln nicht weiter beeinträchtigt. Er kann Dienste starten und der Middleware Auskunft über seine gestarteten Dienste geben, damit diese auch als gutartig von den restlichen Teilnehmern angesehen werden. Nicht autorisierte Knoten kann selbstverständlich keiner davon abhalten, neue Dienste zu starten. Jedoch besitzen diese Knoten keine Möglichkeit, den Autorisierungsknoten mitzuteilen, welche Dienste als gutartig angesehen werden sollen, da die Autorisierungsknoten ja keine Informationen von nicht-autorisierten Knoten annehmen.

Die Autorisierung von Knoten sollte dabei nur bei fest installierten Knoten oder vertrauenswürdigen mobilen Endgeräten vorgenommen werden. Eine zu gutmütige Autorisierung von Knoten kann schnell zu einem Problem werden, wenn Knoten fälschlicherweise autorisiert werden und somit automatisch als gutartig eingestuft werden.

4.3.1.2 Autorisierung von Diensten

Wie bereits erwähnt, kann es in vielen Fällen sehr leichtsinnig sein, komplette Knoten bei der Middleware zu autorisieren. Dynamisch hinzukommende Knoten könnten somit ohne großen Aufwand Schaden am System anrichten, da sie als gleichwertig zu den gutartigen Knoten angesehen werden. Im Gegenzug sollen neu hinzukommende Teilnehmer aber ja auch gerade die Möglichkeit haben, die Dienste und Ressourcen der Middleware nutzen zu können. Aus diesem Grund bietet der Selbstschutz in OCμ die Möglichkeit, nicht nur komplette Knoten, sondern auch nur einzelne Dienste bei der Middleware zu autorisieren.

Dazu wird der zu autorisierende Dienst auf dem neu hinzukommenden Knoten gestartet. Die für den Dienst generierte Identifikationsnummer wird anschließend von einem Administrator bei einem Autorisierungsknoten als autorisierter Dienst registriert. Nun kann dieser Dienst Nachrichten an andere Knoten der Middleware versenden - egal ob er auf einem autorisierten oder nicht-autorisierten Knoten ausgeführt wird. Der Dienst kann also prinzipiell auch seinen ursprünglichen und nicht-autorisierten Knoten verlassen und auf einem beliebigen anderen Knoten weiter ausgeführt werden, ohne dass das Sicherheitssystem dahingehend angepasst werden muss. Dies ist insbesondere bei der Selbstoptimierung von Vorteil, bei der Dienste zur Lastverteilung automatisch zwischen den Knoten migrieren.

4.3.2 Die rollenbasierte Zugriffsverwaltung

Beim bereits beschriebenen Szenario mit dem Bürogebäude, den intelligenten Türschildern und den mobilen Endgeräten kann jeder Dienst auf allen autorisierten Knoten der Middleware ausgeführt werden, sofern die nötigen Ressourcen vorhanden sind. Aber gerade dies ist in einem solchen Szenario nicht erwünscht. Fest installierten Türschildern wird in der Regel ein größeres Vertrauen gegenübergebracht, als den mobilen Endgeräten, die von Besuchern des Gebäudes mitgebracht und dynamisch in das Netzwerk integriert werden. Sicherheitskritische Anwendungen sollten daher beispielsweise nur auf firmeninternen Servern oder den Türschildern ausgeführt und auf keinen Fall an die mobilen Endgeräte übertragen werden.

Um diese Einschränkungen zu ermöglichen, muss in der Middleware eine Zugriffsverwaltung eingeführt werden. Diese Zugriffsverwaltung wurde in OCμ nicht durch eine direkte Zuweisung von Rechten für Dienste für bestimmte Knoten realisiert. Dies würde bei steigernder Dienst- und Knotenanzahl in einer unüberschaubaren Liste aus Rechtedefinitionen enden und eine einfache Administration nahezu unmöglich machen. Ein Rollenkonzept, ähnlich wie in [And03] beschrieben, führt in OCμ eine Abstraktionsschicht zwischen den Rechten der Dienste und der Knoten ein (siehe Abbildung 4.1). Dabei wird jedem Knoten eine bestimmte Rolle zugewiesen und Dienste werden für spezifische Rollen entweder erlaubt oder verboten.

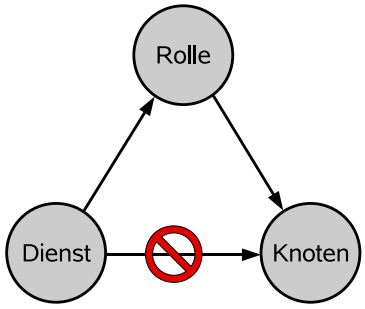


Abbildung 4.1: Rollenbasierte Zugriffsverwaltung in

Die Rollendefinitionen werden bei OCμ in einer XML-Struktur hinterlegt. In dieser XML-Datei sind sowohl alle möglichen Rollen hinterlegt, als auch alle bekannten Dienste mit der Information aufgelistet, auf welchen Rollen sie ausgeführt werden dürfen. Dabei existiert auch der Platzhalter `any`, um Rechte für unbekannte Dienste und unbekannte Rollen zu hinterlegen.

In Abbildung 4.2 ist eine Rollendefinition angegeben, die beispielhaft für das genannte Szenario verwendet werden könnte. In den ersten vier Zeilen werden die im System möglichen Rollen definiert. Dabei werden nur die Namen der Rollen definiert. Welche Rolle ein spezifischer Knoten repräsentiert, wird jedoch nicht in der Datei festgehal-

```
1  <roles>
2    <role name="doorplate">
3    <role name="cellphone">
4  </roles>
5  <services>
6    <service name="rfid" role="any" permission="allow">
7    <service name="prediction" role="doorplate" permission="allow">
8    <service name="prediction" role="any" permission="deny">
9    <service name="any" role="doorplate" permission="allow">
10   <service name="any" role="any" permission="deny">
11 </services>
```

Abbildung 4.2: Beispielhaftes XML-Dokument einer Rollendefinition

ten sondern ist ein interner Parameter des OCμ-Knotens. Ab der fünften Zeile werden dann die bekannten Dienste mit den entsprechenden Rollen und dem zugehörigen Zugriffsrecht aufgelistet. In Zeile 6 wird festgelegt, dass der RFID-Service von jedem Knoten ausgeführt werden darf. Der Prediction-Service darf nur von Knoten mit der Rolle „Türschild“ ausgeführt werden, nicht jedoch von mobilen Endgeräten. Dies macht Sinn, da der Prediction-Service sensible und personenbezogene Daten beinhaltet, die nicht auf fremden Knoten zugreifbar sein sollten. Die Zeilen 9 und 10 legen fest, dass alle unbekannten Dienste auf den Türschildern ausgeführt werden dürfen, aber sonst auf keinem Knoten einer anderen Rolle.

Die Rollenverteilung ermöglicht eine bessere Kontrolle über das Ausführen der Dienste auf den unterschiedlichen Knoten im Netzwerk. Sie wirkt sich aber auch automatisch auf die bereits bestehenden Selbst-X-Eigenschaften von OCμ aus, wie in den folgenden Abschnitten deutlich wird.

4.3.2.1 Auswirkungen des Rollenkonzepts für die Selbstkonfiguration

Bei der Selbstkonfiguration nehmen alle verfügbaren Knoten an einem dezentralen Wahlverfahren teil. Dabei wird ausgehandelt, welche Dienste auf welchen Knoten am besten gestartet werden sollen. Dabei fließen Faktoren wie beispielsweise Speicherplatz, Rechenleistung oder das Vorhandensein erforderlicher Sensoren eine Rolle. Durch das Rollenkonzept wird die Selbstkonfiguration [TKU06] dahingehend beeinflusst, dass sich nicht mehr alle Knoten bei der Wahl eines zu startenden Dienstes beteiligen. Nur noch diejenigen Knoten, deren Rolle es erlaubt, den angefragten Dienst auszuführen, melden sich als potenzielle Kandidaten.

4.3.2.2 Auswirkungen des Rollenkonzepts für die Selbstoptimierung

Zur Hauptaufgabe der Selbstoptimierung in OCμ gehört die Lastverteilung. Dazu werden Dienste zwischen Knoten migriert, um stark in Anspruch genommene Knoten zu entlasten und somit eine systemweit optimale Auslastung der vorhandenen Ressourcen zu erzielen. Das neu eingeführte Rollenkonzept muss nun auch von der Selbstoptimierung berücksichtigt werden. Denn nicht jeder Dienst darf auf einen beliebigen Knoten migriert werden. Nur Knoten mit einer entsprechenden (oder gleichen) Rolle dürfen eine Dienstmigration vollziehen.

4.3.3 Die Datenbasis zur Prüfung der Nachrichten

Wenn die Middleware neu gestartet wird – sich also die teilnehmenden Knoten das erste Mal vernetzen – kommt die Selbstkonfiguration von OCμ zum Tragen [Tru06]. Dabei wird jedem Knoten eine identische Konfigurationsdatei übertragen. In dieser Datei ist hinterlegt, welche initialen Dienste in der Middleware benötigt werden und zu Beginn verteilt werden müssen. Die Knoten handeln selbständig mit einem verteiltem Wahlverfahren je nach vorhandener Ressourcen aus, welcher Dienst auf ihnen gestartet wird, weisen jedem Dienst eine eindeutige ID zu und teilen dies anschließend den anderen Knoten mit. Nach einer erfolgreich abgeschlossenen Konfiguration ist dem OCμ-System bekannt, welche Dienste auf welchen Knoten gestartet wurden.

Jeder Dienst verfügt zusätzlich über eine Schnittstelle, um der Middleware mitzuteilen, welche Nachrichtentypen von ihm aus zu seiner kompletten Laufzeit überhaupt versendet werden können. Die eindeutige Service-ID und die dazugehörigen Nachrichtentypen bilden zusammen das Wissen darüber, welcher Nachrichtenfluss zwischen zwei Knoten in OCμ als gut- oder bösartig eingestuft werden kann.

Service-ID + Nachrichtentyp = Identifizierung von Nachrichten

Bei einer Migration von Diensten zwischen verschiedenen Knoten bleibt die Service-ID erhalten wodurch hierbei keine dynamische Anpassung des Selbstschutzsystems stattfinden muss.

Ob es sich beim Nachrichtenaustausch zwischen zwei Knoten um eine gewollte und somit akzeptierte Kommunikation handelt oder ob die Nachrichten eventuell schädlich für das System sein könnten, wird von einem speziellen Immunsystem in OCμ entschieden. Der Aufbau und die genaue Funktionsweise dieses Immunsystems wird in Kapitel 6 näher beschrieben.

Immer wenn die Middleware eine Rekonfiguration erfährt oder manuell Dienste zur Laufzeit gestartet oder beendet werden, ändert sich damit auch automatisch der Wertebereich

der zulässigen Nachrichten zwischen den Knoten. In beiden Fällen kommen entweder Dienste mit neuen Nachrichtentypen hinzu oder es werden Dienste von Knoten entfernt, wobei ihre Nachrichtentypen in Zukunft nicht mehr in den Kommunikationskanälen von OCμ auftauchen werden. Eine große Fluktuation von Diensten stellt dadurch eine große Herausforderung an den Schutz der Middleware dar, weil sich die Erkennungsverfahren ständig an die neue Situation anpassen müssen. In Abhängigkeit der Rekonfigurationsgeschwindigkeit der verteilten Datenbasis kann es somit vorkommen, dass ein neu gestarteter Dienst fälschlicherweise als bösartig eingestuft wird. Dies ist dann der Fall, wenn der Dienst Nachrichten an andere Knoten in der Middleware versendet, die noch nicht von diesem neuen Dienst in Kenntnis gesetzt worden sind und die ihre Datenbasis bezüglich dieses neuen Dienstes noch nicht aktualisieren konnten. In Kapitel 7 wird näher auf diese Problematik eingegangen.

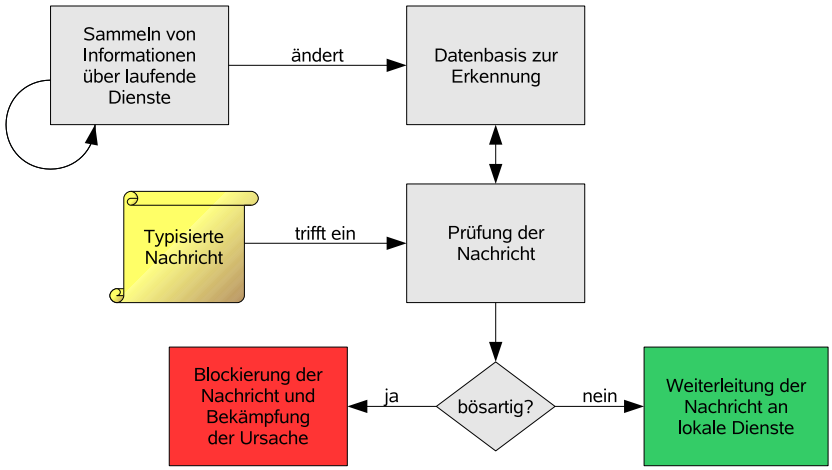


Abbildung 4.3: Ablauf und Funktionsweise des Schutzsystems auf einem Knoten.

In Abbildung 4.3 ist des Schutzsystems eines Knotens schematisch dargestellt. Die Kernaufgaben des Schutzsystems bestehen in der Anpassung des Erkennungssystems bei Veränderung der Dienste oder Knoten der Middleware und der Prüfung der eintreffenden Nachrichten.

4.3.4 Beseitigung von möglichen Bedrohungen

Falls ein Knoten eine Nachricht als bösartig einstuft, muss etwas gegen diesen Sachverhalt unternommen werden. Da in OCμ – und auch in allen gängigen Computer- oder Netzwerksystemen – keine Möglichkeit besteht, auf entfernte Knoten direkt zuzugreifen, muss eine andere Möglichkeit gefunden werden. Aus diesem Grund sendet der Knoten, der die bösartige Nachricht erkannt hat, den anderen Knoten der Middleware eine kurze Nachricht mittels Broadcast und setzt sie damit von diesem Sachverhalt in Kenntnis. Zeitgleich werden von der Middleware zwei parallele Aktionen vorgenommen:

1. *Isolation des Dienstes*
   Alle Knoten werden von dem bösartigen Dienst in Kenntnis gesetzt. Die einzige Möglichkeit diesen Dienst von einer Kommunikation zu isolieren besteht darin, dass alle Knoten alle eingehende Nachrichten von diesem Dienst sofort verwerfen und auch keine Nachrichten mehr von diesem Dienst an andere Knoten weiterleiten.
2. *Aufforderung zur Beendigung des Dienstes*
   Der Knoten, auf dem der bösartige Dienst gestartet wurde, wird aufgefordert, den entsprechenden Dienst zu beenden, um somit das Versenden der ungültigen Nachrichten zu unterbinden.

Nun kann es sein, dass sich der Knoten trotz Aufforderung als nicht kooperativ erweist und weiterhin Nachrichten von diesem bösartigen Dienst versendet. Die beste Bekämpfungsstrategie wäre eine physikalische Isolierung dieses Knotens, damit er das Netzwerk nicht weiter kompromittieren kann. Da dies jedoch in einem normalen Netzwerk und insbesondere in einem Funknetzwerk ein großes Problem darstellt, kann lediglich die Kommunikation mit diesem Knoten von der restlichen Middleware ignoriert werden. Dies wird ebenfalls dadurch erreicht, in dem die restlichen Knoten davon in Kenntnis gesetzt werden, dass nicht nur die Kommunikation mit dem Dienst, sondern auch zusätzlich die Kommunikation mit diesem *Knoten* unterbunden werden soll. Das Kapitel 8 greift die Problematik der Bekämpfung und Beseitigung von Bedrohungen näher auf. Dort wird auch auf die notwendigen Veränderungen und Erweiterungen eingegangen, die an OCμ vorgenommen werden mussten, um ein reaktives System überhaupt zu ermöglichen.

4.4 Fazit

In diesem Kapitel wurde die Gesamtarchitektur des Selbstschutzsystems in OCμ dargestellt. Es besteht aus drei Schichten, die zusammen ein gestuftes Sicherheitssystem ergeben. Die erste Stufe besteht aus der Autorisierung von Diensten und Knoten. Nur autorisierte Knoten dürfen in der Middleware selbständig Dienste starten und nur autorisierte Dienste dürfen in der Middleware mit autorisierten Knoten kommunizieren. In der darunterliegenden Schicht kommt eine rollenbasierte Zugriffskontrolle zum tragen. Durch diese Rollen kann festgelegt werden, welche Dienste auf welchen Knoten ausgeführt werden dürfen und welche nicht. Diese ersten beiden Schichten erfordern die Präsenz eines Administrators. Dieser muss sich um die Autorisierung und die korrekte Rollenvergabe kümmern.

Die dritte und letzte Schicht des Selbstschutzsystems dient dem automatischen Erkennen von fremden und eventuell bösartigen Nachrichten und dem anschließenden Beseitigen der Bedrohung. Diese Schicht agiert vorwiegend autonom und ist nicht zwingend auf

das Vorhandensein einer administrativen Person angewiesen. Durch die Verwendung der Computer-Immunologie wird ein innovativer Ansatz nach dem Vorbild der Natur verwendet. Ebenso wie der menschliche Organismus mit seinen Antikörpern, überprüft das hier verwendete Verfahren die an einem Knoten eintreffenden Nachrichten anhand einer speziellen Signatur auf Anomalien. Die genaue Arbeitsweise und die Integration der Computer-Immunologie in OCμ stellt in dieser Arbeit das Kernthema dar. In den folgenden Kapiteln wird die Funktionsweise, die Vor- und Nachteile, sowie die Effizienz des Schutzsystems genauer analysiert.

5 Computer-Immunologie

5.1 Vom biologischen Immunsystem inspiriert

Unser körpereigenes Immunsystem hat sich über Millionen von Jahren evolutionär entwickelt und hat sich als relativ effektiv herausgestellt. Wenn auch nicht alle Bedrohungen, wie beispielsweise das AIDS-Virus, vom Körper abgewehrt werden können, so kann doch eine erstaunlich große Anzahl von Eindringlingen erfolgreich erkannt und bekämpft werden.

Das Immunsystem besteht bei Wirbeltieren aus mehreren Schichten, in denen jeweils unterschiedliche Techniken zur Abwehr eingesetzt werden. Gemeinsam bilden diese Schichten ein gestuftes System, bei der die Komplexität von außen nach innen zunimmt [SB06]:

Die Haut

Die Haut dient als erste und grundlegende physikalische Barriere für Eindringlinge jeglicher Art. Sie verhindert, dass makroskopische Organismen oder sonstige fremdartige Objekte in das Innere des Körpers eindringen und dort eventuelle Schäden anrichten können.

Der pH-Wert

Die Oberfläche unserer Haut weist einen hohen pH-Wert auf. Für viele Pilze, Bakterien und Viren stellt solch ein basisches Milieu keine günstige Lebensgrundlage dar. Diese einfache aber effektive Technik sorgt somit dafür, dass solche Fremdkörper davon abgehalten werden, sich auf unserer Haut anzusiedeln und dort zu vermehren.

Die angeborene Immunabwehr

Bereits zur Geburt verfügt ein Lebewesen – egal ob Pflanze, Tier oder Mensch – über ein Immunsystem. Es entstand bereits sehr früh in der Stammesgeschichte der Lebewesen und besteht unter anderem aus Mastzellen, Phagozytosen und Killerzellen. Seit seiner Entstehung wurde dieses Abwehrsystem unverändert beibehalten und kann sich daher leider auch nicht dynamisch an neue Situationen der Umwelt anpassen.

Die adaptive Immunabwehr

Wirbeltiere haben das angeborene Immunsystem über Millionen von Jahren um ein sogenanntes adaptives Immunsystem erweitert, um auf Veränderungen der Umwelt dynamisch reagieren zu können. So kann der Organismus mit Antikörpern und etlichen anderen wichtigen Instanzen bisher unbekannte Eindringlinge meistens erfolgreich als solche erkennen und gegebenenfalls beseitigen.

Auf die Erzeugung und Funktionsweise der Antikörper in einem biologischen Organismus wird im Folgenden näher eingegangen, da dies die Schlüsselkomponenten für das verwendete künstliche Immunsystem darstellen. Die Zusammenhänge und Arbeitsweise des biologischen Immunsystems ist in der Realität sehr komplex und umfangreich [JTWS02, Wik08], weshalb hier nur ein vereinfachtes Prinzip zum grundlegenden Verständnis der Zusammenhänge vorgestellt wird.

Die grundlegende Aufgabe des biologischen Immunsystems besteht darin, zwischen körpereigenen und körperfremden Objekten zu unterscheiden und gegebenenfalls ungewollte Eindringlinge zu bekämpfen. Aus diesem Grund ist in der englischen Fachliteratur stets von *Selfs* und *Non-Selfs* die Rede. Zur Gruppe der Selfs zählen dabei alle gutartigen Objekte, die zum körpereigenen Repertoire gehören und (voraussichtlich) keinen Schaden anrichten. Zur Gruppe der Non-Selfs gehören alle Objekte, die nicht der ersten Gruppe angehören - also alle Objekte die der Körper nicht kennt und somit eventuell Schaden anrichten könnten, die sogenannten *Antigene* oder *Pathogene* [OF99].

Um zu erkennen, ob es sich bei einem Objekt um ein Self oder Non-Self handelt, wird das Proteinmuster des Objekts mit dem Proteinmuster der Antikörper verglichen. Bei einer Übereinstimmung der Proteinmuster wird automatisch von einem Non-Self und somit von einer möglichen Bedrohung ausgegangen. Um jedoch überhaupt Antikörper mit korrekten Proteinmustern erzeugen zu können, geht der Körper bei der Produktion der Antikörper nach einem bestimmten Schema vor, das sich *negative Selektion* nennt [STE05]. Bei diesem Prozess werden im Körper ständig sogenannte *T-Zellen* generiert, die ein zufällig generiertes Proteinmuster an ihrer Oberfläche aufgeprägt bekommen. Anschließend werden diese frischen T-Zellen zu einer zentralen Instanz geschickt, die das Wissen über alle körpereigenen Proteinmuster besitzt [SHF97]. Diese Instanz wird Thymus genannt und befindet sich bei Wirbeltieren und Menschen in der Nähe des Brustbeins. Im Thymus werden die frisch erzeugten T-Zellen dann mit den körpereigenen Proteinmustern verglichen. Sobald eine T-Zelle auf ein körpereigenes Proteinmuster reagiert, wird sie vernichtet [HF00]. In diesem Fall würde sie nämlich ein gutartiges Proteinmuster aus dem Self-Set als Non-Self identifizieren und Alarm schlagen, was verhindert werden soll. Erst wenn eine T-Zelle auf kein einziges körpereigenes Proteinmuster reagiert, wird sie an den eigentlichen Organismus übergegeben, der davon Abbilder in Form von Antikörpern erzeugt. Da jede T-Zelle ein zufälliges Proteinmuster besitzt, reagiert jeder Antikörper danach auf einen ganz bestimmtes Proteinmuster eines potenziellen Antigens.

Es existieren im menschlichen Organismus etwa 10^{11} mögliche unterschiedliche Proteinmuster für Antikörper, jedoch befinden sich nur etwa 10^7 unterschiedliche Antikörper zur selben Zeit in unserem Körper [HFP95]. Um trotzdem eine relativ hohe Trefferwahrscheinlichkeit bei der Detektion zu erhalten, tauscht der Körper unbenutzte Antikörper in regelmäßigen Abständen aus. Dies geschieht durch eine fest vorgegebene Lebensdauer für einen Antikörper, der bereits bei der Bildung der T-Zelle vorgegeben ist. Ist diese abgelaufen, wird der Antikörper automatisch vernichtet. Da im Körper ständig neue Antikörper mit zufälligen Rezeptoren erzeugt werden, sind immer wieder andere Rezeptoren vorhanden, die den Körper vor einem potenziellen Eindringling schützen können. Es wird geschätzt, dass das menschliche Immunsystem damit über 10^{16} unterschiedliche Proteinmuster unterscheiden kann [Inm78].

5.2 Übertragung auf die digitale Welt

Das Immunsystem des menschlichen Organismus hat sich über Millionen von Jahren evolutionär enwtickelt und an die vorhandenen Molekülstrukturen, Proteinmmuster und Grundbestandteile unseres Lebens angepasst. Die Idee der Computer-Immunologie wurde von Wissenschaftlern des Department of Computer Science der University of New Mexico und des britischen International Centre for Security Analysis zum Ende der 90er Jahre entwickelt. Erste grundlegende Entwicklungen gab es allerdings bereits 1992 an der Princeton Universität (USA) mit deren IMMSIM-Projekt [FB07].

Eine reine Transformation vom biologischen zum künstlichen Immunsystem erweist sich insofern als schwierig, da in der digitalen Welt nur ein eindimensionaler Speicher zur Verfügung steht und alle Aufgaben seriell abgearbeitet werden müssen[1]. In einem Organismus können Rezeptoren *gleichzeitig* im gesamten dreidimensionalen Raum ihre Aufgabe erfüllen, ohne sich dabei gegenseitig zu beeinträchtigen. Neben dieser Einschränkung existiert auch noch das Problem der Digitalisierung von Daten. Wenn in einem Organismus ein Molekül eine ähnliche Struktur hat wie ein anderes Molekül, dann besitzen diese in der Regel auch eine ähnliche Wirkungsweise. In der digitalen Welt aber kann beispielsweise die Bitkombination 10011 etwas völlig anderes bewirken, als die „optisch“ ähnliche Bitkombination 11001.

Wegen dieser grundlegenden Unterschiede wird in der Computer-Immunologie nicht versucht, das komplette Immunsystem des Menschen nachzubauen. Vielmehr wird in verschiedenen Forschungsarbeiten untersucht, in wie weit sich die Schlüsselkomponenten und die biologischen Grundkonzepte auf die digitale Welt abbilden lassen, worin ihr Nutzen besteht und was die Schwächen eines solchen Systems sind.

[1] Parallelrechner bieten zwar die Möglichkeit, sich von der seriellen Berechnung zu lösen, aber mit der hohen Parallelität unserer dreidimensionalen Welt können sie sich längst nicht messen.

5.3 Andere Forschungsarbeiten und deren Probleme

Die meisten bekannten Forschungsarbeiten aus dem Bereich der Computer-Immunologie befassen sich mit der Thematik, Netzwerkangriffe auf TCP/IP-Basis zu erkennen und gegebenenfalls diese Pakete nicht weiter zu leiten oder entsprechende Schutzmaßnahmen zu aktivieren. Andere Ansätze verfolgen auch das Ziel, dem immensen Auftreten von Spam-Emails entgegenzuwirken und ein künstliches Immunsystem zur Erkennung solcher unerwünschten Emails einzusetzen. Im folgenden werden derartige Systeme aus dem Bereich der Computer-Immunologie vorgestellt und näher auf deren Probleme eingegangen.

5.3.1 Das System LISYS

An der Universität von New Mexico (USA) wurde an einer Architektur für ein künstliches Immunsystem mit dem Namen LISYS geforscht [BEFG02]. Das System ist auf die Erkennung von Netzwerkangriffen ausgelegt und verwendet ein ähnliches Erkennungsmodell wie das in Kapitel 6 beschriebene Verfahren. Anhand von bekannten und gutartigen Nachrichten im Netzwerkverkehr lernt das System sein Normalverhalten und kann darauf hin ungewöhnliche Netzwerkpakete aufspüren. In LISYS wird hauptsächlich das Verfahren der *r-contiguous Bits* [HF00] eingesetzt. Bei diesem Verfahren wird ein generierter Rezeptor mit jedem Teilbereich der Nachricht verglichen und bei Übereinstimmung als schädlich eingestuft. Im Allgemeinen resultiert diese Methode jedoch in einer weitaus schlechteren Erkennungswahrscheinlichkeit als das *r-chunk-Verfahren* [SBE04], bei dem die erzeugten Rezeptoren nur an bestimmten Positionen mit der Nachricht verglichen werden. Die schlechtere Erkennungsrate resultiert aus der Tatsache, dass bei der Anwendung der *r-contiguous Bits* weniger verschiedene Rezeptoren für die zu erkennenden Bitmuster erzeugt werden können. Ein grundlegendes Problem beim Erkennen von bösartigem Netzwerkverkehr liegt aber eigentlich darin begraben, dass der Erkennungsraum nicht abgeschlossen ist und somit nur zufällig eine optimale Länge für die Rezeptoren gefunden werden kann oder vorher eingestellt werden muss [Hof99].

5.3.2 Das System ISNIDS

Das Immune System Network Intrusion Detection System – kurz ISNIDS – der Universität Idaho (USA) wurde ebenso wie LISYS darauf ausgelegt, unbekannte Angriffe auf ein Computernetzwerk aufzuspüren [HF03]. Der Prototyp wurde mit einem regelbasierten System in zwei verschiedenen Simulationsumgebungen verglichen. Es wurde sowohl das Auftreten von fälschlicherweise erkannten Bedrohungen unter normalen Bedingungen geprüft, als auch die Anzahl der nicht erkannten, aber tatsächlich durchgeführten

Attacken gezählt. Laut den Aussagen der Entwickler konnten beide Systeme sechs von acht Angriffen selbständig erkennen: ISNIDS ignorierte eine von zwei Maskierungsattacken auf TCP/IP-Basis und eine Passwort-Attacke mittels Brute-Force, wohingegen das regelbasierte System beide Maskierungsangriffe nicht erkennen konnte. Die im ISNIDS verwendeten Rezeptoren werden evolutionär durch dynamische Annäherung erzeugt. Dieser Prozess benötigt eine lange Zeit und ist bei großen Systemen nur schwer zur Laufzeit zu bewältigen. Im Vergleich dazu können Rezeptoren in OCμ in einer strukturierten Art und Weise generiert werden. Dies ist möglich, da der Middleware durch die Selbstkonfiguration das Self-Set bereits bekannt ist.

5.3.3 Erkennung von Fehlverhalten beim Routing in Ad-hoc-Netzwerken

Die Forschungsarbeiten am EPFL (Ecoles Polytechniques fédérales de Lausanne) in der Schweiz beschäftigen sich mit dem Einsatz der Computer-Immunologie im Bereich von Ad-hoc-Netzwerken im Zusammenhang mit dem Dynamic-Source-Routing [BS04]. Ein solches Netzwerk ist anfällig für ein Störverhalten, wenn bösartige Knoten die korrekten Routingregeln absichtlich oder unabsichtlich nicht beachten. Das künstliche Immunsystem soll dabei helfen, Knoten mit einem solchen Fehlverhalten im Netzwerk aufzuspüren. Dabei werden die Antikörper sowohl mit der negativen Selektion erzeugt, als auch mit dem Verfahren der sogenannten *Clonal-Selection* verändert. Bei der Clonal-Selection werden Rezeptoren aus Kopien von bereits bestehenden Rezeptoren erzeugt, in dem sie Veränderungen an bestimmten Bitpositionen aufweisen. Ebenso wie bei der Architektur von ISNIDS ist dieses Verfahren in der Middleware OCμ nicht von Relevanz, da die Middleware bereits das Wissen darüber besitzt, welche Nachrichten zum Self-Set gehören.

5.3.4 Das Starfish-System

In [KN03] wird das Starfish-System vorgestellt, das Middlewaresysteme gegen ungewollte Eingriffe schützen soll. Es wird dabei keine feste Middleware vorgegeben, sondern beschreibt seine Funktionalität unter anderem mit Jini, CORBA, DCOM und anderen darunterliegenden Plattformen. Das System stützt sich dabei auf die Redundanz der Knoten und Objekte sowie deren Kommunikation im Netzwerk. Diese Redundanz ermöglicht die Erkennung eines Fehlverhaltens anhand des byzantinischen Fehlers [CL99]. Verschiedene Knoten fügen sich zu sogenannten Voting-Groups zusammen, in denen durch spezielle Wahlverfahren das fehlerhafte Objekt ausfindig gemacht werden kann. Dabei berücksichtigt das System sowohl die Verfälschung von Nachrichten, als auch das Fehlverhalten eines Knotens, eines Prozesses oder nur eines Objekts innerhalb

eines Prozesses. Durch die ausgiebige Redundanz in allen Ebenen der Middleware ergibt sich bei diesem System jedoch ein großer Overhead für die Verwaltung und die Kommunikation. Auf speicherarmen Geräten kann dieses System daher vermutlich nicht mit befriedigenden Ergebnissen eingesetzt werden.

