

Bachelorarbeit

**Simulative Analyse von Cyber-Angriffen am
Beispiel des Netzes der Fakultät für Informatik
der TU Dortmund**

Nils Dunker
11. Dezember 2020

Gutachter:
Prof. Dr. Peter Buchholz
Gerd Sokolies

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für praktische Informatik (LS-4)
<http://ls4-www.cs.tu-dortmund.de>

Inhaltsverzeichnis

1 Einleitung	1
1.1 Lösungen	1
1.2 Herangehensweise	2
1.3 Einteilung	2
2 Programme	3
2.1 OMNeT++ und INET	3
2.1.1 Die NED-Sprache	3
2.1.2 Wichtige INET-Module	5
2.1.3 Ini-Datei	8
2.1.4 Analysemöglichkeiten	10
2.2 SEA++	11
2.2.1 Installation	11
2.2.2 Programmkomponenten	11
2.2.3 ASL	12
2.2.4 Erweiterung von SEA++	15
2.2.5 Zusammenfassung des Arbeitsablaufs	15
3 Methodik	16
3.1 Angriffsszenarien	16
3.1.1 Angriffsvariante: ARP-Poisoning	16
3.1.2 Angriffsvariante: Denial of Service	17
3.1.3 Angriffsvariante: Port-Scanner	19
3.2 Das Netzwerk der Fakultät für Informatik	20
3.2.1 Vereinfachter Aufbau	20
3.2.2 Netzwerkverkehr	21
3.2.3 Auslastung	22
4 Aufbau der Simulation	23
4.1 Umsetzung der Netzwerktopologie	23
4.1.1 Innennetz	24
4.1.2 Außennetz	25
4.2 Konfiguration des Netzwerks (ini)	26
4.2.1 Allgemeine Konfigurationen	26
4.2.2 Konfiguration der einfachen Netzwerkmodule	26
4.2.3 Konfigurationen des Internet-Cloud-Moduls	27
4.2.4 Konfigurationen der Anwendungsmodule	28
4.2.5 Angriffskonfigurationen	31

4.3	Netzwerkmodelleinschränkungen	31
4.4	Umsetzung des ARP-Positioning-Angriffs	32
4.4.1	Implementierung des Angriffs	33
4.5	Umsetzung des DoS-Angriffs	34
4.6	Umsetzung des Port-Scanners	35
4.6.1	Fehlerbehebung	36
5	Auswertung	37
5.1	Ergebnisse	37
5.1.1	ARP-Poisoning	37
5.1.2	DoS-Angriff	40
5.1.3	Port-Scanner	43
5.2	Bewertung der Arbeit mit SEA++	45
5.2.1	Zufallszahlen	45
5.2.2	Dokumentation	45
5.2.3	Analyse	46
5.2.4	Einschränkungen durch veraltete Programmkomponenten	46
5.2.5	Flexibilität	47
5.2.6	Modularität	47
5.2.7	Erweiterbarkeit	47
5.2.8	Einschränkungen durch fehlende Module	48
5.2.9	Sprachumfang	48
6	Zusammenfassung	49
A	Ping-Test	52
B	OMNeT++-Auslastung	53
C	OMNeT++-Zufallsverteilung	54
C.1	Konfiguration	54
C.2	Ergebnis	54
D	ASL-Message-Typen	55
D.1	Anwendungsschicht	55
D.2	Transportschicht	55
D.3	Vermittlungsschicht	55
D.4	Sicherungsschicht	55
E	Auswertungen	56
E.1	ARP	56
E.1.1	Summe der Bits im studentischen Bereich	56
E.1.2	Angefangene ARP-Auslösungen	56
E.2	DoS	57
Literatur		59

Abbildungsverzeichnis

2.1	Beispieltopologie	3
2.2	Wichtige Module	6
2.3	Networklayer und EthernetInterface	7
2.4	SEA-Gates	14
3.1	Der ARP-Header nach RFC 826 [20]	17
3.2	TCP-Header nach RFC 739 [21] mit Erweiterung durch RFC 3168 [23]	18
3.3	Ablauf eines normalen TCP-Handshakes und eines SYN-Flooding-Angriffs	18
3.4	Vorgesehene Antwort auf ein unerwartetes TCP-Segment mit gesetztem ACK-Bit RFC 739 [21]	19
3.5	Vereinfachtes Informatiknetzwerk	21
4.1	Modelltopologie	23
5.1	Vollständige ARP-Auflösungen	37
5.2	Übertragene Bits des ARP-Angriffs	38
5.3	IRB-Server-Datenverkehr mit ARP-Angriff	39
5.4	Gesamter Datenverkehr am Router	39
5.5	Durchschnittliche Datenrate pro Sekunde am IRB-Router	40
5.6	IRB-Router Verbindungs auslastung	41
5.7	Summe der übertragenen Bits der jeweiligen Teilnetze	41
5.8	Pingverlust mit DoS-Angriff	42
5.9	RTT zu den Internetservern	43
5.10	Port-Scanner Paketaufkommen	43
5.11	Durchschnittliche Datenrate am IRB-Router	44

Quellcodeverzeichnis

2.1	Beispiel NED-Definition	4
2.2	Beispielkonfiguration	9
4.1	Kabelftypen	25
4.2	Änderungen an der Create.cc	32
4.3	Änderungen an der Change.cc	33
4.4	ARP-Angriff Definition	34
4.5	DoS Definition	35
4.6	Port-Scanner Definition	36
A.1	Ping-Testergebnisse	52
C.1	Erweiterte Durchläufe	54

Tabellenverzeichnis

2.1	Beschreibung wichtiger Module	6
2.2	Kurzbeschreibung Anwendungen	8
2.3	Schichtenzuordnung	13
4.1	Modifizierte Parameter mit zugewiesenen Wert der Netzwerkmodule	26
4.2	Anzahl an Anwendungsmodulen je Host oder Server	28
4.3	Übersicht der Parameter von den TCP-Anwendungen	28
4.4	Übersicht der Parameter von den UDP-Anwendungen	30
4.5	Übersicht der Parameter von den Videostream-Anwendungen	30
4.6	Übersicht der Parameter von den Ping-Anwendungen	31

Kapitel 1

Einleitung

Im Frühjahr des Jahres 2020 musste wegen eines Cyber-Angriffs auf die Ruhr-Universität-Bochum [8] ein Großteil der internen IT-Infrastruktur von dem Netz genommen werden. Dies ist nur ein Beispiel für eine Vielzahl von Angriffen auf die deutsche IT-Infrastruktur. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) sammelte im Zeitraum des Lageberichts 2019 bis zu 11,5 Millionen Schadprogramminfektionsmeldungen von deutschen Netzbetreibern. Es wurden weiterhin tägliche ca. 110.000 Botinfektion auf deutsche Netze gemeldet und 770.000 Mails an das Regierungsnetz mit Schadprogrammen abgefangen [4, Seite 34-35].

Bei diesen Zahlen handelt es sich nur um die erkannten oder vereiteten Angriffe auf Netzwerke. Es ist zu vermuten, dass es eine hohe Dunkelziffer [3] an unerkannten Angriffen gibt. Einige großangelegte staatliche Angriffe, gegen Ziele wie Atomkraftwerke, sind jahrelang nicht bemerkt [15, Kapitel 9] worden. Dies zeigt ein großes Bedrohungspotential von nicht verhinderbaren Cyber-Angriffen und stellt die Frage nach den Auswirkungen dieser Angriffe.

1.1 Lösungen

Um diese Frage zu beantworten, müssen Cyber-Angriffe untersucht werden, wofür es verschiedene Herangehensweisen gibt. Reale Netzwerke können angegriffen werden und dabei analysiert werden, um etwaige Folgen weiterer Angriffe zu minimieren. Weiterhin sind Netzwerksimulationen möglich, um das gleiche Ziel ohne Nutzung eines realen Netzwerks zu realisieren.

Simulative Ansätze haben die Abstrahierung vom realen Fall als Vorteil, wodurch Modellanpassungen die Untersuchung einer Vielzahl von Szenarien erlauben. Ein Nachteil ist die Aussagekraft, welche von der Modellierung abhängt. In dieser Arbeit soll der simulierte Ansatz gewählt werden. Für die Simulation von Netzwerken gibt es eine Vielzahl von möglichen Programmen [19], wie NS2, NS3, OMNeT++, SSFNet und J-Sim. In dieser Bachelorarbeit wird der Event-Simulator OMNeT++ genutzt. Auf OMNeT++ bauen eine Vielzahl von Erweiterungen¹ auf.

Für die Analyse von Auswirkungen einer Cyber-Attacke wurde die Erweiterung SEA++ entwickelt, welche auf INET aufbaut. SEA++ wurde in „SEA++: A Framework for Evaluating the Impact of Security Attacks in OMNeT++/INET“ als flexible und nutzerfreundliche Möglichkeit Cyber-Angriffe zu analysieren, vorgestellt [30, Seite 276]. Zuerst besprochen

¹Eine Übersicht von Erweiterungen kann auf der Projektseite von OMNeT++ gefunden werden.

wurde SEA++ im Jahr 2014 in einer wissenschaftlichen Arbeit [29]. Für SEA++ existiert zusätzlich ein Benutzerhandbuch [22]. SEA++ ist nicht die einzige Erweiterung, welche Sicherheitsthemen im OMNeT++ Ökosystem behandelt. Ein Beispiel für eine alternative Erweiterung ist NETwork Attacks (NETA) [17].

1.2 Herangehensweise

In dieser Arbeit sollen SEA++ und OMNeT++ genutzt werden, um verschiedene Cyberangriffe auf ein Universitätsnetzwerk auszuführen. Der Fokus liegt dabei auf der Arbeit mit SEA++, welche am Ende besprochen wird. Dazu werden die einzelnen Aspekte von SEA++ vorgestellt und das Programmökosystem betrachtet.

Die Wahl der Cyberangriffe spiegelt typische Teileaspekte eines größeren Cyberangriffs wieder. Es soll ein Denial-of-Service-Angriff (DoS-Angriff) untersucht werden, welcher zu den häufigsten Angriffsarten im Internet gehört. Für die Betrachtung eines Angriffs, welcher von einem Gerät innerhalb eines Netzwerks ausgeht, wird eine Variante des ARP-Poisonings implementiert. Auf ARP-Poisoning bauen viele andere unterschiedliche Angriffe auf, wie z. B. ein Man-in-the-Middle-Angriff (MitM). Zuletzt wird ein Port-Scanner betrachtet, welcher unabdingbar ist, um Informationen über ein Netzwerk zu erlangen.

Aufgrund der vermuteten unterschiedlichen Auswirkungen eignen sich die Angriffe zusätzlich für die Betrachtung von SEA++. Es wird vermutet, dass die Auswirkungen von einem ARP-Poisoning und dem DoS-Angriff deutlich im Netzwerk zu sehen sind, der Port-Scanner hingegen eher unauffällig ist.

1.3 Einteilung

Diese Arbeit ist in sechs Teile gegliedert. Nach der Einleitung folgt eine Vorstellung des Programmökosystems 2, mit dem gearbeitet wird. Vor allem die Syntax und Bedienung von OMNeT++ und SEA++ sollen dabei anhand einfacher Beispiele ergründet werden. In Teil 3 folgt eine Vorstellung der Angriffe und eine Übersicht über das reale Netzwerk der Informatikfakultät an der Technische Universität Dortmund (TU). Teil 4 stellt die Modellierung des realen Netzwerks in OMNeT++ vor und zeigt die Implementierung der Angriffe mit SEA++. Insbesondere wird auf die Unterschiede zwischen Modell und Realität eingegangen. Im vorletzten Teil 5 werden die Ergebnisse der Angriffe besprochen sowie die Probleme und Vorteile von SEA++ aufgezeigt. In der Zusammenfassung 6 werden die gewonnenen Ergebnisse zusammengefasst. Rohdaten, wie Grafiken und Dateien, welche in den einzelnen Kapiteln nicht vollständig dargestellt werden können, befinden sich im Anhang A bis E.

Kapitel 2

Programme

Im Mittelpunkt dieser Bachelorarbeit steht SEA++, welches auf OMNeT++ mit INET aufbaut. Im Folgenden werden diese zentralen Programme vorgestellt. Zuerst wird auf OMNeT++ erweitert mit INET als Kernkomponente eingegangen. Danach wird SEA++ vorgestellt, welches Angriffssimulationen auf OMNeT++-Topologien ermöglicht.

2.1 OMNeT++ und INET

OMNeT++ ist eine in C++ geschriebene Event-Simulationsplattform. Zusätzlich zu dem eigentlichen Simulationskern bietet OMNeT++ eine Eclipse-Integrierte Entwicklungsumgebung (IDE). Topologien werden in einer domänenpezifische Sprache (DSL), NED genannt, definiert. [12]

Um Computernetzwerke darzustellen wird OMNeT++ mit INET erweitert. Die Erweiterung INET bietet viele Implementierungen von Netzwerkprotokollen und -geräten an, wie z. B. einen Switch oder Router.

2.1.1 Die NED-Sprache

Zuerst soll auf die NED-Sprache anhand eines Beispiels eingegangen werden. Die Beispieltopologie 1 besteht aus drei Akteuren Alice, Bob und Eve, welche über einen Router kommunizieren können. Eine grafische Darstellung findet sich in der Abbildung 2.1, welche nur Netzwerkmodule darstellt.

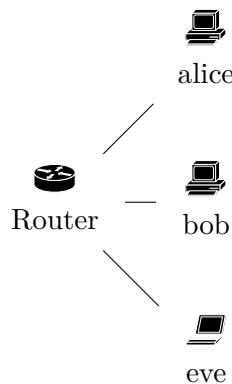


Abbildung 2.1: Beispieltopologie definiert in 1

Beispielimplementierung in der NED-Sprache

Quellcode 1 Auszug aus dem NED Beispiel für die in Abbildung 2.1 dargestellte Topologie.

```

1 network Example
2 {
3   parameters:
4     string attackConfigurationFile = default("none");
5   submodules:
6     configurator: IPv4NetworkConfigurator;
7     globalFilter: GlobalFilter;
8     alice: StandardHost;
9     bob: StandardHost;
10    eve: StandardHost;
11    router: Router;
12    exmachina: ExMachina;
13
14   connections allowunconnected:
15     router.ethg++ <--> Eth1G <--> alice.ethg++;
16     router.ethg++ <--> Eth1G <--> bob.ethg++;
17     router.ethg++ <--> Eth1G <--> eve.ethg++;
18
19     globalFilter.nodes++ <--> alice.global_filter;
20     globalFilter.nodes++ <--> bob.global_filter;
21     globalFilter.nodes++ <--> eve.global_filter;
22     globalFilter.nodes++ <--> router.global_filter;
23 }
```

Um den Aufbau von Topologien in OMNeT++ nachzuvollziehen, sind einige OMNeT++-spezifische Begriffe zu klären. OMNeT++-Simulationen werden aus Knoten (Nodes) und Kanälen (Channel) aufgebaut. Knoten sind im ersten Beispiel 1 die Elemente **alice**, **bob**, **eve**, **router**. Die Verbindungen zwischen den Knoten repräsentieren die Kanäle.

Knoten werden intern als Module umgesetzt. Module können weiter unterschieden werden in zusammengesetzte (compound modules) und einfache Module (simple modules).

Ein einfaches Modul stellt durch eine Implementierung in C++ grundlegende Funktionalitäten zur Verfügung. Es ist für das Aufzeichnen von Statistiken verantwortlich. Komplexere Strukturen, wie zum Beispiel ein Switch, können dann durch einen Zusammenschluss von mehreren einfachen Modulen ohne C++-Code realisiert werden [13]. Zusätzlich zu den Modulen existieren Interfaces, welche von Modulen realisiert werden können [11, Absatz 3.1].

In OMNeT++ wird intern jedem Modul eine **id** zugewiesen, welche für die Zeit der Simulation garantiert einzigartig ist [11, Absatz 4.11]. In dieser Arbeit werden keine neuen Module erstellt, sondern nur vorhandene konfiguriert und in einer Topologie implementiert.

In dem gekürzten Quellcode¹ in Zeile 1 wird ein neues Netzwerk mit dem Namen **Example** erstellt. Ein Netzwerk ist das Modell, welches simuliert werden soll und ist selbst ein Modul [11, Absatz 2.1].

Ein OMNeT++-Modul kann die Abschnitte **types**, **parameters**, **gates**, **submodules** und **connections** beinhalten [11, Absatz 3.4], wobei das Beispielnetzwerk drei dieser Abschnitte nutzt. In dem Parameterabschnitt können Variablen definiert

¹Die Beispieldiagramme lässt einige Details aus. In den ausgelassenen Zeilen wird ähnlich zu Java ein **package** festgelegt und vorgefertigte Modulen geladen. Wenn diese Zeilen inkludiert werden, wird das Beispiel zu lang für die Darstellung auf dieser Seite.

und zugewiesen werden. In diesem Beispiel ist es ein `string attackConfigurationFile`² mit einem leeren Wert `default`.

Unter `submodules` werden die genutzten Module definiert [11, Absatz 3.4]. In Beispielquellcode sind u. a. `alice`, `bob`, `eve` Module des Typs `StandardHost`, welche mit einer späteren Konfiguration Dienste in dem Netzwerk bereitstellen.

Zuletzt nutzt das Beispiel den `connections` Abschnitt, in welchen Verbindungen zwischen den Modulen, mithilfe von Kanälen, hergestellt werden [11, Absatz 3.4].

In dem Quellcode wird z. B. eine Verbindung zwischen dem `router` und den Rechnern `alice`, `bob`, `eve` über ein `Eth1G` Kanal hergestellt. Dieser Kanal ist ein Modell eines 1-Gigabit-Ethernetkabels. Die Option `allowunconnected`, teilt OMNeT++ mit, dass nicht alle Gates verbunden sein müssen. Ein Gate ist dabei, der Kommunikationsendpunkt zwischen Modulen.

Weitere NED-Details

Die nicht genutzten Abschnitte `type` und `gates` werden vor allem für Module in unteren Hierarchieebenen wichtig.

In dem `type`-Abschnitt werden lokale Module und Kanäle definiert und im `gates`-Abschnitt Ports, die in OMNeT++ Gates genannt werden [11, Absatz 3.4]. Diese erlauben die Kommunikation zwischen den einzelnen Modulen. In dem Beispielquellcode besitzt das Modul `StandardHosts` zwei Gates mit den Namen `ethg` und `global_filter`. Der `globalFilter` besitzt ein Gate mit dem Namen `nodes`.

Für die Gates wird ein weiteres, aus Programmiersprachen bekanntes Feature verwendet. In NED ist es möglich Vektoren (Arrays) zu definieren, um z. B. eine über ein Parameter skalierbare Anzahl an Modulen eines spezifischen Typs zu erzeugen oder mehrere Schnittstellen zu anderen Modulen über Gates anzubieten. In dem Quellcode hätten die drei `StandardHost` durch `host[anzahl] : StandardHost` dargestellt werden können, wobei `anzahl` einen Integerwert entspricht. Für die Vektoren unterstützt NED Schleifen. Bei den Gates wird durch die Notation `++` ein nicht genutztes Feld des Vektors `ethg` referenziert. [11, Absatz 3] Diese Sprachfunktionalität wird im Beispiel genutzt, um den Router mit den Hosts zu verbinden, ohne konkret die Felder `ethg[0]`, `ethg[1]` und `ethg[2]` referenzieren zu müssen.

2.1.2 Wichtige INET-Module

Im folgenden werden alle für diese Arbeit genutzten Module vorgestellt. Bereits bekannt sind die Module `StandardHost` 2.2b und `Router` 2.2c. Zusätzlich sollen die Module `EtherSwitch` 2.2a und `InternetCloud` 2.2d betrachtet werden.

Für die Angriffe auf ein Netzwerk, wird die Erweiterung `SEA++` 2.2 benutzt. Diese Erweiterung benötigt das Modul `LocalFilter` in jedem Modul mit welchem `SEA++` zur Laufzeit interagiert. Da `SEA++` keine Integration in die Eclipse-IDE bietet, wurde dieses Modul in Abbildung 2.2 nicht dargestellt.

Die Funktionen der wichtigsten einfachen Module sind in der Tabelle 2.1 beschrieben. Zusammengesetzte Module wie z. B. `L2NodeConfigurator` oder `InterfaceTable` wurden ausgelassen, da diese beim späteren Konfigurationsprozess nicht benutzt werden. Die Implementierungsdetails sind jeweils aus der Dokumentation in den `.ned`-Dateien entnommen, da diese den Entwicklungsstand für die jeweilige INET-Version beschreiben. Zusätzlich

²Der String `attackConfigurationFile` enthält die Angriffsbeschreibung für `SEA++`.