5.3.5 Ein reaktives System für Dienste

In [SLBK05] wird ein reaktives Immunsystem beschrieben, das sich zum Ziel gesetzt hat, Software-Dienste vor Fehlern wie Pufferüberläufen, Division durch Null oder sonstigen illegalen Speicherzugriffen zu schützen. Dazu werden die auszuführenden Programme – oder nur deren Unterprogrammaufrufe – zuerst in einem Emulator gestartet, um das Verhalten zu analysieren. Falls dabei ein Fehlverhalten auftritt, so wird das Programm in den Zustand vor dem letzten Funktionsaufruf zurückversetzt und ein neuer Rückgabewert der fehlerhaften Funktion emuliert. Dieser Vorgang wird so lange wiederholt, bis das Programm keinen der genannten Fehler mehr verursacht. Der Zustand des fehlerfreien Programms wird anschließend vom Emulator an die Prozessoreinheit zur tatsächlichen Ausführung übergeben. Dieses Verfahren bedeutet zwar einen Mehraufwand an Rechenleistung da jedes Programm vor der tatsächlichen Ausführung emuliert werden muss, jedoch können dadurch eventuelle Fehlverhalten angeblich gezielt eliminiert werden. Ein Nachteil besteht möglicherweise darin, dass das modifizierte Programm zwar nicht mehr abstürzt und keinen schädlichen Code durch einen Pufferüberlauf ausführt, aber durch die Veränderung des Rückgabewertes einer Funktion ein global anderes Verhalten aufweisen kann. Obwohl das System das Vorhandensein des Quelltextes voraussetzt, kann die Semantik des Programms nicht automatisiert mit einbezogen werden um geeignete und vor allem „korrekte“ Werte für die Belegung des Rückgabewertes der Unterfunktionen zu finden.

5.3.6 Erkennung von Spam-Emails

Ein heutzutage weit verbreitetes Problem stellt die enorme Anzahl an Spam-Emails dar. Die Carleton Universität in Ottawa (Kanada) hat sich zum Ziel gesetzt, einen Emailfilter zu entwickeln, der auf den Grundtechniken des menschlichen Immunsystems basiert [OW05]. Die dabei verwendeten Lymphozyten beinhalten einen Antikörper in Form eines Strings und eine Gewichtung dieses Antikörpers. Der Antikörper wird mit der Email verglichen und kann auch mit regulären Ausdrücken umgehen. Generiert werden die Antikörper durch zufälliges Wählen aus einem vorher festgelegtem Repertoire, wie beispielsweise einem Wörterbuch. Anschließend werden die Lymphozyten mit Spam und Ham angelernt, um diese für den realen Einsatz vorzubereiten. Um das System bezüglich seiner Erkennungswahrscheinlichkeit hin zu beurteilen, wurde es mit dem frei

verfügbaren und oft eingesetzten Spam-Assasin [Pro07] verglichen. Mit einer Trefferrate von 93,6% und fälschlicherweise als Spam eingestuften 1,1% konnte der Spamfilter mit dem Filter des Apache-Projekts mithalten und zeigt somit, dass der Einsatz der Computer-Immunologie tatsächlich erfolgreich zur Unterscheidung zwischen Spam und Ham eingesetzt werden kann. Ein Problem – mit dem jedoch alle derzeitigen Spamfilter zu kämpfen haben – sind die heutzutage vermehrt auftretenden Spam-Emails, die ihren Text in Bildern mit einem gewissen Rauschen integrieren. Vielleicht können Aspekte der Computer-Immunologie in Zukunft bei der Erkennung derartiger Spam-Emails aber auch weiterhelfen.

5.4 Computer-Immunologie in OCμ

Gerade die freie Verfügbarkeit einer ubiquitären Middelware, stellt diese vor das Problem der Sicherheit. OCμ ist bewusst ein offenes System, damit mobile Knoten an der Kommunikation innerhalb der Middleware teilhaben können. Selbstverständlich kann durch die Verwendung einer Verschlüsselung die Kommunikation von nicht gewollten Teilnehmern unterbunden werden. Aber es existieren auch Nachteile beim Einsatz von Verschlüsselungstechniken:

- Alle Knoten müssen über die selben Algorithmen zur Ver- und Entschlüsselung verfügen.
- Der Rechenaufwand zum Ver- und Entschlüsseln ist enorm – gerade bei eingebetteten Systemen.

Die Verwendung der Computer-Immunologie in OCμ ermöglicht dabei die Realisierung eines Selbstschutzsystems bei gleichzeitig möglichst geringem Ressourcenverbrauch. Ein enormer Vorteil gegenüber einer Verschlüsselung besteht in der Tatsache, dass nicht alle Kommunikationsparter den Selbstschutz aktiviert haben *müssen*, aber dennoch eine Kommunikation zwischen diesen Komponenten möglich ist. Eine verschlüsselte Nachricht kann auf einem Endgerät ohne Verschlüsselung nicht verarbeitet werden. Ein weiterer Vorteil besteht darin, dass unbekannte und noch nie zuvor dagewesene Anomalien erkannt werden können, was mit signaturbasierten Systemen nicht erreicht werden kann.

Es existieren allerdings auch Nachteile der hier verwendeten Computer-Immunologie. Das implementierte Verfahren erreicht zwar meistens eine sehr gute Erkennungsrate von 99% oder mehr. Aber dennoch stellt dies – genauso wie bei unserem menschlichen Immunsystem – keine hundertprozentige Sicherheit dar. Ein weiterer Nachteil besteht auch eventuell in der großen Anzahl an Rezeptoren, die vom System generiert und auf den jeweilgen Knoten abgespeichert werden müssen. Dies kann gerade bei ressourcenschwachen Netzwerkknoten wie Sensoren oder eingebetteten Systemen eine Hürde darstellen.

Im Abschnitt 6.2 werden allerdings Optimierungsverfahren dargestellt, die an diesem Punkt ansetzen und den benötigten Ressourcenverbrauch minimieren. In Kapitel 7.6 wird zusätzlich ein Verfahren vorgestellt, wie Knoten auch mit weniger Rezeptoren effektiv geschützt werden können.

5.5 Vergleich der einzelnen Ansätze

Die hier vorgestellten Projekte setzen alle die Computer-Immunologie ein, um ein bestimmtes Ziel zu erreichen. In Tabelle 5.1 sind die Unterschiede tabellarisch aufgelistet.

System	**LISYS**	**ISNIDS**	**Routing**	**Starfish**	**Reaktiv**	**Spam**	**OCμ**
Basiert auf Middleware	egal	egal	egal	ja	ja	nein	ja
Angriffsbasis	TCP/IP	TCP/IP	Netzwerk	Middleware	Interner Speicher	Email	Middleware
Technik	Negative Selektion	Negative Selektion	Byzantinischer Fehler	Byzantinischer Fehler	Emulation, Veränderung des Programmcodes	Antikörper, Stringvergleich, Reguläre Ausdrücke	Negative und positive Selektion
Ziel	Netzwerkangriffe erkennen	Netzwerkangriffe erkennen	Fehlerhaftes Routing erkennen	Fehlverhalten erkennen	Fehlverhalten eliminieren	Unterscheidung von Spam und Ham	Bösartige Nachrichten erkennen

Tabelle 5.1: Vergleich einiger Middlewaresysteme im Hinblick auf relevante Einflussfaktoren für das Sicherheitskonzept

Ebenso wie OCμ greifen die Projekte LISYS und ISNIDS bei der Erkennung des ungewollten Netzwerkverkehrs auf Rezeptoren mit negativer Selektion zurück. OCμ erweitert dies sogar noch durch die Verwendung der positiven Selektion. Die Problematik von LISYS und ISNIDS liegt jedoch darin begraben, dass sie sich zum Ziel gesetzt haben, den kompletten TCP/IP-Verkehr zu analysieren. Auf Grund der Menge an verschiedenen IP-Adressen und der möglichen TCP/IP-Header-Informationen entsteht ein fast „unendliches Universum“ an möglichen, zu analysierenden Nachrichtenpaketen. Genau hier liegt auch die Schwachstelle der negativen (oder positiven) Selektion begraben. Der biologische Organismus kann dieses Verfahren auch nur deswegen verwenden, weil sein Self-Set bekannt ist und sein „Universum“ auf Grund der maximal vorkommenden Proteinmuster begrenzt ist. Dennoch zeigen diese beiden Forschungsprojekte, dass eine Erkennung

in bestimmten Fällen möglich ist und das Verfahren vom Prinzip her erfolgversprechend sein kann. Ebenso zeigt das Anti-Spam-Projekt, dass die Verwendung von Rezeptoren durchaus zur erfolgreichen Unterscheidung von Self und Non-Self dienen kann.

Andere Projekte, wie das zur Erkennung von fehlerhaftem Routing in Ad-hoc-Netzwerken oder das Starfish-System, setzen verstärkt auf redundante Auslegung von Komponenten und ermitteln fehlerhafte Zustände durch den byzantinischen Fehler. Das Starfish-System ist zusätzlich auf den Schutz einer Middleware ausgelegt, jedoch im Gegensatz zum Selbstschutzsystem in OCμ startet es Dienste auf unterschiedlichen Knoten und vergleicht die unterschiedlichen Ergebnisse, um eventuelle Fehler auszuschließen. Das reaktive System verfolgt im Gegensatz dazu einen völlig anderen Ansatz und verändert den Programmcode des auszuführenden Dienstes solange, bis kein Fehlverhalten mehr auftritt. Diese Methode wird in OCμ nicht verwendet, da anschließend nicht mehr davon ausgegangen werden kann, ob sich der Dienst noch identisch nach seinen Vorgaben verhält.

5.6 Fazit

Das körpereigene Immunsystem hat sich über Millionen von Jahren zu einem autonom arbeitenden System entwickelt, das sich perfekt an unsere Umgebung angepasst hat. Dieses Immunsystem kann nicht einfach direkt in die Computerwelt übertragen werden, da das „digitale Universum“ grundlegend anders aufgebaut ist, als unsere reale Welt. Im Gegensatz zur eindimensionalen und binären Computerwelt, besteht die Realität aus Molekülen im dreidimensionalen Raum und aus ständigen Wechselwirkungen zwischen diesen Molekülen. Das biologische Immunsystem weist bekannte Schwächen auf, ist jedoch in seiner Arbeitweise dennoch effktiv.

Bei der Erforschung des biologischen Immunsystems wurden Mechanismen entdeckt, die sehr wohl in computerbasierten Systemen Anwendung finden können. Eine besonders interessante Fähigkeit besteht darin, dass das biologische Immunsystem mit unbekannten Objekten umgehen und auf diese entsprechend reagieren kann. Kernkomponenten oder bestimmte Techniken können somit aus der Biologie in die Computerwelt transferiert werden. Sie können die Middleware um einen Schutzmechanismus bereichern, der herkömmlichen Methoden in bestimmten Bereichen durchaus überlegen ist.

6 Architektur des künstlichen Immunsystems

Das körpereigene Immunsystem besitzt Antikörper, deren Rezeptoren aus unterschiedlichen Proteinmustern bestehen. Körpereigene Objekte (Selfs) oder körperfremde Objekte (Non-Selfs) werden anhand des spezifischen Proteinmusters erkannt, in dem die Antikörper ihre Rezeptoren auf ihnen testen. In einem computerbasierten System existieren keine Proteinmuster, sondern nur digiale Werte oder allgemein Bitmuster. Die zu testenden Objekte sind somit stets Bitstrings oder allgemein Nachrichten, die im Binärformat kodiert sind.

6.1 Aufbau der Rezeptoren

Die Bedeutung der Nachrichten oder der zu prüfenden Bitmuster soll vorerst keine Rolle spielen. Der folgende Ansatz ist somit universell gültig und ist nicht auf die Verwendung in der Middleware OCμ beschränkt. Grundlegend wird jedoch ein Augenmerk auf die Effizienz gerichtet. Dies bedeutet, dass bei einer Prüfung stets versucht wird, mit wenig Speicherplatz und wenig Rechenzeit auszukommen.

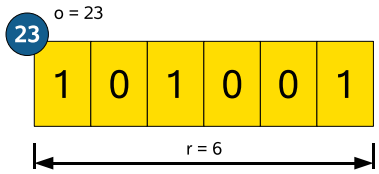


Abbildung 6.1: Beispielhafter Aufbau eines Rezeptors mit Offset 23, Rezeptorlänge 6 und dem festen Bitmuster 101001.

Um die Nachrichten in der Middleware daraufhingehend zu testen, ob sie zur Menge der Selfs oder Non-Selfs gehören, besitzen die künstlichen Rezeptoren ein Bitmuster der Länge r. Das Bitmuster für einen Rezeptor ist dabei fest vorgegeben und wird nicht wieder verändert. In einem biologischen Immunsystem docken die Rezeptoren einfach zufällig an einem (Fremd)körper an. Dies wäre auch in einem künstlichen Immunsystem

möglich, jedoch erzielt eine vordefinierte Andockposition eine bessere Trefferwahrscheinlichkeit als eine freie Platzierung [SBE04]. Aus diesem Grund besitzt jeder künstliche Rezeptor zusätzlich einen Offset o, in dem festgehalten wird, an welcher Bitposition der Rezeptor beginnen soll, sich selbst mit der Nachricht zu vergleichen.

Es existieren mehrere Aspekte, die bei dieser Konfiguration von Rezeptoren von Bedeutung sind und näher betrachtet werden müssen:

- Die Länge der Nachrichten
- Die optimale Rezeptorlänge
- Die Anzahl der verwendeten Offsets

In den folgenden Abschnitten werden diese Parameter näher betrachtet. Um aussagekräftige theoretische Ergebnisse zu bekommen, wurde ein Simulator programmiert, in dem nur die benötigten Schlüsselkomponenten von OCμ implementiert wurden.

6.1.1 Die Länge der Nachrichten

Nachrichten können innerhalb von Systemen in unterschiedlichen Längen auftauchen. Somit besitzen diese Nachrichten auch alle eine unterschiedliche Anzahl an möglichen Offsets, an denen ein Vergleich mit einem Rezeptor gestartet werden kann. Dies hat zur Folge, dass bei Systemen mit unterschiedlich langen Nachrichten niedrigere Offsets viel häufiger vertreten sind als Offsets an weiter hinten gelegenen Offsets, die nur bei langen Nachrichten auftreten. Somit müssten nur wegen der Existenz von wenigen langen Nachrichten Rezeptoren an diesen Offsets erzeugt werden. All diese Rezeptoren wären für diejenigen Nachrichten nutzlos, deren Länge geringer ist, als der Offset des Rezeptors. Um dieses Problem zu umgehen, müssen die Nachrichten auf eine einheitliche Länge l gebracht werden. Es wurden dabei zwei unterschiedliche Verfahren getestet:

1. Auffüllen oder Abschneiden von Nachrichten, um eine einheitliche Länge zu erhalten
2. Vereinheitlichung der Länge durch das Anwenden eines Hash-Verfahrens (MD5)

Bei der ersten Methode werden Nachrichten, die eine Länge unterhalb von l Zeichen aufweisen, so lange wiederholt, bis die gewünschte Länge erreicht ist. Beispielsweise wird die Nachricht `abc` bei einer Nachrichtenlänge von acht Zeichen bei diesem Verfahren zu `abcabcab` erweitert. Andererseits wird dabei allerdings auch bei Nachrichten, die aus mehr als l Zeichen bestehen, der hintere Teil einfach abgeschnitten, um die erforderliche Länge zu erreichen. Die Nachrichtenlänge sollte bei diesem Verfahren daher sinnvollerweise durch eine Maximallänge begrenzt werden, um keinen Informationsverlust durch

das Abschneiden am Ende der Nachricht zu erhalten. In jedem Fall bleibt bei diesem Verfahren die reale Repräsentation einer Nachricht nach ihrer Transformation erhalten.

Beim Hash-Verfahren, wird jede Nachricht durch einen Hash-Algorithmus in einen String konstanter Länge l überführt. Im Testsystem wurde der Hash-Algorithmus MD5 verwendet. Dieser Algorithmus wurde gewählt, da die Hash-Summe ohne großen Rechenaufwand erstellt werden kann und die resultierende Länge von 128 Bit für die Testzwecke ausreicht [Riv92]. Falls MD5 bei Systemen mit einer größeren Anzahl von Nachrichten zu viele Duplikate erzeugen sollte, kann jederzeit ein anderer Hash-Algorithmus verwendet werden, der Summen mit einer längeren Bitfolge erzeugt. Im Folgenden wird vorausgesetzt, dass alle Nachrichten die selbe Länge aufweisen.

Im Gegensatz zum Hash-Verfahren, müssen beim ersten Verfahren in der Regel auch weitaus längere Nachrichten betrachtet werden, um eine sinnvolle Unterscheidung anhand der Zeichen treffen zu können. Dies liegt grundlegend an der Repräsentation der Zeichen und Ziffern in den gängigen Zeichentabellen. Falls ein Zeichen aus acht Bits besteht, kann eine auf 128 Bit beschränkte Nachricht dabei nur aus maximal 16 Zeichen bestehen. Für eine ausreichend gute Auszeichnung einer Nachricht ist aber eine längere Stringrepräsentation normalerweise unumgänglich. Die binäre Repräsentation der Zeichen im ASCII-Code hat außerdem zur Folge, dass alle lateinischen Buchstaben und Ziffern in den ersten vier Bits nur fünf verschiedene Bitkombinationen (`0011` bis `0111`) aufweisen. Dadurch weist die binäre Repräsentation von Nachrichten eine große Regelmäßigkeit auf. Somit entstehen häufig Bereiche in den Nachrichten, bei denen aus diesem Grund keine passenden Rezeptoren erzeugt werden können.

Verwendung der realen Nachrichten

Vorteile:

- Die Originalrepräsentation der Nachricht bleibt erhalten
- Kein großer Rechenaufwand durch Auffüllen oder Abschneiden erforderlich

Nachteile:

- Maximallänge für Nachrichten erforderlich
- Längere Nachrichten müssen betrachtet werden (erfordert mehr unterschiedliche Rezeptoren)

Abbildung 6.2: Vor- und Nachteile beim Auffüllen und Abschneiden von Nachrichten, um eine einheitliche Länge aller Nachrichten zu erreichen

Verwendung eines Hash-Verfahrens

Vorteile:

- Abgeschlossener Wertebereich der möglichen Nachrichten
- Bei gleichverteiltem Hash-Algorithmus sind Berechnungen mit durchschnittlichen Wahrscheinlichkeiten möglich
- Da von der Hash-Nachricht nicht auf die Original-Nachricht geschlossen werden kann, kann keine böse Nachricht systematisch erzeugt werden. Das Wissen über das gehashte Self-Set bringt einem Angreifer reichlich wenig.

Nachteile:

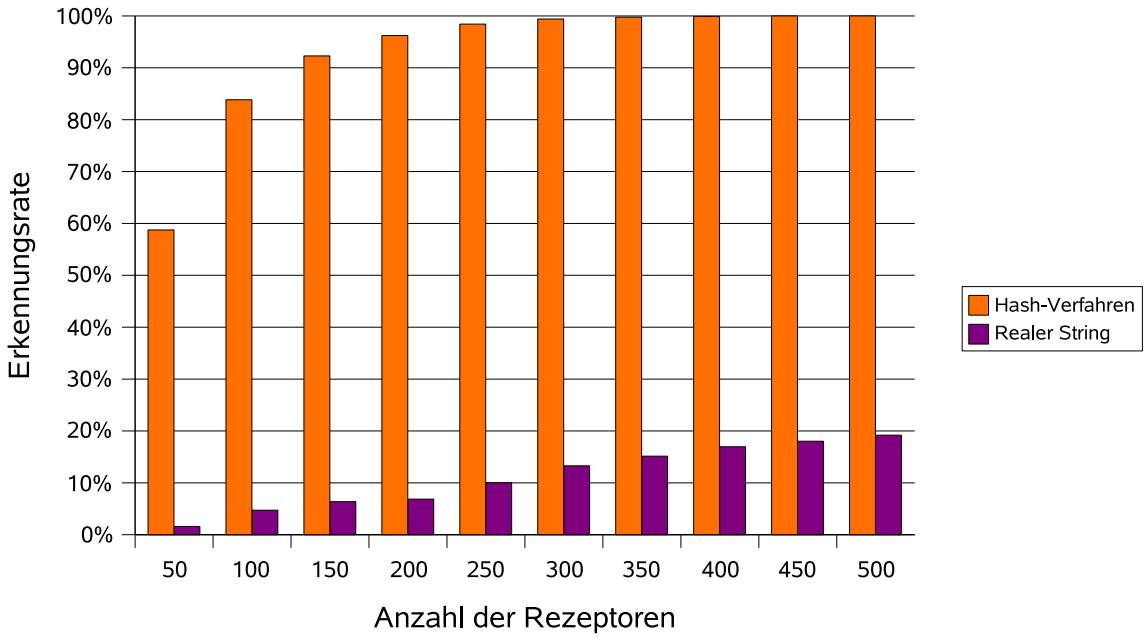
- Zusätzlicher Rechenaufwand auf einem Knoten
- Die Kenntnis über die Originalnachrichten geht verloren
- Eine Nachricht, die nicht zum Self-Set gehört kann den selben Hash-Wert aufweisen, wie eine Nachricht des Self-Sets und somit nicht erkannt werden.

Abbildung 6.3: Vor- und Nachteile um eine einheitliche Länge aller Nachrichten durch ein Hash-Verfahren zu erreichen

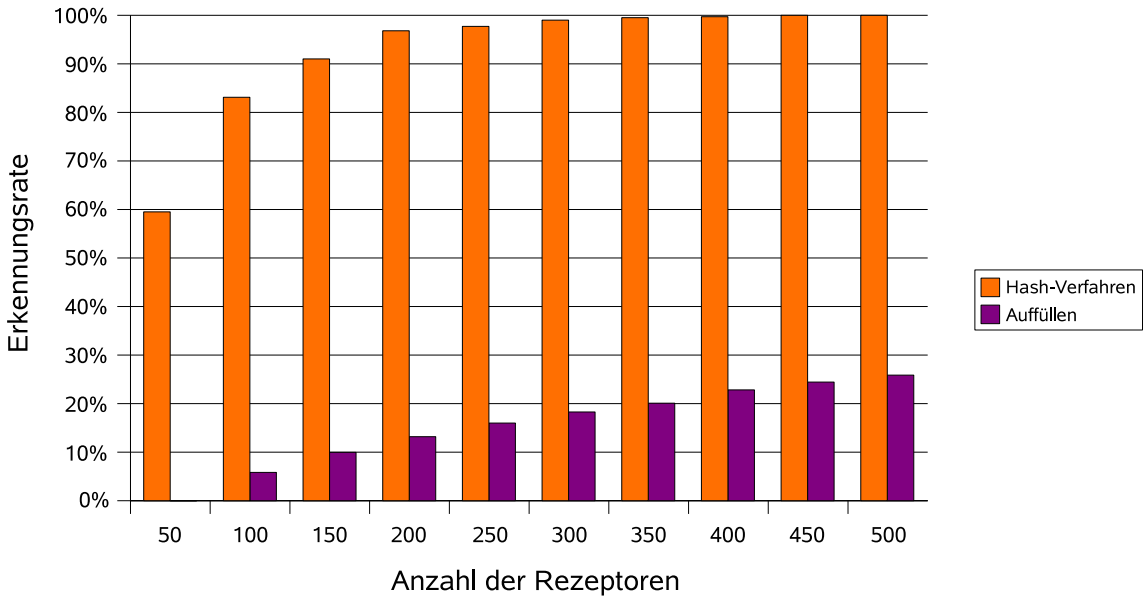
Die einzelnen Vor- und Nachteile der beiden Verfahren sind in den Abbildungen 6.2 und 6.3 nochmals aufgelistet. Auch wenn die Verwendung eines Hash-Verfahrens einige Nachteile beinhaltet, so hat sich in Testläufen allerdings herausgestellt, dass die Trefferwahrscheinlichkeit bei gleicher Rezeptoranzahl trotzdem besser ist, als wenn die Nachrichten in ihrer Ursprungsform erhalten bleiben. Nur ein Hashverfahren ist in der Lage, eine Nachricht auf eine geringe Bytelänge zu reduzieren, bei der jedes Bit annähernd gleich relevant ist. Bei der Verwendung der Originalnachricht ergeben sich auf Grund der verwendeten Zeichentabellen ungünstige Häufigkeiten bei bestimmten Bitmustern. Auch die unterschiedliche Häufigkeit von Buchstaben in den Nachrichten verzerrt die Wertigkeit einzelner Bitpositionen.

In Abbildung 6.4 ist die deutlich bessere Erkennungsrate des Hash-Verfahrens im Gegensatz zur realen Darstellung der Nachricht zu erkennen[1]. Die Ergebnisse beim Hash-Verfahren sind sowohl bei einer gehashten Nachrichtenlänge von 128 Bits als auch von 256 Bits relativ identisch. Das Verfahren, bei dem die String-Repräsentation einer Nachricht erhalten bleibt, zeigt bei einer längeren Nachricht auch eine bessere Trefferwahrschein-

[1]Der genaue Simulationsaufbau wird im Anhang A.1 genauer beschrieben.



(a) 128 Bit Nachrichtenlänge



(b) 256 Bit Nachrichtenlänge

Abbildung 6.4: Erkennungsraten der beiden Varianten in einem System mit 100 Self-Nachrichten und einer gegebenen Rezeptorlänge von sechs Bits.

lichkeit. Dies lässt vermuten, dass die erste Variante ihre Vorteile erst bei sehr vielen langen Nachrichten ausspielen könnte. Allerdings zieht dies auch einen Mehraufwand bei der Erzeugung der Rezeptoren nach sich, da viel mehr mögliche Bitmuster an unterschiedlichen Offsets möglich sind. Da diese Architektur jedoch auch auf kleinen Geräten ihren Einsatz finden soll, wird im weiteren Verlauf der Arbeit nur das Hash-Verfahren berücksichtigt.

6.1.2 Einteilung der Nachrichten in Offsetgruppen

Nun wird die Unterteilung der Nachrichten in unterschiedliche Offsets genauer betrachtet. Jede der s unterschiedlichen Nachrichten, die zum Self-Set des Systems gehören, besitzen bei einer Länge l genau l denkbare Bitpositionen für eventuelle Offsets. Bei einer Rezeptorlänge von r Bits können jedoch nur $p = l - r + 1$ verschiedene Offsets verwendet werden, da die Rezeptoren sonst am Ende über die Nachrichtenlänge hinausragen würden.

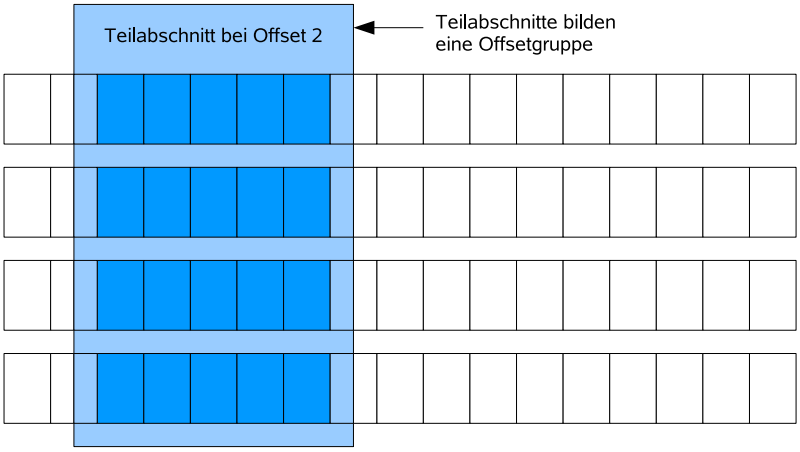


Abbildung 6.5: Beispielhafte Offsetgruppe bei Verwendung einer Rezeptorlänge von fünf Bits und einem Offset an der zweiten Position.

Ein *Teilabschnitt* einer Nachricht ist als der Bereich einer Nachricht definiert, der an Offset o beginnt und bis zur Bitposition $o + r$ reicht. Alle Teilabschnitte aller Nachrichten, die am selben Offset beginnen und die selbe Länge aufweisen werden als *Offsetgruppe* bezeichnet (siehe Abbildung 6.5). Ein Rezeptor mit Offset o und Rezeptorlänge r vergleicht sein Bitmuster demnach immer mit allen Teilabschnitten aus der entsprechenden Offsetgruppe. Daraufhin kann dann entschieden werden, ob die komplette Nachricht als Self oder Non-Self eingestuft wird. Wie bereits erwähnt, können alle Nachrichten, die zur Menge der Selfs gehören, in maximal p unterschiedliche Offsetgruppen eingeteilt werden. In jeder dieser Offsetgruppen weisen die Teilabschnitte der jeweiligen Nachrichten unterschiedliche Bitmuster auf. Von allen möglichen Bitkombinationen der Länge r

ist eine gewisse Anzahl v in jeder Offsetgruppe vorhanden. Der Anteil der vorhandenen Bitkombinationen wird als Nutzungsgrad $u = \frac{v}{2^r}$ bezeichnet und kann bei jeder Gruppe einen anderen Wert aufweisen. Dieser Sachverhalt wird im Beispiel 6.1.1 verdeutlicht.

Beispiel 6.1.1
Es wird angenommen, dass eine Rezeptorlänge von drei Bits verwendet wird. Somit sind maximal acht unterschiedliche Bitkombinationen möglich. An einem bestimmten Offset kommen in den Nachrichten, die zur Gruppe der Selfs gehören, folgende fünf unterschiedliche Bitmuster vor:

{001, 010, 011, 101, 111}

Insgesamt sind jedoch bei Verwendung von drei Bits acht verschiedene Bitmuster denkbar. Für den Nutzungsgrad u dieser Offsetgruppe ergibt sich: $u = \frac{v}{2^r} = \frac{5}{8} = 62{,}5\%$.

Je kürzer die Rezeptorlänge bei gleichbleibender Nachrichtenanzahl gewählt wird, desto höher ist im Mittel der Nutzungsgrad der Offsetgruppen. Im Gegenzug ist dieser durchschnittliche Nutzungsgrad geringer, je länger die Rezeptoren sind.

6.1.3 Negative und positive Selektion

Die aus dem biologischen Immunsystem bekannten Verfahren der negativen und positiven Selektion finden auch in der künstlichen Version Verwendung. Bei der negativen Selektion werden all diejenigen Bitmuster für Rezeptoren herangezogen, die *nicht* in der entsprechenden Offsetgruppe der Self-Nachrichten vorhanden sind. Rezeptoren inklusive Offset werden *Antikörper* genannt.

Bei der positiven Selektion handelt es sich um das exakte Gegenstück zur negativen Selektion. In diesem Fall werden nur diejenigen Bitmuster für Rezeptoren herangezogen, die in der entsprechenden Offsetgruppe auch tatsächlich in der Menge der Self-Nachrichten vorhanden sind. Diejenigen Rezeptoren, die mit der positiven Selektion gewonnen werden, werden *Prokörper*[2] genannt.

Bei einem hohen Nutzungsgrad u einer Offsetgruppe können somit mehr Prokörper als Antikörper generiert werden, wogegen bei einem geringeren Nutzungsgrad mehr Antikörper als Prokörper generiert werden können.

Es gib keine generellen Vor- und Nachteile beim Einsatz der negativen und positiven Selektion. Es ist vielmehr vom Anwendungsszenario anhängig, welches Verfahren vorgezogen werden soll. In einem kleinen abgeschlossenen System, das aus wenigen hundert Self-Nachrichten besteht, ist von Antikörpern, die mit der negativen Selektion erzeugt

[2] In der Natur existiert kein exaktes Gegenstück zu Antikörpern, weshalb der Begriff „Prokörper“ eine Wortneuschöpfung in dieser Arbeit darstellt.

wurden, eher abzuraten. Diese würden bei weitem keine so gute Trefferrate erzielen, wie wenn Prokörper mit der positiven Selektion zum Einsatz kommen würden. Bei einem sehr kleinen Self-Set kann es auch von Vorteil sein, die Nachrichten einfach komplett abzuspeichern. Somit ist ein direkter und absolut treffsicherer Vergleich möglich. Bei herkömmlichen Virenscannern werden beispielsweise die Signaturen des kompletten Non-Self-Sets abgespeichert, um auf eventuellen Schadcode prüfen zu können. Dies entspricht in etwa den hier verwendeten Prokörpern, wobei diese nur einen kurzen Teilabschnitt dieser Signaturen repräsentieren.

Bei dem hier betrachteten Einsatzgebiet werden jedoch Systeme mit mehreren Tausend oder gar Millionen von Self-Nachrichten angenommen. Bei dieser großen Anzahl von unterschiedlichen Nachrichten kann der direkte und exakte Vergleich der Nachrichten einen sehr großen Overhead für das System darstellen. Die Verwendung der Antikörper stellt hierbei ein Mittel zur Verfügung, die Anzahl der Vergleiche zu minimieren. Dies ist jedoch nur mit einem gleichzeitigen Verlust der Trefferwahrscheinlichkeit möglich. Im Folgenden werden Parameter aufgezeigt, die es dennoch ermöglichen, eine bestmögliche Trefferrate bei geringstmöglicher Rezeptoranzahl zu erzielen.