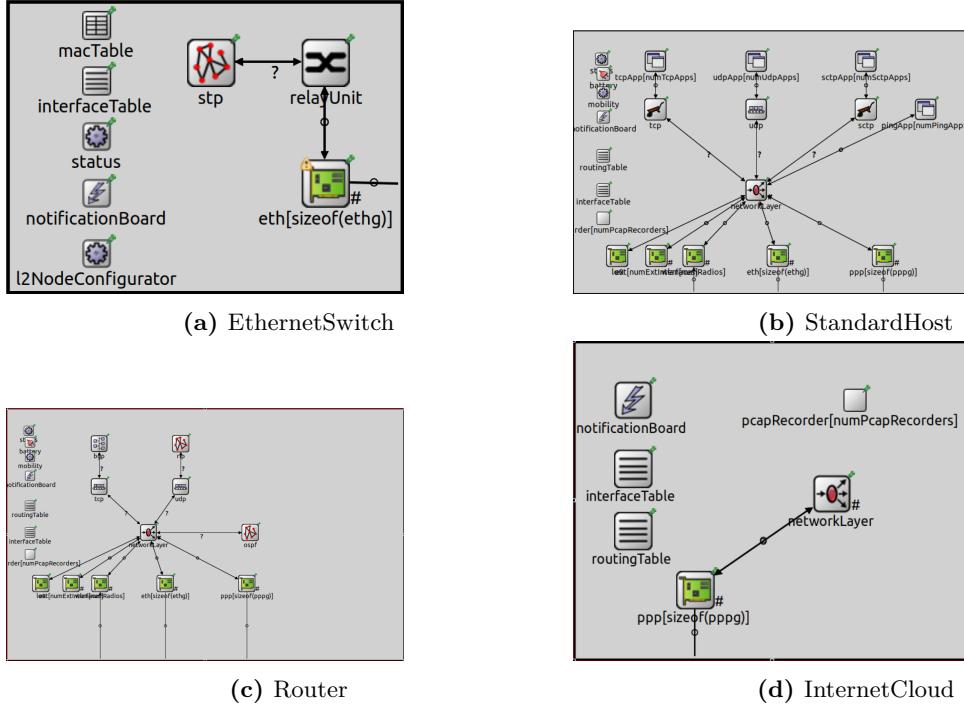


Abbildung 2.2: OMNeT++-Ansicht der genutzten Module

Modul	Beschreibung
IPv4NetworkConfigurator	Konfiguriert IP-Adressen und statisches Routing im IPv4-Netzwerk
EtherMAC	Setzt Sicherungsschicht Funktionen um. Unterstützt IEEE 802.3.
ThrputMeter	Zeichnet Anzahl und Durchsatz für ein- und ausgehende Datenpakete auf.
ARP, TCP, UDP und weitere	Implementierungen der jeweiligen Protokolle

Tabelle 2.1: Kurzbeschreibung wichtiger einfacher Module

finden sich in dieser Datei die Statistiken und Signale, welche von den einfachen Modulen aufgezeichnet werden sowie die Analyse des Modells erlauben.

Zusätzlich zu den einfachen Modulen werden die zusammengesetzten Module **NetworkLayer** 2.3a und **EtherNic** 2.3b von den vorgestellten Netzwerkgeräten verwendet.

EthernetInterface

Um ein Ethernet Network Interface Card (NIC) zu realisieren, ist das zusammengesetzte Modul **EthernetInterface** (2.3b) in allen betrachteten Modulen, außer der **InternetCloud**, welche ein ähnliches Modul für Point-to-Point Protocol (PPP)-Verbindungen verwendet, vorhanden. Es besteht aus Implementierungen der Modulinterfaces **IHook**, **IEtherMAC**, **ITrafficConditioner** und **IWiredNic** sowie einem zusammengesetzten Modul **EtherQoSQueue**.

Das **EthernetInterface** Modul ist dabei wie folgt konfiguriert. Es verwendet das Modul **EtherMAC**, als Implementierung von **IEtherMAC**. Das Interface **IHook** wird von dem

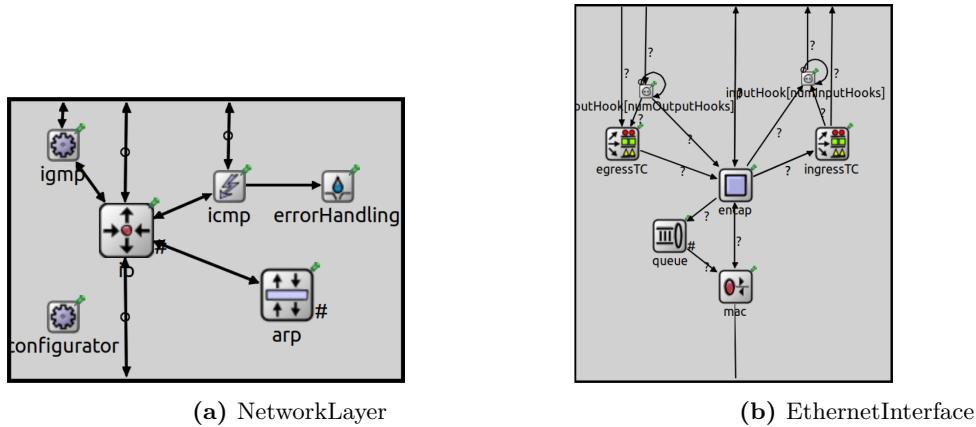


Abbildung 2.3: OMNeT++-Ansicht implizit genutzter zusammengesetzter Module

ThrputMeter realisiert. Das Interface **ITrafficConditioner** wird für QoS-Netzwerke benötigt und bleibt in dieser Arbeit außen vor.

Das zusammengesetzte Modul **EtherQoSQueue** implementiert Queues, und kann mit einer **DropTailQueue** belegt werden.

Networklayer

Das Modul **Networklayer** (2.3a) besteht aus den einfachen Modulen **IPv4**, **IPv4NodeConfigurator**³, **ARP**, **ICMP** und **ErrorHandling**. Diese Module setzen ihrem Namen entsprechend die grundlegenden Netzwerkprotokolle Internet Protocol Version 4 (IPv4), Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP) und Internet Group Management Protocol (IGMP) um, welche für die Kommunikation auf der Vermittlungsschicht wichtig sind.

EtherSwitch

Der Begriff des Switches wird in realen Netzwerken für Sicherungsschicht- und Vermittlungsschichtnetzwerkgeräte verwendet. In OMNeT++ wird ein Modul für die Sicherungsschicht-Switch Funktionalität definiert. Die Abbildung 2.2a zeigt grafisch den Aufbau eines Switches in OMNeT++. Das Modul besteht aus den einfachen Modulen **L2NodeConfigurator**, **InterfaceTable**, **NotificationBoard**, **MACAddressTable**, **MACRelayUnit** und optional einer Implementierung des **ISpanningTree**. Im Gegensatz zu Modulen auf Basis des Moduls **NodeBase** besitzt der **EtherSwitch.ned** kein Gate für den **GlobalFilter**. Dies schränkt den Einsatz mit SEA++⁴ ein.

StandardHost und Router

Die Module **StandardHost** und **Router** erben von dem Modul **NodeBase**, weshalb deren interner Aufbau ähnlich ist. Das Grundmodul **NodeBase** ist in der **NodeBase.ned** definiert und bietet die Grundlage für Ethernet, PPP und Wireless-Verbindungen an. Dies inkludiert das zusammengesetztes Modul **Networklayer**, welches einfache Module zur IP-Kommunikation

³In der bildlichen Darstellung (2.3a) werden Variablennamen angezeigt und nicht die Namen der Module. Beispielsweise wird der **IPv4NodeConfigurator** nur als **configurator** angezeigt.

⁴Die Bedeutung des **GlobalFilter** wird im Abschnitt 2.2 erklärt.

anbietet. Für die Kommunikation auf verschiedenen Übertragungsmedien werden verschiedene Implementierungen des `IWiredNic` Interfaces angeboten. Für diese Arbeit wird das `EthernetInterface` und `PPPIface` Modul genutzt. Das `PPPIface` ist ähnlich zu dem bereits vorgestellten `EthernetInterface`.

Transport Protokoll	Name	Beschreibung
TCP	TCPGenericSrvApp	Akzeptiert TCP-Verbindungen und schickt Antworten falls angefragt.
	TCPBasicClientApp	Sendet Nachrichten bestimmter Größe. Erlaubt es Antwortgrößen anzugeben. Besitzt mehrere Modi für Anfragen.
UDP	UDPVideoStreamSvr	Schickt auf Anfrage simulierte Videodateien bestimmter Größe.
	UDPVideoStreamCli	Fragt beim Server Videodateien an.
	UDPEchoApp	Beantwortet UDP-Nachrichten mit Datagrammen gleicher Größe.
	UDPBasicApp	Schickt UDP-Nachrichten.
Ping/ICMP	PingApp	Sendet ICMP-Ping-Nachrichten.

Tabelle 2.2: Applikationen für den `StandardHost`

StandardHost Aufbauend auf dem `NodeBase`-Modul, werden Implementierungen für Transport- und Anwendungsschicht Protokolle hinzugefügt. Für die Transportprotokolle werden Module für Transmission Control Protocol (TCP) und User Datagram Protocol (UDP) angeboten. Anwendungen können durch Implementierungen von den Modulen `ITCPApp`, `IPingApp`, `ISCTPApp` und `IUDPApp` realisiert werden. In der Tabelle 2.2 sind die wichtigsten Anwendungen für TCP und UDP im Kontext dieser Arbeit vorgestellt.

Router Wie der Standard Host besitzt ein Router Implementierungen von TCP und UDP. Im Gegensatz zu dem Host, werden keine Anwendungsschichtabstraktionen angeboten, sondern Module für die Routingprotokolle, wie z. B. das Routing Information Protocol (RIP) oder Border Gateway Protocol (BGP).

Die meisten der vorgestellten Module besitzen Parameter, mit denen weitere Spezialisierungen vorgenommen werden können. Im Falle des Routers muss z. B. kein Routingprotokoll realisiert werden. Im Fall, dass BGP nicht umgesetzt wird, werden die Module für TCP und UDP nicht initialisiert. In dieser Arbeit wird kein Routingprotokoll vorgeben.

InternetCloud Der Aufbau des Moduls `InternetCloud` ist ähnlich zu dem eines Routers, ohne die Funktionalität von Routing-Protokollen. Zu diesem Zweck wird ein abgewandelter Networklayer mit dem Namen `InternetCloudNetworkLayer` verwendet. Der Hauptunterschied zu dem bereits bekannten NetworkLayer besteht in dem `MatrixCloudDelayer`, welcher ermöglicht Datenpakete zu verzögern oder fallenzulassen (drop).

2.1.3 Ini-Datei

Die Parameterkonfigurationen werden in einer Key-Value-Syntax angegeben. Kommentare können mit einem Doppelkreuz (#) eingeleitet werden [11, Absatz 10.1.2].

Eine Konfigurationsdatei kann aus mehreren Abschnitten bestehen, welche aus eckige Klammern notiert sind und beim Start der Simulation ausgewählt werden. In der Beispieldatei 2 Konfigurationdatei für das Beispielnetzwerk

Quellcode 2 Konfigurationdatei für das Beispielnetzwerk

```

1 [General]
2 # Simulationseinstellungen
3 network = Example
4 sim-time-limit=10s
5
6 # StandardHost-Einstellungen
7 *.eve.numUdpApps = 1
8 *.eve.udpApp[0].typename = "UDPSink"
9 *.eve.udpApp[0].localPort = 222
10
11 *.bob.numUdpApps = 1
12 *.bob.udpApp[0].typename = "UDPSink"
13 *.bob.udpApp[0].localPort = 333
14
15 *.alice.numUdpApps = 1
16 *.alice.udpApp[0].typename = "UDPBasicApp"
17 *.alice.udpApp[0].localPort = 111
18 *.alice.udpApp[0].destPort = 333
19 *.alice.udpApp[0].messageLength = 1250 bytes
20 *.alice.udpApp[0].sendInterval = 0.2s
21 *.alice.udpApp[0].destAddresses = "bob"
22
23
24 [Config AttackConfig]
25 description = "A test attack against the example network."
26 **.attackConfigurationFile = "attack.xml"
```

Konfigurationsdatei 2 wären dies [*General*] und [*Config AttackConfig*]. Die einzelnen Abschnitte unterstützen Vererbung durch das Schlüsselwort *extends* und erben alle von dem Abschnitt *General* [11, Absatz 10.2.3]. In dem Beispiel besteht die allgemeine Konfiguration aus dem Befehl *network = Example*, welcher die genutzte Netzwerktopologie spezifiziert, dem Befehl *sim-time-limit=10s*, welcher die Simulationszeit auf 10 Sekunden begrenzt, und Konfigurationen für die *StandardHosts*, welche in dem Beispiel über ihre Namen *alice*, *bob* sowie *eve* identifiziert werden.

Die Modulidentifizierung nutzt Wildcards. In dieser Arbeit werden dabei hauptsächlich die Sterne (*) benutzt, welche sämtliche validen Zeichen eines Modulnamens entsprechen. Dabei ordnet ein Stern maximal ein Modul zu und zwei Sterne mehrere, da diese, die Punkte, welche zur Trennung der Modulidentifikatoren eingesetzt werden, miteinbeziehen. Für numerische Ausdrücke kann eine Notation mit zwei Punkten(..) verwendet werden, sodass beispielsweise der Befehl [1..4] dem Bereich zwischen eins und vier entspricht.

In der Beispielkonfiguration wird der *StandardHost* *alice* eindeutig über die Notation **.alice.numUdpApps* identifiziert. Der Stern (*) nimmt dabei die Rolle des Netzwerks *Example* ein. Der letzte Teil des Ausdrucks ***.numUdpApps* identifiziert den Parameter *numUdpApps*, welcher Teil des Moduls *StandardHost* ist. Um alle Geräte mit dem Parameter *numUdpApps* zu konfigurieren, kann ***.numUdpApps* geschrieben werden.

Wenn im Modul *StandardHost* der Parameter eins *numUdpApps=1* gesetzt ist, wird ein Modulvektor der Länge eins des Typs *udpApp* erstellt. In den folgenden Konfigurationen wird jeweils das erste Feld des Vektors *udpApp[0]* identifiziert. Mit dem Parameter

typename, der zu dem Modul *udpApp* gehört, wird dabei eine bestimmte Applikationsart erstellt. Die Anwendung *udpApp* wird bei Eve und Bob als *UDPSink* definiert. Ein *UDPSink* implementiert UDP-Sockets. Alices UDP-Applikation ist vom Typ *UDPBasicApp*, welche UDP-Datagrammen senden kann [9, Absatz 11.4].

Zuletzt wird in dem Abschnitt *[Config Example]* eine Beschreibung *description* gesetzt, welche bei der Konfigurationsauswahl angezeigt wird, und der Standardwert des Strings *attackConfigurationFile* auf *attack.xml* überschrieben.

Weitere Konfigurationsmöglichkeiten

In dem minimalen Beispiel wurden nicht alle für diese Arbeit wichtigen Möglichkeiten der Konfigurationsdatei vorgestellt. Um erweiterte Funktionalitäten anzubieten, können Funktionen in der Konfigurationsdatei verwendet werden. Eine Definition dieser findet sich im OMNeT++-Benutzerhandbuch [11, Absatz 22].

Wahrscheinlichkeiten Unter diesen Funktionen finden sich Wahrscheinlichkeitsverteilungsfunktionen, wie *exponential* für Exponentialverteilungen oder *intuniform* für eine Normalverteilung mit ganzen Zahlen. Die Funktionen geben jeweils eine Zufallszahl der entsprechenden Verteilung unter Bedingung ihrer Parameter zurück. Mit diesen Funktionen können Modulvektoren mit zufälligen Werten initialisiert werden, ohne jedem Modul feste Werte zuzuweisen.

NED Funktionen Eine weitere Funktionsklasse kann aus Ausdrücken die jeweiligen Knoten finden. In dieser Arbeit wird die Funktion *moduleListByPath("NodePath")* benutzt, um eine Liste von Knoten entsprechend des Eingabestrings zu erhalten. Die Funktion *choose(zahl, liste)* wird in Kombination mit *moduleListByPath* genutzt, um ein Modul aus der Liste auszuwählen.

2.1.4 Analysemöglichkeiten

Mit der NED Datei und einer Konfigurationsdatei kann ein Netzwerk mit primitiven Diensten modelliert werden. Es stellt sich die Frage, welche Mittel der Analyse OMNeT++ anbietet. Die von den einfachen Modulen erhobenen Daten werden in zwei Dateien gespeichert. Zum einen sind dies Vektordateien *<Simulationsname>.vec* und die Skalardateien *<Simulationsname>.sca*. Zusätzlich kann ein Eventlog aufgezeichnet werden, worauf in dieser Arbeit verzichtet wird.

In den Vektordateien werden fortlaufende Daten der Module und Kanäle gespeichert. [11, Absatz 12.1]. Dies können in dem Quellcode 1, beispielsweise in einem Ethernet Modul, die Anzahl der erstellten Rahmen zu einem bestimmten Zeitpunkt sein. In den Skalardateien werden zusammenfassende Daten über die einzelnen Module gesammelt [11, Absatz 12.1]. Diese Daten können zum Beispiel die Anzahl der gesendeten Datagramme eines UDP-Moduls sein.

Die Intensität der Datensammlung kann durch Konfigurationsparameter kontrolliert werden. Es besteht die Möglichkeit das Sammeln für bestimmte oder alle Module auszuschalten. [11, S. 12.2]

Analysetools

Die Analyse der Vektor- und Skalardateien kann durch Tools, wie das Kommandozeilenprogramm Scave [11, Absatz 12.5], einem in der OMNeT++-IDE integrierten Editor und durch externe Tools wie GNU R oder Python [11, Absatz 12.6] erfolgen. Diese Programme erlauben die grafische Darstellung der Daten. Für diese Bachelorarbeit wird die OMNeT++-IDE genutzt. Um spezifische Ergebnisse von Modulen oder Durchläufen zu betrachten, bietet die IDE Filter an.

2.2 SEA++

Die OMNeT++-Erweiterung SEA++ basiert auf INET. Die meisten Module von INET wurden um SEA++ eigene Module erweitert. Im Folgenden sollen die Kernkomponenten und Ziele von SEA++ vorgestellt sowie auf dessen Einrichtung eingegangen werden.

SEA++ erlaubt es mit einer DSL Angriffe auf ein OMNeT++-Netzwerk durchzuführen. Diese Angriffe sind immer erfolgreich, womit nicht die Analyse des Angriffswegs, sondern deren Folgen [30, Absatz 7.3.1] im Vordergrund stehen. Es handelt sich daher um einen quantitativen Analyseansatz.

2.2.1 Installation

Um die Nachvollziehbarkeit der Ergebnisse zu gewährleisten, sei im Folgenden auf die Einrichtung von SEA++ verwiesen. Im Benutzerhandbuch [22] befindet sich eine Installationsanleitung, welche durch alle wichtigen Schritte der Installation von OMNeT++ und SEA++ führt. Ausgelassen werden die benötigten Pakete für OMNeT++, welche sich in der Installationsanleitung [18] für OMNeT++ 4.6 finden. Zusätzlich wird Python 2 für den Interpreter der DSL benötigt.

2.2.2 Programmkomponenten

SEA++ besteht aus drei Komponenten – der Attack Specification Language (ASL)⁵, dem Attack Specification Interpreter (ASI) und der Attack Specification Engine (ASE). Bei der ASL handelt es sich um eine DSL, welche Angriffsbeschreibungen erlaubt. Der ASI übersetzt diese Beschreibung in eine XML-Konfigurationsdatei, welche zur Laufzeit der Simulation von der ASE ausgeführt wird. [30, Absatz 7.3.1]

Die ASE wird in der Simulation durch zwei Module umgesetzt. In den Netzwerkmodulen von INET wird ein Modul Local Event Processor (LEP) implementiert, welcher Netzwerknachrichten im Kommunikationsstack abfangen, neue Nachrichten erstellen und die physikalischen Eigenschaften der Netzwerkkomponente verändern kann. [30, Absatz 7.3.1] Jedes Modul mit einem LEP muss anschließend mit einem Global Event Processor (GEP) verbunden werden [22, Absatz 4.1]. Dieser GEP verbindet alle LEP und ermöglicht die Evaluation von komplexen Angriffen [30, Absatz 7.3.1].

Der LEP wird mit dem Modul `LocalFilter` und der GEP mit dem Modul `GlobalFilter` umgesetzt. Um andere Module zu deaktivieren, muss im Netzwerk das Modul `ExMachina` vorhanden sein [22, Absatz 3.1].

⁵Die Nomenklatur der Sprache ist uneindeutig. In *SEA++; user manual; SEA++ with SDN support INET-based*, Seite 1 [22] wird von der Attack Description Language (ADL) gesprochen. Allerdings wird in „SEA++: A Framework for Evaluating the Impact of Security Attacks in OMNeT++/INET“, Absatz 7.3.1 [30] von der ASL gesprochen. Im Folgenden wird von ASL ausgegangen.