6.1.4 Anwendung der Rezeptoren auf Nachrichten

Um Nachrichten dahingehend zu testen, ob sie zum Self-Set gehören oder nicht, kann sowohl die positive als auch die negative Selektion verwendet werden. Die Anwendung der beiden Methoden und der daraus folgende Entscheidungsprozess kann jedoch nicht identisch erfolgen, da ihre Erzeugung auch auf unterschiedliche Art und Weise statt findet.

Definition 6.1.1

richtig positiv: *Das Ergebnis eines Tests wird als richtig positiv bezeichnet, wenn korrekt angezeigt wird, dass das gesuchte Ergebnis gefunden wurde. Das gesuchte Ergebnis wurde erkannt und tritt ein.*

richtig negativ: *Das Ergebnis eines Tests wird als richtig negativ bezeichnet, wenn korrekt angezeigt wird, dass das gesuchte Ergebnis nicht gefunden wurde. Das gesuchte Ergebnis wurde nicht erkannt und tritt auch nicht ein.*

falsch positiv: *Das Ergebnis eines Tests wird als falsch positiv bezeichnet, wenn fälschlicherweise angezeigt wird, dass das gesuchte Ergebnis gefunden wurde. Das gesuchte Ergebnis wurde erkannt, tritt aber nicht ein.*

falsch negativ: *Das Ergebnis eines Tests wird als falsch negativ bezeichnet, wenn fälschlicherweise angezeigt wird, dass das gesuchte Ergebnis nicht gefunden wurde. Das*

gesuchte Ergebnis tritt ein, wurde aber nicht erkannt.

Antikörper werden durch den Prozess der negativen Selektion erzeugt. Ihr binärer Rezeptor wird mit einer Nachricht an einer spezifischen Stelle - dem Offset - verglichen. Falls das Bitmuster von Rezeptor und dem entsprechenden Teilabschnitt einer Nachricht identisch ist, wird die Nachricht als Non-Self eingestuft. Wäre dieses Bitmuster in mindestens einer Self-Nachricht vorhanden, wäre dieser Antikörper nicht erzeugt worden. Ein *einziger* Antikörper kann somit dazu beitragen, dass eine unbekannte Nachricht entlarvt wird (*richtig positiv*). Falls kein Antikörper im System auf eine Nachricht reagiert, kann jedoch im Gegenzug *nicht* unbedingt davon ausgegangen werden, dass es sich um eine Nachricht aus dem Self-Set handelt (*richtig negativ*). Es besteht immer die Möglichkeit, dass für eine bestimmte Nachricht kein Antikörper erzeugt werden konnte, da jeder mögliche Antikörper auch eine Nachricht aus dem Self-Set als Non-Self erkennen würde (*falsch negativ*). Diese Fehlerrate gilt es zu minimieren.

Anders verhält es sich bei Verwendung der positiven Selektion und den daraus resultierenden Prokörpern. Einzelne Prokörper haben im Gegensatz zu Antikörpern keinen Nutzen für das Erkennungssystem, sondern müssen immer in Gemeinschaft betrachtet und angewendet werden. Prokörper, die die selbe Rezeptorlänge und den selben Offset aufweisen, werden als *Prokörper-Set* bezeichnet. Bei der Prüfung einer Nachricht an einem bestimmten Offset, muss immer genau ein Prokörper aus einem solchen Set mit dem entsprechenden Bitmuster der Nachricht übereinstimmen. Erst dann kann daraus gefolgert werden, dass dieser Bereich tatsächlich ein Teil einer bekannten Self-Nachricht ist. Es kann dann jedoch noch nicht darauf geschlossen werden, dass die komplette Nachricht auch tatsächlich zum Self-Set gehört. Erst wenn *alle* vorhandenen Prokörper-Sets genau einen Treffer beim Vergleich einer Nachricht aufweisen, kann die Nachricht mit relativ guter Sicherheit als Self eingestuft werden (*richtig negativ*). Wenn mindestens eines dieser Sets keinen passenden Prokörper beinhaltet, kann mit Sicherheit davon ausgegangen werden, dass es sich um eine Nachricht aus dem Non-Self-Bereich handelt (*richtig positiv*).

In Abbildung 6.6 ist das unterschiedliche Entscheidungsverfahren bei Pro- und Antikörpern zu sehen. Bedingt durch die Funktionsweise der Anti- und Prokörper ist es nie möglich, dass ein Vergleich „*falsch positiv*“ ergibt. Falls eine Nachricht als Non-Self eingestuft wird, ist diese Aussage somit immer zu 100% korrekt. Wenn die Nachricht als Self beurteilt wird, kann dies jedoch mit einer Fehleinschätzung behaftet sein (dies soll das Fragezeichen in Abbildung 6.6 verdeutlichen). Da eine Non-Self-Nachricht eventuell ein geschickt gewähltes Bitmuster aufweist, die teilweise immer mit Nachrichten aus dem Self-Set übereinstimmen, jedoch als gesamtes betrachtet keine Nachricht aus dem Self-Set ist, kann sie auch nicht als Non-Self entlarvt werden. Diesen Fehler gilt es zu minimieren. In den folgenden Abschnitten wird daher näher auf die Erzeugung der Rezeptoren und deren Effektivität und Effizienz eingegangen. Da die Prokörper dabei

Antikörper ⇓				**Prokörper** ⇓
mindestens einer matched	$\Longrightarrow$	**Non-Self**	$\Longleftarrow$	mindestens ein Set matched nicht
keiner matched	$\Longrightarrow$	**Self**$^{(?)}$	$\Longleftarrow$	alle Sets matchen

Abbildung 6.6: Pro- und Antikörper besitzen unterschiedliche Regeln, um zwischen Self und Non-Self zu entscheiden

jedoch eher eine spezielle Erweiterung des Erkennungssystems darstellen, wird zuerst nur die Funktionsweise der Antikörper betrachtet. Erst ab Kapitel 6.2.3 wird auch die Verwendung der Prokörper hinzugenommen.

6.1.5 Die optimale Rezeptorlänge

Für ein effektives Erkennungssystem ist es wichtig, welche Länge die verwendeten Rezeptoren aufweisen. Falls ein zu kurzer Rezeptor bei Antikörpern eingesetzt wird, kann es sein, dass überhaupt kein Antikörper produziert werden kann, da alle Bitmuster bereits in der Offsetgruppe der Self-Nachrichten vorhanden sind. Ein zu langer Rezeptor besitzt dieses Problem zwar nicht, dennoch ergeben sich ineffiziente Seiteneffekte. Der Nutzungsgrad der Offsetgruppen ist in diesem Fall sehr gering, was viel Freiraum für die Wahl von Antikörpern lässt. Somit können zwar viel mehr unterschiedliche Rezeptoren an den jeweiligen Offsets erzeugt werden, jedoch nimmt diese Variante auch viel Kapazität in Anspruch, um all diese Rezeptoren zu speichern und zu vergleichen.

In einem Simulator[3] wurde deshalb neben der Effektivität auch die Effizienz der unterschiedlichen Rezeptorlängen getestet. Im Testsystem wurden zuerst zufällige Bitmuster für die Nachrichten des Self-Sets erzeugt und anschließend maximal 1000 Antikörper generiert. Es wurde bei jeweils einer unterschiedlichen Anzahl von Selfs die Wirksamkeit unterschiedlicher Rezeptorlängen getestet. Um die Trefferrate zu bestimmen, wurden 1000 zufällige Testnachrichten erzeugt, die nicht Teil des Self-Sets waren. Je mehr Nachrichten von den Antikörpern als Non-Self eingestuft wurden, desto höher wurde die Wirksamkeit der entsprechenden Konfiguration eingestuft. In Abbildung 6.7 sind die Ergebnisse der einzelnen Testläufe im Diagramm dargestellt. Zur Ermittlung der Testergebnisse wurden verschiedene Simulationsumgebungen aufgesetzt, die sich jeweils in der

[3]Der genaue Simulationsaufbau wird im Anhang A.2 genauer beschrieben.

Größe des Self-Sets unterschieden haben. Da die Self-Nachrichten zufällig generiert wurden, wurden die Testläufe mehrmals durchgeführt und jeweils der Mittelwert berechnet, um eventuelle Nebeneffekte von bestimmten Zufallswerten auszuschließen.

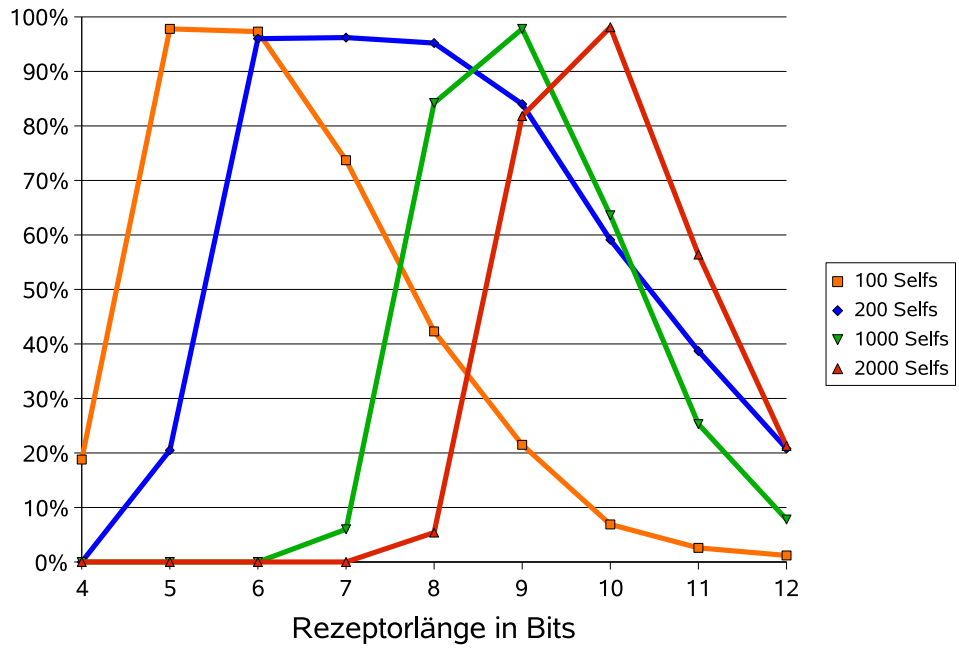


Abbildung 6.7: Trefferwahrscheinlichket bei unterschiedlichen Rezeptorlänge in einer Simulation bei maximal 1000 Antikörpern

Bei einem Self-Set der Größe 100 hat eine Rezeptorlänge von fünf und sechs Bits die beste Trefferquote erzielt, wogegen diese Rezeptorlänge bei einem Self-Set von 2000 Nachrichten überhaupt keine Nachrichten erkannt hat. Dies liegt in dem Nutzungsgrad der Offsetgruppen begraben, der bei 2000 Selfs und bei einer Rezeptorlänge $r < 7$ immer noch bei 100% lag. Erst ab acht Bit konnten überhaupt Rezeptoren für Antikörper erzeugt werden. Eine optimale Trefferrate ergab sich in dieser Konfiguration bei Verwendung von 10 Bits. Eine Rezeptorlänge von 10 Bits war wiederum suboptimal als es um die Erkennung von Non-Selfs in einem System von nur 1000 Selfs ging und eine minimale Anzahl an Rezeptoren erzeugt werden sollte. Eine Verwendung von neun Bits für die Rezeptoren resultierte bei diesem Szenario in einer weitaus besseren Trefferwahrscheinlichkeit.

Beim Durchlauf mehrerer Tests hat sich letztendlich gezeigt, dass es nicht sinnvoll ist, eine feste Rezeptorlänge für ein Erkennungssystem in Betracht zu ziehen. Viel mehr hat sich herausgestellt, dass die Rezeptorlänge in Abhängigkeit der *nicht* zu erkennenden Anzahl von Nachrichten des Self-Sets steht. Dieser Zusammenhang kann auch mathematisch erklärt werden. In einem System mit Nachrichten einer Länge von l Bits kann die Anzahl p der möglichen Offsets bei der Verwendung einer Rezeptorlänge von r Bits

wie folgt bestimmt werden:

$$p = l - r + 1, \qquad r \leq l \tag{6.1}$$

Wenn jeder Antikörper einen fest zugewiesenen Offset besitzt, an dem er sein Bitmuster mit der Nachricht vergleicht, sind maximal a_{all} unterschiedliche Antikörper möglich:

$$a_{all} = 2^r \cdot p \tag{6.2}$$

Angenommen das Self-Set in einem System besteht aus s unterschiedlichen Nachrichten, die nicht von Antikörpern erfasst werden sollen. Dann können im ungünstigsten Fall nur

$$a_{self} = s \cdot p \tag{6.3}$$

Antikörper erzeugt werden, die auch tatsächlich keine Nachricht aus dem Self-Set erkennen. Daraus ergeben sich a_{eff} effektive Antikörper:

$$a_{eff} = a_{all} - a_{self} = 2^r \cdot p - s \cdot p \tag{6.4}$$

Wenn nun angenommen wird, dass maximal so viele Antikörper erzeugt werden sollen, wie die gesamte Anzahl der im System vorhandenen Self-Nachrichten, dann kann die optimale Rezeptorlänge folgendermaßen abgeschätzt werden:

$$\begin{aligned} a_{eff} &\leq s && (6.5)\\ 2^r \cdot p - s \cdot p &\leq s && (6.6)\\ 2^r &\leq s \cdot \frac{1+p}{p} && (6.7)\\ r &\lesssim log_2(s) && (6.8) \end{aligned}$$

Dies zeigt, dass die Länge der Rezeptoren kleiner als der binäre Logarithmus der Anzahl der Self-Nachrichten gewählt werden soll. Die Länge muss aber stets mindestens so groß gewählt werden, dass sich der Nutzungsgrad einer Gruppe unterhalb von 100% befindet, denn sonst besteht keine Möglichkeit, überhaupt Antikörper in einer Offsetgruppe zu erzeugen. In den Testsystemen hat sich gezeigt, dass die Antikörper den besten Erkennungsgrad erzielten, wenn die Gruppen einen durchschnittlichen Nutzungsgrad von unterhalb 95% aufwiesen. Ein höherer Nutzungsgrad resultierte in zu wenig möglichen Antikörpern für eine akzeptable Erkennungsrate.

6.1.6 Koordinierte Erzeugung der Rezeptoren

In unserem Körper werden die Rezeptoren von Antikörpern mit zufälligen Proteinmustern erzeugt und mit dem Mechanismus der negativen Selektion ausgesondert. In der Middleware muss jedoch kein Zufallsprinzip zum Erzeugen der Rezeptoren verwendet werden, denn auf Grund der integrierten Selbstkonfiguration besteht bereits das Wissen über die komplette Menge aller Bitmuster, die zum Self-Set gehört. Die Generierung der Rezeptoren kann somit nach einem geordneten Schema erfolgen.

Algorithmus 1 Die Erzeugung von Anti- und Prokörpern an einem bestimmten Offset

```
r = 1; // initiale Rezeptorlänge ist 1
pattern = 0; // binäre Repräsentation eines Musters
loop
  while pattern < 2^r do
    if groupMatches(pattern) then
      possibleProbody(pattern); // positive Selektion ⇒ Prokörper
    else
      possibleAntibody(pattern); // negative Selektion ⇒ Antikörper
    end if
    pattern++;
  end while
  pattern=0;
  r++;
end loop
```

Der Algorithmus 1 erzeugt alle möglichen Anti- und Prokörper an einem bestimmten Offset. Beginnend mit einer Rezeptorlänge von einem Bit wird der Rezeptor sukzessive verlängert. Da prinzipiell keine obere Schranke für die Rezeptorlänge angegeben ist, terminiert der angegebene Algorithmus nicht selbständig. Ein viel längerer Rezeptor als der binäre Logarithmus von s (bei s unterschiedlichen Nachrichten im Self-Set) macht jedoch im Allgemeinen keinen Sinn, wenn die Anzahl der Rezeptoren gering gehalten werden soll. Ebenso ist eine obere Schranke implizit durch die Nachrichtenlänge l gegeben. Der Algorithmus kann allerdings zu jeder Zeit abgebrochen werden. Je mehr Rezeptoren jedoch erzeugt wurden, desto besser ist die Erkennungswahrscheinlichkeit. Gleichzeitig verschlechtert sich dabei aber auch das Laufzeitverhalten beim Test der Nachrichten. Wieviele Rezeptoren erzeugt werden, ist demnach eine Frage der vorhandenen Ressourcen und der Rechenleistung des Knotens, auf dem die Rezeptoren später zum Einsatz kommen sollen.

6.1.7 Die Wahl der zu verwendenden Offsets

Bei der Verwendung der Technik, die in den vorigen Kapiteln erklärt wurde, ist stets ein Augenmerk darauf gelegt, eine größtmögliche Trefferrate bei geringstmöglicher Anzahl von Rezeptoren zu erreichen. Knoten besitzen keine unendliche Kapazität zum Abspeichern der Rezeptoren. Aus diesem Grund sollten die (wenigen) Rezeptoren wenigstens so gewählt werden, dass sie die bestmögliche Erkennungsrate erzielen. Bei einer festen Anzahl an abspeicherbaren Rezeptoren stellt sich aus diesem Grund die Frage, an welchen Offsets wieviele Rezeptoren generiert und an welchen Offsets eher weniger Rezeptoren erzeugt werden sollen. Es wurden verschiedene Strategien auf ihre Effektivität hin getestet:

- Generierung von Rezeptoren an den Offsetgruppen, die den geringsten Nutzungsgrad u aufweisen
- Generierung von Rezeptoren an den Offsetgruppen, die den höchsten Nutzungsgrad u aufweisen
- Gleichmäßige Generierung von Rezeptoren an allen Offsets

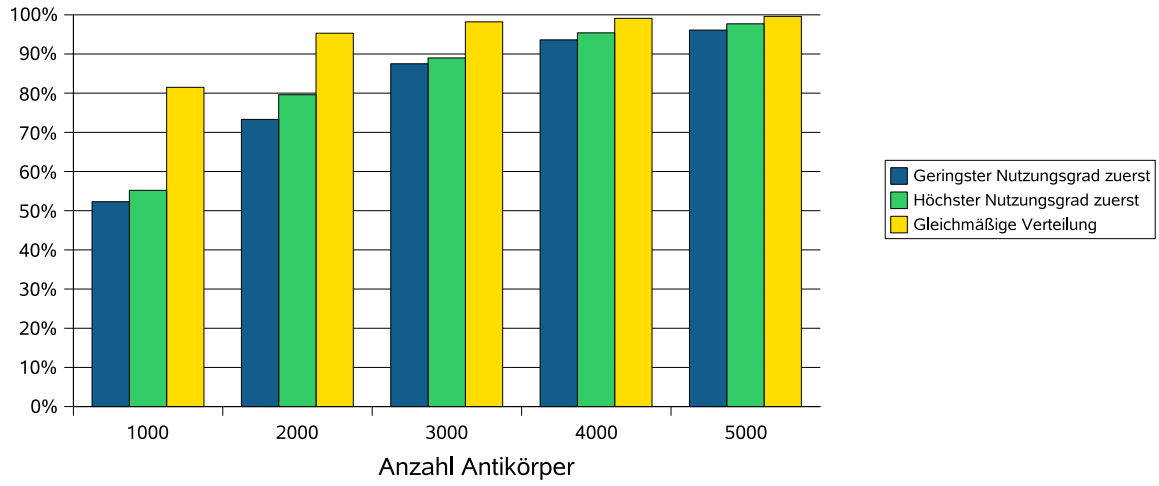


Abbildung 6.8: Trefferwahrscheinlichkeit bei Generierung von Antikörpern an verschiedenen Offsets

In einer Simulationsumgebung[4] mit 1000 zufällig generierten Self-Nachrichten wurde zu jeder der drei Strategien jeweils die gleiche Anzahl von Rezeptoren erzeugt. Auch hier wurde eine mehrmalige Durchführung der Tests durchgeführt und der Mittelwert der Ergebnisse berechnet, um eventuelle Zufallseffekte auszuschließen. In Abbildung 6.8 sind

[4]Der genaue Simulationsaufbau wird im Anhang A.2 genauer beschrieben.

die Ergebnisse der verschiedenen Testdurchläufe im Balkendiagramm dargestellt. Es ist deutlich zu erkennen, dass die Erkennungsrate bei einer gleichmäßigen Verteilung der Antikörper über alle Offsets am höchsten war. Dies ist am deutlichsten zu erkennen, wenn nur 1000 Rezeptoren erzeugt wurden. Als zweitbeste Strategie stellte sich die Erzeugung derjenigen Antikörper heraus, deren Offsets zu den Gruppen gehörten, die den höchsten Nutzungsgrad aufwiesen. Minimal schlechter erwies sich die dritte Strategie, die Antikörper zuerst in den Gruppen zu erzeugen, bei denen die meisten freien Bitmuster für Antikörper vorhanden waren. Dass die drei Strategien bei steigender Anzahl der Antikörper sich immer mehr an die selbe Trefferwahrscheinlichkeit annähern, lässt sich dadurch erklären, dass bei 5000 Antikörpern bereits fast alle möglichen Antikörper erzeugt wurden und somit fast kein Unterschied der drei Methoden mehr vorhanden ist.

In diesem Test ist ersichtlich geworden, dass es für eine bestmögliche Erkennungsrate wichtig ist, dass Antikörper an allen möglichen Offsets zu einem annähernd gleichen Anteil vorhanden sind. Falls Antikörper nur an wenigen verschiedenen Offsets generiert werden, so besteht eine größere Wahrscheinlichkeit, dass Non-Self-Nachrichten existieren, die ein Bitmuster aufweisen, das von keinem Antikörper erkannt werden kann.

6.1.8 Anzahl der notwendigen Rezeptoren

Ein System, das nur eine Teilmenge aller möglichen Rezeptoren enthält, kann maximal gleich viele oder weniger Nachrichten als Non-Self identifizieren. Im Allgemeinen ist anzunehmen, dass eine Reduzierung der Rezeptoren in einer schlechteren Erkennungsrate resultiert. Um eine bessere Vorstellung davon zu bekommen, wie effektiv das System bei einer reduzierten Anzahl von Rezeptoren ist, wurde die Erkennungsrate in verschiedenen Simulationsumgebungen bei unterschiedlichen Rezeptorlängen getestet. Alle Simulationsergebnisse wurden durch ein identisches Verfahren gewonnen:

Zuerst wurde ein System mit einer bestimmten Anzahl an Self-Nachrichten erstellt. Anschließend wurde die optimale Rezeptorlänge bestimmt und automatisch *alle* möglichen Antikörper mit dieser Rezeptorlänge generiert. Zu Beginn der Testläufe wurden keine Rezeptoren auf dem Knoten verwendet, was stets in einer Erkennungsrate von 0% resultierte. Im weiteren Testverlauf wurde die Rezeptoranzahl schrittweise um jeweils fünf Prozentpunkte erhöht, bis alle erzeugten Rezeptoren zum Einsatz herangezogen wurden. Durch die Konfrontation des Knotens mit 1000 zufällig generierten Non-Self-Nachrichten wurde die Erkennungsrate im jeweiligen Testsystem ermittelt.

Es wurden vier unterschiedliche Simulationen durchgeführt, deren Ergebnisse in den Abbildungen 6.9 bis 6.12 zu sehen sind.

Testsystem 1

Das erste Testsystem bestand aus 100 Self-Nachrichten. Bei einer Rezeptorlänge von sechs Bits konnten maximal 1104 unterschiedliche Rezeptoren generiert werden. In Abbildung 6.9 ist der kontinuierliche Anstieg der Erkennungsrate deutlich zu erkennen. Bereits ab einer Anzahl von 331 Rezeptoren (30%) konnte eine ausreichend gute Trefferwahrscheinlichkeit erreicht werden. Bereits ab einer Verwendung von 55% der vorhandenen Rezeptoren konnten in diesem Testsystem stets alle Non-Selfs erkannt werden.

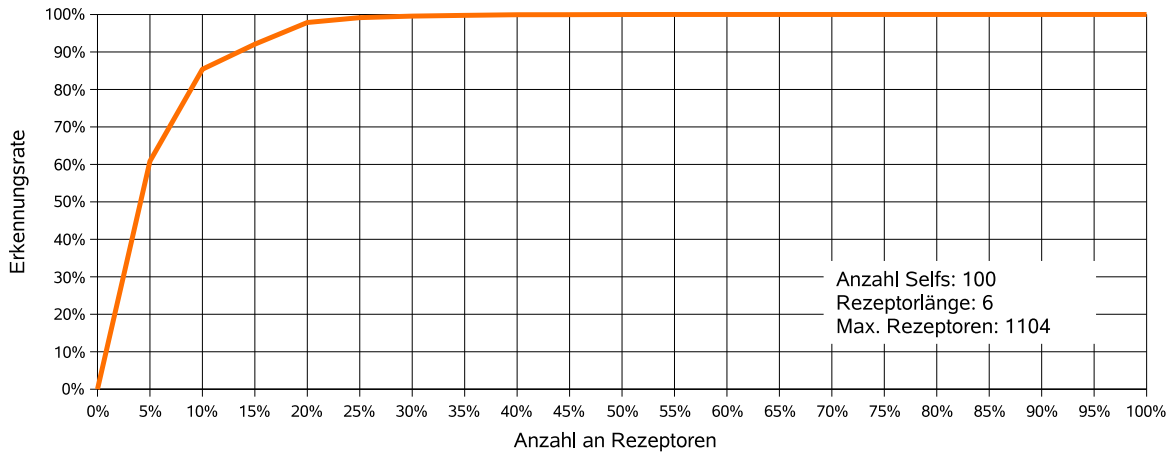


Abbildung 6.9: Erstes Testsystem bei Erhöhung der Rezeptoranzahl

Testsystem 2

Anschließend wurde ein Testsystem mit 1000 Self-Nachrichten generiert. Bei einer Rezeptorlänge von neun Bits konnten hier maximal 6125 unterschiedliche Rezeptoren erzeugt werden. In Abbildung 6.10 ist ebenso der stetige Anstieg der erkannten Non-Selfs zu sehen, wenn die Anzahl der Rezeptoren erhöht wurde. Sobald 40% der Rezeptoren zum Einsatz kamen, betrug die Erkennungsrate bereits über 99%. Eine vollständige Erkennung wurde ab 5206 (85%) unterschiedlichen Rezeptoren erzielt.

Testsystem 3

Der dritte Testaufbau bestand aus einem Self-Set von insgesamt 10000 Nachrichten. Das System errechnete eine optimale Rezeptorlänge von 12 Bits, was in 30389 unterschiedlichen Antikörpern resultierte. Der Anstieg der Kurve ist deutlich langsamer, als bei den vorigen beiden Testsystemen und erst ab 18233 unterschiedlichen Rezeptoren konnte eine Erkennungsrate von über 99% festgestellt werden. Beim Einsatz von allen Rezeptoren wurden jedoch auch hier alle Nachrichten korrekt als Self oder Non-Self erkannt.

Testsystem 4

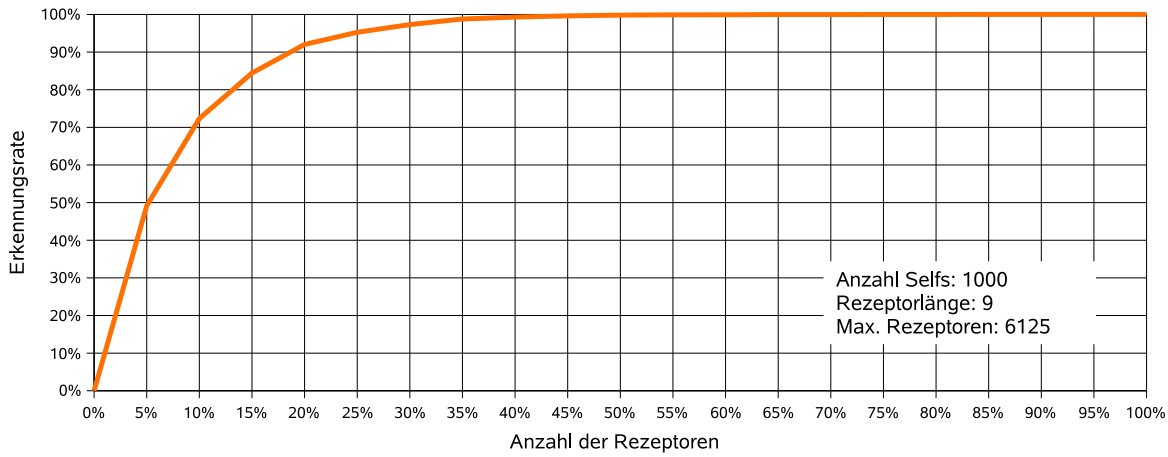


Abbildung 6.10: Zweites Testsystem bei Erhöhung der Rezeptoranzahl

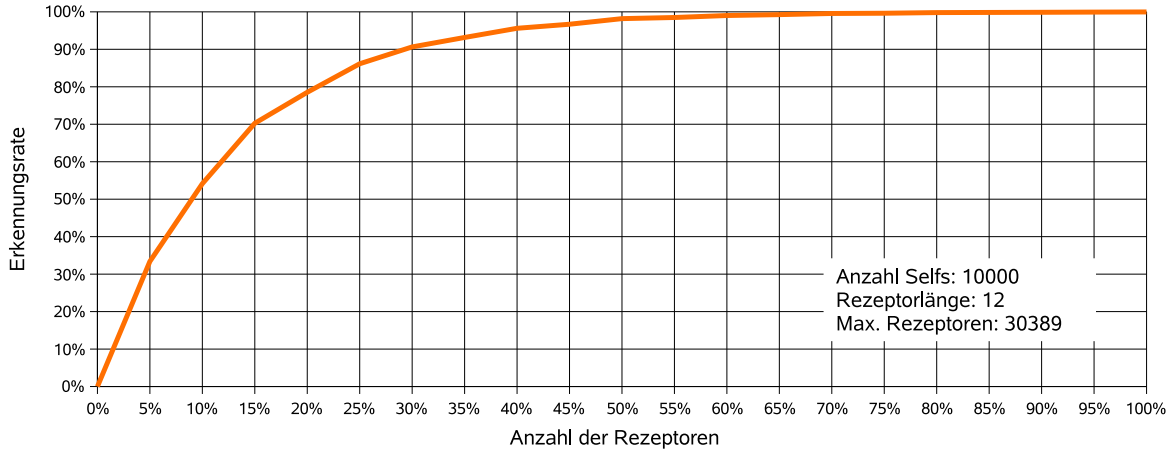


Abbildung 6.11: Drittes Testsystem bei Erhöhung der Rezeptoranzahl

Das interessante an allen drei bisher erwähnten Simulationen ist, dass die Kurven der Erkennungsrate ein identisches Krümmungsverhalten aufweisen. Es scheint so, als würde für alle Simulationsumgebungen ein bestimmter Schwellwert existieren, an dem die Trefferwahrscheinlichkeit – je nach Sichtweise – zu steigen oder zu sinken beginnt. Bei einer genaueren Analyse der Testergebnisse kann man jedoch einen Zusammenhang bezüglich der Anzahl der Self-Nachrichten und der Anzahl der verwendeten Rezeptoren erkennen.