2.2.3 ASL

In der ASL ist ein Angriff eine Eventabfolge. Dabei bietet ASL die Möglichkeit sowohl initialisierte als auch uninitialisierte Variablen zu definieren, welche vor der ersten Benutzung deklariert werden müssen [22, Absatz 3.2.1].

Typen von Angriffen

SEA++ unterscheidet zwischen bedingten 2.2.2 und unbedingten 2.2.1 Angriffen.

Definition 2.2.1. Ein unbedingter Angriff [30, Absatz 7.3.2.3] führt ab einem Zeitpunkt T in einer Periode von P Zeiteinheiten Angriffe aus.

```
1 from T every P do {
2   <Angriffe ... >
3 }
```

Definition 2.2.2. Ein bedingter Angriff [30, Absatz 7.3.2.3] führt für einen Zeitpunkt T von Knoten \langle liste von knoten \rangle Angriffe durch, wenn die Filterbedingungen \langle bedingung \rangle als wahr ausgewertet werden. Diese Bedingung ist dabei ein logischer Ausdruck.

```
1 from T nodes in <liste von knoten> do {
2   filter(<bedingung>)
3   <Angriffe ... >
4 }
```

Der Knoten wird über die Modul-Id identifiziert. Wegen der Modul-Id muss bei jeder Änderung der Topologie geprüft werden, ob sich die Ids verschoben haben. Es gibt ein Listendatentyp, welcher die Identifikation von mehreren Modulen erlaubt. SEA++ unterstützt, in der vorliegenden Version, nicht die Möglichkeit, Knoten über ihre Namen in der Topologie zu identifizieren.

Unterstützte Befehle

Für die Angriffsdefinition bietet ASL zwei unterschiedliche Anweisungsklassen (Primitives) an. Zum einen existieren Anweisungen für die Manipulation von Knoten, vorgestellt in Definition 2.2.3. Zum anderen implementiert SEA++ Anweisungen, um Nachrichten zu modifizieren und zu erzeugen, welche in Definition 2.2.4 vorgestellt werden.

Definition 2.2.3. Zum Modifizieren von Knoten existieren drei Anweisungen [30, Absatz 7.3.2.1]:

- **move(nodeId, t, x, y, z)**
Verschiebt den Knoten mit der Id **nodeId** zum Zeitpunkt **t** zu den Koordinaten **x,y,z**⁶.
- **destroy(nodeId, t)**
Trennt⁷ den Knoten mit der Id **nodeId** zum Zeitpunkt **t** aus dem Netzwerk.
- **disable(nodeId, t)**
Entfernt⁸ den Knoten mit der Id **nodeId** zum Zeitpunkt **t** aus dem Netzwerk.

⁶Über die Koordinaten können Entfernungen in Topologien dargestellt werden. In dieser Bachelorarbeit werden die Entfernungen für die Ethernetverbindungen allerdings vorgegeben.

⁷Im Gegensatz zu dem Entfernen laufen die Applikationen in dem Knoten weiter.

⁸Der Knoten ist nicht mehr Teil des Netzwerkes. Alle Anwendungen stoppen.

Definition 2.2.4. Zum Modifizieren von Nachrichten existieren sieben Anweisungen [30, Absatz 7.3.2.2]:

- ***create(paket, feld, inhalt)***
Erstellt ein neues Datenpaket *paket* und setzt im Feld *feld* den Inhalt *inhalt*.
- ***change(paket, feld, neuerInhalt)***
Im Datenpaket *paket* ändere den Inhalt des Feldes *feld* zu dem Wert *neuerInhalt*.
- ***clone(ursprungsPaket, zielPaket)***
Kopiert das *ursprungsPaket* in das neue Paket *zielPaket*.
- ***retrieve(paket, feld, variable)***
Kopiert den Inhalt des Feldes *feld* aus dem Paket *paket* in die Variable *variable*.
- ***drop(paket, schwelle)*⁹**
Verwirft das Datenpaket *paket*, mit einer Wahrscheinlichkeit *schwelle*.
- ***send(paket, verzögerung)***
Übergibt das Datenpaket *paket* nach einer Verzögerung *verzögerung* an die jeweilige Schicht.
- ***put(paket, empfängerKnoten, richtung, updateStats, verzögerung)***
Platziert das Datenpaket *paket* abhängig von dem Richtungsargument *richtung* entweder in den Transmission- oder Receptionbuffer von allen Knoten in der Liste *empfängerKnoten* mit einer Verzögerung *verzögerung* und einem Parameter *updateStats*.

Einige Anweisungen benötigen eine bestimmte Art von Angriff. Für unbedingte Angriffe können die Anweisungen *move*, *destroy* und *send*¹⁰ nicht ausgeführt werden.

SEA++ Abkürzung	Internet-Protokoll-Stack
APP	Anwendungsschicht
TRA	Transportschicht
NET	Vermittlungsschicht
MAC	Sicherungsschicht

Tabelle 2.3: Zuordnung der SEA++-Typen zum Internet-Protokoll-Stack

Identifikation der Felder Wie aus OMNeT++ bekannt, wird das Feld *feld* mit einer durch Punkte (.) getrennten Syntax identifiziert. Datenpakete werden nicht über ihre Klassennamen, sondern über deren Schicht identifiziert. In der Tabelle 2.3 sind die Abkürzungen der verschiedenen Schichten dargestellt. Die deutschen Namen sind nach *Computernetze ein Top-Down-Ansatz mit Schwerpunkt Internet* [26] gewählt.

⁹Sowohl in der Vorstellung von SEA++ in Tiloca u.a. als auch im Nutzerhandbuch [22] wird das Schwellenfeld nicht erwähnt. Allerdings findet sich in der entsprechenden Implementierung unter *interpreter/interpreter/primitives/drop.py* ein entsprechender Hinweis. Ohne einen Schwellenwert kann die ADL nicht interpretiert werden.

¹⁰In der *UnconditionalAttack.cc* finden sich die entsprechenden Implementierungen.

Erstellen von Datenpaketen Bei dem Erstellen eines Datenpakets mit der Anweisung `create` wird das Feld `[APP/TRA/NET/MAC].type` gesetzt. Der Typ `typ` identifiziert dabei die Art des Datenpakets in der Schicht `[APP/TRA/NET/MAC]` und damit dessen Felder.

Im SEA++-Benutzerhandbuch [22, Absatz 5] ist eine Dokumentation der unterstützten Pakettypen vorhanden. Diese ist allerdings nicht vollständig. Zusätzlich ist durch eine Erweiterung für diese Bachelorarbeit ein neuer Typ **MAC.0040** hinzugekommen. Durch Sichtung des Quellcodes wurde eine neue Übersicht aller Typen erstellt. Diese kann im Anhang D gefunden werden.

Kontroll-Informationen Die meisten Datenpakete besitzen sogenannte **controlInfo**-Objekte [11, Absatz 5.2.4], welche den umliegenden Schichten Informationen geben, die zum Transport der Datenpakete wichtig sind. Mit dem Wissen über den Namen des Datenpakets und dessen **controlInfo**-Objekt, kann im Quellcode, der Name aller Felder in der zugehörigen `<Paket_Name>.msg` Datei nachgeschlagen werden.

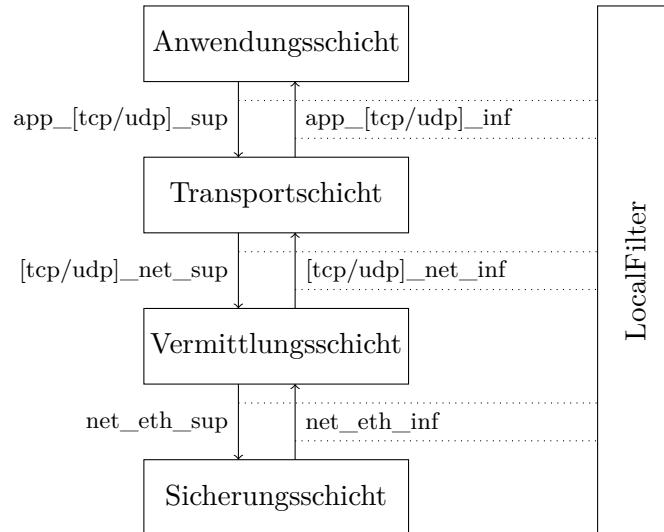


Abbildung 2.4: Die verschiedenen Gates für das `sending.outputGate` Feld. Angelehnt an [22, Abbildung 5.1].

Platzierung des Datenpakets im Kommunikationsstapel Um mit der `put`-Anweisung ein Datenpaket direkt in den Kommunikationsstapel eines Moduls zu platzieren, muss das Gate des LEP bekannt sein. Dabei muss die Richtung des Datenpakets im Internet-Protokoll-Stack beachtet werden. Das Gate kann für ein Datenpaket mit der Anweisung `change(ip4Datagram, "sending.outputGate", «GateName»)` geändert werden. In der Darstellung 2.4 finden sich die Werte für `<GateName>`. Ausdrücke in eckigen Klammern geben Optionen an. So kann entweder `tcp` oder `udp` für die Gates der oberen Schichten genutzt werden.

Mit der Anweisung `change(<paket1>, "[APP/TRA/NET/MAC].payload", <paket2>)` kann ein Datenpaket `paket1` als Payload das Datenpaket `paket2` befördern. Die Position im Internet-Protokoll-Stack muss dabei beachtet werden.

Operatoren Zusätzlich zu den bereits vorgestellten Anweisungen enthält SEA++ logische und algebraische Operatoren. Eine Variable, erstellt und initialisiert mit `var varName = value`, kann u. a. mit den Operator (+) addiert werden. Im Falle, dass `value` ein String ist, würde der Operator eine Konkatenation durchführen. Eine vollständige Übersicht mit allen Operatoren und validen Ausdrücken findet sich im Benutzerhandbuch [22, Absatz 3.2.1].

2.2.4 Erweiterung von SEA++

In dem Benutzerhandbuch [22, Absatz 5] von SEA++ wird vorgestellt, wie das Framework um neue Datenpakete erweitert werden kann. Aufgrund der Wichtigkeit für den geplanten ARP-Angriff, wird die Vorgehensweise kurz vorgestellt.

Gemäß dem Handbuch müssen drei Dateien modifiziert werden.

- Die `Create.cc` muss in der Methode `buildNewPacket(...)` um das entsprechende Datenpaket erweitert werden.
- Die `seapputils.cc` muss in der Methode `getPacketLayer(...)` die entsprechende Schicht des Datenpakets zurückgeben.
- Die `Create.h` muss um den `type_t` des Datenpakets erweitert werden.

Außerdem muss eine `.msg` Datei für das entsprechende Datenpaket vorliegen.

2.2.5 Zusammenfassung des Arbeitsablaufs

Zusammenfassend kann die Arbeit mit SEA++ in fünf Schritte eingeteilt werden.

1. Definieren einer Netzwerktopologie und deren Dienste in OMNeT++ mit INET.
2. Definieren des Angriffs auf das Netzwerk in einer ADL-Datei in der ASL.
3. Übersetzen der ASL Beschreibung mit dem ASI.
4. Durchführen der Simulation in OMNeT++.
5. Auswerten der Daten mit den Tools in OMNeT++.

Insbesondere Schritt eins kann vereinfacht werden, falls das entsprechende Forschungsprojekt bereits in OMNeT++ implementiert ist. Bei Forschungsgruppen, welche im OMNeT++-Programmökosystem arbeiten, kann zusätzlich der Analyseteil durch Skripte vereinfacht werden.

Kapitel 3

Methodik

In diesem Kapitel werden die Angriffsstrategien und deren genutzten Protokolle erklärt. Weiterhin soll ein vereinfachtes Modell des Netzwerks der Fakultät für Informatik an der TU vorgestellt werden.

3.1 Angriffsszenarien

Als Angriff (englisch Attack) wird eine schädliche Aktion gegen ein Netzwerk oder Knoten eines Netzwerks bezeichnet. Ein Angreifer wird dementsprechend die ausführende Partei eines Angriffs genannt. Für diese Bachelorarbeit wurden drei unterschiedliche Typen von Angriffen simuliert, deren Aufbau und Ablauf im Folgenden beschrieben wird.

3.1.1 Angriffsvariante: ARP-Poisoning

Das ARP stellt die Verbindung zwischen Vermittlungs- und Sicherungsschichtprotokollen her. Obwohl die Definition in Request for Comments (RFC) 826 [20] verschiedene Internet- und Netzzugangsprotokolle vorsieht, wird häufig IPv4¹ und Ethernet genutzt. Ungeachtet das IPv4-Adressen verwendet werden, läuft die Kommunikation nur über die Sicherungsschicht im Internet-Protokoll-Stack, womit das Protokoll dieser zuzuordnen ist.

Aufbau

Die ARP Protocol Data Unit (PDU) (3.1) besitzt Felder für die MAC- und IPv4-Adressen des Senders und Empfängers. Zusätzlich existiert ein Headerfeld für den ARP-Datenpakettyp, welcher entweder eine ARP-Anfrage (Request) oder eine ARP-Antwort (Reply) ist. Weitere Felder werden für die Nutzung von anderen Netzzugangs- und Internetprotokollen benötigt [20], wobei deren Werte aus einer Datenbank entnommen werden [24, 2] können.

Ablauf einer Adressauflösung Der Adressauflösungsablauf wird in RFC 1180 [27] vorgestellt. Jeder Client hält in einem Netzwerk einen *ARP-Cache* vor, in dem bereits bekannte Adresszuordnungen gespeichert sind. Im Regelfall werden Einträge in diesem regelmäßig gelöscht. Sollte ein Dienst der Vermittlungsschicht ein neues Paket zum Versenden erhalten, dessen IPv4-Adresse noch nicht bekannt ist, wird eine neue *ARP-Anfrage*

¹IPv6 verwendet ein anderes Protokoll [16].

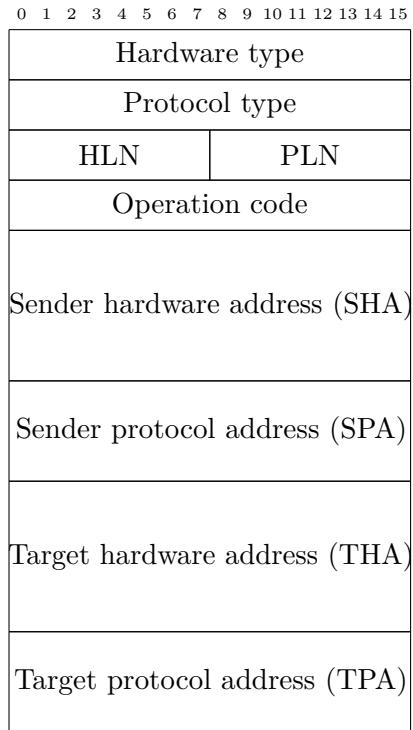


Abbildung 3.1: Der ARP-Header nach RFC 826 [20]

erstellt. Eine Anfrage enthält ein leeres Ziel-IP-Feld (TPA) und wird per Broadcast an alle Geräte in der *ARP-Broadcastdomäne* gesendet

Der Empfänger, mit der geforderten IPv4-Adresse oder dem Gateway zu der entsprechenden IPv4-Adresse, antwortet auf diese Nachricht mit einer *ARP-Antwort*. Diese Antwort wird direkt an den Sender der Anfrage geschickt. Sender- und Empfängerfelder sind in der *ARP-Antwort* im Vergleich zur *ARP-Anfrage* vertauscht.

Der ursprüngliche Sender der Anfrage kann beim Erhalt der Antwort mit der TPA eine neue Zuordnung in seinem *ARP-Cache* speichern und *Rahmen*² an diese Adresse versenden.

Angriffsvektor

Eine fehlende Verifikation der *ARP-Antwort*, ermöglicht falsche oder bösartige Adressauflösungen an ein Netzwerkknoten zu schicken. Dies kann unterschiedliche Auswirkungen auf ein Netzwerk haben. Der Netzwerkverkehr kann durch ungültige Zuordnungen gestört oder gezielt auf bestimmte Knoten umgeleitet werden. Angriffe auf das ARP-System nennt man ARP-Spoofing oder ARP-Poisoning [5].

3.1.2 Angriffsvariante: Denial of Service

Ein DoS-Angriff zielt darauf ab, eine bestimmte Netzwerkkomponente eines Netzes in ihrer Verfügbarkeit zu stören [5]. Dies kann durch eine große Anzahl an Anfragen an einen Dienst dieser Komponente realisiert werden. Aufgrund der großen Netzwerkkapazitäten, welche dafür benötigt werden, ist der Distributed-Denial-of-Service-Angriff (DDoS) entstanden,

²Als Rahmen (Frame) wird eine PDU der Sicherungsschicht bezeichnet.

welche den Denial of Service (DoS) um eine verteilte Menge an Angreifern erweitert. Genauer betrachtet wurden DoS-Angriffe in RFC 4732 [7].

Für einen DoS-Angriff gibt es verschiedene Methoden, die unterschiedliche Protokolle im Internet-Protokoll-Stack ausnutzen. In dieser Bachelorarbeit wird eine Schwachstelle im Verbindungsauflaufbau des TCP ausgenutzt, welche im Folgenden vorgestellt wird.

Transmission Control Protocol

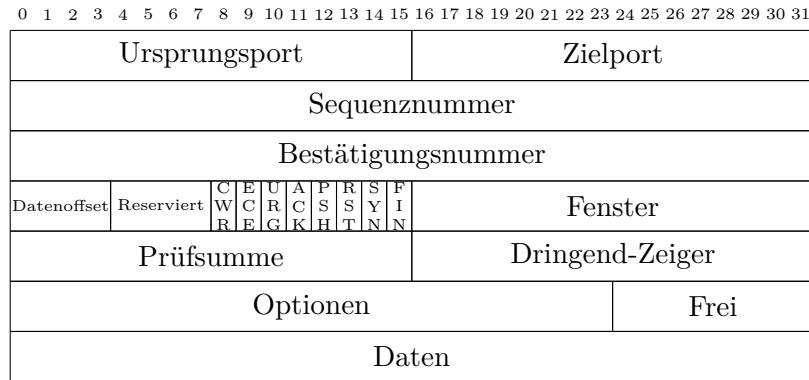


Abbildung 3.2: TCP-Header nach RFC 739 [21] mit Erweiterung durch RFC 3168 [23]

TCP wurde als Protokoll für die zuverlässige Kommunikation zwischen zwei Clients in RFC 739 [21] vorgestellt. Der Header, abgebildet in Abbildung 3.2, besitzt Felder für Quell- und Zielport, sowie Sequenz- und Bestätigungsnummern. Für den folgenden Angriff wichtig sind vor allem die TCP-Flags, welche u. a. für den Verbindungsauflaufbau und -abbau eine wichtige Rolle spielen.

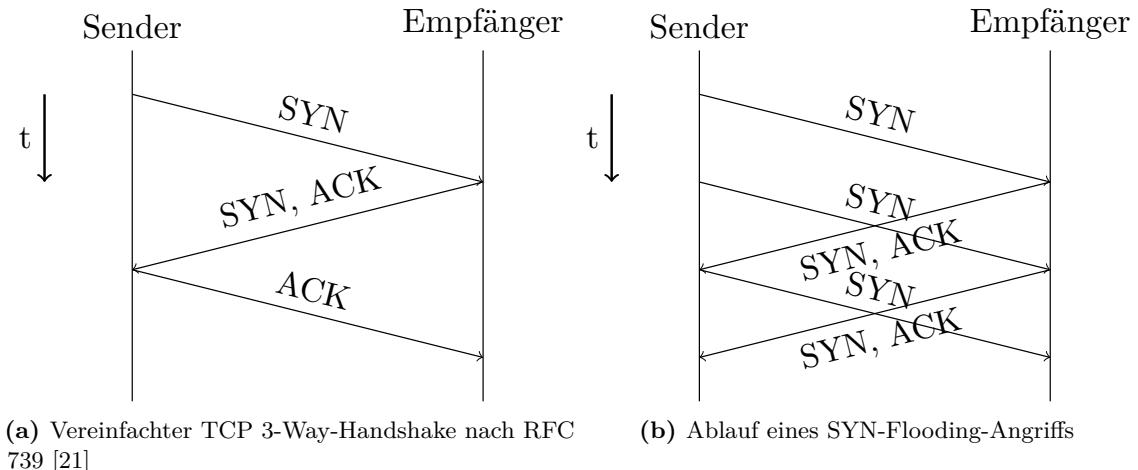


Abbildung 3.3: Ablauf eines normalen TCP-Handshakes und eines SYN-Flooding-Angriffs

Verbindungsauflaufbau Ein Three-Way-Handshake initiiert den TCP-Verbindungsauflaufbau. Durch diesen wird sichergestellt, dass beide Geräte kommunizieren können. In Abbildung 3.3a wird eine vereinfachte Version vorgestellt, welche folgende drei Schritte durchläuft:

1. Zuerst wird ein TCP-Segment von dem Sender, mit gesetzten *SYN*-Bit versendet. Die Sequenznummer ist zufällig und die Bestätigungsnummer auf null gesetzt.
2. Der Empfänger antwortet mit einem Segment, bei dem das *SYN*- und *ACK*-Bit gesetzt sind.
3. Dieses Antwortsegment wird von dem Sender mit einem Segment, in welchem das *ACK*-Bit gesetzt wurde, bestätigt.

Angriff auf den Verbindungsauflbau Ein Angreifer kann diesen Verbindungsauflbau für einen Angriff ausnutzen. Hierzu sendet ein Angreifer viele Segmente mit gesetzter *SYN*-Flag und antwortet nicht auf die Bestätigung des Empfängers. Dies führt dazu, dass ein Angriffsziel auf eine Vielzahl von Verbindungen wartet und dadurch Ressourcen verbraucht. Es entsteht zusätzlich Netzwerkverkehr durch die gesendeten Bestätigungen. Ein solcher Angriff wird SYN-Flooding genannt und gehört zu den häufigsten Angriffsmethoden [33].

Intensität Die Intensität eines DoS-Angriffs kann mit dem verursachten Netzwerkdatendurchsatz (B/s) und mit der Anzahl an Paketen pro Sekunde (p/s) gemessen werden. Cloudflare, ein großer Anbieter für Netzwerkdienstleistungen, veröffentlicht quartalsweise Statistiken über die DDoS auf ihre Kunden. Nach *Network-Layer DDoS Attack Trends for Q2 2020* [33] verursachten im zweiten Quartal 2020 der Großteil der Angriffe ein Paketaufkommen von 50 kp/s bis 1 Mp/s. Die meisten Angriffe dauerten zwischen 30 und 60 Minuten [33].

3.1.3 Angriffsvariante: Port-Scanner

Ports differenzieren unterschiedliche Verbindungen zu einem Host und identifizieren Dienste. Für den zweiten Verwendungszweck werden Zuweisungsregeln in RFCs festlegt. Eine der neusten RFCs hinsichtlich der Portzuweisung ist RFC 7605 [31], in dem zusätzlich der Begriff des Ports definiert wurde. Wegen dieser Nähe zu laufenden Diensten, ist das Wissen über offene Ports essenziell, da so Hinweise auf laufende Anwendungen gefunden werden können. Diese Anwendungen stellen weitere Angriffsvektoren dar. Als Vermittlungsschichtfunktionalität setzt TCP das Portkonzept um. Implementierungen von TCP reagieren unterschiedlich auf Segmente mit gesetzten TCP-Flags an offene oder geschlossene Ports. Das Standardverhalten ist jeweils in der RFC 739 [21] festgelegt.

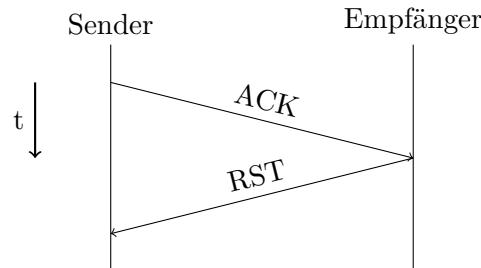


Abbildung 3.4: Vorgesehene Antwort auf ein unerwartetes TCP-Segment mit gesetztem ACK-Bit RFC 739 [21]

TCP-ACK-Scan Der zu implementierende Angriff ist ein sogenannter TCP-ACK-Scan³, welcher auf der in *Transmission Control Protocol* vorgestellten Reaktion auf unbekannte Segmente mit gesetztem *ACK*-Bit basiert. Nach RFC reagiert die TCP-Implementierung, wie dargestellt in Abbildung 3.4, auf solche Segmente mit einer Antwort, bei welcher das *RST*-Bit gesetzt ist. Dabei spielt es keine Rolle, ob der Port geöffnet oder geschlossen ist. Im eigentlichen Sinn handelt es sich deshalb nicht um einen Port-Scanner, sondern um ein Firewall-Scanner.

Ein Host mit Firewall wird in der Regel Ports filtern und Segmente ignorieren oder mit einem ICMP-Segment beantworten. Das Senden von modifizierten Segmenten gibt daher Aufschluss über durch Firewalls gefilterte Ports.

Der Angriff soll wie folgt ablaufen:

1. Gehe von einem infizierten Knoten im Netzwerk aus.
2. Für jedes TCP-Segment an diesem Knoten, welches eine neue Verbindung aufbaut, also ein gesetztes *SYN*-Bit enthält, erstelle ein neues TCP-Segment und
3. sende dieses mit gesetztem *ACK*-Bit an den Sender.

3.2 Das Netzwerk der Fakultät für Informatik

In diesem Abschnitt soll der Aufbau des Netzwerks der Fakultät Informatik an der TU-Dortmund (Informatiknetzwerk) beschrieben werden. Die Informationen über das Netzwerk wurden bei der Informatikrechner-Betriebsgruppe (IRB) und dem IT & Medien Centrum (ITMC) erfragt. Für die Verwaltung des Netzwerks der Informatikfakultät ist die IRB zuständig und für den außerfakultären Bereich das ITMC.

3.2.1 Vereinfachter Aufbau

Das Informatiknetzwerk, vereinfacht abgebildet in der Grafik 3.5, ist ein Teil des Universitätsnetzwerkes. Mit dem Internet verbunden ist die TU über das Deutsche Forschungsnetz (DFN) mit zweimal 5 Gbit/s⁴. Das Informatiknetz ist ein reines Ethernet, wobei Wireless Local Area Network (WLAN) Funktionalitäten im Bereich des ITMCs liegen.

Topologisch wird das Informatiknetzwerk durch die Aufteilung der Informatikfakultät auf drei Gebäude, welche aufgrund ihres Standorts an der Otto-Hahn-Straße (OH), im weiteren Verlauf OH12, OH14 und OH16 genannt werden, bestimmt.

In der OH12 befindet sich der Übergabepunkt zum Universitätsnetz. Von der OH12 ausgehend sind die OH14 und OH16 verbunden. Das Informatiknetzwerk, kann in einen Campus- und einen Datacenterbereich, welcher in der OH12 liegt, aufgeteilt werden. Der Datacenterbereich ist mit 40 GB/s und die Campusbereiche jeweils mit zweimal 20 GB/s angebunden.

Über eine Sterntopologie sind alle Geräte eines Gebäudes an einem zentralen Verteilerraum angebunden, welcher mit dem IRB-Router verbunden ist. Da alle Gebäude an dem zentralen Router angebunden sind, setzt sich die Sterntopologie zu diesem fort. Der zentrale Router besteht aus zwei mit Virtual Switching System (VSS) verbundenen Routern.

Eine Differenzierung der einzelnen Netze wird nicht über dedizierte Hardware, u. a. eigene Switchinghardware für jeden Lehrstuhl, sondern über den Router mithilfe von

³Namen von Angreifen können sich unterscheiden. Der Name wurde aufgrund der Nutzung von *nmap*, einem bekannten Netzwerkscanner, gewählt. <https://nmap.org/book/scan-methods-ack-scan.html>

⁴Die Universität Bochum und Duisburg-Essen sind jeweils mit 10 Gbit/s angebunden

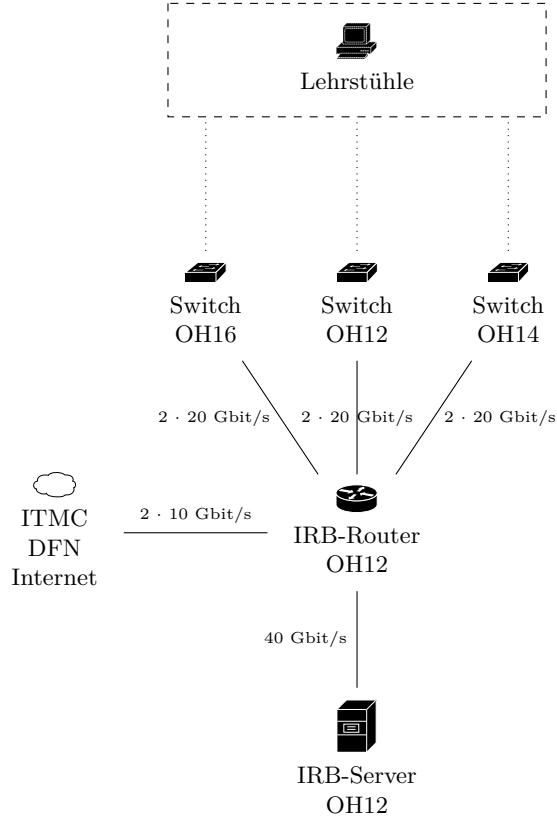


Abbildung 3.5: Vereinfachte Darstellung des Netzwerks der Fakultät für Informatik an der TU-Dortmund

Virtual Local Area Network (VLAN) vorgenommen. Entsprechend sind über die VLANs die Broadcastdomainen definiert.

Das gesamte Netzwerk ist in ungefähr 90 Subnetze eingeteilt, mit durchschnittlich 700 aktiven Geräten, welche ihre IP-Adressen je nach Subnetz, aus dem Adressbereich 129.217.1.1 bis 129.217.63.255 beziehen.

Vereinfacht kann davon ausgegangen werden, dass jedes Gerät mit 1 Gbit/s an den Switches angebunden ist. Im Datacenterbereich liegen die Anbindungen bei 10 Gbit/s. Das Netzwerk ist durch verschiedene Sicherheitsmaßnahmen geschützt, auf welche nicht weiter eingegangen wird, da die Modellierungsmöglichkeiten in dieser Bachelorarbeit fehlen.

3.2.2 Netzwerkverkehr

Im Netzwerk werden verschiedene Dienste angeboten und genutzt. Da in OMNeT++ keine echten Applikationen auf den Geräten laufen, sind die verschiedenen genutzten Protokolle von vorrangigem Interesse.

Der TCP-Netzwerkverkehr wird durch eine Vielzahl von Diensten im Informatiknetzwerk erzeugt. Viele Lehrstühle und Studenten benutzen Versionierungssysteme wie Git und SVN. Außerdem wird auf verschiedene HTTP-Dienste, wie interne und externe Webseiten zugegriffen. Von dem IRB werden einige Dienste wie zum Beispiel das BigBlueButton (BBB), DHCP, VPN, ein Datensicherungsdienst sowie Dienste für den E-Mail-Empfang und -Versand, bereitgestellt. Durch die tägliche Nutzung dieser Dienste ist TCP eine wichtige

Komponente des Netzwerkverkehrs. Das BBB und weitere Videostreamingservices sind Quellen von UDP-Netzwehrverkehr. Zusätzlich fällt ICMP Netzwerkverkehr an.

3.2.3 Auslastung

Aufgrund von Schwankungen kann eine genaue Auslastung des Netzwerks nicht bestimmt werden. Die Frage, wo die Netzwerksauslastung zu messen ist, kann nicht eindeutig beantwortet werden. Für diese Arbeit wird eine Auslastung des Netzes am zentralen Router der IRB gemessen, da dieser im Mittelpunkt des Netzes steht.

Stark vereinfacht, kann festgestellt werden, dass das Netzwerk nur leicht belastet ist. Im Folgenden wird von einer Auslastung von ungefähr 1 Gbit/s an den Ports zum Datacenterbereich ausgegangen. Die Ports zum Außennetz sind geringer ausgelastet.

Kapitel 4

Aufbau der Simulation

Im Folgenden wird auf die Implementierung des in Kapitel 3 beschriebenen Modells eingegangen. Die Gliederung folgt dem OMNeT++-Arbeitsablauf, indem zuerst die Topologie umgesetzt wird und diese danach konfiguriert wird. Zuletzt werden die Angriffe mit SEA++ umgesetzt.

4.1 Umsetzung der Netzwerktopologie

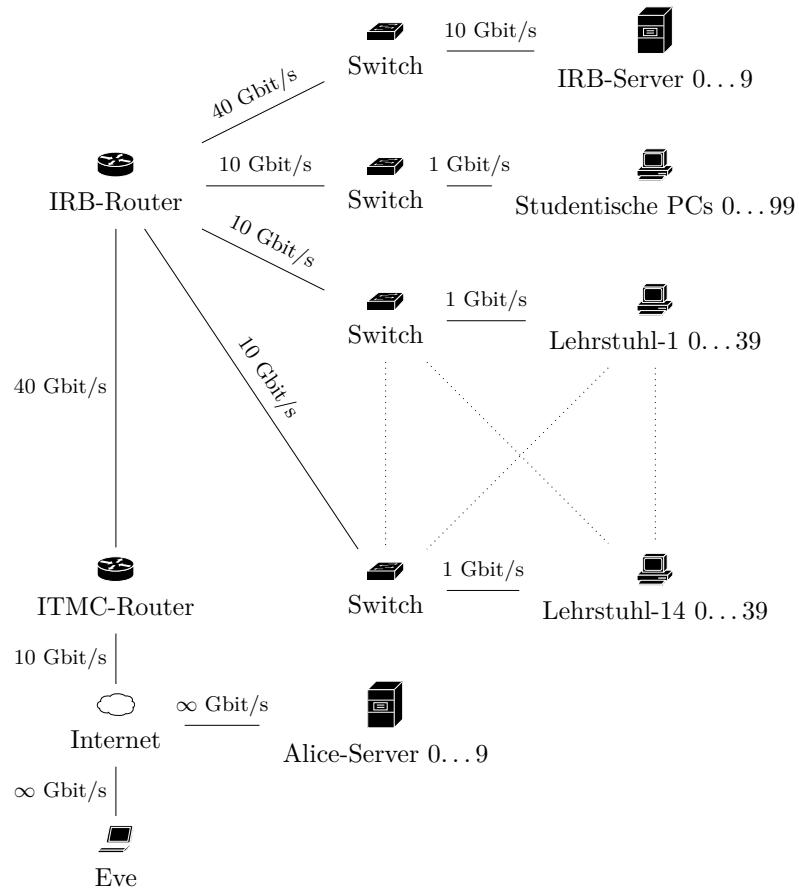


Abbildung 4.1: Vereinfachte Netzwerktopologie der Informatikfakultät an der TU-Dortmund.

Für das Modell des Informatiknetzwerks wird ein OMNeT++-Netzwerk mit dem Namen `TuCsNet` erstellt, welches aus den zusammengesetzten Modulen `StandardHost`, `Router`, `InternetCloud` und `EtherSwitch` besteht. In Abbildung 4.1 wird eine vereinfachte Netzwerktopologie dargestellt. Grob kann das Netzwerk in zwei Teile aufgeteilt werden,

1. einem Innennetz, in diesem Fall das Netzwerk der Fakultät für Informatik
2. sowie einem Außennetz, welches bei der Verbindung zwischen dem Router der IRB und des ITMC beginnt und das restliche Netzwerk umfasst.

4.1.1 Innennetz

Das Innennetz stellt ein vereinfachtes Modell des Netzwerks der Fakultät für Informatik an der TU-Dortmund dar. Es kann in einen zentralen Router sowie mehrere Netzbereiche mit Rechnern und Servern unterteilt werden.

IRB-Router

An einem zentralen Router der IRB werden alle Netzbereiche über Switches angebunden. Dies entspricht der Sternkopologie des realen Netzwerks. In dem echten Netzwerk besteht der zentrale Router aus zwei einzelnen Routern, welche unter Zuhilfenahme von VSS zu einem Router zusammen geschaltet sind. Das proprietäre Protokoll wird in Version 2.6 von INET nicht unterstützt, weshalb der zentrale Router nur aus einem zusammengesetzten Modul `Router` besteht.

Netzbereiche

Im Unterschied zu den ca. 90 Netzbereichen des realen Netzwerks, sind in der Simulation nur Bereiche für die Fakultäten und Studenten sowie ein Datacenterbereich der IRB vorgesehen. Die Bezeichnung der Netzbereiche richtet sich nach dem Kürzel `ls` gefolgt von der Nummer des Lehrstuhls `1,2,4-9,11-14` und den Bezeichnern `studentHost` sowie `irbServer`.

In jedem der Bereiche sind die Netzwerkgeräte repräsentiert durch `StandardHost`-Module, welche mit einem `EtherSwitch`- und `GlobalFilter`-Modul verbunden sind. Jeder Switch ist direkt mit dem Router verbunden. Auf eine Modellierung von zentralen Punkten, an denen alle Geräte eines Gebäudes zusammenlaufen, wird verzichtet. Weil das reale Netzwerk VLAN nutzt, um die verschiedenen Bereiche des Netzwerks zu trennen. Die Funktionalität für VLAN, realisiert durch IEEE802.1Q, wird in INET erst in Version 4.1.0 [10] hinzugefügt, welche zusätzlich nur mit OMNeT++ in Version 5.4.1 kompatibel ist. Aufgrund des fehlenden VLAN besitzen die Lehrstühle keine Server in der eigenen Broadcastdomäne.

Anzahl der Netzwerkgeräte Die Größe der Bereiche ist stark vereinfacht. Es wird von ungefähr 700 aktiven Netzwerkgeräten zum Zeitpunkt der Simulation ausgegangen. Diese werden so aufgeteilt, dass im studentischen Bereich 100 Geräte vorhanden sind, in jeden Lehrstuhlbereich 40 und im IRB-Serverbereich jeweils 10. Diese Konfiguration wird mit den Parametern `hostCount`, `serverCount` und `studentCount` als Standardwerte übergeben.

Quellcode 3 Verwendete Kabeltypen um die Geräte in dem `tucsnet` zu verbinden.

```

1      channel IdealChannel extends DatarateChannel
2      {
3          parameters:
4              delay = 0;
5      }
6
7      channel EthGebaeudeKabel   extends Eth10G
8      {
9          parameters:
10             length = 500m;
11     }
12
13     channel EthRaumKabel    extends Eth1G
14     {
15         parameters:
16             length = 100m;
17     }
18
19     channel EthIrbServerKabel  extends Eth10G
20     {
21         parameters:
22             length = 50m;
23     }
24
25     channel EthServerRaumKabel  extends Eth40G
26     {
27         parameters:
28             length = 50m;
29     }

```

Verbindungen Im Innennetz werden die Kabeltypen `EthGebaeudeKabel`, `EthRaumKabel`, `EthServerRaumKabel` und `EthIrbServerKabel` verwendet, welche sich in der Länge¹ und Übertragungsrate unterscheiden. Die Länge der Kabel liegt zwischen 50 m und 500 m.

Um die Verbindungen zwischen Switch und Netzwerkgeräten außerhalb der Serverbereiche zu realisieren, werden `EthRaumKabel` mit 1 Gbit/s verwendet, was dem echten Netzwerk entspricht. Im realen Netzwerk wurden mehrere Netzbereiche über den gleichen Switch angebunden. In dem Modell ist jeder Bereich über einen eigenen dedizierten Switch mit einem Kabel `EthGebaeudeKabel` bei einer Übertragungsrate von 10 Gbit/s angebunden.

Die restlichen Kabeltypen mit jeweils 10 Gbit/s und 40 Gbit/s werden im Serverbereich eingesetzt, sowohl für die Verbindung von Server zum Switch, als auch von Switch zum Router. Zusätzlich zu den Verbindungen mit Ethernetkabeln muss jeder `StandardHost` und Router mit dem `GobalFilter` verbunden werden.

4.1.2 Außenetz

An dem ITMC-Router ist eine `InternetCloud internet` angeschlossen, um ein grobes Modell des Internets zu erzeugen. Mehrere Module des Typs `StandardHost` bilden Server im Internet ab. Die Module sind, der typischen Terminologie im Sicherheitsbereich folgend,

¹Die Längen haben für die eigentliche Simulation nur eine geringe Bedeutung, da im heutigen Ethernet die Latenzen nur minimal über die Kabellänge bestimmt werden.

`eve` und `alice` benannt. Bei der Bezeichnung `alice` handelt es sich um einem Modulvektor der Größe 10 und bei `eve` um ein einzelnes Modul.

Alle im Netzwerk genutzten Kabeltypen sind in dem Quellcode 3 dargestellt. Das ITMC ist mit einer 10 Gbit/s Leitung an das Internet angebunden, wobei dies einer Vereinfachung der doppelten 5 Gbit/s Anbindung entspricht. Eine 40 Gbit/s Leitung wird zwischen dem ITMC und IRB Netz eingesetzt. Alle Geräte im Internet nutzen eine ideale Anbindung. Bei den Verbindungen im Außennetz handelt es sich um PPP-Verbindungen.

4.2 Konfiguration des Netzwerks (ini)

Die Netzwerkkonfiguration `omnetpp.ini` ist in kleine Abschnitte eingeteilt, welche spezifische Konfigurationen enthalten. In den Abschnitten `[ServerConfig]` und `[ClientConfig]` werden die Server- und Clientknoten konfiguriert, wobei geteilte Konfigurationen der Knoten in der `[NodeConfig]` vermerkt sind. Am Ende folgen die Abschnitte der Angriffe.