Sobald die Anzahl der Rezeptoren ein gewisses Vielfaches der Anzahl der Self-Nachrichten unterschreitet, beginnt die Trefferwahrscheinlichkeit in der selben Art und Weise abzufallen. Die drei bisher aufgezeigten Testumgebungen lassen sich jedoch nicht so einfach vergleichen, da sie alle eine unterschiedliche Größe der Self-Sets, der Rezeptorlänge und der Anzahl der generierten Rezeptoren aufweisen. Aus diesem Grund wurde eine vierte Simulationsumgebung konzipiert, die sich mit der dritten Simulation

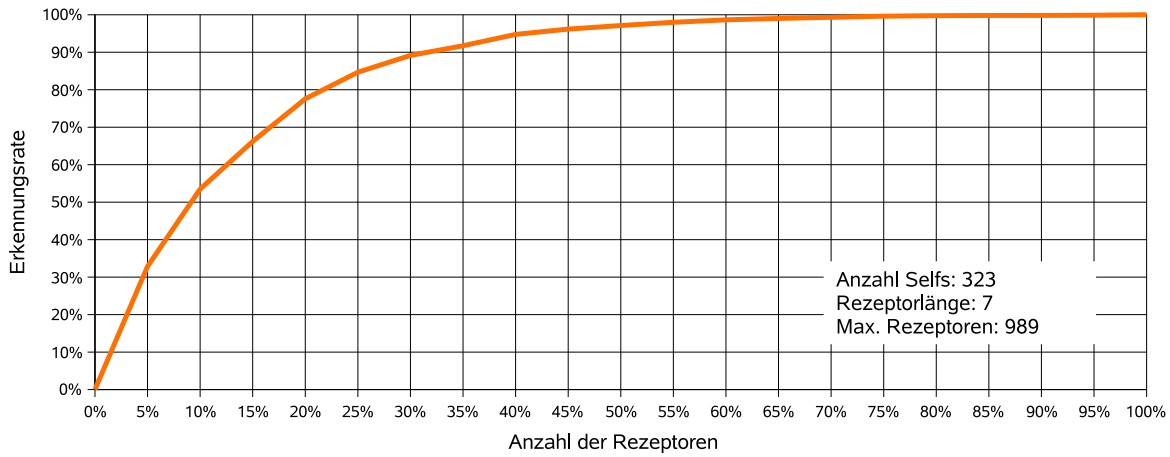


Abbildung 6.12: Viertes Testsystem bei Erhöhung der Rezeptoranzahl

gut vergleichen lässt und somit nähere Einblicke über den Zusammenhang liefert.

Bei der dritten Simulation konnten in etwa drei mal so viele Rezeptoren generiert werden, wie Self-Nachrichten im System existierten. Mit diesem Anhaltspunkt wurde das vierte Testsystem generiert, bei dem das selbe Verhältnis zwischen Self-Nachrichten und maximaler Anzahl an Antikörpern auftritt. Bei einem System von 323 Self-Nachrichten konnten in etwa auch drei mal so viele Rezeptoren (989 Antikörper) erzeugt werden. Bei einer Rezeptorlänge von sieben Bits ergab sich ein fast identischer Kurvenverlauf, wie in der dritten Simulationsumgebung (vgl. Abb. 6.11 und Abb. 6.12). Daraus wird ersichtlich, dass die Anzahl der Self-Nachrichten keinen großen Einfluss auf den Kurvenverlauf hat. Vielmehr ist das Verhältnis von Self-Nachrichten zu Rezeptoren ausschlaggebend für die Erkennungsrate.

Endergebnis der vier Testläufe

Alle vier Testläufe haben gezeigt, dass die Erkennungsrate nach einem bestimmten Schema ansteigt, wenn man die Rezeptoranzahl erhöht. Dieser stetig steigende Kurvenverlauf wird durch das Verhältnis von Self-Nachrichten zu der Anzahl der vorhandenen Rezeptoren bestimmt und ist unabhängig von der alleinigen Anzahl der Self-Nachrichten im System. Die Tests lassen darauf schließen, dass es bei jeder Konfiguration eine minimale Anzahl an Rezeptoren gibt, um eine gute Trefferwahrscheinlichkeit zu erreichen. Bei allen vier Simulationsumgebungen erreicht die Erkennungsrate einen Wert von 100%, sobald die Anzahl der Rezeptoren das drei- oder vierfache der Anzahl der Self-Nachrichten erreicht hat.

6.1.9 Speicherverbrauch der Rezeptoren

Es stellt sich zwangsläufig die Frage, ob nicht eine Speicherung der Self-Nachrichten an sich effizienter wäre, anstatt die vielen kurzen Rezeptoren zu erzeugen. Eine mathematische Gegenüberstellung der beiden Methoden gibt Aufschluss über den jeweiligen Speicherverbrauch. Das System besteht aus S unterschiedlichen Self-Nachrichten, die mit einem Hash-Verfahren auf eine einheitliche Länge von l Bits gebracht werden. Für die Abspeicherung der Self-Nachrichten werden somit $s \cdot l$ Bits benötigt. Bei der Abspeicherung der Rezeptoren muss zuerst festgelegt werden, wie viele Rezeptoren erzeugt werden sollen. Die Tests in Kapitel 6.1.8 haben gezeigt, dass eine ausreichend gute Trefferquote erzielt wird, wenn etwa viermal so viele Rezeptoren verwendet werden, wie Self-Nachrichten im System vorhanden sind. Ferner wurde in Kapitel 6.1.5 gezeigt, dass die optimale Rezeptorlänge bei s Self-Nachrichten bei ungefähr $log_2(s)$ Bits liegt. Ein Rezeptor besteht demnach aus dieser Länge und zusätzlichen $log_2(l)$ Bits, um den Offset des Rezeptors zu speichern. Der Speicheraufwand all dieser Rezeptoren berechnet sich daher mit $4 \cdot s \cdot (log_2(s) + log_2(l))$. Eine Gegenüberstellung der beiden Formeln zeigt, bei welcher Anzahl von Self-Nachrichten die beiden Varianten den selben Speicherplatz benötigen:

$$l \cdot s = 4 \cdot s \cdot (log_2(s) + log_2(l)) \tag{6.9}$$

$$\frac{l}{4} = log_2(s) + log_2(l) \tag{6.10}$$

$$2^{\frac{l}{4} - log_2(l)} = s \tag{6.11}$$

Die Speicherung der Rezeptoren benötigt somit weniger Speicherplatz, wenn im System weniger als $2^{\frac{l}{4} - log_2(l)}$ Self-Nachrichten vorhanden sind. Bei der Verwendung des MD5-Hash-Verfahrens werden Strings mit einer Länge von 128-Bit generiert. Mit

$$2^{\frac{128}{4} - log_2(128)} = s \tag{6.12}$$

$$2^{25} = s \tag{6.13}$$

ergibt sich somit erst ab 2^{25} Self-Nachrichten ein Mehrverbrauch an Speicherplatz. Es muss jedoch auch bedacht werden, dass es sich dabei um eine rein mathematische Betrachtung handelt. Bei so vielen unterschiedlichen Self-Nachrichten wird es eher schwer sein, überhaupt noch genügend Rezeptoren zu finden, deren Bitmuster nicht bereits in den Self-Nachrichten vorkommen. In diesem Fall muss die Hash-Länge der Self-Nachrichten nach oben korrigiert werden.

6.2 Optimierungen

Die in den vorigen Kapiteln beschriebenen Verfahren zur Erkennung von unbekannten Nachrichten haben zu sehr guten Ergebnissen bei der Erkennung von unbekannten Nachrichten geführt, obwohl das Wissen über das Self-Set nicht bekannt war. Jedoch ist der Speicheraufwand der Rezeptoren sehr groß. Ebenfalls nimmt es sehr viel Rechenzeit in Anspruch, um alle notwendigen Rezeptoren mit der Nachricht zu vergleichen. Aus diesem Grund werden im Folgenden einige Optimierungsverfahren vorgestellt, die sowohl den erforderlichen Speicherplatz für die Rezeptoren, als auch den notwendigen Zeitaufwand bei Vergleichen zwischen Rezeptoren und Nachricht minimieren. Ferner wird gezeigt, dass die Trefferwahrscheinlichkeit durch den zusätzlichen Einsatz von Prokörpern erhöht werden kann und dass der Einsatz von Permutationsmasken keinen Einfluss auf die Qualität der Erkennung hat.

6.2.1 Minimieren des Speicherplatzes

Im Vergleich zu den Nachrichten weisen Rezeptoren nur eine geringe Anzahl von abzuspeichernden Bits auf. Das Abspeichern einer großen Anzahl von Rezeptoren resultiert jedoch ebenso in viel Speicherplatz. Verschiedene Rezeptoren, die den selben Offset aufweisen, lassen sich aber an bestimmten Bitpositionen vereinen, um somit Speicherplatz zu sparen.

Zwei Rezeptoren mit einem identischen Rezeptor können trivialerweise komplett miteinander zu ein und dem selben Rezeptor verschmolzen werden. Falls sie sich an genau nur einer Bitposition unterscheiden, kann aus beiden ein neuer Rezeptor gebildet werden, der an dieser unterschiedlichen Bitposition ein Jokerzeichen und im Rest das identische Bitmuster aufweist. Bei einem Vergleich mit einer Nachricht kann der Wert an der Position des Jokerzeichens ignoriert werden, da sowohl die 1 als auch die 0 als Treffer gewertet werden. Analog können auch Rezeptoren, die bereits Jokerzeichen enthalten, zu neuen Rezeptoren verschmolzen werden. Dazu dürfen sich die beiden Rezeptoren ebenso nur an einer Bitposition unterscheiden, wobei an dieser einen Position an keinem der beiden Rezeptoren ein Jokerzeichen vorhanden sein darf. Dieses Verfahren kann beliebig oft wiederholt werden, um den Speicherbedarf zu minimieren. In Abbildung 6.13 ist beispielhaft eine Verschmelzung von vier unterschiedlichen Rezeptoren zu einem einzigen Rezeptor mit zwei Jokerzeichen abgebildet, worin die Jokerzeichen mit einem Minuszeichen symbolisiert sind.

In einem Testsystem mit der unterschiedlichen Anzahl von 1000 und 2000 Self-Nachrichten wurde die Effizienz dieses Verfahrens ermittelt (siehe Abbildung 6.14). Dabei hat sich nach mehreren Durchläufen herausgestellt, dass im Mittel etwa 30% der Rezeptoren eingespart werden konnten. Die Verschmelzung ergab bei den Testsystemen

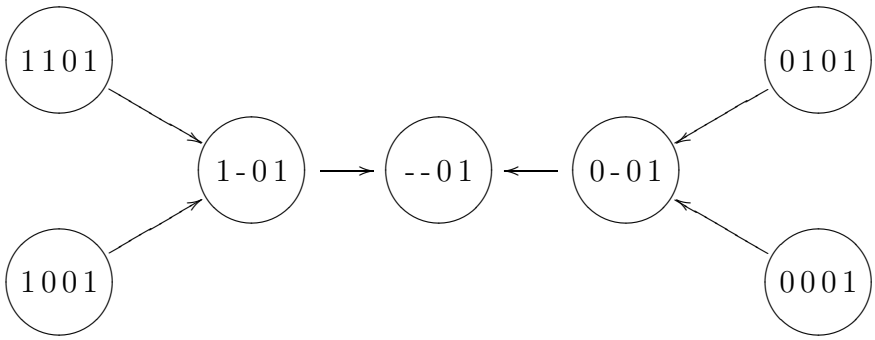


Abbildung 6.13: Beispiel einer Verschmelzung von vier Rezeptoren

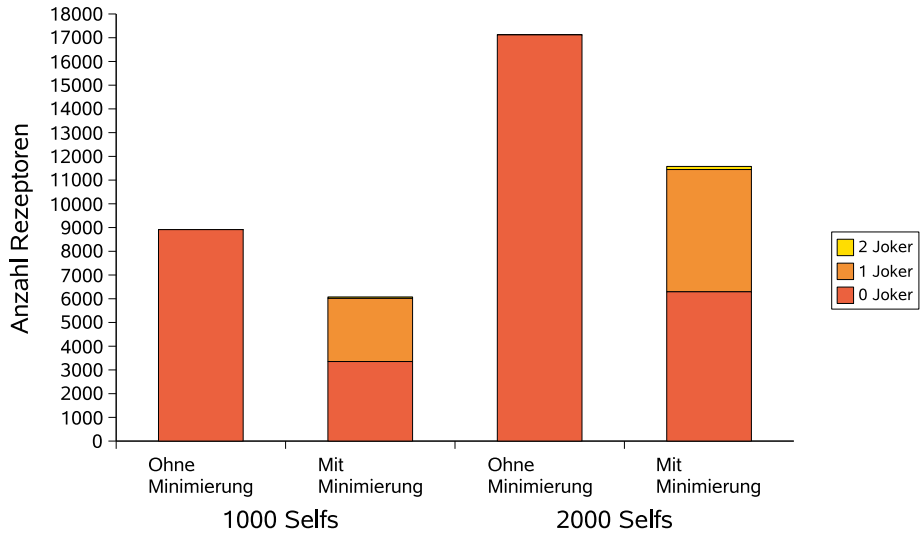


Abbildung 6.14: Optimierung des Speicherverbrauchs von Rezeptoren durch Verschmelzung mittels Jokerzeichen

jeweils nur wenige Rezeptoren mit zwei Jokerzeichen und keinen mit drei Jokerzeichen. In einem System mit mehr Self-Nachrichten und dadurch längeren Rezeptoren treten Kombinationen mit mehreren Jokerzeichen wahrscheinlichkeitsbedingt immer öfter auf.

6.2.2 Minimierung des Laufzeitverhaltens

In einem biologischen Immunsystem bewegen sich die Antikörper in einem dreidimensionalen Raum und zusätzlich werden die Vergleiche zwischen den verschiedenen Proteinmustern parallel und zeitgleich erledigt. Die Nachteile bei einem computerbasierten System sind einerseits der eindimensionale Arbeitsspeicher und andererseits die serielle Abarbeitung der einzelnen Vergleiche zwischen den Rezeptoren und der ankommenden Nachricht. In ersten Implementierungen wurden m verschiedene Rezeptoren von An-

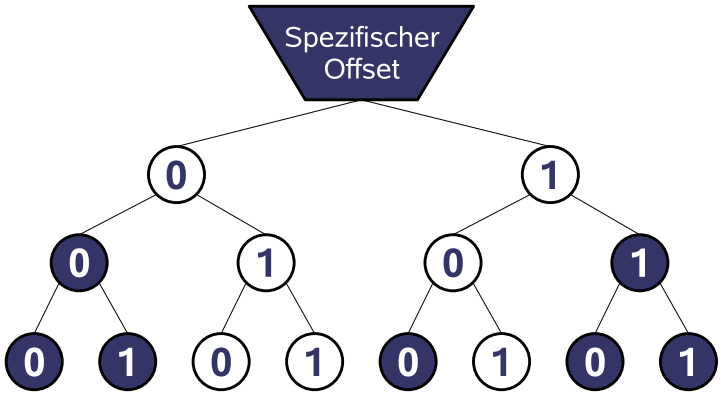


Abbildung 6.15: Beispielhafte Darstellung eines Binärbaums, der Rezeptoren an einem definierten Offset mit einer Länge von drei Bits aufnehmen kann. Die weiß eingefärbten Knoten repräsentieren hierbei „freigeschaltete Bitfolgen“. In diesem Baum sind somit die Rezeptoren 010, 011 und 101 enthalten.

tikörpern in einem Feld abgespeichert. Der Vergleich einer Nachricht mit allen Rezeptoren, der Länge r ergibt sich damit eine Zeitkomplexität von $O(m \cdot r)$. Dies stellt in einem System mit vielen Rezeptoren einen sehr großen Rechenaufwand dar.

Um dennoch ein akzeptables Laufzeitverhalten zu erreichen musste deshalb eine Methode gefunden werden, die diesen Zeitaufwand bei den Vergleichen minimiert. Da viele Rezeptoren an identischen Stellen gleiche Bitmuster aufweisen, wurden die Rezeptoren in einem binären Baum angeordnet (siehe Abbildung 6.15). Für alle unterschiedlichen Offsets im System, muss ein eigener vollständiger Binärbaum - ohne Wurzelknoten - mit Höhe r verwendet werden. Die Rezeptoren werden dann im Baum von oben entsprechend eingefügt. Dabei werden diejenigen Knoten im Baum „freigeschalten“, die den Bitmustern der Rezeptoren entsprechen.

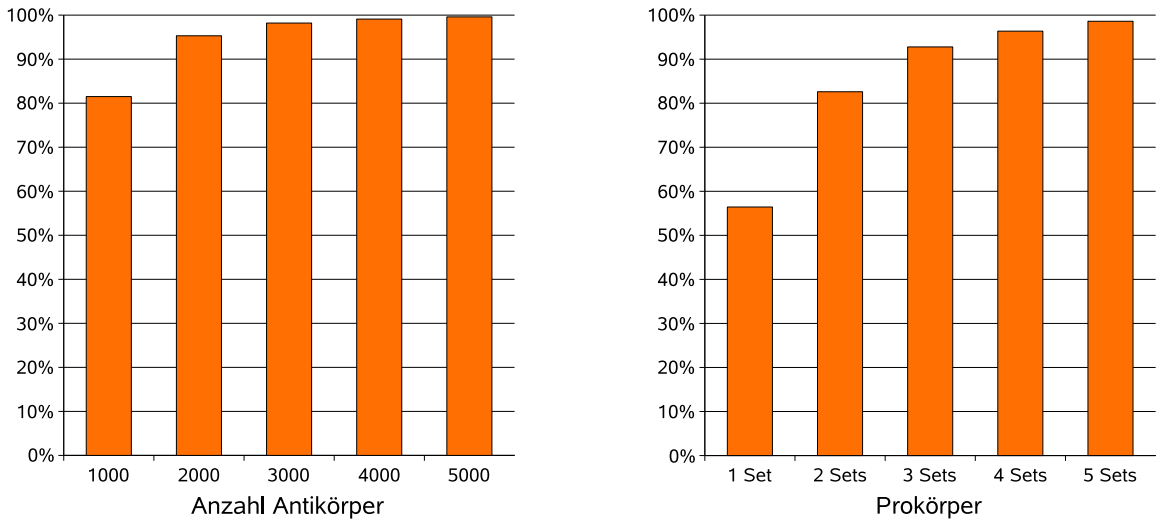
Ein Vergleich beginnt somit an der Wurzel des Baums und verfolgt die Knoten entsprechend dem zu vergleichenden Bitmuster der Nachricht. Falls ein kompletter Weg von der Wurzel zu einem Blatt des Baumes existiert, weist ein Rezeptor des Baums ein identisches Bitmuster auf.

Da jeder Binärbaum vollständig sein muss und jeder Konten einen binären Wert aufweist ob der Knoten freigeschaltet ist oder nicht, benötigt jeder Baum einen Speicherplatz von $2^{r+1} - 2$ Bits. Bei einem Vergleich einer Nachricht muss der Baum im schlimmsten Fall komplett von oben nach unten verglichen werden. Die Zeitkomplexität eines Vergleichs beträgt damit $O(p \cdot r)$ für alle möglichen Offsets. Da die Anzahl der Offsets p im Allgemeinen viel kleiner ist, als die Anzahl der produzierten Rezeptoren m, stellt dieses Verfahren eine Optimierung des Laufzeitverhaltens dar. Ferner stellen p und r in einem System Konstanten dar, was dadurch in einem mathematischen Laufzeitverhalten

von $O(1)$ resultiert. In den Testläufen hat sich durch den Einsatz der Binärbäume ein Geschwindigkeitsgewinn von 30% gezeigt.

6.2.3 Positive Selektion für die Steigerung der Erkennungsrate

Bis jetzt wurde nur die negative Selektion und die damit erzeugten Antikörper zur Erkennung von fremdartigen oder unbekannten Bitmustern herangezogen. Im Folgenden wird die Effektivität zusätzlich bei Verwendung der positiven Selektion näher beleuchtet. Das Ziel bei der Erkennung von unbekannten Nachrichten liegt nicht auf der 100%-igen Treffsicherheit, sondern bei einer akzeptablen hohen Trefferwahrscheinlichkeit bei geringstmöglichem Rechen- und Speicheraufwand. In Anbetracht dieser Faktoren wurden zwei Simulationsumgebungen aufgesetzt, in denen die Erkennungsrate von Antikörpern und Prokörpern getestet wurde. Beide Simulationsumgebungen bestanden aus 1000 zufällig generierten Self-Nachrichten. Es wurden mehrere tausend zufällig generierte Nachrichten erzeugt und getestet, ob sie vom System richtig als Self oder Non-Self eingestuft wurden.



(a) Ausschließliche Verwendung von Antikörpern

(b) Ausschließliche Verwendung von Prokörpern

Abbildung 6.16: Trefferwahrscheinlichkeiten in unterschiedlichen Testumgebungen bei Verwendung von Antikörpern oder Prokörpern

In Abbildung 6.16 sind die Testergebnisse als Balkendiagramm dargestellt. Das linke Diagramm zeigt darin die Trefferwahrscheinlichkeit bei ausschließlicher Verwendung von Antikörpern bei einer Anzahl von 1000 bis 5000 Rezeptoren. Obwohl nur neun Bits für die Rezeptoren der Antikörper verwendet wurden, ergab sich bereits bei Verwendung

von 2000 Antikörpern eine Trefferrate von 95,3%. Bei Erzeugung von 5000 Antikörpern wurden sogar 99,6% aller erzeugten unbekannten Nachrichten korrekt als solche erkannt. Der benötigte Speicherplatz lag dabei weit unter dem Speicherplatz, der benötigt worden wäre, wenn die Self-Nachrichten mit ihrer Länge von 128 Bits anstelle der kurzen Rezeptoren abgespeichert worden wären.

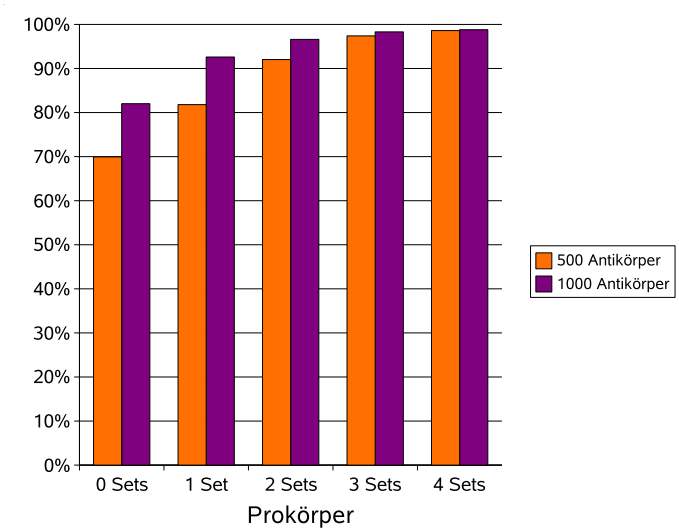


Abbildung 6.17: Trefferwahrscheinlichkeiten beim kombinierten Einsatz von Antikörpern und Prokörpern

Im rechten Diagramm in Abbildung 6.16 ist die Trefferwahrscheinlichkeit bei ausschließlicher Verwendung von Prokörpern zu sehen. In den Testumgebungen bestand ein Prokörper-Set aus durchschnittlich 250 Rezeptoren. Die Abweichungen ergeben sich durch die zufällige Wahl des Self-Sets und dem damit zusammenhängenden unterschiedlichen Nutzungsgrad der einzelnen Offsetgruppen. Die Erkennungsrate lag bei Verwendung von fünf Prokörper-Sets bei 98,6%, was etwa dem Ergebnis der Verwendung von 5000 Antikörpern entspricht.

Ein Nachteil der Verwendung von Prokörpern ist, dass diese immer nur in Gemeinschaft auftreten können. Antikörper können hingegen auf den Knoten in der Middleware einzeln verteilt werden und können auch einzeln zur Detektion herangezogen werden. Um dennoch den Vorteil der Prokörper zu ermitteln, wurde die Erkennungsrate bei gleichzeitiger Verwendung von Prokörpern und Antikörpern getestet. In Abbildung 6.17 ist deutlich zu sehen, dass die Erkennungsrate stetig ansteigt, wenn Antikörper und Prokörper gleichzeitig zum Einsatz kommen. Ein weiterer Vorteil daran war die geringere Anzahl an Rezeptoren, die insgesamt erzeugt werden mussten. Jedoch kann aus diesen Testergebnissen keine Aussage über den genauen Zusammenhang zwischen Pro- und Antikörpern gemacht werden, da jede Steigerung der Rezeptoranzahl trivialerweise in einer besseren Erkennungsrate resultiert. Aus diesem Grund wurde eine weitere Testumgebung aufge-

setzt, in der ein besserer Zusammenhang der unterschiedlichen Wirkungsgrade deutlich wird. Dazu wurden sowohl Antikörper als auch Prokörper-Sets erzeugt, jedoch immer maximal eine insgesamte Anzahl von 1250 Rezeptoren. Die Testgrundlagen waren die selben wie bei den vorigen Tests. Ein Prokörper-Set bestand somit auch hier wiederum aus etwa 250 Rezeptoren.

Anzahl der Antikörper	∼1250	∼1000	∼750	∼500	∼250	0
Anzahl von Prokörper-Sets	0	1	2	3	4	5
Erkennungsrate	87,03%	94,12%	93,90%	91,63%	84,26%	79,87%

Tabelle 6.1: Trefferwahrscheinlichkeit der Erkennung von unbekannten Nachrichten bei Verwendung von Antikörpern und Prokörpern bei einer insgesamten Anzahl von etwa 1250 Rezeptoren.

In der Ergebnistabelle 6.1 ist deutlich zu sehen, dass die reine Verwendung von Antikörpern eine bessere Erkennungsrate erzielt, wie wenn die ausschließliche Verwendung von Prokörpern zum Einsatz kommt. Dieser Sachverhalt kann damit erklärt werden, dass die Rezeptoren der Antikörper an allen möglichen Offsets erzeugt werden können und gleichmäßig über alle möglichen Offsets verteilt sind, wohingegen fünf Sets von Prokörpern nur an ausgewählten fünf unterschiedlichen Offsets generiert werden können. Die beste Trefferrate wurde bei dieser Simulationsumgebung allerdings erzielt, wenn ein Set von Prokörpern und etwa 1000 Antikörper erzeugt wurden. Eine Erhöhung der Anzahl von Antikörpern oder Prokörpern hat immer eine bessere Erkennungsrate zur Folge, allerdings hat dieser Test die unterschiedliche Mächtigkeit der beiden Verfahren gezeigt.

6.2.4 Anwendung von Permutationsmasken

Eine andere Art der Optimierung haben Steven A. Hofmeyr und Stephanie Forrest an der Universität von New Mexico untersucht [HF00]. Bei ihren Forschungsarbeiten wurden sie ebenso auf die grundlegende Problematik [SMT05] der negativen Selektion aufmerksam: Auch wenn alle möglichen Antikörper einer Rezeptorlänge erzeugt wurden und beim Erkennungsvorgang verfügbar sind, besteht immer die Möglichkeit, dass eine Non-Self-Nachricht von keinem Antikörper erkannt wird. In Beispiel 6.2.1 wird exemplarisch ein Szenario dargestellt, das diesen Sachverhalt verdeutlicht.

Beispiel 6.2.1
Ein System bestehe aus den drei Nachrichten `101`, `011` *und* `100`, *die zum Self-Set gehören. Daraus können maximal drei unterschiedliche Antikörper mit einer Rezeptorlänge von zwei Bits generiert werden. Zwei Antikörper mit Rezeptor* `00` *und* `11` *an Offset* `0` *und ein Antikörper mit Rezeptor* `10` *an Offset* `1`. *Nun lässt sich eine Non-Self-Nachricht*

generieren, die von keinem der beiden Antikörper erkannt werden kann. Diese Nachricht kann beispielsweise aus dem Bitmuster `001` *bestehen. Dieses Muster kann von den beiden generierten Antikörpern nicht erkannt werden, weil sowohl die ersten beiden Bits, als auch die letzten beiden Bits jeweils ein Teilbereich einer Self-Nachricht sind.*

Bei den Nachrichten, die von den Antikörpern nicht erkannt werden, handelt es sich um sogenannte *Löcher (engl. holes)* [HF00]. Um die Problematik der Löcher [JD05] in den Griff zu bekommen, kamen Steven A. Hofmeyr und Stephanie Forrest auf die Idee, sogenannte *Permutationsmasken* bei den Rezeptoren anzuwenden.

Beim Verfahren der Permutationsmasken werden die Nachrichten aus dem Self-Set nach einem bestimmten Muster verändert. Diese Veränderung kann beispielsweise eine Rotation oder ein Vertauschen von Bits oder bestimmten Teilbereichen der binären Nachricht sein. Dadurch ergibt sich eine komplett neue Repräsentation des Self-Sets, auf dessen Grundlage eigene Antikörper generiert werden. Das Beispiel 6.2.2 zeigt wie ein solches Verfahren dabei helfen kann, zuvor nicht zu erkennende Non-Self-Nachrichten trotzdem als solche zu identifizieren.

Beispiel 6.2.2
Es sei das selbe System wie in Beispiel 6.2.1 gegeben. In der linken Tabelle ist das Self-Set, die erzeugten Antikörper und ein nicht erkennbarer Non-Self zu sehen. In der rechten Tabelle ist die permutierte Repräsentation abgebildet, wobei in diesem Beispiel exemplarisch nur das letzte Bit an die vorderste Stelle rotiert wurde. Aus dem daraus resultierenden Self-Set wurden die entsprechenden Antikörper erzeugt.

System ohne Permutationsmaske	
Self-Set	1 0 1 0 1 1 1 0 0
Antikörper	0 0 1 1 1 0
Non-Self	0 0 1

System mit Permutationsmaske	
Self-Set	1 1 0 1 0 1 0 1 0
Antikörper	0 0 1 1 0 0
Non-Self	1 0 0

Die Non-Self-Nachricht mit dem Bitmuster `001` *muss nach dem selben Rotationsmechanismus verändert werden, bevor es von den Antikörpern im rechten System getestet wird. Im linken Original-System kann der Non-Self nicht erkannt werden, wohingegen das durch die Permutation veränderte System den Non-Self mit dem Antikörper* `00` *an den letzten beiden Bits als solchen entlarven kann.*

In der Tat kann das Permutationsverfahren mehr Non-Self-Nachrichten erkennen, als ein System, das ohne diese Erweiterung auskommt. Ein Nachteil dieses Verfahrens ist jedoch,

dass pro Permutationsverfahren in etwa die selbe Anzahl an Antikörpern hinzu kommt und somit mehr Speicherplatz benötigt wird. Es stellt sich zwangsläufig die Frage, ob ein System ohne Permutationmasken mit x Antikörpern eine schlechtere Erkennungsrate erzielt, als ein System mit Permutationsmasken, bei dem in zwei Repräsentationen jeweils $\frac{x}{2}$ Antikörper erzeugt werden.

In einem Testsystem wurde dieser Sachverhalt untersucht. Das System A bestand aus den Nachrichten, die lediglich mit dem MD5-Algorithmus gehasht und daraus die entsprechenden Antikörper erzeugt wurden. Im System B wurde das selbe Verfahren angewendet. Zusätzlich wurde jedoch eine Permutationsmaske verwendet, bei der die binären Nachrichten um die Hälfte ihrer Länge rotiert wurden. Anschließend wurden die durch diese Rotation veränderten Nachrichten ebenso mit dem MD5-Algorithmus gehasht. Das System B kann damit sowohl die Antikörper in der Originalrepräsentation, als auch die der permutierten Repräsentation erzeugen. Mehrere Testläufe mit unterschiedlichen Anzahlen an Self-Nachrichten und Antikörpern zeigten jedoch, dass sich durch die Anwendung der Permutationsmasken keine deutliche Verbesserung der Erkennungsrate ergibt. Die Ergebnisse sind in der Tabelle 6.2 zu sehen.

	System ohne Permutationsmaske		System mit Permutationsmaske	
	Anzahl Antikörper	Erkennungsrate	Anzahl Antikörper	Erkennungsrate
$s = 100, r = 6$	200	95,37%	100 + 100	95,35%
$s = 1000, r = 9$	800	78,20%	400 + 400	78,23%
$s = 10000, r = 12$	1000	20,87%	500 + 500	21,07%

Tabelle 6.2: Vergleich der Erkennungsrate von Systemen mit und ohne Permutationsmaske (s ist die Anzahl der Self-Nachrichten, r die Rezeptorlänge

6.3 Fazit

In diesem Kapitel wurde die Computer-Immunologie-Komponente der Architektur des Selbstschutzsystems in OCμ vorgestellt. Dabei wurde auf die Einflussfaktoren, wie die Länge der Nachrichten, den Aufbau von Rezeptoren und den Mechanismus der negativen und positiven Selektion eingegangen. Die Relevanz der einzelnen Faktoren wurde in unterschiedlichen Simulationsumgebungen getestet. Es stellte sich dabei heraus, dass die Rezeptorlänge in Abhängigkeit der Anzahl der Self-Nachrichten im System gewählt werden muss und die Rezeptoren gleichmäßig über die Nachrichtenlänge verteilt werden sollten, um eine bestmögliche Erkennungsrate zu erzielen. Des weiteren wurden sinnvolle Optimierungen im Laufzeitverhalten und bei der Speicheranforderung vorgestellt. Zwei unterschiedliche Methoden zur Steigerung der Erkennungsrate wurden ebenfalls simuliert. Die Verwendung von Prokörpern stellte sich unter bestimmten Umständen

als sinnvoll heraus, wogegen die Anwendung von Permutationsmasken im Gegensatz zu anderen Forschungsergebnissen zu keiner Verbesserung der Erkennungsrate führte.

7 Integration der Computer-Immunologie in OCμ

Bei OCμ handelt es sich um eine nachrichtenbasierte Middleware, deren Grundkomponenten aus Transport-Connector, Event-Dispatcher, Monitor-Queues, Diensten und Monitoren besteht. Es liegt daher nahe, auch den Selbstschutz der Middleware durch den ausschließlichen Einsatz dieser Komponenten zu realisieren. Dafür wurde das im Folgenden beschriebene Konzept entwickelt und in OCμ eingebaut. Der Aufbau von OCμ wurde bereits in Kapitel 2.5 beschrieben, weshalb auf die Grundkomponenten hier nicht mehr näher eingegangen wird. Vielmehr wird in diesem Kapitel der detaillierte Aufbau und das Zusammenspiel der einzelnen Komponenten des Schutzsystems beschrieben. Zuerst wird ein kurzer Überblick über die Schlüsselkomponenten gegeben und anschließend werden die Aufgaben dieser einzelnen Komponenten detailliert erklärt.

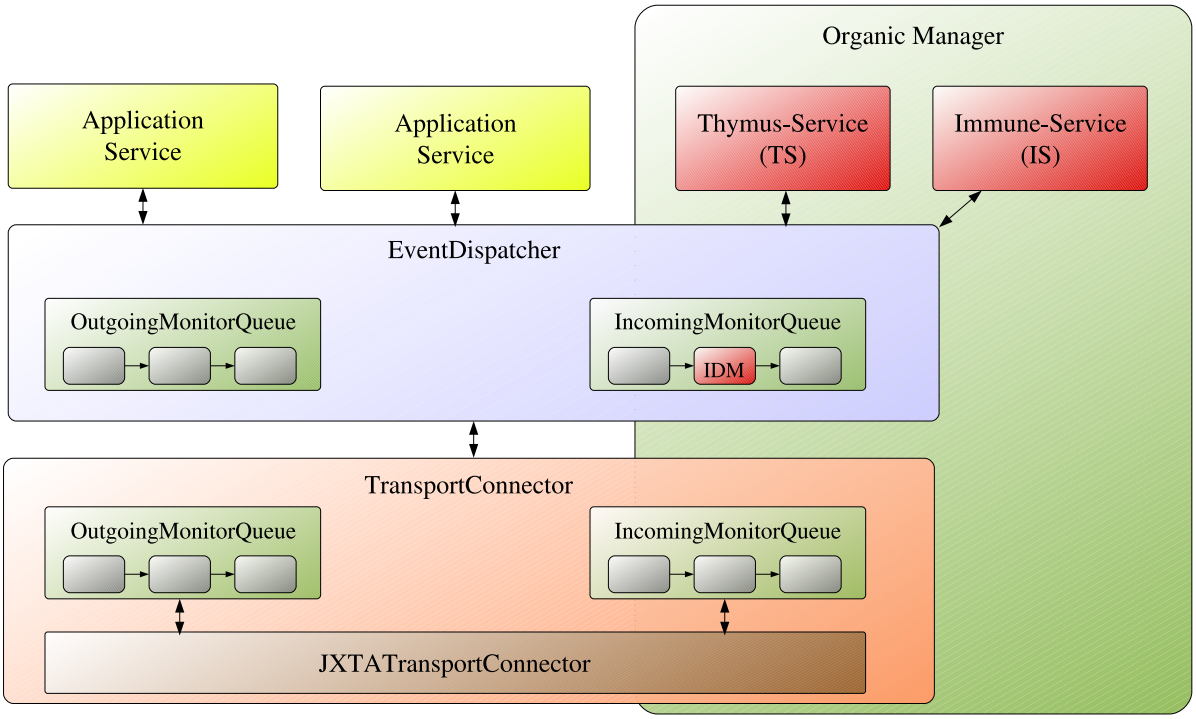


Abbildung 7.1: Integration der Computer-Immunologie in der Middleware OCμ

Die grundlegende Architektur des Selbstschutzsystems besteht aus drei Komponenten:

1. dem Thymus-Service (TS)
2. dem Immune-Service (IS)
3. dem Intrusion-Detection-Monitor (IDM)

In Abbildung 7.1 ist die Integration der drei Komponenten in OCμ schematisch veranschaulicht.

7.1 Der Thymus-Service

Der Thymus-Service ist für die Erzeugung und Verteilung der Rezeptoren verantwortlich. Da es sich bei diesen Tätigkeiten um rechenintensive Aufgaben handelt, sollte er nur auf leistungsstarken Knoten der Middleware gestartet werden. Seine Funktionen umfassen:

- Die Erzeugung des kompletten Self-Sets der Middleware
- Die Generierung der entsprechenden Rezeptoren
- Die Anpassung der Rezeptoren bei System- oder Konfigurationsänderungen
- Die Einteilung der registrierten Immune-Services in Gruppen
- Die Verteilung der Rezeptoren an die registrierten Gruppen
- Das Informieren der registrierten Gruppen und der bekannten Thymus-Knoten über eventuell erkannte Bedrohungen
- Das Beenden von eventuell bösartigen lokalen Diensten

Hinweis: Da auf einem Knoten nur maximal ein Thymus-Service gestartet ist, werden die Begriffe Thymus-Service und Thymus-Knoten in dieser Arbeit oft als Synonym verwendet.

7.2 Der Immune-Service

Der Immune-Service kümmert sich um die Analyse der eingehenden Nachrichten und entscheidet, ob eine Nachricht als Self oder Non-Self eingstuft wird. Der Dienst muss auf jedem Knoten der Middleware gestartet werden, um die Funktionalität des Selbstschutzes gewährleisten zu können. Seine Aufgaben sind:

- Die Registrierung bei einem Thymus-Service
- Das Senden von Informationen über lokale Ressourcen an den Thymus

- Das Empfangen der Gruppenzugehörigkeit vom Thymus-Service
- Das Empfangen der Rezeptoren
- Das Weiterleiten von Rezeptoren an andere Gruppenmitglieder
- Das Melden von eventuellen Bedrohungen an den Thymus-Service

Selbstverständlich arbeitet ein Knoten in OCμ auch problemlos ohne einen gestarteten Immune-Service. Aber für ein funktionierendes Schutzsystem ist dieser Dienst unabdingbar.

7.3 Der Intrusion-Detection-Monitor

Der Intrusion-Detection-Monitor befindet sich in der Incoming-Monitor-Queue des Event-Dispatchers und greift die Nachrichten ab, bevor der Knoten sie an die entsprechenden Dienste weiterleitet. Prinzipiell kann dieser Monitor auch in der Monitor-Queue der Transportschicht angesiedelt werden. Da die Transportschicht aber auf jedem Knoten je nach gewähltem Kommunikationsmedium anders implementiert sein kann, wurde der Monitor eine Ebene nach oben verschoben, um von der Kommunikationsinfrastruktur unabhängig zu sein. Der Monitor muss – ebenso wie der Immune-Service – auf jedem Knoten der Middleware gestartet werden, um die Funktionalität des Selbstschutzsystems gewährleisten zu können. Zu seinem Gebiet zählen die folgenden Aufgaben:

- Bevor eintreffende Nachrichten an die entsprechenden Dienste weitergeleitet werden, wird die Nachricht an den Immune-Service zur Überprüfung weitergeleitet
- Anschließende Weiterleitung einer gutartigen Nachricht an den ursprünglichen Empfänger, oder
- Das Verwerfen einer als Non-Self erkannten Nachricht (der Immune-Service übernimmt weitere Aufgaben)
- Das Verwerfen einer Nachricht eines als bösartig gemeldeten Dienstes oder Knotens

7.4 Verteilung der Thymus-Services auf bestimmte Knoten

Der Thymus-Service ist sowohl für das Sammeln der in der Middleware vorkommenden Self-Nachrichten, als auch für die Generierung und Verteilung der Rezeptoren zuständig. Das Sammeln aller Self-Nachrichten und das Erstellen der unterschiedlichen Rezeptoren ist ein rechenintensiver Vorgang, der nicht unbedingt von leistungsschwachen Knoten in der Middleware ausgeführt werden sollte.

Auch wenn der Thymus-Service prinzipiell auf jedem Knoten gestartet werden kann, würde dies keinen großen Vorteil oder Gewinn bringen, ihn auch tatsächlich auf allen Knoten laufen zu lassen. Da alle Thymus-Services letztendlich die selben Rezeptoren erzeugen, würden sie somit nur redundante Rechenarbeiten erledigen. Der einzige Vorteil bestünde lediglich darin, dass die Rezeptoren nicht an andere Knoten geschickt werden müssten, wenn jeder Knoten seinen eigenen Thymus-Service besäße. Somit würde allein der Netzwerkverkehr minimiert werden. OCμ ist jedoch auch für die Verwendung von mobilen Endgeräten angedacht, die derzeit lediglich eine Taktrate von wenigen hundert Megahertz aufweisen [WR05]. Einen so rechenaufwändigen Dienst auf diesen leistungsschwachen Knoten zu aktivieren, würde den Knoten sehr schnell an seine Grenzen stoßen lassen. Vielmehr macht es Sinn, diese Rechenarbeit den leistungsstarken Knoten im Netzwerk zu überlassen und nur die erstellten Rezeptoren an die leistungsschwachen Knoten zu übertragen.

Prinzipiell wäre es auch möglich, die Thymus-Services verteilt arbeiten zu lassen, so dass keine redundante Arbeit erledigt werden würde. Da die Erzeugung der Rezeptoren jedoch für normale Desktop-PCs in der Regel keine große Hürde darstellt, würde ein verteilter Algorithmus nur unnötig Netzwerkkapazität zur Synchronisation der verteilten Arbeiten in Anspruch nehmen, als dass dadurch tatsächlich ein Mehrwert an Geschwindigkeit entstehen würde.

Ein einzelner zentraler Thymus-Knoten im Netzwerk würde dabei jedoch die andere Extremsituation darstellen. Einerseits würde in diesem Fall zwar der Rechenaufwand auf einen einzigen Knoten reduziert werden, da nur ein einzelner Knoten diesen Aufwand zu bewältigen hätte. Andererseits würde dadurch aber bei großen Netzwerken das Nachrichtenaufkommen in den Kommunikationskanälen der Middleware zunehmen. Je nach Größe und Leistung des Netzwerkes kann dies enorme Einbußen für den restlichen Nachrichtenaustausch in der Middleware bedeuten.

In Tabelle 7.1 sind die Vor- und Nachteile der beiden Verteilungsstrategien gegenüber gestellt. Ein geringer durchschnittlicher Rechenaufwand resultiert in einem höheren Kommunikationsaufkommen und umgekehrt. Dies zeigt, dass ein Optimum nur nach eigener Gewichtung gefunden werden kann. Je nach Größe des Netzwerks, der jeweiligen Leitungsbandbreite und Leitungsauslastung sollten daher mehr oder weniger Knoten im Netzwerk einen Thymus-Service gestartet haben. Deren Aufgabe ist es dann, die notwendigen Rezeptoren zu generieren und anschließend an die restlichen Knoten im Netzwerk zu verteilen. Eine Angabe der optimalen Anzahl oder Verteilung vom Thymus-Services ist dabei jedoch nicht trivial, da zu viele Parameter wie beispielsweise die Netzwerktopologie, die Leitungsqualität oder das Vorhandensein von Netzwerk-Switches in das Modell mit einfließen. Eine gut gewählte Anzahl der Thymus-Knoten im Netzwerk minimiert aber gleichzeitig die gesamte Rechenauslastung der Middleware und den Kommunikationsaufwand zwischen den Knoten.

	wenige Thymus-Knoten	viele Thymus-Knoten
∅ **Rechenaufwand**	gering	hoch
Kommunikationsaufwand	hoch	gering

Tabelle 7.1: Vor- und Nachteile beim Einsatz von wenig oder vielen Thymus-Knoten in einer vernetzten OCμ-Umgebung

7.5 Zuordnung der übrigen Knoten

Da die Thymus-Services idealerweise nur auf dedizierten und rechenstarken Knoten der Middleware ausgeführt werden, müssen die dort erzeugten Rezeptoren auch anschließend auf die restlichen Knoten der Middleware übertragen werden. Dies geschieht durch spezielle Nachrichten, in denen die Rezeptoren versendet werden. Um Rezeptoren empfangen zu können, muss ein Empfängerknoten den Immune-Service gestartet haben, denn nur dieser Dienst stellt die Empfangsroutine bereit und kann die erhaltenen Rezeptoren entsprechend verwalten.

Eine triviale Lösung wäre eine Broadcast-Nachricht eines Thymus-Services an alle Immune-Services in der Middleware, um die relevanten Rezeptoren zu verteilen. Somit würden alle Rezeptoren früher oder später auf allen Knoten der Middleware eintreffen. Dieses Vorgehen bringt allerdings zwei Nachteile mit sich:

1. Die Verwendung von mehreren Thymus-Services im Netzwerk wäre hinfällig
2. Es existieren im Netzwerk möglicherweise Knoten, die nicht die erforderliche Kapazität[1] aufweisen, um alle versendeten Rezeptoren im internen Speicher ablegen zu können.

Um diese Nachteile zu umgehen, wird im Selbstschutzsystem von OCμ eine intelligentere Verteilungsstrategie verwendet, auf die im Folgenden genauer eingegangen wird.

7.5.1 Registrierung der Knoten bei einem Thymus-Knoten

Um die Problematik der Broadcast-Methode zu umgehen, müssen sich die Immune-Services bei *genau einem* Thymus-Knoten ihrer Wahl registrieren. In OCμ wählt jeder Immune-Service automatisch den Thymus-Knoten mit der geringsten Latenzzeit. Diesen Wert kann ein Immune-Service im Laufe des Betriebs automatisch vom `NetworkLatencyMonitor` erfragen, der auf jedem Knoten gestartet werden kann. Alle

[1] Gerade die Hersteller von mobilen Endgeräten bemessen den Speicher aus Kostengründen oft recht knapp, so dass dieser gerade für die Anwendungen des Benutzers ausreichend ist.

Knoten, die sich beim selben Thymus-Knoten registriert haben, werden als *Thymusgruppe* bezeichnet. Durch diese eindeutige Zuordnung von einem normalen Knoten zu einem Thymus-Knoten wird der Kommunikationaufwand in der Middleware gering gehalten, da die darauf gestarteten Thymus-Services hauptsächlich mit ihren registrierten Knoten kommunizieren. In Abbildung 7.2 wird eine solche Registrierung beispielhaft dargestellt. Die Thymus-Knoten sind darin mit einem „T“ gekennzeichnet und die Latenzzeiten sind an den Leitungen symbolisch als Zahlen angegeben. Je größer die Zahl, desto mehr Zeit wird für eine Nachrichtenübertragung zwischen den beiden Knoten benötigt. In diesem Beispiel bilden sich auf Grund der kürzeren Latenzzeiten automatisch zwei Gruppen, die durch die gestrichelten Linien gegeneinander abgegrenzt sind.

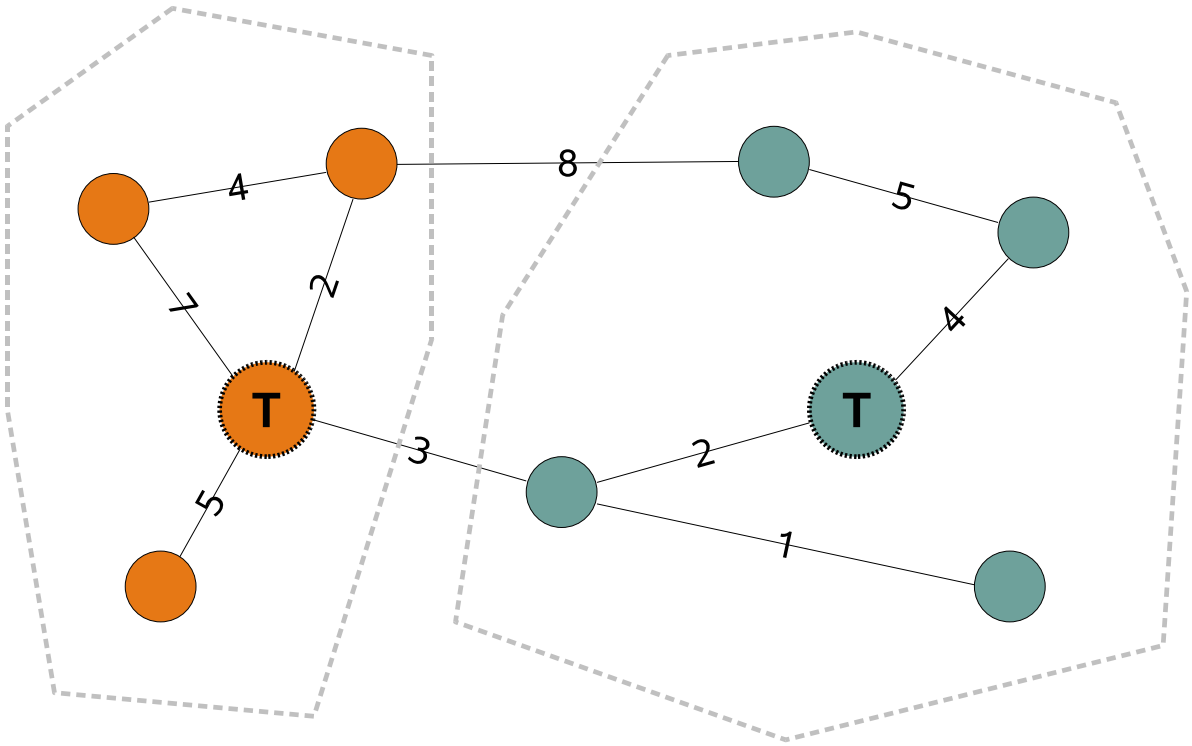


Abbildung 7.2: Beispielhafte Registrierung von Knoten zu verschiedenen Thymus-Knoten im Netzwerk

7.5.2 Gruppierung der Knoten innerhalb einer Thymusgruppe

Jeder Knoten, auf dem ein Immune-Service gestartet wurde, besitzt eine unterschiedliche Größe an verfügbarem Speicherplatz für die Rezeptoren und eine unterschiedliche Rechenleistung zum Prüfen von eingehenden Nachrichten. Es können in einer Thymusgruppe sowohl Knoten vorhanden sein, die über ausreichend Ressourcen verfügen und somit alle erforderlichen Rezeptoren bei sich abspeichern können. In einer solchen Thy-

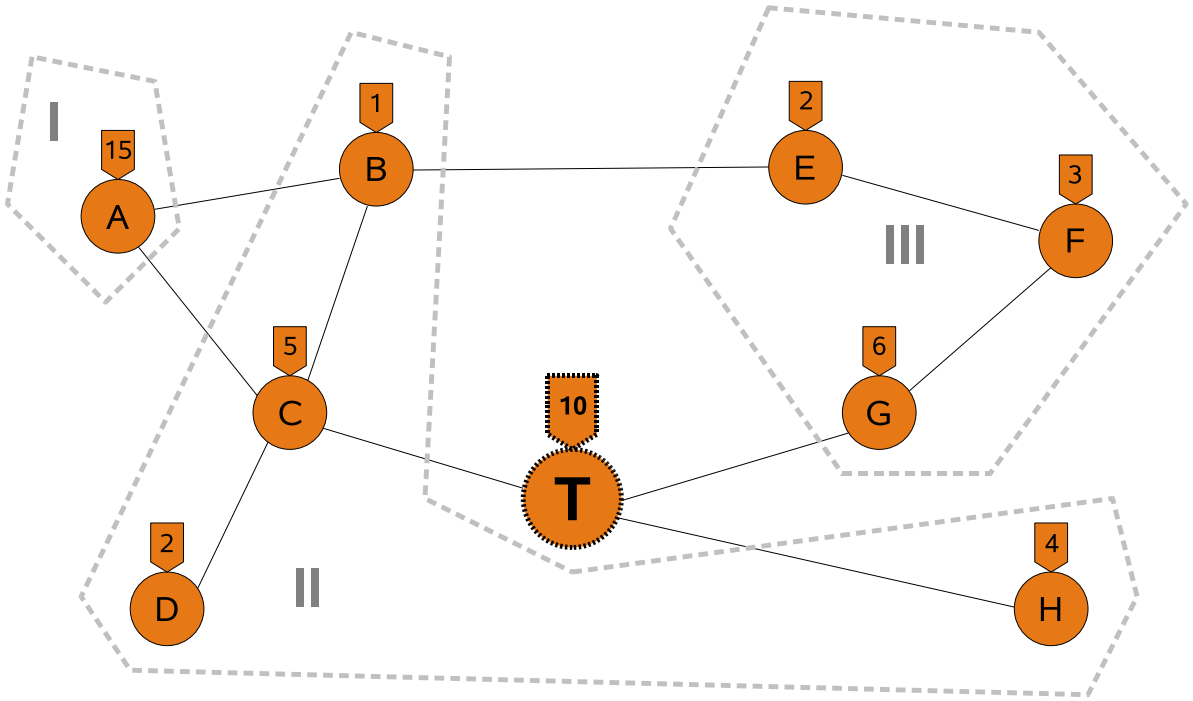


Abbildung 7.3: Beispiel einer Gruppierung von Knoten innerhalb einer Thymusgruppe. Es werden automatisch Knoten derart gruppiert, dass ihre summierte Kapazität mindestens so groß ist, wie die vom Thymus geforderte Kapazität von zehn Rezeptoren.

musgruppe können allerdings auch leistungsschwache Knoten – wie beispielsweise Sensoren – beheimatet sein. Für diese schwächeren Knoten, muss eine gesonderte Lösung zur Realisierung des Selbstschutzsystems gefunden werden. Dazu werden die Knoten einer Thymusgruppe wiederum in sogenannte *Rezeptorgruppen* unterteilt.

Um die Problematik besser zu veranschaulichen, dient die Thymusgruppe in Abbildung 7.3. Die jeweilige Speicherkapazität ist dabei mit einer Zahl an jedem Knoten gekennzeichnet. Diese Information wird bei der Registrierung von jedem Immune-Service automatisch an den jeweiligen Thymus-Service mitgesendet. Dabei besitzt der Thymus-Knoten in diesem Beispiel zehn verschiedene Rezeptoren, die er an seine registrierten Knoten übertragen muss. Leistungsstarke Knoten, die keine Schwierigkeit damit haben, alle ihnen zugesandten Rezeptoren im Speicher abzulegen und diese beim Nachrichteneingang zu vergleichen, können von einem Thymus-Service als eigenständige Knoten betrachtet werden. Dabei befindet sich der Knoten in einer Rezeptorgruppe, in der er das einzige Mitglied ist. Im Beispiel ist dies durch den einzelnen Knoten A mit einer Kapazität von 15 in der Gruppe I dargestellt.

Falls sich jedoch Knoten registrieren, die einen zu kleinen Speicher aufweisen, kann ihnen nicht das gesamte Rezeptor-Repertoire zugesendet werden. Im Beispiel sind dies die übrigen Knoten B bis H, da sie alle keine zehn Rezeptoren speichern können. Daher werden diese Knoten vom Thymus automatisch in Gruppen zusammengefasst, die die nötige Kapazität *zusammen* zur Verfügung stellen. Im Beispiel werden die Knoten B, C, D und H zur Rezeptorgruppe II mit einer Gesamtkapazität von 12 und die Knoten E, F und G zur Rezeptorgruppe III mit einer Gesamtkapazität von 11 zusammengefasst. Jeder Teilnehmer einer solchen Rezeptorgruppe erhält anschließend nur einen ganz bestimmten Anteil der vom Thymus-Service verteilten Rezeptoren. Diese Verteilung führt auf diesen leistungsschwachen Knoten selbstverständlich erstmal zu einer schlechteren Erkennungsrate, jedoch stellt sie die einzige Möglichkeit dar, diese Knoten überhaupt mit einem Schutzsystem auszustatten.

Allgemein gilt: Wenn sich k Knoten bei einem Thymus registrieren, müssen die Knoten so in Gruppen eingeteilt werden, dass ihre Speicherkapazität zusammengezählt mindestens alle erzeugten Rezeptoren aufnehmen können. Jede Gruppenkapazität sollte diesen Wert jedoch auch nicht unbedingt überschreiten, da sonst Speicherkapazität unnötig verschwendet würde. Diese Gruppierung ähnelt somit stark dem eindimensionalen Behälterproblem, welches ein NP-vollständiges Optmierungsproblem darstellt [LL85]. Der Algorithmus 2 verwendet eine First-Fit-Variante des Behälterproblems und gruppiert k Knoten derart, dass die summierte Kapazität in einer Gruppe mindestens den Wert c_{req} erreicht.

Der Algorithmus erstellt auch Gruppen, die lediglich aus einem einzelnen Knoten bestehen. Dies ist dann der Fall, wenn ein Knoten bereits alleine über eine ausreichende Kapazität verfügt. Der Thymus liefert seine erzeugten Rezeptoren also *immer* an Grup-

Algorithmus 2 Die Gruppierung von registrierten Knoten zu Rezeptorgruppen

```
1: collection nodes; // k registrierte Knoten
2: collection groups; // die gruppierten Knoten
3: collection tempgroup; // die temporäre Gruppe
4: while nodes.count() > 0 do
5:    while tempgroup.capacity() < c_req AND nodes.count() > 0 do
6:       // der Knoten mit der größten Kapazität
7:       // wird der aktuellen Gruppe hinzugefügt
8:       node = getMaxCapacityNode(nodes);
9:       tempgroup.append(node);
10:      // und aus der Knotenliste gestrichen
11:      nodes.remove(node);
12:   end while
13:   if tempgroup.capacity() >= c_req then
14:      // die temporäre Gruppe wird zu den relevanten Gruppen hinzugefügt
15:      groups.append(tempgroup);
16:      // und anschließend wieder geleert
17:      tempgroup.clear();
18:   end if
19: end while
20: if groups.count() > 0 then
21:   while tempgroup.count() > 0 do
22:      // nun müssen noch die übrigen Knoten der temporären Gruppe
23:      // auf die relevanten Gruppen verteilt werden
24:      group = getMinCapacityGroup(groups);
25:      group.append(tempgroup.firstElement();
26:      tempgroup.removeFirstElement();
27:   end while
28: else
29:   // es existiert nur eine einzige Rezeptorgruppe
30:   // mit einer zu geringen Gesamtkapazität
31:   groups.append(tempgroup);
32: end if
```

pen aus - auch wenn diese nur aus einem einzigen Knoten bestehen.

7.6 Verteilung der Rezeptoren

Wenn ein Thymus alle seine Rezeptoren erzeugt hat, müssen diese anschließend an seine registrierten und zu Rezeptorgruppen zusammengefassten Knoten ausgeliefert werden. Dabei müssen bei diesen Gruppen folgende drei Fälle unterschieden werden:

1. Die Gruppe besteht nur aus einem Knoten, da seine Kapazität c für die Aufnahme der benötigten Rezeptoren ausreicht.
2. Die Gruppe besteht aus n Knoten, deren jeweiligen Kapazitäten $c_1, c_2, ..c_n$ erst zusammen die erforderliche Gesamtkapazität von c_{req} ergeben.
3. Die Gruppe besteht aus n Knoten, deren jeweiligen Kapazitäten $c_1, c_2, ..c_n$ zusammen *nicht* die erforderliche Gesamtkapazität von c_{req} ergeben.

Der erste Fall ist trivial, denn der Thymus-Knoten muss lediglich alle generierten Rezeptoren an diesen Knoten senden. Dafür erzeugt der Thymus-Dienst eine neue Nachricht, in der die Rezeptoren als Nachrichtenelemente angehängt werden. Da OCμ derzeit auf dem Java-Framework JXTA aufsetzt, dürfen die Nachrichten eine maximal festgelegte Größe in Bytes nicht überschreiten. Der Thymus-Dienst teilt die Übertragung der Rezeptoren aus diesem Grund in mehrere kleinere Nachrichten auf, die jeweils maximal 200 Rezeptoren beinhalten.

Ein komplexeres Vorgehen muss dann angewendet werden, wenn eine Gruppe aus mehr als einem Knoten besteht. Das Problem dabei ist, dass die Knoten erst in ihrer Summe die benötigte Gesamtkapazität c_{req} zum Aufnehmen der Rezeptoren erbringen können. Da jeder Knoten i in der Gruppe nur über seine spezifische Kapazität c_i verfügt und dadurch nur eine begrenzte Anzahl von Rezeptoren aufnehmen kann, können die Rezeptoren in diesem Fall nicht alle einfach an die Knoten gesendet werden. Es muss hierfür ein anderes Verfahren gewählt werden.

Bereits zu dem Zeitpunkt, an dem sich ein Knoten bei einem Thymus-Dienst registriert, wird auch seine spezifische Kapazität c_i mitgeliefert. Anhand dieser Information findet auch die Einteilung in Gruppen beim Thymus statt (siehe Kapitel 7.5.2). Durch diese Information besitzt der Thymus-Knoten zusätzlich das genaue Wissen darüber, welche Kapazitäten in jeder Gruppe vorhanden sind und kann dadurch berechnen, wieviele Rezeptoren er an einen Knoten maximal senden darf. Vor der Auslieferung der Rezeptoren an eine Gruppe mit n Knoten, teilt der Thymus-Knoten hierfür sein gesamtes Repertoire in kleinere Rezeptorpakete auf, deren Größe sich proportional zur Knotenkapazität verhält. Die Größe eines solchen Rezeptorpakets $\sharp r_i$ für den Knoten i ergibt sich aus folgender Formel:

$$\sharp r_i = \lfloor \frac{c_i \cdot \sharp r_{all}}{\sharp c_{req}} \rfloor \tag{7.1}$$

Um sicher zu stellen, dass die Kapazität eines Knotens auf keinen Fall überschritten wird, wird der Bruch nach unten auf eine Ganzzahl abgerundet. Jedes erzeugte Rezeptorpaket beinhaltet andere Rezeptoren, damit alle generierten Rezeptoren aus dem gesamten Repertoire auf mindestens einem Knoten Verwendung finden.

Der dritte Fall ist als Sonderfall zu betrachten. In der Regel sollte er nicht auftreten, da eine Rezeptorgruppe mit zu geringer Gesamtkapazität mit dem gezeigten Algorithmus am Schluss auf die bereits erzeugten Rezeptorgruppen aufgeteilt wird. Somit erhöht sich die Gesamtkapazität der einzelnen Gruppen, aber anschließend weisen alle Rezeptorgruppen eine ausreichende Kapazität auf. Nur wenn in einer Thymusgruppe die Kapazitäten *aller registrieren* Knoten zusammen geringer ist, als die erforderliche Kapazität, bildet der Algorithmus eine Rezeptorgruppe mit unzureichender Kapazität. In diesem Fall bleibt dem Thymus-Knoten nichts anderes übrig, als nur den entsprechenden Bruchteil der erzeugten Rezeptoren an seine Mitglieder zu übertragen.

7.6.1 Die Problematik der schwachen Rezeptorgruppen

Eine Rezeptorgruppe, die nur aus einem einzigen Knoten besteht, der dazu noch über ausreichend Kapazität verfügt, um alle erforderlichen Rezeptoren bei sich abzuspeichern, bereitet in der Regel keine Schwierigkeiten. Der Knoten erhält alle Rezeptoren und verfügt somit über einen optimalen Schutz gegenüber ungewünschten Nachrichten.

Die Zusammenfassung von schwächeren Knoten zu einer Rezeptorgruppe stellt das System jedoch erstmal vor ein Problem: Die Knoten können nur einen Bruchteil der Rezeptoren aufnehmen und im schlimmsten Fall können sogar alle Knoten zusammen nicht einmal alle Rezeptoren abspeichern. Das Schutzsystem kann jedoch nur dann zuverlässig arbeiten, wenn tatsächlich alle Rezeptoren im Einsatz sind. Es existieren zwei alternative (und auch kombinierbare Ansätze), um die geschilderte Problematik zu umgehen:

- Das Rotieren von Rezeptoren innerhalb einer Rezeptorgruppe
- Das Rotieren von eintreffenden Nachrichten in der Rezeptorgruppe

Im Folgenden werden beide Varianten auf ihre Vor- und Nachteile hin untersucht.

7.6.1.1 Rotation der Rezeptoren innerhalb einer Rezeptorgruppe

Beim menschlichen Immunsystem befinden sich auch nicht immer *alle* möglichen Ausprägungen von Antikörpern zur selben Zeit im Organismus. Ein regelmäßiger Austausch

von Antikörpern bietet jedoch in der Biologie erfahrungsgemäß einen ausreichenden Schutz gegenüber potenziellen Angreifern oder Eindringlingen.

Ein ähnliches Verfahren findet sich auch in OCμ bei den vom Thymus-Service erzeugten Gruppen wieder. Jede Gruppe bekommt vom Thymus-Service eine eindeutige ID zugewiesen. Jeder der darin enthaltenen Knoten erfährt nach abgeschlossener Gruppierung von seinem Thymus-Service die entsprechende Gruppen-ID und alle zu dieser Gruppe gehörenden Knoten-IDs.