4.2.1 Allgemeine Konfigurationen

In dem Abschnitt `[General]` wird die Zeit der Simulation auf 100 s begrenzt, da nur in diesem Zeitraum Anwendungsdatenverkehr erzeugt wird. Zusätzlich würden längere Simulationen die Größe der aufgezeichneten Daten und die Fähigkeiten der zur Verfügung stehenden Hardware übertreffen. Im Anschluss wird mit der Anweisung `seed-set` der Startwert des Zufallsgenerators auf 1 bis 5 in Abhängigkeit des durchgeföhrten Durchlaufs gesetzt. Danach können die Ordner für aufgezeichneten Daten in diesem Abschnitt und die Aufzeichnungsfähigkeiten der Knoten angepasst werden.

4.2.2 Konfiguration der einfachen Netzwerkmodule

Module	Parameter	Wert
ARP	retryTimeout	1 s
	retryCount	3
	cacheTimeout	180 s
EthernetInterface	num[Out/In]putHooks	1
	[out/in]putHook	"ThruputMeter"
	queueType	"DropTailQueue"
TCP	nagleEnabled	true
	limitedTransmitEnabled	true
	increasedIWEEnabled	true
	windowScalingSupport	true
	timestampSupport	true
	tcpAlgorithmClass	"TCPNewReno"
	sackSupport	false
	delayedAcksEnabled	true

Tabelle 4.1: Modifizierte Parameter mit zugewiesenen Wert der Netzwerkmodule

Die meisten Module werden in ihrer Standardkonfiguration betrieben, wobei abgeänderte Parameterwerte in der Tabelle 4.1 zusammengefasst sind.

ARP-Modul

Außer bei Durchführung des ARP-Angriffs sind die Werte für das ARP-Modul dem echten Netzwerk nachempfunden. Praktische Bedeutung hat der *cacheTimeout* in der aktuellen Simulationskonfiguration nicht, da diese deutlich kürzer läuft. Um einen Cachetimout zu beobachten, wird bei dem ARP-Angriff ein Timeout von 30s genutzt. Das ARP Modul ist zusätzlich so konfiguriert, dass es drei ARP-Anfrageversuche durchführt und die Wartezeit dazwischen jeweils 1s beträgt.

Ethernet-Modul

Das zusammengesetzte Modul `EthernetInterface` ist so konfiguriert, dass ein- und ausgehender Netzwerkverkehr mit dem Modul `ThruputMeter` gemessen wird. Da der *csmacd*-Parameter nicht gesetzt wurde, benutzt die Ethernet-Implementierung eine Voll-Duplex-Verbindung, bei der Überlasten in einer "*DropTailQueue*" zwischengespeichert werden.

TCP-Modul

Die Konfiguration des TCP-Moduls folgt den Standardeinstellungen des Linux TCP-Stapels [28], da große Teile der Netzwerkgeräte der Fakultät für Informatik mit Linux betrieben werden. Im Gegensatz zu der Konfiguration in Linux, werden selektive Bestätigungen (Selective Acknowledgment [14]), über den Parameter *sackSupport* gesteuert, deaktiviert, da andernfalls die Simulation nach wenigen Events abstürzt². Der Grund für den Absturz ist eine fehlgeschlagene Assertion in der TCP-Implementierung. Zu beachten ist, dass sowohl die Implementierungen in INET als auch im Linux Kernel Abweichungen zum RFC-Standard beinhalten. In der INET-Implementierung des Moduls in der Datei *TCP.ned* wird auf die Unterschiede hingewiesen.

4.2.3 Konfigurationen des Internet-Cloud-Moduls

Das Modul `InternetCloud` wird genutzt, um Datenpakete über das Internet zu verzögern und einen leichten Paketverlust zu simulieren. Die beiden Funktionen `truncnormal` und `uniform` werden genutzt um Zahlen aus einer positiven Normalverteilung und Gleichverteilung zu generieren.

Durch die Normalverteilung `5ms + truncnormal(10ms, 10ms)` wird eine durchschnittliche Verzögerung von ungefähr 15 ms erreicht, womit die Round Trip Time (RTT) bei 30 ms mit starken Schwankungen liegen sollte. Der minimal Wert von 5 ms wurde wegen eines durchgeführten Pingtest A aus dem Fakultätsnetzwerk zu dem Server 8.8.8.8³ gewählt. Es wurde eine zusätzliche Verzögerung von durchschnittlich 10 ms mit einer Standardabweichung von 10 ms hinzugefügt, um schlechtere Verbindungen zu simulieren. Es kann davon ausgegangen werden, dass die meisten Geräte schlechtere Anbindungen als Google besitzen und dass ICMP Datenverkehr bevorzugt behandelt wird.

Ein leichter Paketverlust wird durch die Zufallsverteilung `uniform(0, 10000) < 1` erreicht. Genaue Werte für einen Paketverlust liegen nicht vor. Daher wird diese Modellierung genutzt, die im Durchschnitt 0,1 % der Datenpakete nicht weiterleitet.

Bereich	Anwendung	Anzahl
Server	TCP	50 (alice) 30(irb)
	UDP	0
	Videostream	50 (alice) 20 (irb)
	Ping	0
Host	TCP	2
	UDP	1
	Videostream	1
	Ping	je 10 Server 2

Tabelle 4.2: Anzahl an Anwendungsmodulen je Host oder Server

4.2.4 Konfigurationen der Anwendungsmodule

Der Netzwerkverkehr wird von den Anwendungsmodulen erzeugt, von den es sieben unterschiedliche Grundtypen im Netzwerk gibt. In der Tabelle 2.2 werden die wichtigsten Module vorgestellt. Nur Serveranwendungen sind exklusiv in Modulen aus dem Serverbereich konfiguriert. Die jeweiligen Netzbereiche werden durch ihre Namen differenziert. Alle Geräte, welche mit ***Host*** identifiziert werden können, sind Clientgeräte. Server werden durch ***Server***⁴ identifiziert.

Da in jedem Netzwerkbereich unterschiedlich viele Anwendungsmodule eingesetzt werden, ist die genaue Anzahl pro **StandardHost** in den jeweiligen Bereichen in der Tabelle 4.2 vermerkt. Dabei besitzen Server mehr Anwendungsmodule pro Gerät, da es insgesamt weniger Server gibt.

Eine Ausnahme in der Konfiguration bildet das Modul **eve**, welches ein Modul **TCPGenericSrvApp** besitzt. Dies garantiert das Vorhandensein eines TCP-Protokollstacks, wodurch mit SEA++ Angriffe von diesem Knoten ausgeführt werden können.

TCP-Datenverkehr

App	Parameter	Wert
TCPGenericSrvApp	localPort	80-93
	connectAddress	irbServer[*]/aliceServer[*]
	connectPort	intuniform(80,93)
	requestLength	Unterschiedlich
	replyLength	Unterschiedlich
	numRequestsPerSession	min(10, truncnormal(1, 2))
	idleInterval	min(100ms, truncnormal(10ms, 50ms))
	thinkTime	min(2s, truncnormal(10ms, 250ms))
	dataTransferMode	"object"
	startTime	Unterschiedlich

Tabelle 4.3: Übersicht der Parameter von den TCP-Anwendungen

²Mit der Konfiguration **[KeinAngriffSackCrash]** kann der Absturz reproduziert werden.

³Bei 8.8.8.8 handelt es sich um einen DNS Server von Google [6].

⁴Die Vektoren **aliceServer[]** und **irbServer[]** sind die einzigen Server in der Simulation.

In der Tabelle 4.3 sind alle genutzten Parameter für die Server- und Clientgeräte dargestellt. Auf den Servern werden jeweils die ersten 13 TCP-Anwendungen mit dem Modul `TCPGenericSrvApp` belegt und an die Ports 80 bis 93 gebunden.

Die `thinkTime` und `idleInterval` geben die Wartezeiten zwischen den Anfragen und dem Senden einzelner Segmente an. Beide Parameter werden für alle Anwendungen mit Zufallsvariablen initialisiert. Die Wartezeit zwischen zwei Anfragen ist maximal 2 s lang und beträgt im Durchschnitt 10 ms mit einer großen Standardabweichung von 250 ms. Zwischen zwei Datenpaketen liegt eine Wartezeit von maximal 10 ms bei einem gleichen Durchschnitt mit geringerer Standardabweichung von 50 ms. Beide Werte können nur ein grobes Modell von tatsächlichen TCP-Anwendungsdatenverkehr erreichen, da die Charakteristika für jede Anwendung und jedes Netzwerk unterschiedlich sind.

Die Anfrageanzahl wird durch die Verteilung `min(10, truncnormal(1, 2))` für alle Geräte festgelegt, was dem Verhalten neuerer Hypertext Transfer Protocol (HTTP) Versionen⁵ entspricht. Der `dataTransferMode` wird auf `object` gesetzt, damit die Server jeweils eine Antwort der Länge `replyLength` verschicken. Mit diesen und dem Parameter `requestLength` können Up- und Downloaddatenströme mit TCP modelliert werden. Ist die Anfrage größer als die Antwort, so wird eine Datei hochgeladen und anders herum. Für die Server wird jeweils von Downloaddatenverkehr ausgegangen, welcher mit der Zufallsverteilung `min(1MiB, 20B + truncnormal(500B, 500B))` für Anfragen und `min(16MiB, 20B + truncnormal(2MiB, 2MiB))` für Antworten modelliert wird. Für den Download wurde der Mittelwert von 2 MB entsprechend der Durchschnittsgröße einer Website [32] im Jahr 2019 gewählt.

Im Clientbereich werden die beiden TCP-Anwendungen unterschiedlich konfiguriert. Die erste Anwendung folgt der Konfiguration der Server und verbindet sich mit den IRB-Servern. Die zweite TCP-Anwendung nutzt für Anfrage und Antwort die Verteilung `min(16MiB, 20B + truncnormal(2MiB, 2MiB))`, um einen gemischten Netzwerkverkehr zu erzeugen. Diese Datenverkehrsmodellierung wird genutzt, da die zweite Anwendung sich mit den Internetservern (alice) verbindet und in diesem Szenario Up- und Downloads typisch sind. Im Serverbereich verbinden sich die Anwendungen mit den Geräten aus dem jeweiligen anderen Bereich. Um jedem Gerät im Vektor einen anderen Verbindungspartner zuzuweisen, wird eine Kombination aus Funktionen genutzt. Der Parameter `choose(intuniform(0, 9), moduleListByPath("*.<Serverbereich>[*]"))` besteht aus den Funktionen `choose`, `intuniform` und `moduleListByPath`. In Kombination wählt `choose` aus der Liste an Geräten, welche `moduleListByPath` mit dem Ausdruck `*.<Serverbereich>/[]` findet, einen zufälligen Index durch `intuniform` aus.

Mit einer Gleichverteilung werden die Verbindungsports aus dem Bereich 80 bis 93 für alle Geräte festgelegt. Die Startzeit unterscheidet sich für die unterschiedlichen Bereiche. Geräte, welche über das Internet kommunizieren, starten früher, da sonst in der Simulation in den späteren Minuten zu einem starken Anstieg der Belastung kommt. Der Grund dafür sind die Verzögerungen und Verluste im Internet, welche den Netzwerkdatenverkehr beeinflussen. Dies erschwert die Simulation des Internetdatenverkehrs, weil vor Ausführung der Simulation nicht zu erkennen ist, wie sich Änderungen auf diesen Datenverkehr auswirken. Der gewählte Parameter für den Start liegt für die Simulation bei `uniform(0s, 90s)`. Clientgeräte starten zwischen dem Anfang und Ende der Simulation.

App	Parameter	Wert
UDPBasicApp	destAddresses	*Server[*]
	destPort	intuniform(9080,9085)
	messageLength	min(508B, 8B + truncnormal(256B, 64B))
	sendInterval	min(100ms, truncnormal(50ms, 50ms))
	startTime	uniform(0s,1s)
UDPEchoApp	localPort	9080-9085

Tabelle 4.4: Übersicht der Parameter von den UDP-Anwendungen

Generischer UDP-Datenverkehr

Die geänderte Parameter der UDP-Module sind in der Tabelle 4.4 vermerkt. Module des Typs UDPEchoApp werden an die Ports 9080 bis 9085 gebunden.

Auf der Clientseite wird ein UDPBasicApp genutzt, welches zufällig an die Serverports gebunden wird. Die Länge der Nachrichten ist auf 508 B begrenzt, womit im Regelfall keine Fragmentierungen der Datagramme durchgeführt werden müssen. Im Normalfall beträgt die Größe 262 B mit einer Standardabweichung von 64 B. Die Datagramme werden maximal alle 100 ms verschickt. Durchschnittlich werden alle 50 ms mit einer Standardabweichung von 50 ms Datagramme gesendet. Alle UDPBasicApp starten in der ersten Sekunde der Simulation und verbinden sich mit den *irbServer*-Bereich.

UDP-Videodatenverkehr

App	Parameter	Wert
UDPVideoStreamCli	serverAddress	**.*Server[*]
	serverPort	intuniform(8080,8085)
	startTime	uniform(0s,90s)
UDPVideoStreamSrv	localPort	8080-8085
	videoSize	$2^{2,3,4,5,6,7}$
	sendInterval	min(100ms, truncnormal(50ms, 50ms))
	packetLen	min(508B, 128B + truncnormal(256B, 64B))

Tabelle 4.5: Übersicht der Parameter von den Videostream-Anwendungen

Die veränderten Parameter für die Videostream-Module sind in der Tabelle 4.5 zusammengefasst. Ähnlich zu den UDP-Modulen werden die Server an die Ports 8080 bis 8085 gebunden. Beide Modultypen nutzen das gleiche Intervall um Datenpakete zu versenden. Die Datagramme des Videostreamservers sind im Gegensatz zu den anderen UDP-Modulen leicht größer und liegen bei mindestens 128 B sowie einer Durchschnittsgröße von 384 B. Ein großer Unterschied zu den UDP-Modulen besteht hinsichtlich der Übertragungsdauer. Die UDP-Module senden bis zu einem festen Zeitpunkt, wohingegen das Modul UDPVideoStreamSrv eine feste zu übertragene Datengröße besitzt. Mit dem Parameter *videoSize* wird diese festgelegt und ist an das jeweilige Modul gebunden. Gleich den Ports werden den Modulen jeweils in zweifacher Potenz aufsteigende Dateigrößen zugewiesen.

⁵HTTP 1.0 nutzt für jedes Objekt eine neue Verbindung [1].

Die Videoclienten verbinden sich jeweils zu den IRB- oder Internetservern. Im Serverbereich wird jeweils eine Verbindung zu den anderen Serverbereich hergestellt und alle Clientgeräte verbinden sich mit den IRB-Servern.

ICMP-Datenverkehr

App	Parameter	Wert
*Host[*9]	destAddr	irbServer
*Host[*8]	destAddr	aliceServer
Host[]	sendInterval	1s
	startTime	exponential(1s)
	stopTime	100s

Tabelle 4.6: Übersicht der Parameter von den Ping-Anwendungen

Mit einem Ping-Modul wird ICMP-Datenverkehr erzeugt. Module dieses Typs sind in den achten und neunten Clientgeräten vorhanden und senden zu einem beliebigen IRB- und Internetserver im Intervall von einer Sekunde Ping-Nachrichten. Alle Module fangen in den ersten Sekunden an zu senden.

4.2.5 Angriffskonfigurationen

Die Angriffe werden in den Abschnitten, welche mit dem Wort „Angriff“ enden, konfiguriert, wobei alle Abschnitte ähnlich aufgebaut sind. Als Erstes wird die Konfiguration ohne Angriff erweitert. Danach wird eine kurze Beschreibung des jeweiligen Angriffs mit *description* gegeben und der String *attackConfigurationFile* auf den jeweiligen Angriff überschrieben.

4.3 Netzwerkmodelleinschränkungen

Durch die Einschränkungen des Programmökosystems weicht das Modell vom realen Netzwerk ab. Auf topologischer Ebene sind durch fehlende Redundanzen in den Geräteverbindungen und in einfachen Ausführungen ansonsten redundanter Netzwerkgeräte, Angriffe nicht zielführend, wenn diese auf den Ausfall jener Angriffspunkte abzielen. Aufgrund der Abweichungen durch die fehlende VLAN-Implementierung sind Aussagen zu Überlastungen der Lehrstuhlverbindungen nicht möglich.

Weiterhin ist das Datenverkehrsmodell durch die fehlenden Fakultäten im ITMC-Netz eingeschränkt. Überdies fehlt die WLAN-Komponente des Universitätsnetzes und eine weitgehende Internetmodellierung. Zusätzlich kann auf Anwendungsebene nur ein grobes Datenverkehrsmodell erstellt werden. Da OMNeT++ in Verbindung mit SEA++ keine gesonderten Netzwerkdatenverkehrsgeneratoren unterstützt, kann nur mit den Standardanwendungen gearbeitet werden. Insbesondere ist es aufwändig viele unterschiedliche Portbereiche zu nutzen, da diese manuell im Server- und Clientgeräten erstellt werden müssen.

Des Weiteren wird ohne Netzwerkdatenverkehrsgeneratoren die Modellanpassbarkeit auf verschiedene Übertragungsraten erschwert, was sich in der Flexibilität des Netzwerks widerspiegelt. Dadurch ist die Simulationsdauer nicht anpassbar, ohne alle Anwendungsmodulkonfigurationen zu ändern. Zuletzt fehlen Module aus dem Sicherheitsbereich, womit die Antworten der Standardmodule stark von den realen Anwendungen abweichen.

4.4 Umsetzung des ARP-Positioning-Angriffs

Im weiteren Verlauf wird die Umsetzung der Angriffe beschrieben. Für den ARP-Angriff ist eine Erweiterung von SEA++ nötig, da im Ursprungszustand ARP nicht unterstützt wird. Deshalb wird der in Abschnitt 2.2.4 beschriebene Weg, ein neues Datenpaket zu SEA++ hinzuzufügen, befolgt. Um dieser Anleitung zu folgen wird zuerst überprüft, ob eine entsprechende Datei `ARP-.msg` vorhanden ist. Unter dem Pfad `networklayer/arp/ARPPacket.msg` findet sich diese Datei.

Erweiterung der `seapputils.cc` Als Nächstes wird geprüft, ob eine Schicht für ein ARP-Datenpaket zurückgegeben wird. Die Methode `getPacketLayer(...)` in der Datei `seapputils.cc` gibt den entsprechenden Wert 2 für die Sicherungsschicht zurück.

Erweiterung der `Create.h` Um mit der Anweisung `create` ein neues ARP-Datenpaket zu erstellen, wird in der `Create.h` unter dem Pfad `/actions/create/Create.h` das Enumerate `type_t` erweitert. Der neu eingefügte Typ heißt **ARPFRAME**.

Quellcode 4 Auszug aus der Create.cc

```

1 ...
2 case 2: {
3     switch (type) {
4         ...
5         case type_t::ARPFRAME: {
6             *packet = new ARPPacket("CreatedPacket-ArpPacket", 0);
7             break;
8         ...
9         switch (type) {
10            case type_t::ARPFRAME: {
11                controlInfo = new Ieee802Ctrl();
12                (*packet)->setControlInfo(controlInfo);
13                return;
14            ...

```

Erweiterung der `Create.cc` Zuletzt muss nach Anleitung in der `Create.cc` unter dem Pfad `/actions/create/Create.cc` die Methode `buildNewPacket(...)` modifiziert werden, um die Logik für das Erstellen des Datenpakets einzufügen. Diese Modifizierung wird in dem Quellcode 4 skizziert. Im ersten Teil wird, falls ein Datenpaket des Typs **ARPFRAME** erkannt wurde, ein neues Objekt **ARPPaket** erstellt. Danach wird für dieses Objekt ein `controlInfo`-Objekt des Typs `Ieee802Ctrl` gesetzt. Sollte dieser Code fehlen, werden die Datenpakete nicht richtig von den Modulen in der Simulation behandelt, da Informationen über den Empfänger oder Sender fehlen.

Erweiterung der `Change.cc` Obwohl im Benutzerhandbuch nicht angegeben, reichen die Änderungen nicht aus, um ein funktionierendes neues Datenpaket zu erstellen. Da ein ARP-Datenpaket hauptsächlich Informationen zu IP- und MAC-Adressen enthält⁶ und diese durch die `change`-Anweisung besonders behandelt werden, muss die entsprechende

⁶Die anderen Felder werden von der ARP-Implementierung automatisch ausgefüllt.

Quellcode 5 Auszug aus der Change.cc

```

1 ...
2 else if (fieldName == "srcMACAddress" && packetClassName == "ARPPacket") {
3   if (isRandom) {
4     (check_and_cast<ARPPacket*> (encapsulatedPacket))->setSrcMACAddress(
5       MACAddress(stoull(value)));
6   }
7   else {
8     (check_and_cast<ARPPacket*> (encapsulatedPacket))->setSrcMACAddress(
9       MACAddress(value.c_str()));
10 }
11 ...

```

Klasse im Pfad */actions/change/Change.cc* so bearbeitet werden, sodass die Methode **executeOnField()** für die entsprechenden Felder einen Typecast ausführt. In Quellcode 5 ist eine solche Änderung skizziert. Es werden Abfragen für die Felder ***src&destMACAddress*** und ***src/destIPAddress*** des ARP-Datenpakets erstellt. In diesen werden die Strings zu Objekten umgewandelt, welche dann über entsprechende Methoden gesetzt werden können.