Durch die bereits beschriebene Verteilung der Rezeptoren befinden sich innerhalb einer Gruppe in Summe immer alle vom Thymus-Service versendeten Rezeptoren. Ein Austausch von x Rezeptoren innerhalb einer Gruppe von n Knoten geschieht anhand der Reihenfolge, in der der Thymus die Knoten $k_1, k_2, ...k_n$ gruppiert hat (siehe Abb. 7.4):

Knoten k_1 sendet x Rezeptoren an seinen Nachbarknoten k_2 und löscht diese bei sich. Knoten k_2 sendet wiederum seine ältesten x Rezeptoren an Knoten k_3 und ersetzt diese durch die eingetroffenen x Rezeptoren von Knoten k_1. Nach diesem Schema verfährt jeder Knoten reihum, wobei der Knoten k_n seine Rezeptoren wieder an den Knoten k_1 sendet. Dieser Austausch stellt somit einen Zyklus dar, in dem jeder Knoten in bestimmten Abständen alle möglichen Rezeptoren besitzt.

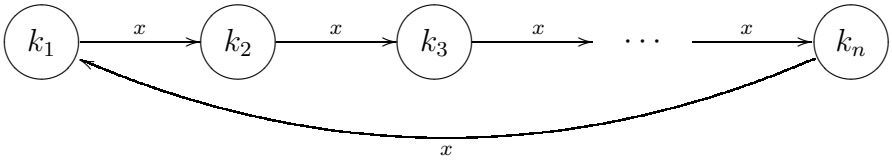


Abbildung 7.4: Rotation von x Rezeptoren innerhalb einer Gruppe von n Knoten

Eine Gruppe von Knoten besteht jedoch nicht immer aus Knoten mit identischer Kapazität. Vielmehr weisen in der Regel alle Knoten eine unterschiedliche Kapazität auf. Ein Problem ergibt sich beispielsweise dann, wenn ein Knoten k_1 mit Kapazität $c_1 = 300$ sein gesamtes Rezeptor-Set an einen Knoten k_2 mit Kapazität $c_2 = 250$ senden würde. Der Knoten k_2 müsste somit 50 Rezeptoren verwerfen, da er diese nicht abspeichern kann. Somit würden diese Rezeptoren aus der Gruppe für immer verschwinden. Aus diesem Grund müssen die Mitglieder innerhalb einer Gruppe auf den Knoten mit der geringsten Kapazität c_{min} Rücksicht nehmen.

Um diesen Problem zu lösen, teilt der Thymus-Service zum Zeitpunkt der Gruppierung allen Gruppenmitgliedern die kleinste in dieser Gruppe vorhandene Kapazität mit. Ein Austausch von Rezeptoren wird in einer Gruppe dann ausschließlich in Einheiten mit dieser Kapazität durchgeführt. Dabei können keine Rezeptoren beim Austausch verloren gehen.

Durch die Rotation der Rezeptoren zwischen den Knoten steigert sich die Wahrschein-

lichkeit *nicht*, dass ein Knoten unbekannte und möglicherweise bösartige Nachrichten auch als solche identifizieren kann. Da immer nur ein Bruchteil, der verfügbaren Rezeptoren vorrätig ist, kann die Trefferquote auch einen bestimmten Wert nicht überschreiten. In Kapitel 6.1.8 wurde bereits analysiert, wie gut die Trefferwahrscheinlichkeit beim Einsatz von weniger Rezeptoren ist. Die Abbildungen 6.9 bis 6.12 der Testsimulationen verdeutlichen dies. Erst wenn drei- oder viermal so viele Rezeptoren, wie Self-Nachrichten an einem Knoten verfügbar sind, erreicht die Trefferquote annähernd 100%. Die Rotation von Rezeptoren zwischen den Knoten schützt diese immerhin zu unterschiedlichen Zeiten jeweils gegen andere Angriffe.

Nachteile:

Auch wenn die Rezeptoren in regelmäßigen Abständen ausgetauscht werden, befindet sich nie das komplette Rezeptor-Repertoire auf einem Knoten.

Das Austauschen der Rezeptoren erfordert zusätzlichen Kommunikationsaufwand innerhalb einer Rezeptorgruppe.

Vorteile:

Da auf allen Knoten unterschiedliche Rezeptoren vorhanden sind, können nie alle Knoten innerhalb einer Rezeptorgruppe für die selbe bösartige Nachricht anfällig sein.

Sobald ein Knoten in der Rezeptorgruppe eine Nachricht als bösartig einstuft, kann er dies den anderen Knoten miteilen.

Wenn die Gesamtkapazität aller beim Thymus-Knoten registrierten Knoten nicht die erforderliche Kapazität erreicht, stellt dieses Verfahren die einzige Möglichkeit dar, die Knoten wenigstens mit einem wechselnden Schutzsystem auszustatten.

7.6.1.2 Rotation der Nachrichten innerhalb einer Rezeptorgruppe

Wenn in einer vom Thymus-Knoten gebildeten Rezeptorgruppe eine ausreichende Gesamtkapazität zur Verfügung steht, dann beinhaltet die Gruppe nach der Verteilung auch alle erzeugten Rezeptoren. Dabei sind die Rezeptoren zwar nicht gleichzeitig auf einem Knoten, aber immerhin auf allen Knoten verteilt. Dieser Sachverhalt kann ausgenutzt werden, um diese Knoten dennoch mit dem kompletten Schutz auszustatten, den auch leistungsstarke Knoten erfahren.

Falls eine Nachricht bei einem Knoten in einer Rezeptorgruppe eintrifft, wird zuerst das lokal verfügbare Rezeptor-Repertoire mit der Nachricht verglichen. Falls dabei keine Entscheidung getroffen werden konnte, wird die Nachricht an den nächsten Knoten der Rezeptorgruppe übergeben, der daraufhin sein lokales Rezeptor-Repertoire mit der

Nachricht abgleicht. Dieser Vorgang wird so lange in der Rezeptorgruppe der Reihe nach fortgesetzt, bis entweder ein Knoten die Nachricht als bösartig einstuft oder die Nachricht wieder beim ursprünglichen Knoten eintrifft, dadurch automatisch als gutartig eingestuft und wie vorgesehen verarbeitet wird. Somit kann eine Rezeptorgruppe als Einheit betrachtet in den selben Genuss des Selbstschutzsystems kommen, wie leistungsstarke Knoten.

Nachteil:

Das Herumreichen einer eintreffenden Nachricht sorgt für eine Verzögerung, da die Nachricht erst jeden Knoten einmal passieren muss, bevor sie verarbeitet werden kann.

Vorteil:

Der leider einzige aber durchaus erhebliche Vorteil dieser Methode liegt darin, dass sich ressourcenschwächere Knoten in einer Gruppe gegenseitig beim Nachrichtenvergleich helfen und dadurch letztendlich den selben Schutz erfahren, wie leistungsstarke Knoten, die das gesamte Rezeptor-Repertoire aufnehmen können.

7.6.1.3 Optimierung der Gruppenbildung

Ein Nachteil beider Ansätze stellt die – unter Umständen nicht unerheblichen – Verzögerungen beim Nachrichtenaustausch dar. Beim einen Ansatz sind es die in regelmäßigen Abständen auszutauschenden Rezeptoren, wogegen beim anderen Ansatz die Nachrichten der Reihe nach innerhalb der Rezeptorgruppe rotiert werden müssen. Um diesen negativen Effekt der Verzögerung zu minimieren muss der Algorithmus 2 modifiziert werden, um Laufzeitaspekte darin abbilden zu können. In seiner ursprünglichen Version gruppiert der Algorithmus die Knoten nur anhand ihrer Kapazität. Das Ziel ist dabei lediglich, Gruppen zu erzeugen, die eine Gesamtkapazität oberhalb der erforderlichen Kapazität aufweisen – egal wie „weit“ die Knoten voneinander entfernt sind. Nun muss der Algorithmus die Knoten noch nach einem anderen Aspekt gruppieren: Die Knoten innerhalb einer Rezeptorgruppe sollen zusätzlich zur Mindestkapazität auch noch eine möglichst geringe Latenzzeit beim Nachrichtenaustausch aufweisen.

Dazu müssen die beiden verwendeten Methoden

- `getMaxCapacityNode(collection nodes)`
  ermittelt den Knoten mit der größten Kapazität
- `getMinCapacityGroup(collection groups)`
  ermittelt die Knotengruppe mit der geringsten Gesamtkapazität

im Algorithmus durch die zwei neuen Methoden

- `getNearestNode(collection nodes, group)`
  ermittelt den Knoten, der die geringste Latenzzeit zur angegebenen Gruppe aufweist
- `getNearestGroup(node, collection groups)`
  ermittelt die Knotengruppe, der die geringste Latenzzeit zum angegebenen Knoten aufweist

ausgetauscht werden. Die beiden neuen Methoden berücksichtigen die Latenzzeiten zwischen den Knoten und ermitteln so bei jeder Iterationsstufe die jeweils besten Kandidaten für eine Gruppenbildung.

Für die Latenzzeit zwischen zwei Knoten wird die mittlere Laufzeit einer Nachricht zwischen diesen beiden Knoten herangezogen. Die Latenzzeit innerhalb einer Knotengruppe ist dabei nicht so einfach zu ermitteln. Dieser Wert hängt in der Realität davon ab, wieviel die Knoten innerhalb einer Gruppe miteinander kommunizieren und vor allem, *welche* Knoten mit *welchen* Knoten in regem Kontakt stehen. Da dieses Wissen jedoch nicht im Voraus bekannt ist, wird eine Gleichverteilung der Kommunikation innerhalb einer Rezeptorgruppe angenommen. Die Latenzzeit innerhalb einer Gruppe entspricht dann dem Mittelwert aller Latenzzeiten zwischen allen Kommunikationspartnern in dieser Gruppe. Die Latenzzeit zwischen einem Knoten und einer Gruppe wird als Mittelwert der Latenzzeiten zwischen diesem Knoten und allen einzelnen Gruppenmitgliedern angesehen.

Es existiert jedoch ein genereller Nachteil beim Gruppieren der Knoten nach kurzen Latenzzeiten. Gerade nach dem Start der Middleware besitzt das System noch keine Informationen über die Latenzzeiten im Netzwerk. Diese Werte kann das System nur zur Laufzeit ermitteln. Erst wenn zwei Knoten das erste mal miteinander kommunizieren, kann eine Aussage über die Latenzzeit zwischen diesen beiden Knoten getroffen werden.

In OCμ existiert kein globales Wissen über die Entfernung der verschiedenen Knoten oder die Latenzzeiten beim Übertragen von Nachrichten zwischen den Knoten. Der `NetworkLatencyMonitor` verfügt jedoch über die Möglichkeit, diese Daten zur Laufzeit mitzuschneiden. Allerdings muss dafür auf jedem Knoten der Monitor zum Messen der Laufzeitverzögerungen aktiviert sein. Da die Gruppierung innerhalb einer Thymusgruppe vom jeweiligen Thymus-Knoten durchgeführt wird, muss dieser natürlich auch über das Wissen der Latenzzeiten *zwischen* seinen registrierten Knoten verfügen. Da sein lokal gestarteter `NetworkLatencyMonitor` jedoch auch nur die Latenzzeiten zu seinen Kommunikationspartnern kennt, müssen die Knoten ihr Wissen dem Thymus-Knoten mitteilen können. Dazu wurde ein neuer Nachrichtentyp eingeführt, mit dem der Thymus-Knoten von einem registrierten Knoten die Latenzzeiten den anderen registrierten Knoten in Erfahrung bringen kann. Erst wenn der Thymus-Knoten über das

Wissen über alle diese Latenzzeiten verfügt, kann eine sinnvolle Gruppierung mit dem modifizieren Algorithmus stattfinden.

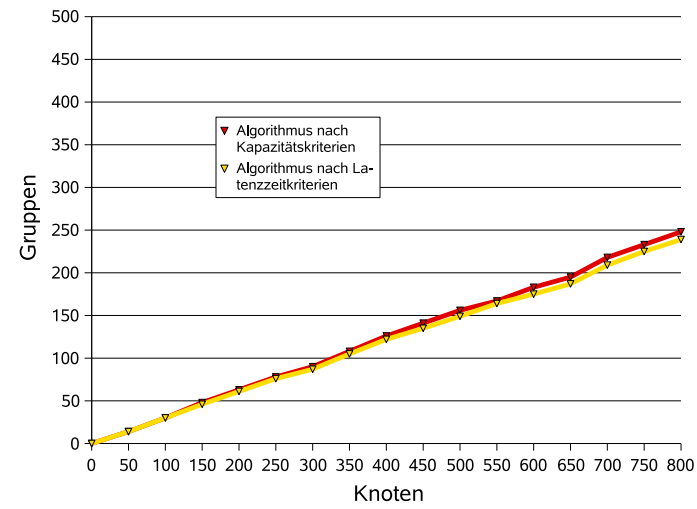
Um die Effizienz der Gruppenbildung und die Verbesserung durch Einbeziehung der Latenzzeiten zu erfassen, wurde ein Simulator entwickelt, der die Algorithmen bei verschiedenen Szenarien durchgeführt hat[2].

Es wurden mehrere Testumgebungen generiert, bei denen jeweils eine unterschiedliche Anzahl von Knoten erzeugt wurde. Diese Knoten sind alle automatisch dem selben – und in der Testumgebung auch einzigem – Thymus-Knoten zugewiesen. Dieser Thymus-Knoten besitzt eine bestimmte Anzahl an Rezeptoren, die er an seine registrierten Knoten verteilen muss. Die zu verteilende Menge wird als 100% festgelegt. Jedem der generierten Knoten wurde eine zufällige Kapazität zwischen 10% und 110% zugewiesen. Somit können im Mittel 10% der Knoten alle Rezeptoren alleine abspeichern, wohingegen die restlichen Knoten zu Rezeptorgruppen zusammengeschlossen werden müssen. Des Weiteren wurden die Latenzzeiten zwischen den Knoten als zufällige Ganzzahlwerte zwischen 1 und 100 generiert. Diese Zahl dient als rein abstrakter Wert, kann dadurch aber auch gut verglichen werden.

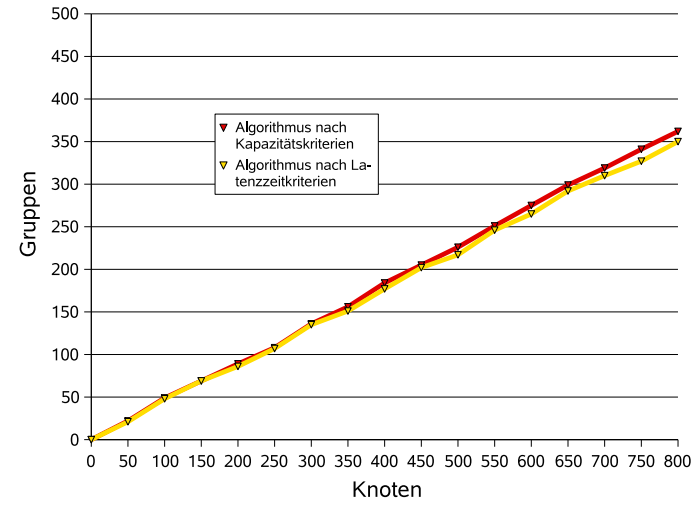
In unterschiedlichen Testläufen wurden der Reihe nach bis zu 800 Knoten generiert und die beiden Algorithmen auf die Knoten angewendet. In Abbildung 7.5 sind die Ergebnisse der Simulationen grafisch dargestellt. In der ersten Simulation (a) wurden nur Knoten generiert, die eine Kapazität zwischen 10% und 60% der erforderlichen Gesamtkapazität aufweisen. In der zweiten Simulation (b) waren die Kapazitäten im Bereich zwischen 10% und 110% angesiedelt. Die dritte Simulation (c) bestand aus Knoten, denen eine Kapazität von 60% bis 100% zugewiesen wurde. Allen drei Grafiken ist eine stetige und nahezu lineare Steigung der Kurven zu entnehmen. Daraus lässt sich ablesen, dass beide Algorithmen stets Gruppen mit einer durchschnittlich konstanten Anzahl von Knoten generieren. Die rote Linie repräsentiert dabei den Algorithmus, der nur die reinen Kapazitäten der Knoten berücksichtigt, wohingegen die gelbe Linie die Ergebnisse des auf die Latenzzeit optimierten Algorithmus widerspiegelt. Bei wenigen Knoten erzeugen beide Algorithmen nahezu die selbe Anzahl von Rezeptorgruppen. Aber bei steigender Knotenanzahl generiert der Algorithmus, der auch die Latenzzeiten der Knoten berücksichtigt, etwas weniger Gruppen, als sein Vorgänger. Selbstverständlich befinden sich dann im Durchschnitt auch mehrere Knoten in einer solchen Gruppe, aber dieser negative Effekt ist bei einer so großen Knotenanzahl fast zu vernachlässigen. Gerade in der dritten Simulation, bei der die Knoten im Mittel über eine größere Kapazität verfügen, ist deutlich erkennbar, dass der Algorithmus mit Latenzzeitberücksichtigung weniger Gruppen erzeugt, als der erste Algorithmus.

Ein in der Abbildung nicht aufgeführter Nebeneffekt ist die Art der Gruppenbildung

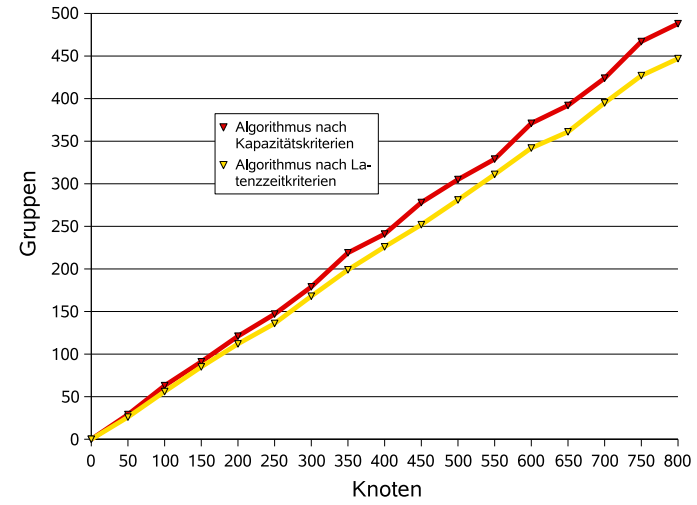
[2]Der genaue Aufbau dieses Simulators ist im Anhang A.3 genauer beschrieben.



(a) Gruppenbildung bei Knoten mit Kapazitäten von 10% bis 60% der erforderlichen Gesamtkapazität



(b) Gruppenbildung bei Knoten mit Kapazitäten von 10% bis 110% der erforderlichen Gesamtkapazität



(c) Gruppenbildung bei Knoten mit Kapazitäten von 60% bis 110% der erforderlichen Gesamtkapazität

Abbildung 7.5: Die Erzeugung von Rezeptorgruppen mit beiden Varianten des Algorithmus bei steigender Knotenanzahl und unterschiedlichen Kapazitätseigenschaften der zu gruppierenden Knoten

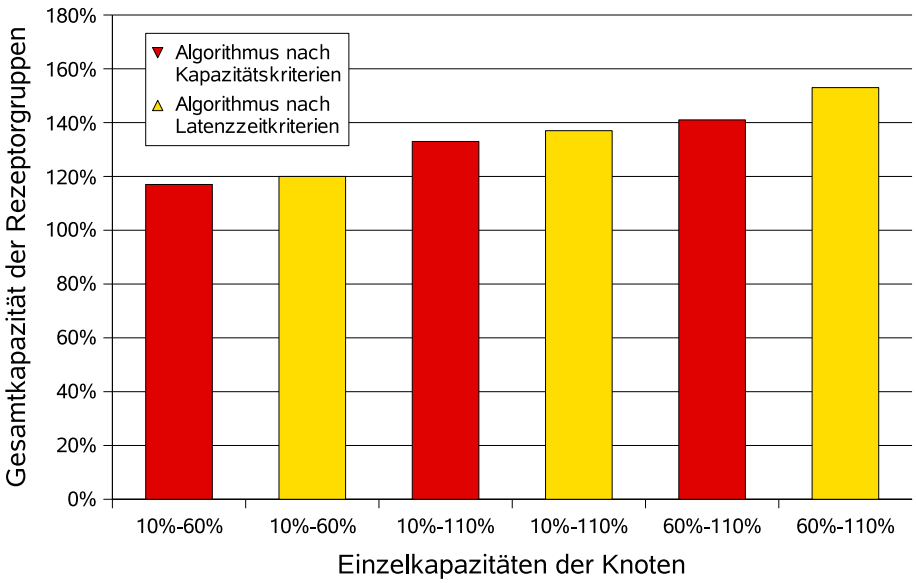


Abbildung 7.6: Durchschnittliche Gesamtkapazität der erzeugten Rezeptorgruppen mit beiden Varianten des Algorithmus bei unterschiedlichen Kapazitätseigenschaften der zu gruppierenden Knoten

beim optimierten Algorithmus. Die erste Version des Algorithmus erzeugte oft Gruppen mit wenig Mitgliedern, jedoch großen Einzelkapazitäten. Aber oft wurden auch Gruppen aus sehr vielen Knoten gebildet, die über wenig Kapazität verfügten. Letzteres hat gerade in diesen Gruppen eine negative Auswirkung. Denn diese Gruppen müssen ja Nachrichten oder Rezeptoren des öfteren zwischen den Gruppenmitgliedern rotieren und verwenden dazu das Netzwerk als Transportmedium. Je weniger Mitglieder eine Gruppe besitzt, desto weniger wird das Netzwerk belastet und Verzögerungen somit vermieden.

Ein weiterer Unterschied der beiden Algorithmen besteht in den resultierenden Gesamtkapazitäten der gebildeten Rezeptorgruppen. Da der erste Algorithmus in der Regel mehr Gruppen erzeugt als der auf die Latenzzeit optimierte Algorithmus, besitzen die Gruppen im Durchschnitt auch weniger Gesamtkapazität. Es findet zwar dadurch eine bessere Ausnutzung des Speicherplatzes statt. Allerdings ist es nicht das Ziel des Selbstschutzsystems, den Speicher eines Knotens optimal auszunutzen. Vielmehr ist es von Vorteil, wenn die Knoten trotz Schutzsystem noch über ausreichend Restspeicher verfügen. In Abbildung 7.6 sind die Auswirkungen der beiden Algorithmen deutlich zu erkennen. Wie bei der vorigen Simulation wurden auch hier Netzwerke generiert, bei denen die Knoten eine Kapazität von 10% bis 60%, von 10% bis 110% und von 60% bis 110% besaßen. Der linke (rote) Balken repräsentiert dabei immer den Algorithmus, der die Gruppen streng nach Kapazitäten bildet, wohingegen der rechte (gelbe) Balken die Ergebnisse des modifizierten Algorithmus darstellt. Bei allen Simulationen erzeugte der zweite Algorithmus Gruppen mit einer größeren Gesamtkapazität. Dies hat zur Folge,

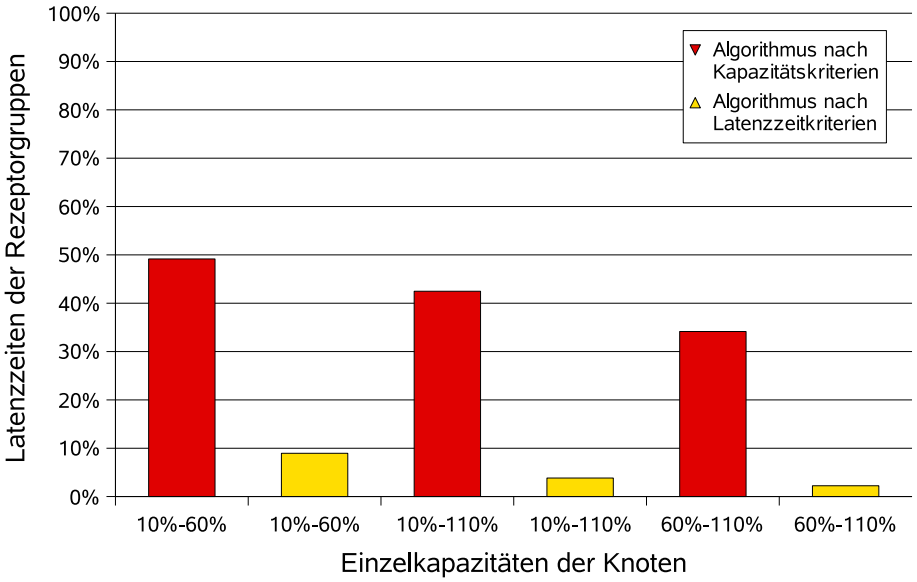


Abbildung 7.7: Durchschnittliche Gesamtkapazität der erzeugten Rezeptorgruppen mit beiden Varianten des Algorithmus bei unterschiedlichen Kapazitätseigenschaften der zu gruppierenden Knoten

dass jeder Knoten im Durchschnitt weniger Rezeptoren aufnehmen muss, als er zum Speichern in der Lage wäre.

Der interessanteste Aspekt ist allerdings die Betrachtung der Latenzzeiten in den gebildeten Rezeptorgruppen. In den Simulationen wurden die Latenzzeiten zwischen den Knoten zufällig als Zahl zwischen 1 und 100 gewählt. Dadurch lassen sich die Latenzzeiten relativ leicht als Prozentwerte abbilden. 100% entspricht dabei der maximalen Entfernung oder Verzögerung zwischen zwei Knoten. Je niedriger dieser Wert ist, desto näher liegen die Knoten im Netzwerk nebeneinander. In Abbildung 7.7 sind die Ergebnisse der Simulationen dargestellt. Dabei wurden die selben Verteilungen der Kapazitäten verwendet, wie bei den beiden vorigen Simulationen. Der linke (rote) Balken in einem Kapazitätsintervall zeigt die Ergebnisse des ersten Algorithmus, wogegen der rechte (gelbe) Balken die Ergebnisse der Gruppierung nach Latenzzeitminimierung widerspiegelt. Da der erste Algorithmus keinerlei Informationen über die Entfernung zweier Knoten berücksichtigt und die Entfernungen zufällig generiert wurden, bewegt sich die durchschnittliche Entfernung einer Rezeptorgruppe auch bei 50% der möglichen Latenzzeiten, wenn viele unterschiedliche Knoten in eine Gruppe aufgenomen werden. Dies ist der Fall, wenn die Knoten nur geringe Kapazitäten aufweisen. Je mehr Kapazität die Knoten bereits selbst in die Gruppe einbringen, desto schneller hat die Gruppe ihre Kapazität erreicht. Aus diesem Grund nimmt die interne Kommunikationsdauer der Gruppe ab, wenn sich mehr Knoten im Netzwerk befinden, die einen großen Speicher aufweisen. Der veränderte Algorithmus erzeugte dagegen bei allen Testläufen Gruppen mit weitaus

geringerer Verzögerung zwischen den Knoten. Eine enorm niedrige Latenzzeit ergibt sich auch hier bei einer Gruppenbildung von vorwiegend Knoten mit großer Kapazität (sechster Balken in der Abbildung). Die internen Latenzzeiten bewegen sich dabei im Durchschnitt nur noch knapp oberhalb von 2%. Das lässt sich dadurch erklären, dass es bei dem Szenario mit Kapazitäten von 60% bis 110% nur maximal zwei Gruppenmitglieder geben kann. Ab einer gewissen Anzahl von Knoten findet sich somit immer ein Knotenpaar mit einer geringen Latenzzeit auf dem Kommunikationsmedium.

Die Testläufe haben gezeigt, dass der Gruppierungsalgorithmus allein durch die Berücksichtigung der Latenzzeiten zwischen den Knoten, viel effizientere Gruppen bildet. Durch die geringere Latenzzeit innerhalb einer Rezeptorgruppe kann der Overhead für die interne Kommunikation so gering wie möglich gehalten werden.

7.6.2 Generierung der Rezeptoren nach Konfiguration der Middleware

Die Middleware OCμ besitzt die Fähigkeit der Selbstkonfiguration. Jeder Dienst der Middleware stellt spezifische Anforderungen an seinen Knoten und benötigt eventuell bestimmte Ressourcen. Aus diesem Grund kann nicht jeder Dienst auf jedem Knoten ausgeführt werden. Manche Dienste sind beispielsweise auch an einen speziellen Knoten gebunden oder können zur Laufzeit frei in der Middleware von Knoten zu Knoten migrieren. All diese Eigenschaften werden in der Konfiguration der Middleware hinterlegt.

Die Selbstkonfiguration besteht aus einer Beschreibungsdatei im XML-Format, die Informationen über die benötigten Dienste und deren individuellen Anforderungen beinhaltet. Diese Informationen werden beim Starten an die Knoten der Middleware gesendet und durch ein Wahlverfahren [Tru06] werden die Dienste der Reihe nach auf die passenden Knoten verteilt.

Anhand dieser Konfiguration erhält auch jeder Thymus-Knoten implizit das Wissen über alle in der Middleware gestarteten Dienste. Jeder Thymus-Service erhält somit das Wissen über die vom Dienst verwendeten Nachrichtentypen, woraus zuerst das Self-Set generiert wird und anschließend die notwendigen Rezeptoren gebildet werden können. Sobald der Thymus alle notwendigen Rezeptoren gebildet hat, müssen diese selbstverständlich an die entsprechenden Knoten übertragen werden.

7.6.3 Erneuerung der Rezeptoren nach Rekonfiguration der Middleware

In einem Organismus bleibt das Self-Set über die gesamte Lebensdauer relativ konstant, da der Körper seine Eigenschaften, wie beispielsweise die Blutgruppe, über die gesamte Lebenszeit nicht verändert. Lediglich in der Pubertät ändert sich der Hormonhaushalt und der Stoffwechsel – und somit das Self-Set – worauf das Immunsystem angepasst werden muss [TFR00]. Im Gegensatz zu den organischen Lebewesen stellt die Middleware kein solch ein statisches System dar, das eingeschalten wird und seinen Zustand bezüglich seines Self-Sets ab dann nicht mehr ändert. Vielmehr soll OCμ dem Anwender ein dynamisches System zur Verfügung stellen, in dem Knoten und Dienste zur Laufzeit hinzukommen oder sich von der Middleware jederzeit wieder abmelden können. Somit ändert sich auch zur Laufzeit die Information darüber, was zum Self-Set gehört und was nicht zum vertrauenswürdigen Teil der Middleware gezählt werden kann. Diese Dynamik stellt das Immunsystem der Middleware vor eine große Herausforderung.

Immer dann, wenn sich neue Teilnehmer am System anmelden oder bestimmte Dienste die Middleware verlassen, ändert sich das Self-Set des Systems und damit auch die effektiven Rezeptoren. Der Knoten auf dem der neue Dienste gestartet wird oder der einen Dienst beendet, meldet diese Veränderung bei dem Thymus-Service bei dem er registriert ist. Bei neu hinzukommenden Diensten berechnet der Thymus-Service anschließend, welche Antikörper nun als ungültig eingestuft werden, weil sie eine Nachricht aus dem Self-Set erkennen würden, und löscht diese. Falls entsprechende Prokörper-Sets vorhanden sind, können die Antikörper auch direkt in Prokörper umgewandelt werden und dem entsprechenden Set hinzugefügt werden. Für beendete Dienste generiert der Thymus-Service automatisch neue Antikörper, falls diese nicht bereits vorhanden sind. Wenn Prokörper-Sets vorhanden sind, in denen relevante Rezeptoren für die abgemeldeten Dienste existieren, werden diese Sets um die entsprechenden Bitmuster reduziert.

Durch das Hinzukommen neuer Dienste kann sich das Self-Set derart vergrößern, dass die bisher verwendete Rezeptorlänge nicht mehr ausreichend ist oder überhaupt keine freien Rezeptoren für einen Antikörper zur Verfügung stehen. In diesem Fall wird die minimale Rezeptorlänge dementsprechend erhöht und neue Rezeptoren mit dieser neuen Länge generiert.

Jede Veränderung im Repertoire der Rezeptoren muss selbstverständlich auch mit den restlichen Thymus-Services im Netzwerk abgeglichen werden, die anhand der übermittelten Veränderungen eine identische Aufbereitung der Rezeptoren vornehmen. Diese Synchronisation wird durch einen speziellen Nachrichtenaustausch der Thymus-Services realisiert.