4.4.1 Implementierung des Angriffs

Mit diesen Erweiterungen kann ein ARP-Datenpaket mit ***create(PaketAddress, "MAC.type", "0040")*** erstellt werden. Die Informationen im Objekt ***controlInfo*** können nach der bekannten Syntax verändert werden.

In der Angriffsbeschreibung, dargestellt in Quellcode 6, wird die Erweiterung genutzt, um den ARP-Ablauf zu stören. Der bedingte Angriff zielt dabei auf den Server ***irbServer[0]*** ab, welche mit der Id 3 identifiziert ist und beginnt ab Sekunde 0.

Die Filterbedingung wird erfüllt, wenn das ***opcode***-Feld 1 beinhaltet und das ARP-Packet als Ursprungs-MAC-Adresse nicht die Adresse des Servers enthält. Der erste Parameter zeigt an, dass es sich um ein ARP-Antwort [20] handelt.

Sollte die Filterbedingung erfüllt sein, wird ein neues ARP-Datenpaket ***arpPacket*** erstellt und Variablen für die Ursprungs-MAC-Adresse ***srcMac***, Ursprungs-IPv4-Adresse ***srcIp*** und Ziel-IPv4-Adresse ***dstIp*** erstellt. Daraufhin werden diese Variablen mit den Werten aus dem abgefangenen ARP-Datenpaket gefüllt.

In dem neu erstellten ARP-Datenpaket wird der ***opcode*** 2 gesetzt, welcher signalisiert, dass es sich um eine ARP-Antwort [20] handelt. Für eine ARP-Antwort müssen die Adressfelder der ARP-Anfrage getauscht werden.

Um das Datenpaket über die Sicherungsschicht zu versenden, müssen die ***controlInfo***-Informationen für die Ziel-MAC-Adresse gesetzt werden. Zusätzlich wird mit dem ***etherType 2054*** signalisiert, dass es sich um ein ARP-Datenpaket [25, 24] handelt. Mit diesen Informationen wird das Datenpaket an den Netzwerklayer übergeben und das Original Datenpaket wird aus der Simulation entfernt.

Eine zweite Variante dieses Angriffs zielt auf den studentischen Bereich ab und verhält sich in allen anderen Aspekten gleich. Im Gegensatz zu der normalen Netzwerkkonfiguration, wird der ARP-Cache-Timout auf 30 Sekunden gestellt, um mehrere ARP-Auflösungen im Simulationszeitraum zu beobachten.

Quellcode 6 Angriffsdefinition für den ARP-Angriff auf den Server

```

1  list dstList = {3}
2
3  from 0 nodes in dstList do {
4      filter ("MAC. opcode" == 1 and "MAC. srcMACAddress" != "0A-AA-00-00-00-0C")
5
6      packet arpPacket
7
8      var srcMac = "Empty"
9      var srcIp = "Empty"
10     var dstIp = "Empty"
11
12     retrieve(original, "MAC. srcMACAddress", srcMac)
13     retrieve(original, "MAC. srcIPAddress", srcIp)
14     retrieve(original, "MAC. destIPAddress", dstIp)
15
16     create(arpPacket, "MAC. type", "0040")
17
18     change(arpPacket, "MAC. opcode", 2)
19     change(arpPacket, "MAC. srcMACAddress", "0A-AA-00-00-00-0C" )
20     change(arpPacket, "MAC. destMACAddress", srcMac)
21     change(arpPacket, "MAC. srcIPAddress", dstIp)
22     change(arpPacket, "MAC. destIPAddress", srcIp)
23
24     change(arpPacket, "controlInfo.src", "0A-AA-00-00-00-0C")
25     change(arpPacket, "controlInfo.dest", srcMac)
26     change(arpPacket, "controlInfo.etherType", 2054)
27
28     change(arpPacket, "sending.outputGate", "net_eth_inf$o[0]")
29
30     drop(original, 0)
31
32     send(arpPacket, 0.001)
33 }
```

4.5 Umsetzung des DoS-Angriffs

Aufgrund des rudimentären Aufbaus der Internetsimulation wird anstatt eines DDoS-Angriffs, ein DoS-Angriff implementiert, welcher im Quellcode 7 dargestellt ist .

Der Angriff wird von dem Knoten `eve` durchgeführt, welcher mit der ID 617 identifiziert ist. Nach 75 Sekunde wird jede $0,1\mu s$ ein TCP-Segment (**0010**) erstellt. Dies entspricht 1 Mp/s und liegt im Rahmen häufiger Angriffsintensitäten 3.1.2.

Mit dem `change`-Befehl wird das TCP-Segment konfiguriert. Um ein SYN-Flooding zu simulieren wird das SYN-Bit gesetzt. Alle anderen Flags verbleiben im Standardzustand. Eine zufällige Zuweisung der Sequenznummern ist nicht möglich, da SEA++ dabei einen Fehler erzeugt. Daher wird für diesen Angriff das `sequenceNo`-Feld auf den festen Wert 123 gesetzt. Dies sollte kein Unterschied für die Simulation ergeben, da der Wert von der TCP-Implementierung nicht geprüft wird. Die Bestätigungsnummern werden entsprechend dem Standard mit 0 vorkonfiguriert.

Um die Segmente versenden zu können, wird ein `controlInfo`-Objekt mit der Zieladresse und dem transportierten Protokoll angefügt. Die Zieladresse ist dabei 129.217.13.1, was dem Modul `irbServer[0]` entspricht. Da es sich um ein TCP-Segment handelt, welches von IPv4 transportiert wird, muss das `protocol`-Feld mit 6 initialisiert [25, 24] werden.

Quellcode 7 Angriffsdefinition für den großen DoS-Angriff

```

1 list dstList = {617}
2
3 from 75 every 0.0000001 do {
4   packet tcpSegment
5
6   create(tcpSegment, "TRA.type", "0010")
7
8   change(tcpSegment, "TRA.synBit", "true")
9   change(tcpSegment, "TRA.srcPort", 71)
10  change(tcpSegment, "TRA.destPort", 71)
11  change(tcpSegment, "TRA.sequenceNo", 123)
12  change(tcpSegment, "TRA.ackNo", 0)
13
14  change(tcpSegment, "controlInfo.destAddr", "129.217.13.1")
15  change(tcpSegment, "controlInfo.protocol", 6)
16
17  change(tcpSegment, "sending.outputGate", "tcp_net_inf$0[0]")
18  put(tcpSegment, dstList, TX, FALSE, 0)
19 }

```

Durch die Änderung des ***outputGates*** auf ***tcp_net_inf*** wird das Segment an die Netzwerkschicht übergeben, wenn es mit ***put*** in den Buffer gelegt wird.

Der Angriff wird in zwei zusätzlichen Abstufungen ausgeführt, welche jeweils eine Zehnerpotenz weniger Pakete pro Sekunde versendet. Für jede dieser Abstufungen existiert zusätzlich ein Szenario, bei dem der betroffene Knoten 5 Sekunden nach Beginn des Angriffs deaktiviert wird. Dies wird durch den ***destroy***-Befehl realisiert.

4.6 Umsetzung des Port-Scanners

Auf der Paketebene arbeitet ein klassischer Port-Scanner ähnlich zu einem DoS Angriff. Um ein möglichst unterschiedlichen Angriff zu erzeugen, wird ein passiver Scan durchgeführt, welcher nur auf SYN-Segmente reagiert. In dem Quellcode 8 wird die Implementierung dargestellt.

Der Angriff wird von dem Knoten mit der ID 3, welche dem ***irbServer[0]*** entspricht, ausgeführt. Bei dem Simulationsstart wird durch eine Filterbedingung geprüft, ob es sich um TCP-Segmente mit gesetzten SYN-Bit handelt, welche von Geräten mit einer anderen IP gesendet wurden. Es muss vermieden werden, dass das Segment von dem ***GlobalFilter*** erzeugt wurde, da sonst eine Schleife entsteht.

Für alle Pakete, welchen den Filter erfüllen, werden Variablen für die IPv4-Adresse und den Ursprungsport erstellt und jeweils mit den Werten aus dem abgefangenen Transportpaket und TCP-Segment ausgefüllt. Die Variablen müssen mit einem Wert initialisiert werden, da SEA++ sonst einen Fehler⁷ beim Übersetzen anzeigt.

Wie bei dem DoS-Angriff, wird ein neues TCP-Segment erstellt, bei dem anstatt des SYN-Bits das ACK-Bit gesetzt wird. Der Ursprungsport ist aufgrund einer fehlenden Zufallsportfunktion⁸ fixiert. Im Feld des Zielports wird der ausgelesene Ursprungsport genutzt. Da das Paket an die Internetschicht übergeben wird, muss ein ***controlInfo***-Objekt

⁷Error: variable 'srcPort' must be initialized - Line 16

⁸Die Zufallsfunktion für ein Shortinteger wird nicht in dem Portfeld unterstützt.

Quellcode 8 Angriffsdefinition für den passiven ACK-Scan

```

1  list dest = {3}
2  from 0 nodes in dest do {
3      filter("NET.transportProtocol" == 6 and "TRA.synBit" == true and "NET.
4      srcAddress" != "129.217.13.1" and "attackInfo.fromGlobalFilter" == 0)
5      packet tcpPacket
6
7      var srcIp = "empty"
8      var srcPort = "empty"
9
10     retrieve(original, "NET.srcAddress", srcIp)
11     retrieve(original, "TRA.srcPort", srcPort)
12
13     create(tcpPacket, "TRA.type", "0010")
14
15     change(tcpPacket, "TRA.srcPort", 45515)
16     change(tcpPacket, "TRA.destPort", srcPort)
17     change(tcpPacket, "TRA.ackBit", "true")
18     change(tcpPacket, "controlInfo.destAddr", srcIp)
19     change(tcpPacket, "controlInfo.protocol", 6)
20     change(tcpPacket, "sending.outputGate", "tcp_net_inf$0[0]")
21
22     send(tcpPacket, 0)
23 }
```

erzeugt werden und mit der Zieladresse und dem zu transportierenden Protokoll ausgefüllt werden.

Zum Schluss muss das ***outputGate*** wie bei dem DoS-Angriff gesetzt werden und das Paket mit ***send*** an die Internetschicht übergeben werden. Im Gegensatz zu dem DoS-Angriff, sollte dieser Angriff deutlich weniger Netzwerkverkehr generieren.

4.6.1 Fehlerbehebung

Wenn die Angriffsbeschreibung 8 ausgeführt wird, kommt es nach einer kurzen Zeit zu einem Fehler in der Simulation, welcher die Ausführung stoppt. Dieser Fehler kann auf die Nutzung von einem bedingten Angriff zurückgeführt werden. Für die Filterbedingung wird jede Nachricht, welche den **LocalFilter** passiert, geprüft. Bei der Prüfung wird die Schicht der Nachricht durch Auslesen der **packetClassName** festgestellt. Sollte das Paket nicht bekannt sein, führt dies zu einem Fehler in der Simulation.

In der aktuellen Konfiguration der Simulation werden drei, für SEA++ unbekannte Datenpakete, mit den Klassennamen erstellt:

1. **GenericAppMsg**,
2. **PingPayload**,
3. **ICMPMessage**

Diese Datenpakete müssen eine Schicht zurückgeben. Daher wird die *seapputils.cc* in der **getPacketLayer**-Methode um drei Konditionalabfragen für die jeweiligen Pakete erweitert.

Kapitel 5

Auswertung

In diesem Kapitel sollen die Ergebnisse vorgestellt werden, welche aus dem Modell gewonnen werden konnten. Nach der Betrachtung der Ergebnisse, werden die Probleme, welche bei der Arbeit mit SEA++ entstanden sind, aufgezeigt.

5.1 Ergebnisse

Bei der Ergebnisanalyse wird vorrangig auf die Paketdurchsätze eingegangen, welche sich je nach Angriff verändern könnten. Falls weitere Daten durch den Dienst gespeichert werden, welcher vom Angriff ausgenutzt wird, werden diese zusätzlich betrachtet. Sollte der Paketdurchsatz kein eindeutiges Bild zeigen, wird auf die Ping-Module zurückgegriffen.

5.1.1 ARP-Poisoning

Der ARP-Angriff wurde in zwei unterschiedlichen Varianten durchgeführt, welche sich in dem Ziel des Angriffs unterscheiden.

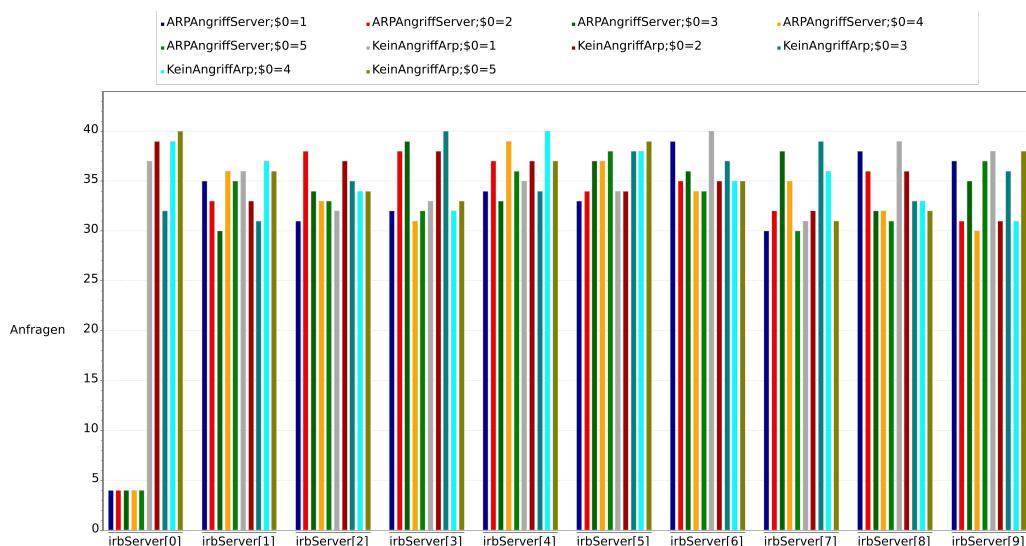


Abbildung 5.1: Vollständige ARP-Auflösungen im Serverbereich der IRB

In der Grafik 5.1 werden die vollständig beendeten ARP-Auflösungen im Serverbereich für jeden Server und den fünf Durchläufen ohne und mit Angriff betrachtet. Besonders

auffällig ist hierbei der erste Server, von welchem der Angriff ausgeht. In den fünf Ausführungen der Simulation mit Angriff werden jeweils vier ARP-Auflösungen beendet, was einen starken Rückgang zu den sonst 32 bis 40 Auflösungen darstellt. Unter Beachtung der initiierten ARP-Auslösungen E.1.2, kann festgestellt werden, dass bekannte Einträge im ARP-Cache während des Angriffs in unter 30 s genutzt werden, so dass diese nicht gelöscht werden. Bei allen anderen Servern kann kein eindeutiges abweichendes Verhalten festgestellt werden. Weitere Erkenntnisse können durch die Betrachtung des Netzwerkdatenverkehrs gewonnen werden.

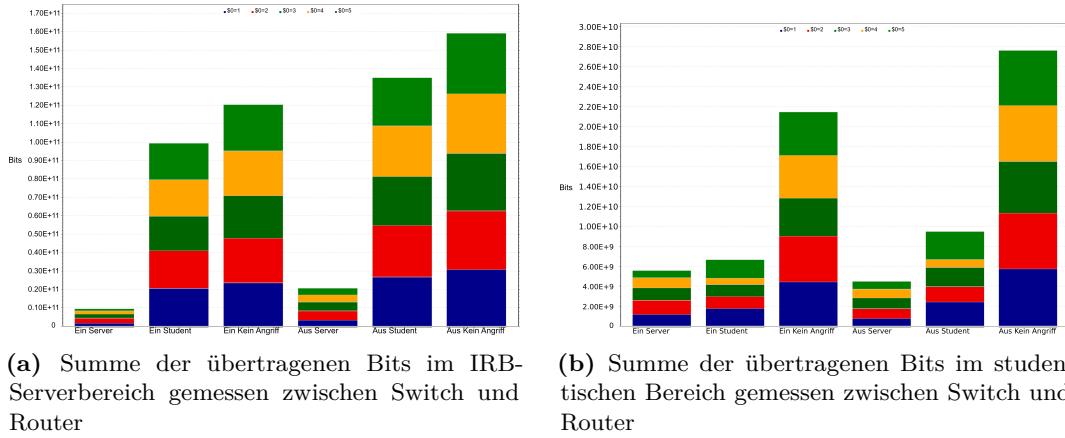


Abbildung 5.2: Summe der übertragenen Bits im studentischen Bereich und Serverbereich

In Abbildung 5.2a wird der Datenverkehr zwischen dem IRB-Router und IRB-Switch dargestellt. Dies stellt die gesamte Anzahl an übertragenen Bits zwischen den Servern und dem Restnetz dar. Es wird unterschieden in ein- und ausgehenden Datenverkehr. Die fünf Durchläufe sind jeweils in den Farben blau, rot, dunkelgrün, gelb und hellgrün als Balken übereinander dargestellt.

Der erste und vierte Balken geben jeweils den ein- und ausgehenden Datenverkehr des ARP-Angriffs auf den Server an und sind im summierten Netzwerkverkehr der fünf Durchläufe etwa 90 % niedriger, als in einem normalen Durchlauf.

In der Angriffskonfiguration auf den studentischen Bereich, dargestellt in dem zweiten und fünften Balken, sind die Auswirkungen geringer. Der kumulierte Datenverkehr sinkt um ungefähr 19 %.

Es ist ein unterschiedliches Ergebnis zu beobachten in Abbildung 5.2b. Hier sind die Auswirkungen beider Angriffsvarianten zu erkennen. In der Summe sind die Folgen des Angriffs auf den Serverbereich, mit Ausnahme des ersten Durchlaufs, im ein- und ausgehenden Datenverkehr größer. Dort ist der eingehende Datenverkehr im Serverangriffsszenario größer als im studentischen Szenario. Nach der Betrachtung des gesamten Netzwerksegments, werden die einzelnen Geräte untersucht.

Die Grafik 5.3 zeigt die Summe des ein- und ausgehenden Datenverkehrs der IRB-Server, mit farblich übereinander gestapelten Durchläufen. Alle Server außer dem angreifenden Server (0) zeigen weniger Netzwerkaktivität. Nur im zweiten Durchlauf (rot) ist eine geringere Aktivität von Server zwei zu erkennen. Begründet werden kann dies mit der Implementierung des Angriffs, welcher leicht verzögert mit einer gefälschten ARP-Antwort reagiert. Falls die Originalantwort später eintrifft, überschreibt diese die manipulierte Antwort und die Kommunikation wird ermöglicht.

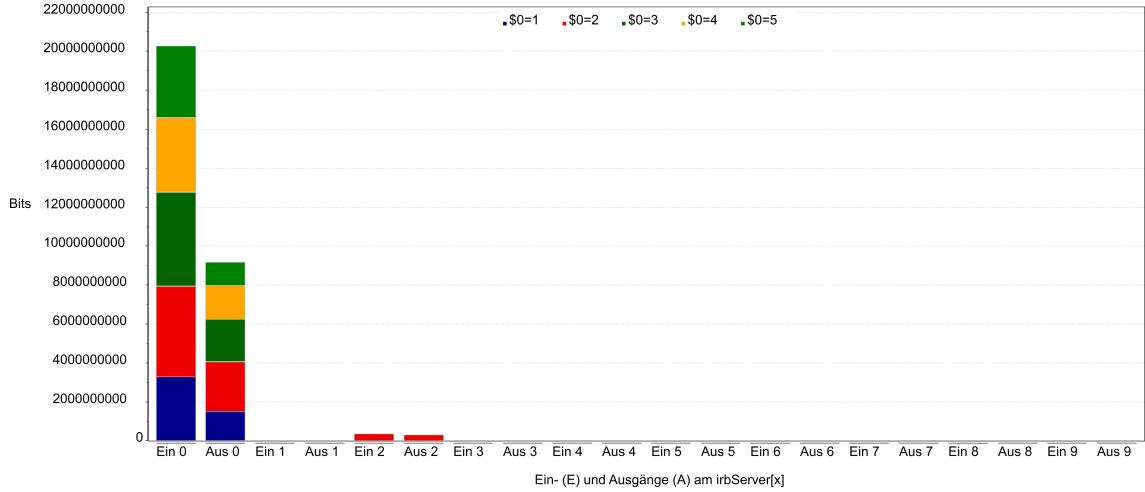


Abbildung 5.3: Der ein- und ausgehende Datenverkehr aller IRB-Server bei Ausführung des ARP-Angriffs

Die gleichen Beobachtungen E.1.1 können beim Angriff auf den studentischen Bereich gemacht werden. Als Nächstes wird der gesamte Netzwerkdatenverkehr am IRB-Router betrachtet.

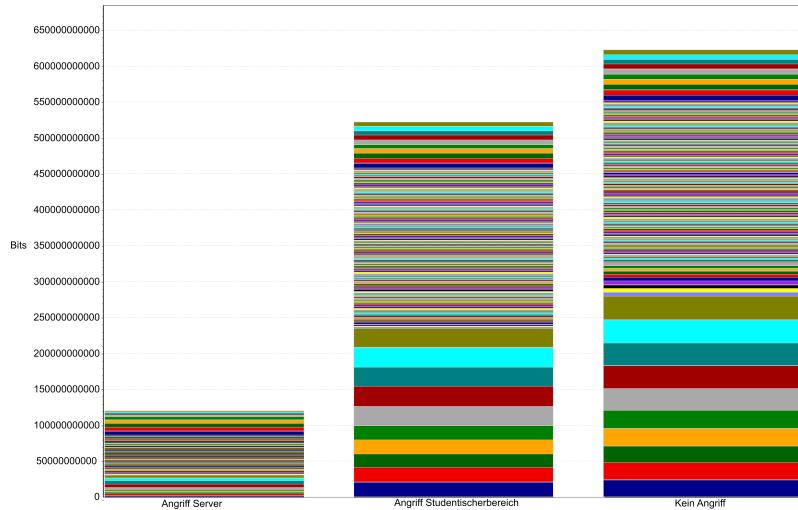


Abbildung 5.4: Gesamter Datenverkehr am Router mit und ohne ARP-Angriff

In der Abbildung 5.4 werden die aufsummierten Netzwerkdaten der fünf Wiederholungen von den drei unterschiedlichen Angriffskonfigurationen dargestellt. Da OMNeT++ nicht genug unterschiedliche Farben zur Unterscheidung der Ports zur Verfügung stellt, werden diese im einzelnen nicht genauer betrachtet. Im Vergleich der Konfigurationen (Balken), ist zu erkennen, dass bei dem ARP-Poisoning im Serverbereich der Netzwerkverkehr um ca. 80% einbricht. Dies zeigt eine Störung des gesamten Netzwerks.

Wenn der normale Durchlauf mit dem ARP-Angriff auf den studentischen Bereich verglichen wird, kann ein Rückgang von ca. 16% festgestellt werden. Dies ist mit ca. 15 % der Netzwerkgeräteanteil am Gesamtnetzwerk. Erwartungsgemäß sind die Auswirkungen eines Angriffs auf einen wichtigeren Teil des Netzwerks größer.

Im Bezug auf den Netzwerkverkehr kann zusammenfassend geschlossen werden, dass der ARP-Angriff fatale Folgen für den betroffenen Netzwerkbereich hat. Wenn viele Verbindungen auf diesen Bereich aufbauen, wie in dieser Simulation der Serverbereich, kann das ganze Netzwerk gestört werden. Im besten Fall fällt ein Teilbereich des Netzwerks vollständig aus. Eine Betrachtung von weiteren Parametern ist aufgrund der Deutlichkeit der Ergebnisse nicht nötig.

5.1.2 DoS-Angriff

Die DoS-Angriffe wurden in drei unterschiedlichen Intensitäten und zwei verschiedenen Konfigurationen durchgeführt. Im Folgenden werden die Intensitäten mit klein, mittel und groß sowie die unterschiedlichen Konfigurationen als DoS mit und ohne Ausfall des Servers bezeichnet.

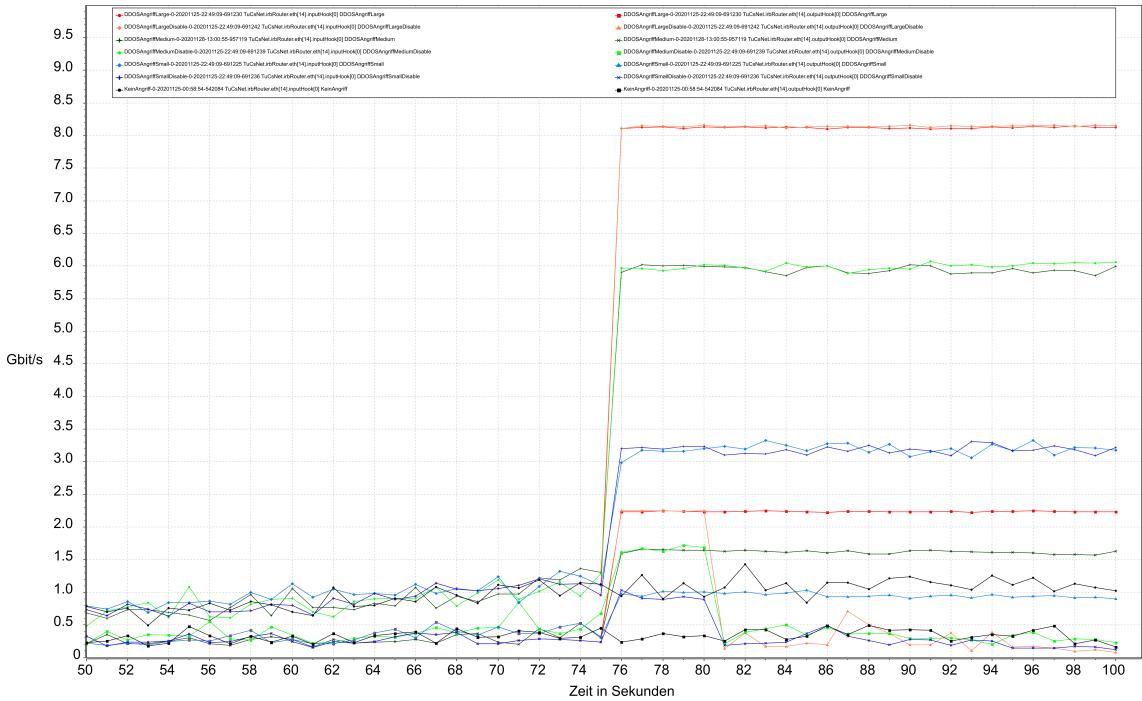


Abbildung 5.5: Durchschnittliche ein- und ausgehende Datenrate pro Sekunde am IRB-Router unter Beobachtung der einzelnen DoS-Angriffe

Abbildung 5.5 stellt den Netzwerkdurchsatz aller DoS-Angriffe dar. Es wird zwischen ein- und ausgehendem Durchsatz unterschieden. Alle Angriffe zeigen deutliche Auswirkungen auf den Netzwerddatenverkehr. Die Angriffsvariante mit einer großen Intensität erzeugt 8 Gbit/s eingehenden und 2,25 Gbit/s ausgehenden Datenverkehr inklusive der ungefähr 1 Gbit/s, welche im Normalzustand zustande kommen.

Ein Unterschied zwischen der Variante mit dem Ausfall des Servers ist im ausgehenden Datenverkehr zu erkennen. Dieser fällt bei der Deaktivierung des Servers auf ein normales Level zurück. Analog kann diese Beobachtung für alle Angriffsintensitäten gemacht werden. Es ist zu vermuten, dass der ausgehende Datenverkehr aus Bestätigungssegmente für den Verbindungsaufbau besteht.

Der kleine und mittlere Angriff erzeugen eine niedrigere Belastung für das Netzwerk. Im eingehenden Datenverkehr können 6 Gbit/s für den mittleren Angriff und 3,2 Gbit/s für den kleinen Angriff beobachtet werden. Der ausgehende Datenverkehr ist analog reduziert.

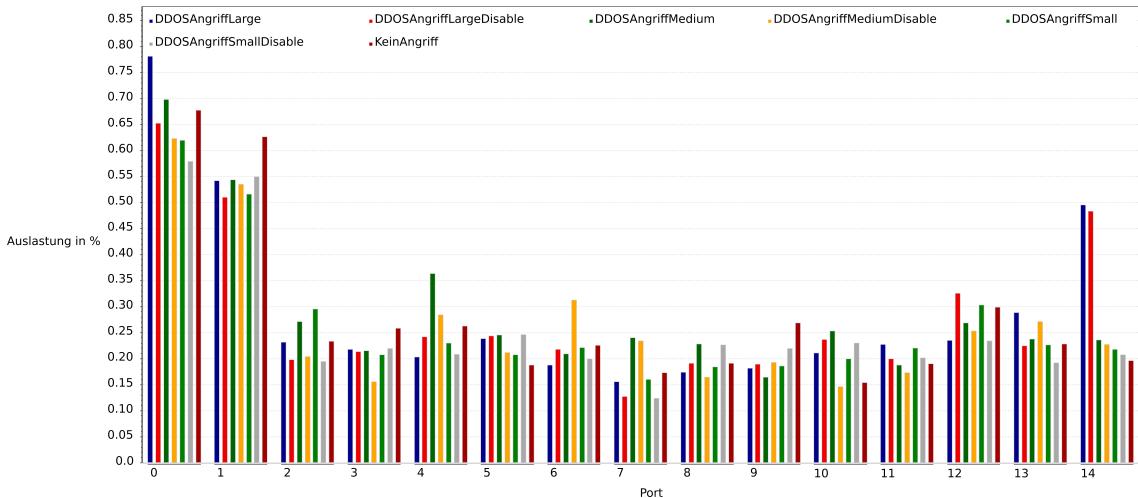


Abbildung 5.6: Auslastung der Verbindungen am IRB-Router mit und ohne DoS-Angriff

Bei den Datenraten der Angriffe ist zu vermuten, dass das Netzwerk seine Funktionalität beibehält und nur bei bedeutend größeren Angriffen zusammenbricht. Um dieser Vermutung nachzugehen wird die Auslastung am Router betrachtet. In Abbildung 5.6 wird diese für alle Konfigurationen außer denen mit ausgefallenen Servern betrachtet. Die jeweiligen Durchläufe sind farbig gekennzeichnet und nach Nummer des Port gruppiert.

In keiner Konfiguration ist ein Router Port überlastet. Die höchste Auslastung kommt am Port 0 mit 80% zustande. Es sei angemerkt, dass die Verbindungskapazitäten der einzelnen Ports unterschiedlich sind, und damit der Balken von Port 0 und 14 eine höhere absolute Kapazität aufweisen, als die restlichen Ports.

Die 80 % Auslastung des Ports kann von außen nur minimal erhöht werden. Um eine Überlastung der Router-Ports zu simulieren, müsste die restliche Universität Daten produzieren und mit der Informatik austauschen. Da dies in dieser Simulation nicht geschieht, kann maximal die Universitätsanbindung ausgelastet werden, womit 10 Gbit/s möglich sind, was unterhalb der zentralen Verbindungen des Fakultätsnetzes liegt.

Da festgestellt wurde, dass die simulierten Angriffe nicht reichen, um das Netzwerk zu überlasten, wird erwartet, dass die Auswirkungen des DoS-Angriffs auf die restlichen Teile des Netzwerkes sich nicht in der Datenübertragung widerspiegeln.

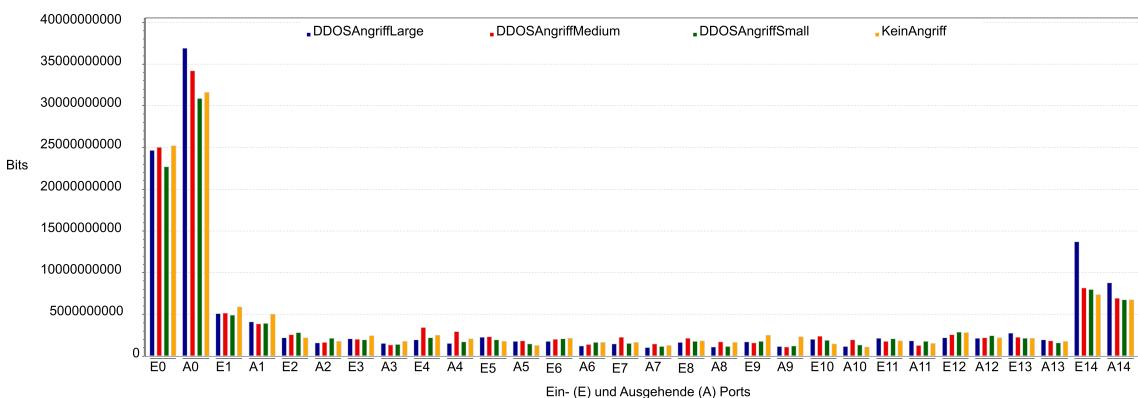


Abbildung 5.7: Summe aller übertragenen Bits der einzelnen Teilnetze am IRB Router bei Ausführung der drei DoS Angriffe und einem Durchlauf ohne Angriff

Um dies zu zeigen, werden die gesamt übertragenen Bits an den jeweiligen Router Ports genutzt. In Abbildung 5.7 werden diese für die Angriffe ohne Ausfall des Servers dargestellt. Die Ergebnisse sind analog zu der vollständigen Darstellung, welche sich im Anhang E.2 befindet.

Eine Auffälligkeit ergibt sich zwischen dem Durchlauf ohne und dem Durchlauf mit kleinem Angriff. Obwohl der Netzwerkverkehr bei letzterem höher liegen sollte, ist dies nicht in der Darstellung zu erkennen. Eine Erklärung für dieses Verhalten könnte eine unterschiedliche Reihenfolge der Zufallsvariablen sein. Auf diesen Punkt wird in Abschnitt 5.2.1 genauer eingegangen.

Abgesehen von den Ports zum Internet und zu den Servern kann in der Grafik 5.7 kein großer Unterschied bei dem Netzwerkverkehr der Lehrstühle festgestellt werden. Dies deutetet darauf hin, dass das Gesamtnetzwerk nicht eingeschränkt ist.

Wenn die Auswirkungen des Angriffs sich nicht in einer Reduzierung des Datenverkehrs widerspiegeln, können trotzdem Quality-of-Service-Parameter betroffen sein. Um dies zu untersuchen wird der Pingverlust im Internet und die RTT betrachtet.

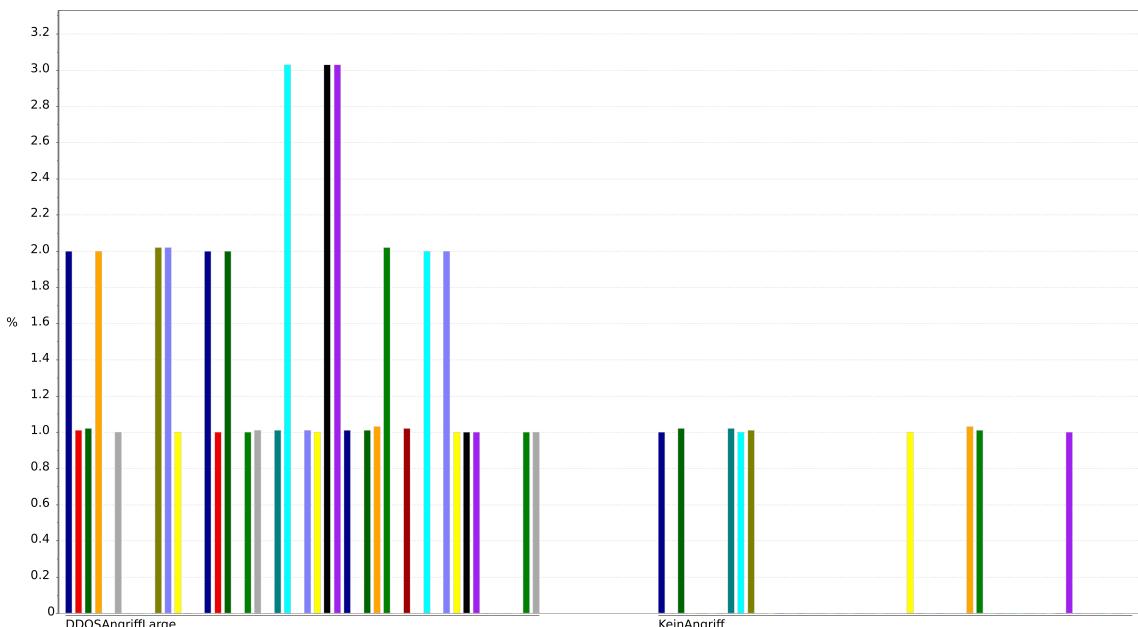


Abbildung 5.8: Pingverlust mit und ohne DoS-Angriff

Der Pingverlust wird für den großen Angriff und dem normalen Durchlauf betrachtet (Abbildung 5.8). Bei dem Vergleich der beiden Durchläufe kann ein Anstieg der Pingverlustintensität bemerkt werden. In dem normalen Durchlauf zeigen neun Module 1 % an Pingverlust. Hingegen zeigen bei dem Angriff insgesamt 29 Module einen Paketverlust mit bis zu 3 %. Es sei angemerkt, dass die Pingmodule in Sekundenintervallen Pings senden und deshalb nur relativ wenige Pings als Grundlage für diese Beobachtung dienen. Da die Simulation Schwankungen zeigt, sollten weitere Untersuchungen zum Pingverlust gemacht werden. Zuletzt wird die RTT der Module betrachtet.

In Grafik 5.9 werden die RTT für die Pingmodule, welche die Internetserver anpingen, dargestellt. Die Abbildung 5.9a stellt den großen DoS-Angriff dar und Abbildung 5.9b den normalen Durchlauf. Mit den OMNeT++-Analysemöglichkeiten kann kein Unterschied zwischen beiden Abbildungen festgestellt werden und daher kein abschließendes Fazit bezüglich der Netzwerkqualität gezogen werden. Hierfür müssten die Histogrammdaten

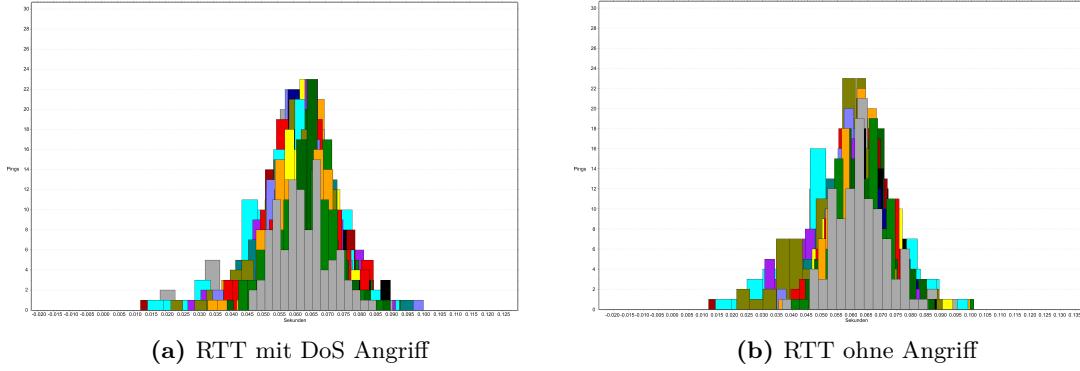


Abbildung 5.9: Vergleich der RTT zu den Internetservern mit und ohne Server

genauer betrachtet und aggregiert werden. Dies ist nicht mit der OMNeT++-IDE möglich. Der Angriff ist jedoch auf Netzwerkdatenverkehrsebene zu erkennen. Hilfreich für die weitere Analyse wären Ausführungen mit mehr Datenpaketen pro Sekunde, welche für diese Arbeit zu lange gedauert hätten.

Der DoS-Angriff zeigt Limitierungen bei der Arbeit mit OMNeT++ und SEA++. Insbesondere bessere TCP-Anwendungen könnten weitere Daten liefern, da so nur ein grobes Modell eines realen Ausfalls durch Deaktivierung des Knotens simuliert werden konnte. Eine genaue Betrachtung der Probleme folgt im zweiten Teil 5.2.

5.1.3 Port-Scanner

Für den Port-Scan werden jeweils fünf Durchläufe ohne und mit Angriff betrachtet, um Veränderungen im Verhalten des Netzes zu erkennen.

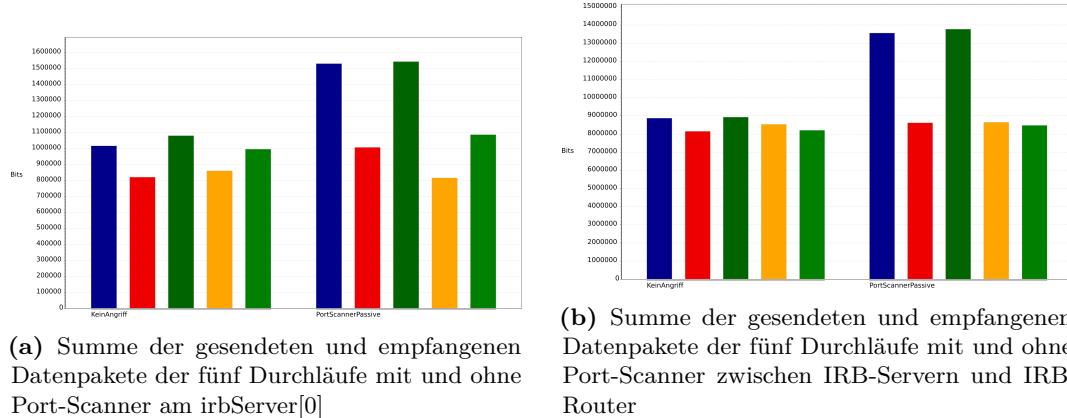


Abbildung 5.10: Summe der gesendeten und empfangenen Datenpakete der fünf Durchläufe mit und ohne Port-Scanner am Server und dem gesamten Serverbereich

Zuerst werden die gesendeten und empfangenen Datenpakete des ersten Servers, welcher den Angriff ausführt, in Abbildung 5.10a betrachtet. In den fünf Durchläufen produziert der Port-Scan-Angriff ein viermal höheres Paketaufkommen. Insbesondere bei dem ersten (blau) und dritten Durchlauf (dunkelgrün), fallen mehr Datenpakete an.