7.7 Erkennen von bösartigen Knoten und Diensten

Wenn eine Nachricht an einem Knoten ankommt, wird sie zuerst vom Transport-Connector entgegen genommen und von dort aus an den Event-Dispatcher weitergeleitet. Im Event-Dispatcher wird die Nachricht dann zuerst durch alle aktiven Monitore seiner Incoming-Monitor-Queue geschleust, bis sie letztendlich an die registrierten Dienste ausgeliefert wird. Da der Intrusion-Detection-Monitor ebenfalls in dieser Incoming-Monitor-Queue des Event-Dispatchers angesiedelt ist, durchläuft eine eingehende Nachricht automatisch die Überprüfung, ob sie zum Self-Set gehört oder zur Menge der unbekannten Non-Self-Nachricht zählt. Um diese Unterscheidung treffen zu können, wird die Nachricht vom Intrusion-Detection-Monitor an den lokalen Immune-Service weitergereicht. Dieser beinhaltet die für die Überprüfung notwendigen Rezeptoren und kann die Nachricht dahingehend untersuchen, ob sie als gut- oder bösartig eingestuft werden soll.

Wenn es sich dabei um eine Nachricht aus dem Self-Set handelt, wird sie vom Monitor unverändert weitergeleitet und der Nachrichtenfluss nimmt ungehindert seinen Lauf. Dies gilt allerings nur für Knoten, die das gesamte Rezeptor-Repertoire bei sich vorrätig haben oder sich keine Rezeptoren mit anderen Knoten teilen. Ein leistungsschwacher Knoten kann – wie in diesem Kapitel bereits erwähnt – die Mitglieder in seiner Rezeptorgruppe um Hilfe bitten, in dem er die Nachricht reihum an alle Knoten der Gruppe sendet. Dazu wird nur der entsprechende Nachrichtentyp in eine neue Nachricht gepackt. Diese Nachricht wird von den anderen Knoten geöffnet, der enthaltene Nachrichtentyp separat mit dem jeweiligen Rezeptor-Set geprüft und eine Statusmeldung an den ursprünglichen Knoten gesendet. Erst wenn alle Knoten die Nachricht als gutartig empfunden haben, wird die Nachricht normal am ursprünglichen Knoten weiterverarbeitet.

Sobald ein Knoten – oder ein Knoten aus der zugehörigen Rezeptorgruppe – die Nachricht als Non-Self identifiziert hat, wird die Nachricht keinesfalls an die darüber liegenden Dienste weitergeleitet. Vielmehr wird davon ausgegangen, dass der Dienst, der die Nachricht versendet hat, die Absicht hatte, dem lokalen Knoten eventuell Schaden zuzuführen. Da jede Nachricht in OCμ die Information darüber beinhaltet, von welchem Dienst und von welchem Knoten sie erzeugt und versendet wurde, kann anhand dieser Information die eindeutige Knoten-ID und eindeutige Service-ID in Erfahrung gebracht werden.

7.8 Einsatz in realen Systemen

Die bisherigen Evaluierungen stützen sich auf Erkenntnisse, die hauptsächlich in Simulatoren gewonnen wurden. Testläufe in realen Middlewaresystemen stehen noch aus. Da derzeit jedoch noch kein reales Netzwerk existiert, in dem ausreichend OCμ-Knoten zur Verfügung stehen und auf denen eine Vielzahl von unterschiedlichen Diensten gestartet

ist, können nur Vermutungen getroffen werden, was den Einsatz des Selbstschutzsystems in großen realen Systemen betrifft. Selbstverständlich verzögert das Schutzsystem die Antwortzeiten eines Knotens in der Middleware. Die bisher gewonnen Erkenntnisse zeigen allerdings auf, dass das System annähernd linear skaliert. Die Anzahl der benötigten Rezeptoren ist direkt proportional zur Anzahl der im System vorhandenen Self-Nachrichten und die Zeitkomplexität reduziert sich bei Rezeptorvergleichen auf Grund von Optimierungen auf $O(1)$. Dies stellt eine gute Voraussetzung dar, um sich in einem realen Einsatzgebiet mit anderen Konzepten und Architekturen messen zu können. Genaue Laufzeitdaten sind leider nicht bekannt, da ein derartiges System von vielen unterschiedlichen Parametern, wie beispielsweise der Netzwerktopologie, den Ressourcenunterschieden der beteiligten Knoten oder von der bereits vorhandenen Auslastung der Knoten abhängt.

Wenn gängige Desktop-PC-Systeme als OCμ-Knoten eingesetzt werden, dann kann der Overhead des Schutzsystems nahezu vernachlässigt werden. In Testläufen war selbst in einem System mit 4.000 Self-Nachrichten bei 20.000 unterschiedlichen Rezeptoren kein nennenswerter Zeitverlust für die Vergleiche beim Eintreffen von Nachrichten zu verzeichnen. Auf mobilen Endgeräten oder leistungsschwachen Sensoren sieht die Situation aber sicherlich anders aus. Da die Middleware derzeit auf eingebettete Systeme angepasst und übertragen wird, kann eventuell in naher Zukunft ein realer Test auf eingebetteten Java-Prozessoren durchgeführt werden. Die Ergebnisse der Testläufe und der Einsatz von Speicherplatzoptimierungen lassen jedoch vermuten, dass selbst diese eingebetteten Systeme keine großen Einbußen durch ein aktiviertes Schutzsystem verzeichnen werden. Gerade auch die Möglichkeit der Gruppenbildung von ressourcenarmen Knoten und der darin implementierten Teamwork-Funktionalität, lässt das System auch in diese Richtung gut skalieren. Falls ein Knoten aber tatsächlich zu wenig Ressourcen für das Schutzsystem zur Verfügung hat, kann er auch jederzeit ohne ein aktiviertes Schutzsystem mit anderen Knoten in der Middleware kommunizieren.

7.9 Fazit

Dieses Kapitel hat gezeigt, dass sich die Computer-Immunologie sehr gut in OCμ abbilden und implementieren lies. Die verschiedenen Instanzen des Selbstschutzsystems werden als Dienste und Monitore umgesetzt. Die Eckpfeiler des Systems stellen dabei die sogenannten Thymus-Knoten dar, die der Administrator in OCμ festlegt. Die übrigen Knoten der Middleware schließen sich nach einem bestimmten Algorithmus zu einer Thymus-Gruppe zusammen. Jede dieser Gruppen pflegt eine Beziehung zu genau einem Thymus-Knoten, welcher die Gruppenmitglieder mit den notwendigen Rezeptoren versorgt. Der Thymus-Knoten hält seine Gruppenmitglieder auch bei Konfigurationsänderungen stets auf dem neuesten Stand, wenn in der Middleware beispielweise neue

Dienste hinzugekommen oder alte Dienste entfernt wurden.

Falls manche Mitglieder innerhalb einer Thymus-Gruppe nicht über die ausreichende Kapazität verfügen, werden kleinere Teams oder sogenannte Rezeptorgruppen innerhalb der Thymus-Gruppe gebildet. Zusammen verfügen diese ressourcenarmen Knoten dann über eine ausreichende Kapazität, um alle notwendigen Rezeptoren aufnehmen zu können. Ein Gruppierungsalgorithmus erzeugt dabei effiziente Gruppen mit sehr kleinen Kommunikationsverzögerungen. Dies ist wichtig, da die Mitglieder einer Rezeptorgruppe ihr Rezeptor-Set gemeinsam nutzen. Eintreffende Nachrichten werden dann nicht nur von einem einzigen Knoten, sondern eventuell auch von den anderen Team-Mitgliedern überprüft. Dadurch können auch schwache Knoten in den Genuss eines effektiven Schutzssystems kommen.

8 Reaktion auf Bedrohungen und mögliche Beseitigungen

In Kapitel 7.7 wurde beschrieben, wie eventuell schädliche Nachrichten von bösartigen Diensten oder Knoten in der Middleware aufgespürt werden können. Falls ein solches Objekt vom System erkannt wird, müssen Schritte eingeleitet werden, die die korrekte Funktionalität der Middleware aufrecht erhalten. Anhand der erkannten Bedrohung muss unterschiedlich reagiert werden.

8.1 Bösartige Dienste

Wenn das künstliche Immunsystem eines Knotens einen potenziell bösartigen Dienst in OCμ erkannt hat, dann wird dessen eindeutige Service-ID umgehend dem Event-Dispatcher mitgeteilt. Der Event-Dispatcher leitet ab diesem Zeitpunkt keine einzige Nachricht mehr von diesem Dienst an darüberliegende registrierte Dienste weiter. Lediglich ein Administrator kann diese Sperrung wieder aufheben. Andere Knoten in der Middleware haben aber möglicherweise bereits selbst erkannt, dass es sich bei dem sendenden Dienst um einen potenziell schadhaften Dienst handelt und haben diesen ebenfalls ihrem Event-Dispatcher gemeldet. Somit blockieren auch diese Knoten automatisch eingehende Nachrichten von diesem Dienst. Es kann jedoch nicht davon ausgegangen werden, dass andere Knoten diesen Dienst auch tatsächlich selbst erkannt haben. Denn wie im vorigen Kapitel beschrieben, verfügt nicht jeder Knoten über die selben Möglichkeiten zur Erkennung von Non-Self-Nachrichten. Aus diesem Grund meldet jeder Knoten, der einen potenziell bösartigen Dienst erkannt hat, dessen Service-ID an seinen Thymus-Service bei dem er registriert ist. Dieser Thymus-Service wiederum leitet diese Nachricht an alle seine registrierten Knoten weiter, und setzt sie somit von der Existenz des bösartigen Dienstes in Kenntnis. Um nun noch alle restlichen Knoten der Middleware, die bei anderen Thymus-Knoten im Netzwerk registriert sind, von der Bedrohung zu informieren, sendet jeder Thymus-Dienst, der eine Bedrohung als solche identifiziert hat, diese Information an alle ihm bekannten Thymus-Knoten im Netzwerk.

Ein Thymus-Dienst besitzt die alleinige Kontrolle über die Auslieferung dieser Bedrohungsbenachrichtigungen an seine registrierten Knoten. Aus diesem Grund werden in-

nerhalb dieser Gruppe keine Nachrichten mehrfach versendet. Beim Austausch der Information zwischen den Thymus-Knoten ist jedoch keine zentrale Instanz vorhanden, die sich um die Zustellung kümmert. Jeder Thymus-Knoten handelt autonom und prinzipiell ohne Kenntnis über die Tätigkeiten der anderen Thymus-Knoten in OCμ. Dadurch ist es möglich, dass die Meldungen über bösartige Dienste mehrfach an einem Knoten eintreffen. Um diese Duplikate zu minimieren, prägt jeder Thymus-Service der Nachricht noch die Zusatzinformation auf, dass er selbst die Nachricht bereits erhalten hat, an welche Knoten die Nachricht bereits gesendet wurde und an welche sie noch gesendet wird. Jeder Thymus-Knoten, der eine solche Benachrichtigung erhält, wertet diese Information vor dem erneuten Versenden aus und leitet die Nachricht nur noch an die Knoten weiter, die in dieser Liste noch nicht aufgeführt sind. Im schlimmsten Fall erreicht das Nachrichtenaufkommen in OCμ zwischen t Knoten, die einen Thymus-Dienst gestartet haben, einen Maximalwert von

$$(2 \cdot \sum_{n=0}^{t-1} n) - (t-1) \tag{8.1}$$

Nachrichten. In einem Netzwerk von t Knoten senden im schlimmsten Fall alle Knoten ihre Benachrichtigungen über einen potenziell bösartigen Dienst an alle anderen Knoten. Somit wird jeder logische Kanal zwischen zwei Knoten doppelt verwendet. Da sich der initiale Thymus-Knoten jedoch bereits zu Beginn in die Liste der erhaltenen Knoten einträgt, bekommt er keine Nachricht mehr zurückgesendet. Aus diesem Grund werden $t-1$ Nachrichten weniger erzeugt. Zu dieser Anzahl von Nachrichten kommt noch hinzu, dass jeder Thymus-Service die Nachricht an seine registrierten Knoten sendet, wovon er selbst ausgenommen ist. Bei einer Gesamtanzahl von k Knoten, erweitert sich die Formel daher auf

$$(2 \cdot \sum_{n=0}^{t-1} n) - (t-1) + k - t \tag{8.2}$$

$$= (2 \cdot \sum_{n=0}^{t-2} n) + (t-1) + k - t \tag{8.3}$$

$$= (2 \cdot \sum_{n=0}^{t-2} n) + k - 1 \tag{8.4}$$

Nachrichten, die bei der Benachrichtigung von Bedrohungen maximal ausgetauscht werden müssen. Diese Anzahl an maximal auftretenden Nachrichten sollte in Betracht gezogen werden, wenn die Anzahl der zu startenden Thymus-Dienste in der Middleware dimensioniert und festgelegt wird.

Wenn ein Knoten beim Eintreffen einer solchen Benachrichtigung erkennt, dass der bösartige Dienst auf ihm läuft, wird zuerst jegliche Kommunikation mit dem Dienst im Event-Dispatcher des Knotens unterbunden. Dies gilt sowohl für eingehende als auch ausgehende Nachrichten. Anschließend versucht der Knoten, den Dienst zu beenden. Je nach Implementierung der Middleware kann das Beenden zur Laufzeit unterschiedlich erfolgreich sein. Falls die gesamte Middleware beispielsweise in einer virtuellen Java-Runtime ausgeführt wird und die Dienste als Threads realisiert sind, kann der Dienst rein technisch nicht einfach beendet werden, ohne die gesamte Virtual-Machine zu stoppen. Damit würde allerdings auch die Middleware beendet werden, was eventuell noch mehr Schaden anrichten würde. In OCμ wurden die einzelnen Dienste daher nachträglich als eigene Prozesse im Betriebssystem realisiert. Somit ist es möglich, einen Dienst über die Betriebssystem-Funktionen zu beenden ohne dass die Middleware davon beeinträchtigt wird. Bei eingebetteten Systemen muss daher bei der Realisierung der Middleware-Komponenten auf die Möglichkeit geachtet werden, dass sich Dienste sicher beenden lassen.

8.2 Unkooperative Knoten

Ein Knoten kann als unkooperativ eingestuft werden, wenn er trotz Anweisung einen auf ihm laufenden bösartigen Dienst nicht gestoppt hat. Dies kann daran erkannt werden, dass weiterhin Nachrichten von diesem bösartigen Dienst versendet werden, obwohl der sendende Knoten bereits darüber informiert wurde, dass er den entsprechenden Dienst zu deaktivieren oder zu beenden hat.

In diesem Fall kann nicht sehr viel unternommen werden, da über das Netzwerk kein direkter Zugriff auf einen Knoten existiert. In OCμ teilen die Knoten, die einen anderen Knoten als bösartig eingestuft haben, den restlichen Knoten mit, dass es sich bei der entsprechenden Knoten-ID um einen bösartigen Knoten handelt. Die Verteilung der Nachrichten geschieht dabei nach dem selben Schema, wie beim Benachrichtigen von bösartigen Diensten. Das Nachrichtenaufkommen nimmt in diesem Fall gleiche Ausmaße an, wie in den Formeln 8.1 bis Formel 8.4 beschrieben wurde.

8.2.1 Isolation des Knotens

Die einzige Möglichkeit für die gutartigen Knoten besteht nun darin, in ihrem Event-Dispatcher die bösartige Knoten-ID in eine Blacklist einzutragen. Eintreffende Nachrichten werden mit dieser Liste verglichen, bevor sie an die entsprechenden Dienste weitergeleitet werden. Ist die ID des sendenden Knotens in der Blacklist aufgeführt, wird die Nachricht schlichtweg verworfen und ignoriert.

Dieser Mechanismus schützt die Knoten allerdings nur bedingt vor einem bösartigen Knoten. Es kann zwar keine Nachricht eines bösartigen Knotens mehr ins Innere des Systems eindringen, jedoch kann ein gutartiger Knoten beispielsweise mit Nachrichten überschwemmt werden, so dass dieser keine freie Kapazität mehr für die Abarbeitung seiner Dienste besitzt. In Kapitel 3 wurde jedoch bereits angemerkt, dass diese Art von Bedrohungen in dieser Arbeit nicht betrachtet werden, da zur Bekämpfung dieser Angriffe tiefere Eingriffe in der Netzwerkschicht und Firewallkonfigurationen notwendig sind oder gar Netzwerkadministratoren eingeschaltet werden müssen.

8.2.2 Der positive Effekt der Selbstoptimierung

Wenn ein Knoten einen bösartigen Dienst beherbergt, diesen jedoch nach erfolgreicher Identifizierung und Benachrichtigung nicht selbst beendet, dann kann die in OCμ implementierte Selbstoptimierung ihren Teil zur Beseitigung beitragen. Angenommen, der bösartige Dienst konsumiert relativ viele Rechen- oder Speicherressourcen auf einem Knoten, so wird die Selbstoptimierung früher oder später eine Umlagerung des Dienstes in Erwägung ziehen und den Dienst auf einen anderen Knoten verschieben. Da alle Knoten in der Middleware bereits zuvor von diesem bösartigen Dienst und seiner Service-ID in Kenntnis gesetzt wurden, weiß der neue Knoten automatisch, dass er nun einen bösartigen Dienst zur weiteren Ausführung erhalten wird. Anstatt diesen nun wieder zu aktivieren, kann der Knoten den Dienst gleich verwerfen und somit die Beseitigung anstelle des ursprünglichen Knotens vornehmen.

Dieses Vorgehen funktioniert allerdings nur dann, wenn die für die Selbstoptimierung relevanten Nachrichten weiterhin in der Middleware versendet und empfangen werden, obwohl der Knoten bereits in der oben genannten Blacklist aufgeführt ist. Nur dadurch kann sichergestellt werden, dass die Selbstoptimierung auch unabhängig von dem Selbstschutzsystem funktionstüchtig bleibt.

Alternativ kann die Selbstoptimierung auch proaktiv handeln, um das Problem der unkooperativen Knoten zu beseitigen. Normalerweise entscheidet die Selbstoptimierung erst nach einer gewissen Zeit, ob die Verschiebung eines Dienstes einen besseren Gesamtressourcenverbrauch der Middleware nach sich zieht. Im Fall eines unkooperativen Knotens kann der als bösartig eingestufte Dienst unmittelbar von der Selbstoptimierung zur Migration veranlasst werden. Dadurch wird die Beseitigung alternativ von einem anderen Knoten vorgenommen.

Die Unterstützung der Selbstoptimierung kann in allen Fällen nur dann erfolgen, wenn der unkooperative Knoten noch über eine funktionierende Selbstoptimierung verfügt. Wenn die Selbstoptimierung beispielsweise vom bösartigen Dienst gestoppt oder manipuliert wurde, dann kann der Dienst auch nicht für eine Migration vorbereitet werden.

8.3 Bösartige Knoten

Eine weitere Möglichkeit für einen Angriff besteht darin, dass ein Knoten ständig neue bösartige Dienste mit zufälligen Sender-IDs generiert, oder einen bösartigen Dienst laufen hat, der seine Sender-ID fortlaufend ändert. In beiden Fällen ist das Immunsystem von OCμ relativ machtlos, denn der Dienst wird zwar als bösartig erkannt, jedoch schlägt eine Beseitigung des Dienstes fehl, da der Dienst mit dieser ID ja bereits nach dem Senden nicht mehr existiert. Der gesamte Knoten muss folglich als bösartig eingestuft werden. Um diese Einstufung vollziehen zu können, besitzt jeder Knoten intern einen Zähler für jeden anderen Knoten im Netzwerk. Dieser Zähler wird immer dann inkrementiert, wenn eine eintreffende Nachricht von diesem Knoten als Non-Self erkannt wurde. Falls dieser Zähler einen bestimmten und vorher festgelegten Schwellwert erreicht hat, verhält sich der Knoten identisch wie in 8.2 beschrieben. Leider kann das Selbstschutzsystem in OCμ gegen bösartige Knoten nicht viel ausrichten - insbesondere dann nicht, wenn sich die Identität immer wieder ändert. Aber jedes System ist machtlos gegenüber einem Feind der sich stets anders maskiert.

8.4 Falsch eingestufte Nachrichten

In Kapitel 6 wurde gezeigt, dass das Immunsystem keine Aussagen trifft, die falsch positiv sein können (siehe Abb. 6.6 in Kapitel 6). Die einzige Fehlerrate besteht darin, dass eine Nachricht als Self und somit als falsch negativ eingestuft wird, obwohl sie unbekannter oder bösartiger Natur ist. In der Theorie stimmen diese beiden Aussagen. Jedoch fließen im laufenden Betrieb noch andere Faktoren mit ein. Durch Rekonfigurationen erfährt die Middleware OCμ immer Änderungen am Self-Set. Dadurch sind die Thymus-Knoten angehalten, die generierten Rezeptoren auf dem neuesten Stand zu bringen, und ihre registrierten Knoten damit zu versorgen.

Da es in einem Netzwerk immer zu Laufzeitverzögerungen kommt, besitzt kein Knoten bei einer Änderung des Self-Sets umgehend ein aktualisiertes Rezeptor-Repertoire. Die Folge davon ist, dass ein Knoten eine eintreffende Nachricht fälschlicherweise als Non-Self einstuft, falls er noch keine Aktualisierung seines Rezeptor-Sets erfahren hat. Dieser Knoten versendet nun logischerweise eine Bedrohungsbenachrichtigung an seinen Thymus-Knoten. Dessen Aufgabe wiederum ist es nun, die Thymus-Knoten in seiner Nachbarschaft auch davon in Kenntnis zu setzen. Dieser Sachverhalt führt zu einem Konflikt. Da aber mindestens *ein* Thymus-Knoten im Netzwerk über den neu hinzugekommenen Dienst Bescheid weiß – nämlich der Thymus-Knoten bei dem der ausführende Knoten registriert ist – muss dieser Thymus-Knoten eine Entwarnungsbenachrichtigung versenden, wenn er selbst die Benachrichtigung über die Bedrohung zugesendet bekommt. Diese Entwarnung wird nach demselben Prinzip versendet, wie die ursprüngliche Be-

drohungsbenachrichtigung und erreicht somit auch früher oder später alle Knoten. Ein neuer Dienst sollte daher in OCμ nicht sofort nach seinem Starten Kontakt mit anderen Knoten aufnehmen, sondern je nach Netzwerk eine gewisse Zeit verstreichen lassen, bevor Nachrichten von im ausgeliefert werden. Sobald aber alle Rezeptor-Sets auf allen Knoten der Middleware aktualisiert wurden, wird der neue Dienst automatisch als gutartig eingestuft und kann somit ungestört mit anderen Diensten kommunizieren.

Der umgekehrte Fall, dass eine Nachricht als gutartig eingestuft wird, obwohl sie eigentlich bösartiger Natur ist, kommt in der Praxis eher selten vor. Dieses Fehlverhalten kann dann auftreten, wenn ein Angreifer eine Nachricht versendet, die durch das Hashverfahren identisch zu einer im Self-Set aufgelisteten Nachricht ist. Dies ist allerdings sehr unwahrscheinlich und schwer zu konstruieren. Eine zweite Möglichkeit eröffnet sich für einen Angreifer, nachdem ein Dienst seine Arbeit erfolgreich beendet hat und in der Middleware beendet wurde. Bis das Rezeptor-Set der Knoten aktualisiert ist, werden weiterhin Nachrichten von dem beendeten Dienst von der Middleware akzeptiert. Dies kann durch einen Angreifer ausgenutzt werden, in dem Nachrichten mit der entsprechenden Service-ID generiert werden. Da die Nachrichten jedoch typisiert sind und der Typ der Nachricht in die Signatur zum Überprüfen der Nachrichten mit einfließt, bietet sich für einen Angreifer in der Regel keine große Angriffsfläche. Dennoch stellen diese beiden Szenarien ungelöste Schwachstellen des Systems dar. Aber wie in der Natur, gibt es auch in elektronischen Systemen nie eine hundertprozentige Sicherheit.

8.5 Fazit

Dieses Kapitel hat gezeigt, wie gegen bösartige Dienste und Knoten in OCμ vorgegangen wird. Prinzipiell werden immer alle Knoten informiert, wenn bösartige Dienste oder Knoten in der Middleware identifiziert werden. Die Bekämpfung von bösartigen Diensten ist dabei auch einfacher, als die Bekämpfung bösartiger Knoten. Dienste laufen in OCμ in eigenen Prozessen und können vom Betriebssystem ohne größere Beeinträchtigung des Knotens beendet werden. Bösartige Knoten stellen jedoch ein größeres Problem dar. Diese können nicht vom Netzwerk ausgeschlossen oder gar entfernt werden, da in der Regel kein physikalischer Zugriff auf diese Knoten besteht. Die einzige Lösungsstrategie besteht darin, die Nachrichten eines solchen Knotens zu ignorieren. Dies zeigt deutlich, dass die Beseitigung von Bedrohungen in der Realität ein viel größeres Problem darstellt, als ihre eigentliche Erkennung.

9 Zusammenfassung und Ausblick

Der vorgestellte Selbstschutz-Mechanismus der Organic Ubiquitous Middleware OCμ wird in diesem Kapitel zusammengefasst. Anschließend wird ein Ausblick auf mögliche Erweiterungen und Verbesserungen gegeben.

9.1 Zusammenfassung

In dieser Arbeit wurde ein gestuftes Selbstschutzsystem für die Middleware OCμ aufgezeigt und evaluiert. In einer ersten Stufe werden Knoten und Dienste in der Middleware von einem Administrator autorisiert. Die zweite Stufe besteht aus einer Rollenvergabe, die bestimmten Knoten nur eingeschränkte Rechte einräumt. Als letzte Stufe kommt das künstliche Immunsystem zum Tragen, welches sich um das Erkennen von unbekannten Nachrichten und um das Beseitigen der Bedrohungen kümmert. Es hat sich in diversen Testumgebungen gezeigt, dass das Schutzsystem ein gutes und effizientes Werkzeug zum Erkennen von unbekannten Bedrohungen darstellt und diese auch erfolgreich isolieren oder gar beseitigen kann. Ein Vorteil des vorgestellten Systems ist, dass es nicht auf einer Verschlüsselung basiert, was die Verschlüsselung für alle Teilnehmer erzwingen würde. Es können immer alle Knoten miteinander kommunizieren – egal ob das Selbstschutzsystem bei ihnen aktiviert oder deaktiviert ist.

9.1.1 Das künstliche Immunsystem

Für die künstliche Variante wurde das biologische Immunsystem als Vorbild genommen und auf die digitale Kommunikation in der Middleware adaptiert. Das biologische Immunsystem funktioniert durch ein komplexes Zusammenspiel von vielen Einzelkomponenten, die sich über Millionen von Jahren evolutionär entwickelt haben. Während dieses Entwicklungsprozesses hat sich dieses Schutzsystem auch extrem an die Bedürfnisse des Organismus auf der Erde angepasst und ist nicht ohne weiteres auf die Computerwelt übertragbar. Dennoch lassen sich gewisse Grundkonzepte des biologischen Vorbilds extrahieren und die Methodik des Immunsystems in der Middleware anwenden. Die negative Selektion ist dabei ein zentral herausstechendes Merkmal. Das biologische Immunsystem vergleicht die Moleküle und Proteine im Körper ständig, ob sie zur Gruppe der

gutartigen oder zur Gruppe der bösartigen Elemente gehören. Dies geschieht allerdings nicht durch den Vergleich mit den körpereigenen Molekülen, sondern genau mit der Gegenmenge. Es werden im Körper zufällig Rezeptoren gebildet, die auf kein körpereigenes Proteinmuster reagieren. Sobald solch ein Rezeptor im Organismus mit einem Objekt reagiert, kann davon ausgegangen werden, dass es sich um ein körperfremdes Objekt handelt. Dieses gilt es zu bekämpfen.

Im künstlichen Immunsystem wurde diese Idee aufgegriffen und als Rezeptor ein Bitmuster einer bestimmten Länge verwendet. Da die Middleware alle ihre registrierten Knoten und Dienste inklusive deren möglichen Nachrichtentypen kennt, können die Rezeptoren nach einem strukturierten Prinzip generiert werden. Dabei entstehen nur Rezeptoren, die auf keine Nachricht reagieren, die zum bekannten Teil der Middleware gehört. Damit eine größere Anzahl an möglichen Bitmustern erreicht werden kann, werden die Rezeptoren immer für eine spezielle Position innerhalb des Nachrichtenkopfs erzeugt.

Im weiteren Verlauf der Arbeit wurde die optimale Rezeptorlänge bestimmt, um die beste Trefferrate zu erzielen bei gleichzeitig möglichst geringer Gesamtanzahl unterschiedlicher Rezeptoren. Durch Testläufe im Simulator hat sich auch gezeigt, dass diese optimale Länge der Rezeptoren nur von der Gesamtanzahl der unterschiedlichen Nachrichtentypen in der Middleware abhängig ist. Die Trefferrate hat sich dabei von der alleinigen Anzahl der Self-Nachrichten als unabhängig gezeigt. Allerdings ist das Verhältnis von verfügbaren Rezeptoren zur Menge der Self-Nachrichten für eine gute Erkennungswahrscheinlichkeit verantwortlich. Sobald drei- oder viermal so viele Rezeptoren wie Self-Nachrichten vorhanden sind, kann mit einer sehr guten Erkennung von nahezu 100% ausgegangen werden.

Zusätzlich wurden auch diverse Optimierungen vorgenommen, um den Speicherbedarf der Rezeptoren zu minimieren und die Ausführungszeit beim Vergleich von Nachrichten und Rezeptoren zu verkürzen. Dazu wurden Rezeptoren mit ähnlichen Bitmustern zusammengefasst und Vergleiche in einer Baumstruktur vorgenommen. Die Verwendung der positiven Selektion und der daraus resultierenden Prokörpern hat die Erkennungsrate unter bestimmten Umständen zusätzlich gesteigert. Der Einsatz von Permutationsmasken hat sich aber im Gegensatz zu Meinungen aus anderen Forschungsarbeiten nicht positiv auf die Minimierung der sogenannten „Löcher“ ausgewirkt.

9.1.2 Integration in die Middleware

Um die theoretisch gewonnenen Erkenntnisse des künstlichen Immunsystems auch praktisch anwenden zu können, wurde es in die Middleware OCμ integriert. Da die Middleware im Grunde nur aus Diensten und Monitoren auf einem Knoten besteht, wurde auch beim Selbstschutzsystem auf diese bekannte Architektur gesetzt. Leistungsstarke Knoten

werden dabei explizit vom Administrator als sogenannte Thymus-Knoten ausgezeichnet. Die Aufgabe dieser Knoten ist es, die entsprechenden Rezeptoren zu erzeugen und in der Middleware auf die restlichen Knoten zu verteilen. Bei leistungsstarken Desktop-PCs stellt die gesamte Architektur keine große Einschränkung dar. Aber gerade in einer ubiquitären Umgebung sind sehr oft auch leistungsschwache Knoten, wie beispielsweise mobile Endgeräte oder Sensoren, an der Kommunikation beteiligt. Bei der Integration wurden diese Knoten besonders berücksichtigt und es wurde gezeigt, wie durch eine geschickte Gruppierung und eine intelligente Kommunikation innerhalb dieser Gruppen selbst diese ressourcenarmen Knoten mit dem künstlichen Immunsystem geschützt werden können. Verschieden ausgeprägte Algorithmen berücksichtigen dazu die Entfernung (Latenzzeit) und Kapazitäten der einzelnen Knoten und gruppieren automatisch schwächere Knoten zusammen, damit diese in ihrer Summe die notwendige Kapazität erreichen. Diese Algorithmen kümmern sich auch um die korrekte Verteilung der Rezeptoren innerhalb der entstandenen Gruppen. Durch die Rotation der Rezeptoren oder der eintreffenden Nachrichten innerhalb einer solchen Gruppe können letztendlich die selben Erkennungswahrscheinlichkeiten erreicht werden, die sonst nur ein Knoten mit hoher Kapazität erreichen könnte.

Das hier vorgestellte Immunsystem kann seine Stärke erst in großen umfangreichen Netzwerken richtig entfalten. Bei einer geringen Anzahl an Knoten und Diensten ist sicher ein direkter Vergleich der eintreffenden Nachrichtentypen die bessere Herangehensweise. Wenn jedoch mehrere hundert oder gar tausend Dienste ihre Arbeit auf hunderten von Knoten erledigen, greift der positive Effekt der negativen Selektion und das Konzept der kurzen Rezeptoren im Gegensatz zu kompletten Vergleichen. Bei der hier vorgestellten Architektur wurde immer ein besonderes Augenmerk auf Effektivität bei gleichzeitig effizientem Ressourcenverbrauch gerichtet und in den anschließenden Simulationen wurden sehr gute Ergebnisse bei der Erkennung und der benötigten Rechen- und Speicherkapazität erzielt. Das Schutzsystem eignet sich daher insbesondere für den Einsatz in mobilen Endgeräten oder Sensoren, stellt aber auch bei leistungsstarken Knoten eine ressourcenschonende Möglichkeit dar, das System gegen unbekannte Bedrohungen erfolgreich zu schützen.

9.2 Ausblick

Das vorgestellte Schutzsystem nennt sich zwar „Selbstschutzsystem“, aber wie bereits in manchen Kapiteln angesprochen, ist das Vorhandensein eines Administrators derzeit noch unabdingbar. Dieser kümmert sich um die Autorisierung von Diensten und Knoten, um die korrekte Rollenvergabe und teilweise um die Parametrisierung des Immunsystems. Gerade beim Aspekt der Sicherheit und des Schutzes in unserer realen Welt spielt aber auch das gegenseitige Vertrauen eine sehr große Rolle. Möglicherweise lässt sich

dieses Verhaltensmuster aus der realen Welt ebenso in der digitalen Kommunikation von OCμ abbilden?

Durch das Einbeziehen von Vertrauensaspekten könnten mehrere Aufgaben im Selbstschutzsystem der Middleware automatisiert werden, um die notwendigen Eingriffe eines Administrators auf ein Minimum zu reduzieren. Dazu müssen jedoch erst Modelle und Parameter für ein solches Vertrauen zwischen Knoten und Diensten entwickelt werden. Andererseits könnte gerade ein Selbstschutzsystem auch selbst als Parameter für die Einstufung von Vertrauen dienen. Knoten, die bösartige Dienste ausführen, könnte man beispielsweise weniger Vertrauen entgegenbringen als anderen Knoten.

Vorstellbar wäre auch eine effizientere und dynamischere Anwendung des Immunsystems. Nachrichten von sehr vertrauenswürdigen Knoten müssen eventuell gar nicht durch das Sicherheitssystem abgedeckt und dadurch auch keiner Prüfung unterzogen werden. Dies würde die Verarbeitung oder Weiterleitung von Nachrichten auf Knoten mit wenigen Ressourcen beschleunigen. Andererseits könnten auch Nachrichten von Knoten, denen kein Vertrauen entgegengebracht wird, einfach abgeblockt werden. Generell stellt sich bei der Hinzunahme von Vertrauensaspekten die Frage, ob ein Schutzsystem dann überhaupt noch notwendig ist. Aber gerade bei Knoten, deren Vertrauensstatus unbekannt ist oder nicht exakt bestimmt werden kann, kann der Einsatz des vorgestellten Selbstschutzsystems Vorteile bringen.

Literaturverzeichnis

[AB05] Apel, Sven und Klemens Bohm: *Towards the Development of Ubiquitous Middleware Product Lines.* In: *ASE'04 SEM Workshop*, Band 3437 der Reihe *Lecture Notes in Computer Science*. Springer, 2005.

[And03] András Belokosztolszki and David M. Eyers and Peter R. Pietzuch and Jean Bacon and Ken Moody: *Role-Based Access Control for Publish/Subscribe Middleware Architectures.* In: *DEBS '03: Proceedings of the 2nd International Workshop on Distributed Event-Based Systems*, Seiten 1–8, New York, NY, USA, 2003. ACM.

[AT05] Andrews, Paul S. und Jon Timmis: *Inspiration for the Next Generation of Artificial Immune Systems.* In: *4th International Conference on Artificial Immune Systems (ICARIS 2005)*, Band 3627 der Reihe *Lecture Notes in Computer Science*, Seiten 126–138. Springer-Verlag, 2005.

[BEFG02] Balthrop, Justin, Fernando Esponda, Stephanie Forrest und Matthew Glickman: *Coverage and Generalization in an Artificial Immune System.* In: Langdon, William (Herausgeber): *Genetic and Evolutionary Computation Conference (GECCO 2006)*, New York, USA, July 2002.

[Ber04] Berbeco, Robert: *Exploration of Computer Immune Systems.* Technischer Bericht 1.4b, SANS Institute, GIAC Security Essentials Certification (GSEC), August 2004.

[BHSR04] Becker, Christian, Marcus Handte, Gregor Schiele und Kurt Rothermel: *PCOM - A Component System for Pervasive Computing.* In: *PERCOM'04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, Seite 67, Washington, DC, USA, 2004. IEEE Computer Society.

[BS04] Boudec, Jean-Yves Le und Slaviša Sarafijanović: *An Artificial Immune System Approach to Misbehavior Detection in Mobile Ad-hoc Networks.* In: *In Proceedings of Bio-ADIT – The First International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, Seiten 96–111, Lausanne, Schweiz, Januar 2004.

[BSGR03] Becker, Christian, Gregor Schiele, Holger Gubbels und Kurt Rothermel: *BASE - A Micro-Broker-Based Middleware for Pervasive Computing.* In: *PERCOM'03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, Seite 443, Washington, DC, USA, 2003. IEEE Computer Society.

[BSLG06] Ballesteros, Francisco J., Enrique Soriano, Katia Leal und Gorka Guardiola: *Plan B: An Operating System for Ubiquitous Computing Environments.* In: *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'06)*, Seiten 126–135, Washington, DC, USA, 2006. IEEE Computer Society.

[CL99] Castro, Miguel und Barbara Liskov: *Practical Byzantine Fault Tolerance.* In: *OSDI '99: Proceedings of the Third Symposium on Operating Systems Design and Implementation*, Seiten 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[EMS+02] Elkeelany, Omar S., Mustafa M. Matalgah, Khurram P. Sheikh, M. Thaker, Ghulam Chaudhry, Deepankar Medhi und Jihad Qaddour: *Performance Analysis of IPSec Protocol: Encryption and Authentication.* In: *IEEE International Conference on Communications (ICC 2002)*, Band 2, Seiten 1164–1168, New York, NY, USA, April 2002.

[FB07] Forrest, Stephanie und Catherine Beauchemin: *Computer Immunology.* Immunological Reviews, 216(1):176–197, 2007.

[GJMV+05] Goncharova, Larisa B., Yannick Jacques, Carlos Martin-Vide, Alexander O. Tarakanov und Jonathan I. Timmis: *Biomolecular Immune-Computer: Theoretical Basis and Experimental Simulator.* In: *4th International Conference on Artificial Immune Systems (ICARIS 2005)*, Band 3627 der Reihe *Lecture Notes in Computer Science*, Seiten 72–85. Springer-Verlag, 2005.

[HF00] Hofmeyr, Steven A. und Stephanie Forrest: *Architecture for an Artificial Immune System.* In: *Evolutionary Computation 8*, Nummer 4, Seiten 45–68, Massachusetts Institute of Technology, 2000.

[HF03] Hall, John M. und Deborah A. Frincke: *An Architecture for Intrusion Detection Modeled After the Human Immune System.* In: *Proceedings of the International Conference on Computer, Communication and Control Technologies*, Band 6, Seiten 75–78, Orlando, USA, 2003.

[HFP95] Hightower, Ron, Stephanie Forrest und Alan S. Perelson: *The Evolution of Emergent Organization in Immune System Gene Libraries.* In: Eshelman, Larry (Herausgeber): *Proceedings of the Sixth International*

Conference on Genetic Algorithms, Seiten 344–350, San Francisco, USA, 1995. Morgan Kaufmann.

[Hof99] Hofmeyr, Steven Andrew: *An Immunological Model of Distributed Detection and Its Application to Computer Security*. Doktorarbeit, University of New Mexico, Mai 1999.

[HT05] Hart, Emma und Jonathan Timmis: *Application Areas of AIS: The Past, The Present and The Future*. In: *4th International Conference on Artificial Immune Systems (ICARIS 2005*, Band 3627 der Reihe *Lecture Notes in Computer Science*, Seiten 483–497. Springer-Verlag, 2005.

[Inm78] Inman, J. K.: *The Antibody Combining Region: Speculations on the Hypothesis of General Multispecifity*. In: Bell, G. I., A. S. Perelson und Jr. G. H. Pimbley (Herausgeber): *Theoretical Immunology*, Seiten 243–278, New York, USA, 1978.

[JD05] Ji, Zhou und Dipankar Dasgupta: *Estimating the Detector Coverage in a Negative Selection Algorithm*. In: *Genetic and Evolutionary Computation Conference (GECCO 2005)*, Seiten 281–288, Washington DC, USA, Juni 2005. ACM.

[JTWS02] Janeway, Charles A., Paul Travers, Mark Walport und Mark Shlomchik: *Immunologie, 5. Auflage*. Spektrum Akademischer Verlag GmbH, Heidelberg, September 2002.

[KC03] Kephart, Jeffrey O. und David M. Chess: *The Vision of Autonomic Computing*. IEEE Computer Society, Seiten 41–50, Januar 2003.

[KN03] Kihlstrom, Kim Potter und Priya Narasimhan: *The Starfish System: Providing Intrusion Detection and Intrusion Tolerance for Middleware Systems*. In: *8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, Seiten 191 – 199, Guadalajara, Mexico, Januar 2003. IEEE Computer Society.

[KSC04] Kapadia, Apu, Geetanjali Sampemane und Roy H. Campbell: *KNOW Why Your Access Was Denied: Regulating Feedback for Usable Security*. In: *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, Seiten 52–61. ACM Press, 2004.

[Lab07] Labs, Bell: *Plan 9*. `http://plan9.bell-labs.com/plan9`, 2007.

[LL85] Lee, C. C. und D. T. Lee: *A Simple On-Line Bin-Packing Algorithm*. J. ACM, 32(3):562–572, 1985.

[MR04] Mirkovic, Jelena und Peter Reiher: *A Taxonomy of DDoS Attack and DDoS Defense Mechanisms*. SIGCOMM Computer Communication Review, 34(2):39–53, 2004.

[MS04] Müller-Schloer, Christian: *Organic Computing Initiative*. published as PDF, April 2004.

[OF99] Oprea, Mihaela und Stephanie Forrest: *How the Immune System Generates Diversity: Pathogen Space Coverage with Random and Evolved Antibody Libraries*. In: Banzhaf, Wolfgang, Jason M. Daida, A. E. Eiben, Max H. Garzon, Vasant Honavar, Mark J. Jakiela und Robert E. Smith (Herausgeber): *Genetic and Evolutionary Computation Conference (GECCO 2006)*, Orlando, USA, July 1999. Morgan Kaufmann.

[OW05] Oda, Terri und Tony White: *Immunity for Spam: An Analysis of an Artificial Immune System for Junk Email Detection*. In: Jacob, Christian, Marcin L. Pilat, Peter J. Bentley und Jonathan Timmis (Herausgeber): *Artificial Immune Systems: 4th International Conference*, Band 3627 der Reihe *Lecture Notes in Computer Science*, Seiten 276–289, Banff, Kanada, August 2005. Springer-Verlag.

[Pro07] Project, The Apache SpamAssassin, 2007. `http://spamassassin.apache.org/`.

[PSTU06a] Pietzowski, Andreas, Benjamin Satzger, Wolfgang Trumler und Theo Ungerer: *A Bio-Inspired Approach for Self-Protecting an Organic Middleware with Artificial Antibodies*. In: Meer, Hermann de und James P. G. Sterbenz (Herausgeber): *Self-Organising Systems, First International Workshop (IWSOS 2006)*, Band 1, Seiten 202–215, Passau, Deutschland, September 2006. Springer.

[PSTU06b] Pietzowski, Andreas, Benjamin Satzger, Wolfgang Trumler und Theo Ungerer: *A Practical Computer Immunology Approach for Self-Protection Enhanced by Optimization Techniques*. Journal of Autonomic and Trusted Computing (JoATC), 2006. unveröffentlicht.

[PSTU06c] Pietzowski, Andreas, Benjamin Satzger, Wolfgang Trumler und Theo Ungerer: *Using Positive and Negative Selection from Immunology for Detection of Anomalies in a Self-Protecting Middleware*. In: Hochberger, Christian und Rüdiger Liskowsky (Herausgeber): *Informatik 2006, Informatik für Menschen*, Band P-93, Seiten 161–168, Dresden, Deutschland, Oktober 2006. Gesellschaft für Informatik e.V., LNI.

[PTU06] Pietzowski, Andreas, Wolfgang Trumler und Theo Ungerer: *An Artificial Immune System and its Integration into an Organic Middleware for Self-Protection*. In: al., Maarten Keijzer et (Herausgeber): *Genetic and Evolutionary Computation Conference (GECCO 2006)*, Band 2, Seiten 129–130, Seattle, Washington, USA, Juli 2006. ACM, ACM Press.

[PvST02] Popescu, Bogdan, Maarten van Steen und Andrew S. Tanenbaum: *A Security Architecture for Object-Based Distributed Systems.* In: *18th Annual Computer Security Applications Conference*, Seiten 161–171. IEEE, Dec 2002.

[Rau04] Rau, Alexander: *Entwicklung eines Middleware-Teilsystems zur Unterstutzung nichtfunktionaler Parameter in ubiquitären Rechnersystemen.* Diplomarbeit, Universität Stuttgart, Institut für parallele und verteilte Systeme, Universitätsstraße 38, 70569 Stuttgart, 2004.

[RHC+02a] Roman, M., C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell und K. Nahrstedt: *A Middleware Infrastructure for Active Spaces.* Pervasive Computing, IEEE, 1(4):74–83, 2002.

[RHC+02b] Román, Manuel, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell und Klara Nahrstedt: *Gaia: A Middleware Platform for Active Spaces.* Mobile Computing and Communications Review, 6(4):65–67, 2002.

[Riv92] Rivest, Ronald: *The MD5 Message-Digest Algorithm.* Technischer Bericht Request for Comments: 1321, Internet Engineering Task Force (IETF), April 1992.

[SB06] Schütt, Christine und Barbara Bröker: *Grundwissen Immunologie.* Spektrum Akademischer Verlag GmbH, 2006.

[SBE04] Stibor, Thomas, Kpatscha M. Bayarou und Claudia Eckert: *An Investigation of R-Chunk Detector Generation an Higher Alphabets.* In: *Genetic and Evolutionary Computation Conference*, Seiten 299–307. Springer-Verlag, 2004.

[SG02] Sousa, João Pedro und David Garlan: *Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments.* In: *WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, Seiten 29–43, Deventer, Niederlande, 2002. Kluwer, B.V.

[SHF97] Somayaji, Anil, Steven Hofmeyr und Stefanie Forrest: *Principles of a Computer Immune System.* In: *New Security Paradigms Workshop*, Seiten 75–82, Cumbria, Großbritanien, 1997. ACM.

[SLBK05] Sidiroglou, Stelios, Michael E. Locasto, Stephen W. Boyd und Angelos D. Keromytis: *Building a Reactive Immune System for Software Services.* In: *USENIX Annual Technical Conference, General Track*, Seiten 149–161, Anaheim, CA, USA, 2005. USENIX.

[SMT05] Stibor, Thomas, Philipp Mohr und Jonathan Timmis: *Is Negative Selection Appropriate for Anomaly Detection?* In: *Genetic and Evolutionary*

Computation Conference (GECCO 2005), Seiten 321–328, Washington DC, USA, Juni 2005. ACM.

[SNC02] Sampemane, Geetanjali, Prasad Naldurg und Roy H. Campbell: *Access Control for Active Spaces.* In: *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*, Seite 343, Washington, DC, USA, 2002. IEEE Computer Society.

[STE05] Stibor, Thomas, Jonathan Timmis und Claudia Eckert: *A Comparative Study of Real-Valued Negative Selection to Statistical Anomaly Detection Techniques.* In: *4th International Conference on Artificial Immune Systems*, Seiten 262–275, 2005.

[TBPU04] Trumler, Wolfgang, Faruk Bagci, Jan Petzold und Theo Ungerer: *Towards an Organic Middleware for the Smart Doorplate Project.* In: Dadam, Peter und Manfred Reichert (Herausgeber): *GI Jahrestagung (2)*, Band 51 der Reihe *Lecture Notes in Informatics*, Seiten 626–630. GI, September 2004.

[TBPU05] Trumler, Wolfgang, Faruk Bagci, Jan Petzold und Theo Ungerer: *AMUN - Autonomic Middleware for Ubiquitous Environments Applied to the Smart Doorplate.* ELSEVIER Advanced Engineering Informatics, 19(3):243–252, 2005.

[TFR00] Tanchot, Corinne, Henrique Veiga Fernandes und Benedita Rocha: *The Organization of Mature T-cell Pools.* Philosophical Transactions of the Royal Society B: Biological Sciences, 355(1395):323–328., März 2000.

[TKU06] Trumler, Wolfgang, Robert Klaus und Theo Ungerer: *Self-configuration Via Cooperative Social Behavior.* In: Yang, Laurence Tianruo, Hai Jin, Jianhua Ma und Theo Ungerer (Herausgeber): *ATC*, Band 4158 der Reihe *Lecture Notes in Computer Science*, Seiten 90–99. Springer, 2006.

[TPBU06] Trumler, Wolfgang, Jan Petzold, Faruk Bagci und Theo Ungerer: *AMUN: An Autonomic Middleware for the Smart Doorplate Project.* Personal Ubiquitous Computing, 10(1):7–11, 2006.

[Tru06] Trumler, Wolfgang: *Organic Ubiquitous Middleware.* Doktorarbeit, Universität Augsburg, Juli 2006.

[TS03] Thayer, Scott M. und Peter Steenkiste: *An Architecture for the Integration of Physical and Informational Spaces.* Personal Ubiquitous Comput., 7(2):82–90, 2003.

[TTU06] Trumler, Wolfgang, Tobias Thiemann und Theo Ungerer: *An Artificial Hormone System for Self-organization of Networked Nodes.* In:

IFIP Conference on Biologically Inspired Cooperative Computing, Seiten 85–94, Santiago de Chile, August 2006. Springer-Verlag.

[Wik08] Wikipedia: *Immunsystem — Wikipedia, Die freie Enzyklopädie*. `http://de.wikipedia.org/w/index.php?title=Immunsystem&oldid=47221491`, 2008. [Online; Stand 17. Juni 2008].

[WR05] Wei, John und Chris Rowen: *Implementing Low-Power Configurable Processors — Practical Options and Tradeoffs*. In: *Design Automation Conference (DAC 2005)*, Seiten 706–711, Anaheim, CA, USA, Juni 2005. ACM.

[WV03] Wang, Yao und Julita Vassileva: *Trust and Reputation Model in Peer-to-Peer Networks*. In: *P2P'03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, Seite 150, Washington, DC, USA, 2003. IEEE Computer Society.

[ZCC00] Zwicky, Elizabeth D., Simon Cooper und D. Brent Chapman: *Building Internet Firewalls (2nd Edition)*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

Tabellenverzeichnis

Abbildungsverzeichnis

A Aufbau der Simulationsumgebungen

Für die Evaluierung der verschiedenen Ansätze und verwendeten Techniken in dieser Arbeit wurden unterschiedliche Simulationsumgebungen aufgebaut. Dabei wurden allerdings keine realen OCμ -Knoten zum Testen herangezogen. Der Nachteil an realen Knoten ist, dass jeder Knoten auf einem dedizierten Rechner gestartet werden muss oder dass jedem Knoten einen extra Port im Netzwerkprotokoll zugewiesen werden muss, wenn mehrere Knoten auf dem selben Rechner gestartet werden. Bei einer Simulation mit mehreren Knoten würde dies daher einen großen Administrationsaufwand bedeuten. Für Simulationen in OCμ wurde am Lehrstuhl der Simulator *OCSim* entwickelt. Dieser Simulator ermöglicht es, beliebig viele Knoten in einer grafischen Oberfläche (siehe Abbildung A.1) miteinander zu vernetzen und bestimmte Dienste auf den Knoten zu starten. Dabei läuft die komplette Kommunikation innerhalb einer Java-Virtual-Machine ab, wobei jeder Knoten und jeder Dienst als Thread realisiert sind.

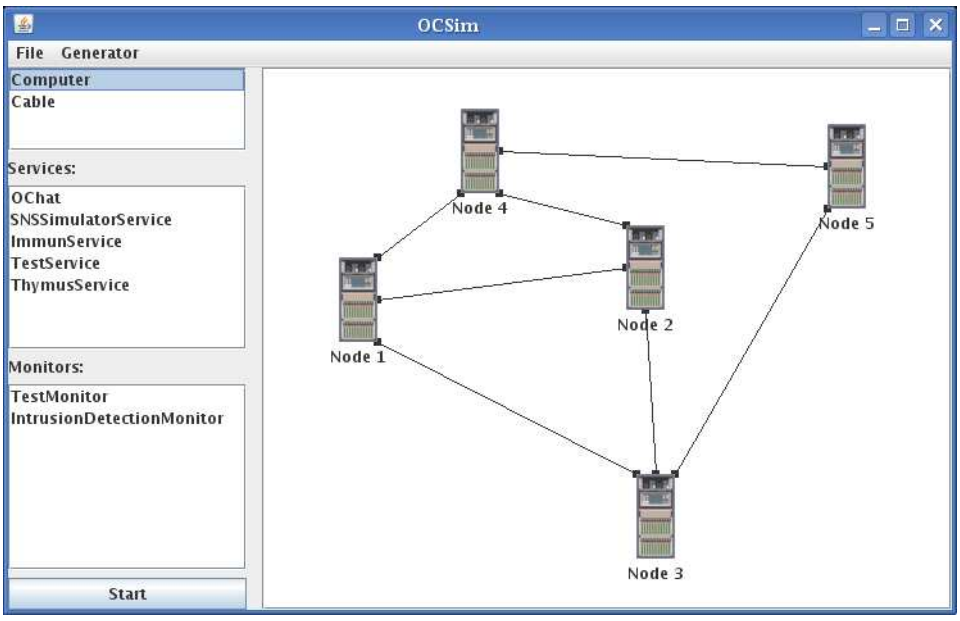


Abbildung A.1: Die Simulationsumgebung OCSim zum Entwurf von Netzwerklandschaften für die Evaluierungen der Middleware OCμ

Die API in OCSim wurde absichtlich – bis auf wenige Änderungen – komplett von OCμ übernommen. Dies hat den Vorteil, dass Monitore und Dienste aus OCμ ohne

große Änderungen direkt in den Simulator überführt werden können und umgekehrt. Für alle Simulationen wurde zusätzlich ein spezieller Simulator-Service entwickelt, der es ermöglicht, gezielt eine bestimmte Anzahl an Self- oder Non-Self-Nachrichten an andere Knoten zu versenden. Damit konnte die Erkennungsrate der Knoten sehr gut evaluiert werden.

A.1 Simulationsaufbau bei Tests für die Wahl des Normierungsverfahrens für Nachrichten

In Kapitel 6 wurde getestet, ob die reale Repräsentation einer Nachricht oder ihre gehashte Variante einen besseren Ausgangspunkt für die Architektur des Selbstschutzsystems darstellt. Hierfür wurde die interne Routine zum Erstellen des relevanten Bitmusters im Simulator in vier unterschiedlichen Versionen implementiert. Einmal eine Version, die die Nachrichten auf 128 Bits normierte und eine zweite, die sie auf die doppelte Länge, also auf 256 Bits normierte. Zudem wurde bei beiden Längen jeweils ein Hash-Verfahren und die reale Repräsentation des Strings implementiert.

Hash-Verfahren mit 128 Bits

Hierfür wurde die Nachricht lediglich mit dem MD5-Hash-Algorithmus zu einer 128-Bit-Repräsentation transformiert.

Hash-Verfahren mit 256 Bits

Um bei diesem Verfahren dennoch den schnellen MD5-Algorithmus verwenden zu können, wurde die Nachricht einmal in ihrer realen Repräsentation mit dem MD5-Algorithmus gehasht. Anschließend wurde die Nachricht in ihrer umgekehrten Darstellung erneut gehasht, was in der Regel in einem völlig anderen Hash-Wert resultiert. Die beiden Hash-Werte wurden schließlich zusammengefügt, um die 256-Bit-Repräsentation der Nachricht zu erhalten.

Reale Repräsentation mit 128 oder 256 Bits

Um die Nachrichten in ihrer originalen String-Repräsentation zu erhalten, aber dennoch eine uniformierte Länge der Nachrichten zu erreichen, mussten zwei Vorkehrungen getroffen werden:

1. Längere Nachrichten wurden ab der entsprechenden Bitposition einfach abgeschnitten.

2. Falls die Nachrichten unterhalb der gewünschten Bitlänge waren, wurden sie intern solange an sich selbst angehängt, bis die entsprechende Länge erreicht wurde.

A.2 Simulationsaufbau bei Tests für die optimalen Rezeptorattribute

Bei den Tests zur Bestimmung der optimalen Rezeptorlänge und der zu verwendenden Offsets wurde eine einfache Simulationsumgebung mit nur zwei verbundenen Knoten aufgebaut. Der erste Knoten diente dabei der Generierung der Rezeptoren, der zweite zur Ermittlung der Erkennungsrate.

Bei der Generierung kann in einer Oberfläche festgelegt werden, welche Rezeptorlänge und welche Offsets verwendet werden sollen und welche Anzahl von Antikörpern und Prokörpern erzeugt werden soll. Ferner können auch gezielt Rezeptoren kombiniert und miteinander verschmolzen werden, um Speicherplatz einzusparen. Diese diversen Parameter ermöglichten es, die unterschiedlichsten Rezeptoren, Antikörper und Sets von Prokörpern zu erstellen und auf ihre Wirksamkeit hin zu untersuchen.

Um die Trefferwahrscheinlichkeit zu ermitteln, wurde von einem anderen Knoten aus eine bestimmte Anzahl von Self- und Non-Self-Nachrichten generiert und wiederum an den ersten Knoten gesendet. Am ersten Knoten konnte dadurch die Erkennungsrate mit dem aktuellen Rezeptor-Repertoire in Erfahrung gebracht werden.

A.3 Simulationsaufbau bei Tests für die Gruppierung von Knoten innerhalb einer Thymus-Gruppe

Für die Gruppierung wurde auf die Verwendung des Simulators OCSim verzichtet. Gerade beim Gruppieren von Knoten ist es wichtig, das Verhalten bei mehreren hundert Knoten zu analysieren. Nur so kann erkannt werden, wie gut das System skaliert und ob die erzeugten Gruppen die gewünschten Eigenschaften aufweisen. In OCSim wäre ein solcher Testaufbau zwar möglich, das Erzeugen des Testaufbaus ist aber mit großen Mühen verbunden, da der Simulator grafisch bedient wird. Aus diesem Grund wurde ein separater Simulator programmiert, in dem ausschließlich die Gruppierung programmiert wurde. Damit können problemlos mehrere hundert oder tausend Knoten generiert werden, die anschließend nach verschiedenen Algorithmen gruppiert werden. Für den Gruppierungsalgorithmus aus Kapitel 7.5.2 werden folgende Attribute der Knoten benötigt:

- **Kapazität eines Knotens**
  Die Kapazität eines Knotens kann im Simulator als ganzzahliger Bereich angegeben werden und repräsentiert die Anzahl der Rezeptoren, die der Knoten aufnehmen kann. Der angegebene Bereich gilt für alle Knoten gleichermaßen. Beim Generieren der Knoten weist der Simulator jedem Knoten einen zufälligen Wert aus diesem Bereich zu.
- **Latenzzeit zwischen den Knoten**
  Die Latenzzeit zwischen den Knoten erfolgt ebenso nach dem Zufallsprinzip. Dazu wird intern eine zweidimensionale Matrix aufgespannt. Die Werte in der Matrix entsprechen den Latenzzeiten zwischen den Knoten im Netzwerk und werden zufällig aus dem Wertebereich von 1 bis 100 ausgewählt. Dabei entspricht 100 der maximalen und 1 der minimalen Latenzzeit, was sich anschließend sehr gut als Prozentwerte abbilden lässt.
- **Erforderliche Gesamtkapazität einer Gruppe**
  Jeder Thymus-Knoten hat eine bestimmte Anzahl von Rezeptoren, die er seinen registrierten Knoten zur Verfügung stellen möchte. Diese Anzahl kann vor der Gruppierung im Simulator eingegeben werden.

Durch diese drei Parameter können die unterschiedlichen Testsysteme im Simulator generiert werden und die verschiedenen Algorithmen zum Gruppieren von Knoten getestet werden. Dabei wurde sowohl der Algorithmus implementiert, der die Gruppierung nur anhand der Kapazitätseigenschaften der Knoten vorgenommen hat, als auch der optimierte Algorithmus, der zusätzlich die Latenzzeiten zwischen den Knoten berücksichtigt. Um die beiden Algorithmen und ihr Verhalten besser vergleichen zu können, wurde jeder Testlauf jeweils mit dem selben Ausgangssystem einmal mit der einfachen und anschließend mit der optimierten Variante durchgeführt. Zu beachten ist, dass in diesem Simulator der Thymus-Knoten nicht als Instanz vertreten ist sondern nur virtuell existiert. Alle generierten Knoten sind implizit bei diesem Thymus-Knoten registriert. Der Simulator kann daher auch nur eine einzige Thymus-Gruppe emulieren, was für die Art der Testläufe jedoch ausreichend ist.

B Eigene Veröffentlichungen

2005

Petzold, Jan, Andreas Pietzowski, Faruk Bagci, Wolfgang Trumler und Theo Ungerer: *Prediction of Indoor Movements Using Bayesian Networks.* In: Strang, Thomas und Claudia Linnhoff-Popien (Herausgeber): *LoCA*, Band 3479 der Reihe *Lecture Notes in Computer Science*, Seiten 211–222. Springer, 2005.

2006

Pietzowski, Andreas, Benjamin Satzger, Wolfgang Trumler und Theo Ungerer: *A Bio-Inspired Approach for Self-Protecting an Organic Middleware with Artificial Antibodies.* In: Meer, Hermann de und James P. G. Sterbenz (Herausgeber): *Self-Organising Systems, First International Workshop (IWSOS 2006)*, Band 1, Seiten 202–215, Passau, Deutschland, September 2006. Springer.

Pietzowski, Andreas, Benjamin Satzger, Wolfgang Trumler und Theo Ungerer: *A Practical Computer Immunology Approach for Self-Protection Enhanced by Optimization Techniques.* Journal of Autonomic and Trusted Computing (JoATC), 2006. unveröffentlicht.

Pietzowski, Andreas, Benjamin Satzger, Wolfgang Trumler und Theo Ungerer: *Using Positive and Negative Selection from Immunology for Detection of Anomalies in a Self-Protecting Middleware.* In: Hochberger, Christian und Rüdiger Liskowsky (Herausgeber): *Informatik 2006, Informatik für Menschen*, Band P-93, Seiten 161–168, Dresden, Deutschland, Oktober 2006. Gesellschaft für Informatik e.V., LNI.

Pietzowski, Andreas, Wolfgang Trumler und Theo Ungerer: *An Artificial Immune System and its Integration into an Organic Middleware for Self-Protection.* In: al., Maarten Keijzer et (Herausgeber): *Genetic and Evolutionary Computation Conference (GECCO 2006)*, Band 2, Seiten 129–130, Seattle, Washington, USA, Juli 2006. ACM, ACM Press.

2007

Satzger, Benjamin, Andreas Pietzowski, Wolfgang Trumler und Theo Ungerer: *A New Adaptive Accrual Failure Detector for Dependable Distributed Systems.* In: *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, Seiten

551–555, New York, NY, USA, 2007. ACM.

Satzger, Benjamin, Andreas Pietzowski, Wolfgang Trumler und Theo Ungerer: *Variations and Evaluations of an Adaptive Accrual Failure Detector to Enable Self-healing Properties in Distributed Systems.* In: *ARCS*, Seiten 171–184, 2007.

Trumler, Wolfgang, Jörg Ehrig, Andreas Pietzowski, Benjamin Satzger und Theo Ungerer: *A Distributed Self-healing Data Store.* In: Xiao, Bin, Laurence Tianruo Yang, Jianhua Ma, Christian Müller-Schloer und Yu Hua (Herausgeber): *ATC*, Band 4610 der Reihe *Lecture Notes in Computer Science*, Seiten 458–467. Springer, 2007.

Trumler, Wolfgang, Markus Helbig, Andreas Pietzowski, Benjamin Satzger und Theo Ungerer: *Self-configuration and Self-healing in AUTOSAR.* In: *14th Asia Pacific Automotive Engineering Conference (APAC-14)*, Hollywood, CA, USA, 2007.

Trumler, Wolfgang, Andreas Pietzowski, Benjamin Satzger und Theo Ungerer: *Adaptive Self-optimization in Distributed Dynamic Environments.* In: *SASO*, Seiten 320–323. IEEE Computer Society, 2007.

2008

Satzger, Benjamin, Andreas Pietzowski, Wolfgang Trumler und Theo Ungerer: *A Lazy Monitoring Approach for Heartbeat-Style Failure Detectors.* In: *ARES*, Seiten 404–409. IEEE Computer Society, 2008.