Da der Durchlauf mit Port-Scanner immer mehr Datenpakete erzeugen sollte, kann der Durchlauf vier (gelb), in dem weniger Datenpakete übertragen wurden, nicht erklärt werden. Ein Problem mit der Reihenfolge der Zufallsvariablen scheint sich wie in Abschnitt 5.1.2

zu wiederholen. Diese Vermutung wird bestärkt durch den stark erhöhten Paketdurchsatz des ersten und dritten Durchlaufs. Durch diese Unstimmigkeit lässt sich kein klarer Trend bei der Grafik 5.10a erkennen.

Werden die akkumulierten Datenpakete aller Server bei der Verbindung zwischen Router und IRB-Switch betrachtet (5.10b), wiederholt sich das Bild, dass der erste und dritte Durchlauf mehr Datenpakete produzieren bei dem Port-Scanner. Die Anomalie, dass der vierte Durchlauf weniger Datenpakete im Vergleich zum Durchlauf ohne Angriff erzeugt, ist nicht mehr auffindbar. Dies ist ein weiteres Indiz dafür, dass die Zufallsverteilung anders abgelaufen ist und daher das Netzwerk anders konfiguriert wurde.

Der gesamte Serverbereich überträgt abgesehen von dem ersten und dritten Durchlauf zwischen 1 % und 5 % Datenpakete mehr.

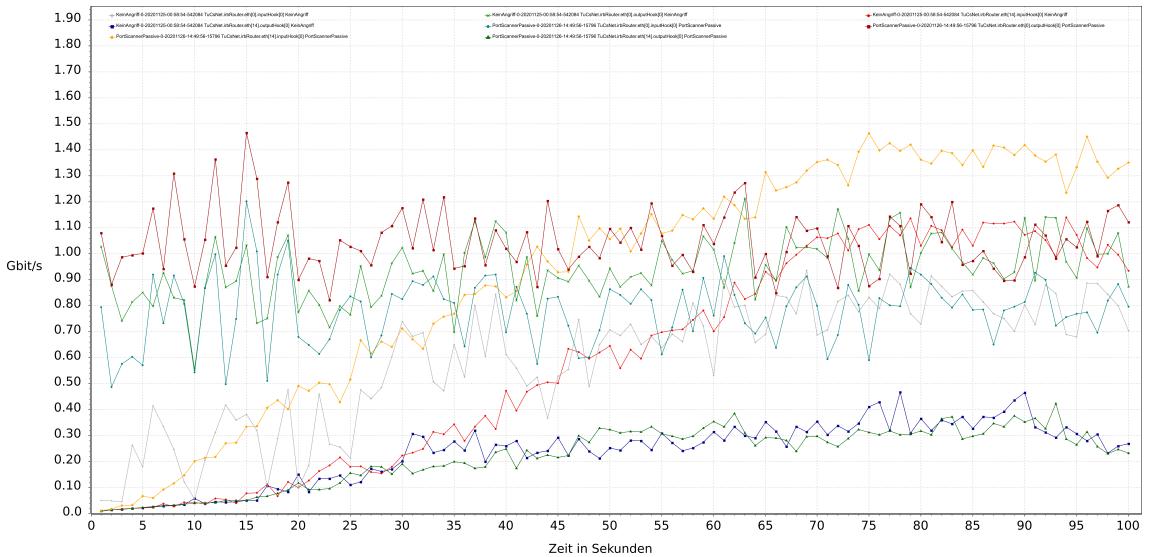


Abbildung 5.11: Durchschnittliche Datenrate der ein- und ausgehenden Verbindungen an den Ports zum Internet und Serverbereich des IRB-Routers

In der Grafik 5.11 wird der durchschnittliche ein- und ausgehende Netzwerkdurchsatz pro Sekunde an dem zentralen Router im Mittel aller Durchläufe betrachtet. Die Verbindungen zu den Lehrstühlen wurden aus Übersichtlichkeitsgründen nicht dargestellt. Der erhöhte Datenverkehr bei dem ersten und dritten Durchlauf zeigt sich in einem erhöhten eingehenden Netzwerkdurchsatz am Port 14. Dies ist zu erkennen an der gelben und roten Kurve, welche etwa 20 Mbit/s voneinander abweichen.

Außer dieser Abweichung kann bei Betrachtung der jeweiligen Verläufe der Kurven, festgestellt werden, dass die Abweichungen zwischen Normal- und Angriffsdurchlauf minimal sind. Beide Durchläufe liegen teilweise über oder unter der jeweils anderen in ihrer Durchschnittsdatenrate. Durch leichte Verzögerungen können Datenpakete bei einem Beobachtungsfenster von einer Sekunde bereits viele Datenpakete in ein anderes Zeitfenster fallen und somit dieses Bild erzeugen.

Insgesamt kann kein eindeutiger Schluss aus dem Netzwerkverkehr gezogen werden. Für die weitere Analyse wäre ein Zugriff auf die gesendeten ACK- und empfangenen RST-Segmente interessant. Allerdings werden ohne eine Modifikation des TCP-Moduls keine Skalarwerte für die empfangenen RST-Segmente gespeichert. Darüber hinaus werden die

ACK-Segmente nur in Vektoren gespeichert, was die Analyse bei der großen Datenmenge erschwert¹ hat.

5.2 Bewertung der Arbeit mit SEA++

Die Arbeit mit SEA++ hat gezeigt, dass es mit der Erweiterung möglich ist, Angriffe in einer DSL zu formulieren. Jeder der drei durchgeführte Angriffe, hätte ohne die Erweiterung in OMNeT++ durchgeführt werden können. Hierfür hätten eigene Module implementiert werden können. Mit SEA++ ist es im Gegensatz zu einer nativen Implementierung möglich die Angriffe schneller umzusetzen und einfacher zu modifizieren. Ein gutes Beispiel dafür ist der DoS-Angriff, welcher ohne Aufwand in mehreren Varianten durchgeführt werden konnte. Änderungen am Netzwerk sind im Gegensatz zu einer Implementierung über Module nicht mehr nötig.

Im Gegensatz zu dem guten Eindruck bei der Formulierung von Angriffen, hat die Arbeit mit SEA++ einige Schwächen und Probleme gezeigt, welche im Folgenden einzeln betrachtet werden sollen.

5.2.1 Zufallszahlen

In OMNeT++ werden Zufallszahlen über einen festgelegten Hashwert generiert. Dieses Vorgehen sichert, dass bei gleichen Hashwert in jedem Durchlauf die gleiche Reihenfolge an Zufallszahlen generiert werden [11, Absatz 7.4].

Allerdings sind bei dem Vergleich zwischen kleinem und keinem DoS-Angriff und dem Port-Scanner Unstimmigkeiten aufgetreten. Der normale Ablauf hat mehr Datenpakete oder Daten erzeugt hat als das Szenario mit Angriff. Durch die gleichen Zufallszahlen und Simulationsdauer sollte dies nicht geschehen.

Bestätigt werden kann das Verhalten durch mehrere Durchläufe, welche von der gleichen Konfiguration erben. Mit den zusätzlichen Konfigurationen `KeinAngriffEmptyAttack`, `KeinAngriffLateAttack` und `KeinAngriffNoAttack`, welche jeweils eine leere, eine nach Simulationsende startende und eine nicht vorhandene XML-Angriffskonfigurationsdatei verwenden, wird ein Experiment zur Verifikation erstellt. Dabei zeigen alle Varianten (C) unterschiedliche Auslastungen der Server, obwohl kein Angriff ausgeführt oder geladen wurde. Damit ist die Abweichung gesichert, aber kein Grund für diese gefunden. Weitere Experimente sind nötig, um dieses Verhalten abschließend zu klären.

5.2.2 Dokumentation

Weder von der ASL noch von SEA++ existiert eine allumfassende Dokumentation, was dazu führt, dass die Einarbeitung in SEA++ umständlich ist. Im Moment gibt es drei Quellen für die einzelnen Aspekte von SEA++:

- das Benutzerhandbuch [22],
- ein Papier [29], indem SEA++ zuerst vorgestellt wurde
- und einem Journalbeitrag über Neuerungen im OMNeT++-Umfeld citeTiloca2019.

¹Die Analysetools von OMNeT++ stürzen bei zu großen Dateien ab. Nutzung von alternativen Analysetools wurde nicht getestet, da für diese die Daten in andere Formate konvertiert werden mussten und dies zu ähnlichen Problemen führte. Eine weitere Lösung wäre mehr RAM-Speicher im Rechner zu nutzen, da OMNeT++ einen hohen Speicherverbrauch zeigt (B).

Alle drei Quellen unterscheiden sich in Details. Beispielsweise wird in dem Benutzerhandbuch [22, Listing 3.13] ein Beispiel gegeben, indem in der Filterbedingung ***UDP.sourcePort*** als Identifikator eines Feldes genutzt wird. Dies widerspricht der Definition [30, Absatz 7.3.2.5] und kann mit der aktuellen Version von SEA++ nicht übersetzt werden.

Zusätzlich ist in allen drei Veröffentlichungen die Definition für den ***drop***-Befehl abweichend hinsichtlich der aktuellen Umsetzung² beschrieben. Der aktuelle Interpreter benötigt einen zusätzlichen Verzögerungsparameter, welche in den Veröffentlichungen nicht genannt wurde. Es ist zu vermuten, dass die Veröffentlichungen sich jeweils auf unterschiedliche Versionen von SEA++ beziehen. Da es keine Hinweise dazu gibt und keine fortlaufende Versionssummierung existiert, führen diese Ungenauigkeiten zu Problemen, die ADL nachzuvollziehen. Weitere Unschärfen in der Dokumentation sind beispielsweise die unklare Bezeichnung der ADL/ASL, welche in Kapitel 2 angemerkt wurde.

Eine zentrale Dokumentation der Komponenten, inklusive der ADL, würde dieses Problem beheben. Hierbei sollte darauf geachtet werden, eine fortlaufende Versionierung von SEA++ zu erstellen, um Funktionen, welche auf eine bestimmte Version beschränkt sind, sofort zu erkennen.

5.2.3 Analyse

Die Analyse von Angriffen ist einer der Hauptaspekte, mit welchem SEA++ beworben wird [30, Absatz 7.5]. In der aktuellen Version basieren die Analysemöglichkeiten vollständig auf OMNeT++. Dies hat den Vorteil, dass Weiterentwicklungen der OMNeT++-Analysetools gleichzeitig eine SEA++-Verbesserung bedeuten. Mit dem Versionswechsel zu Version 6 von OMNeT++ werden die Analysemöglichkeiten stark überarbeitet und auf Python umgestellt. Von dieser Änderung kann SEA++ stark profitieren, da, solange die Datenformate der Ergebnissedateien nicht geändert werden, die neuen Analysemöglichkeiten genutzt werden können. Änderungen am Analyseverhalten sind vorteilhaft, da bei dem Erstellen der Grafiken mit OMNeT++ für die Bachelorarbeit Skalierungsprobleme und Abstürze häufig vorkamen. Zusätzlich fehlen Möglichkeiten Skalarwerte effektiv zu filtern und zusammenzufassen. Des Weiteren fehlten nach dem Export bei den Grafiken die y-Achsen-Beschriftungen, sodass diese manuell hinzugefügt werden mussten.

Weiterhin sollte SEA++ eigene Daten sammeln. Beispielsweise ist es im Sicherheitskontext interessant, welche Knoten von modifizierten Datenpaketen erreicht wurden. Im Standardzustand speichert dies kein Modul. SEA++ könnte dies durch den **LocalFilter** implementieren.

5.2.4 Einschränkungen durch veraltete Programmkomponenten

Die Einschränkung von SEA++ auf ältere Versionen von INET und OMNeT++ schränkt die Aussagekraft von komplexeren Netzwerken stark ein. Technologien wie VLAN sind wichtig zur korrekten Darstellung von z. B. dem Fakultätsnetzwerk.

Zusätzlich wird die Wiederverwendbarkeit von Modellen stark eingeschränkt, da OMNeT++ und INET zwischen Version 4 und 5 beziehungsweise Version 2 und 4 einige inkompatible Änderungen erfahren haben. Beispielsweise wurde das Modul **StandardHost**³ im Anwendungsschichtbereich stark überarbeitet. Neuentwicklungen der Anwendungsmodule sind deshalb inkompatibel mit den alten INET Versionen.

²Als aktuelle Umsetzung wird die am 30. Dezember 2019 veröffentlichte Version bezeichnet.

³Aktuelle Version des Moduls: <https://doc.omnetpp.org/inet/api-current/neddoc/inet.node.inet.StandardHost.html>

Ein Update auf die neue OMNeT++-Version würde diese Probleme beheben und zusätzlich die Einrichtung und Einarbeitung durch bessere Dokumentation der neueren Versionen erleichtern.

5.2.5 Flexibilität

Der direkte Bezug auf die Parameter schränkt die Portabilität der Simulation stark ein. Konkret zeigt sich das Problem bei kleinen Änderungen an einem bestehenden Modell. Jede Änderung an der Simulation kann möglicherweise die NodeId ändern und jede Änderung kann die IP- und MAC-Adressen ändern. Dies führt leicht zu Fehlern und schränkt die Wiederverwendbarkeit des Modells stark ein.

Weiterhin werden Definitionen von Angriffen durch fehlenden Zugriff auf die aktuellen Parameter der Simulation stark eingeschränkt. Ein Beispiel hierfür wären dynamisch zugewiesene Adressen, welche nicht aus den jeweiligen Speichern der Knoten ausgelesen werden können. Das Wissen des Angreiferknotens ist für einige Angriffe relevant, um Beispielsweise bestimmte Geräte aus der ARP-Tabelle anzugreifen.

Diese Probleme könnte durch die Implementierung von Funktionen, wie beispielsweise `moduleListByPath` in OMNeT++, gelöst werden. Insbesondere eine Funktion für die Knoten-Id würde die Arbeit mit SEA++ erleichtern.

5.2.6 Modularität

Die in dieser Arbeit genutzte Netzwerktopologie spiegelt nicht den ersten Ansatz für die Modellierung wieder. Ursprünglich war geplant das Netzwerk modular aus selbst erstellten zusammengesetzten Modulen zu modellieren, welche eine bestimmte Anzahl von StandardHost zu einer Funktionseinheit zusammenfassen. Von dieser Modellierung wurde abgewichen, da kein dokumentierter Weg bestand, die einzelnen LEP in den Host dieser Funktionseinheit zusammenzufassen. Eine solche Lösung wäre vorteilhaft, da die Darstellung und Implementierung des Netzwerks dadurch erleichtert werden.

5.2.7 Erweiterbarkeit

Bei der Implementierung von Angriffen mit Filterbedingungen stürzt ein unmodifiziertes SEA++ ab, weil einige Nachrichten in dem Netzwerk nicht implementiert wurden. Dieses Verhalten führt dazu, dass SEA++ für jede Nachricht erweitert werden muss, ungeachtet davon, ob die entsprechende Nachricht abgefangen werden soll.

Zusätzlich sind nicht alle Nachrichtentypen, welche bei einer Simulation mit der genutzten INET Version auftreten können, implementiert. Erweiterungen von SEA++ sind daher ohne Modifikation nötig gewesen.

Die Erweiterbarkeit ist allerdings eingeschränkt, da nicht alle Erweiterungsdetails dokumentiert wurden. Weiterhin steigt die Quellcodekomplexität durch verschachtelte und lange Konditionalabfragen für die einzelnen Datenpakete. Weniger fest implementierte (Hardcoded) Funktionen, in denen neue Datenpakete definiert werden, könnten dieses Problem lösen. Dateien in einer strukturierten Sprache wie Extensible Markup Language (XML) könnten Definitionen für neue Datenpakete enthalten und damit eine einfachere Erweiterbarkeit von SEA++ sicherstellen.

Bei der Erweiterbarkeit ist zusätzlich ein Problem mit der Dokumentation aufgekommen, welche nicht erwähnte, dass für Adresstypen weitere Erweiterungen notwendig sind. Adresstypen, wie IPv4- oder MAC-Adressen benötigen in der aktuellen Implementierung

spezielle Methoden, welche ein Cast zu den entsprechenden C++-Typen durchführt. Ohne diese Modifizierung ist ein Setzen der Felder mit *change* nicht möglich. Weitere Programmmethoden, um solche Prozesse zu automatisieren wären vorteilhaft.

5.2.8 Einschränkungen durch fehlende Module

Ein weiteres Problem, welches im Rahmen dieser Arbeit auftrat und die Ausdrucksfähigkeit von SEA++ einschränkte, sind fehlende Module. Es fehlen zum einen Module um Netzwerk-sicherheitskomponenten wie z. B. Firewall und Paketfilter zu implementieren. Zum anderen fehlen viele Protokolle, welche im Sicherheitskontext relevant sind, wie z. B. Transport Layer Security (TLS). Das Fehlen dieser Protokolle wurde bereits von den Erstellern angemerkt [30, Absatz 7.5].

Zusätzlich sollten Module, welche typischen Netzwerkverkehr generieren, kompatibel zu SEA++ gemacht werden. Das Fehlen dieser Module, schränkt die Anpassbarkeit des in dieser Arbeit untersuchten Netzwerks stark ein.

5.2.9 Sprachumfang

Angriffe, welche beispielsweise auf bestimmten Portbereiche Segmente erzeugen, sind im Moment nur eingeschränkt zu realisieren. In einem bedingten Angriff, wie dem Port-Scanner, kann nicht generisch ein Paketarray mit einem Portbereich erstellt werden. Aus diesem Grund müsste, um einen größeren Bereich an Ports zu scannen, für jeden Port ein neues Segment mit allen Zuweisungen des Angriffs erstellt werden.

Zusätzlich wäre es bei den Port-Scanner hilfreich, Daten zu speichern, um beispielsweise bereits bekannte Server nicht erneut zu kontaktieren. Eine Erweiterung der Listendatenstruktur könnte dies realisieren.

Innerhalb einer Abfrage kann zusätzlich nicht auf ein bestimmtes Feld reagiert werden. Funktionen um Konditionalabfragen und Schleifen in einem Angriff durchzuführen, würden dieses Problem beheben. Die Komplexität der Sprache würde allerdings steigen und neue Fehlervektoren hinzukommen.

Kapitel 6

Zusammenfassung

SEA++ wurde von dessen Autoren als Möglichkeit der Cyberangriffanalyse im OMNeT++ Ökosystem vorgestellt. Dies wurde anhand eines DoS-SYN-Flooding-Angriffs, eines ARP-Poisoning-Angriffs und eines Port-Scanners untersucht. In jedem der Teilschritte zum Erstellen des Modells und der Simulation haben sich Probleme und Vorteile von SEA++ gezeigt.

Bei dem ersten Schritt der Implementierung einer Netzwerktopologie kamen Probleme bei der Nutzung des Programmökosystems auf. Wie in Absatz 5.2.6 beschrieben, konnte eine anfänglich geplante Modularität des Netzwerks nicht erreicht werden. Dies führte zu einer Einschränkung in der Darstellung des Netzwerks und zusätzlichem Implementierungsaufwand.

Ein weiteres Problem bei der Implementierung waren Einschränkungen durch den Umfang von INET in der alten Version 2.6, auf welche in Absatz 5.2.4 eingegangen wurde. Hier zeigten sich fehlende Umsetzungen von Kerntechnologien des Informatiknetzwerks, welche eine Abweichung von der realen Topologie verlangten. Trotz dieser Probleme konnte eine vereinfachte Version des Netzwerks der Fakultät für Informatik an der TU, mit geringer Einarbeitungszeit, implementiert werden.

Im nächsten Schritt, der Konfiguration des Netzwerks, sind weitere Probleme aufgetreten. Vor allem fehlende Möglichkeiten im Bereich der Anwendungen haben die Aussagekraft des Modells eingeschränkt. Absatz 5.2.8 führt insbesondere Module für Netzwerksdatenverkehr und Sicherheitsfeatures als Beispiele auf. Trotz dieser Einschränkungen konnte ein grobes Modell von dem Netzwerksdatenverkehr erstellt werden, welches wenig Flexibilität zeigte.

Nachdem die Topologie erstellt wurde, zeigten sich Probleme bei der Einarbeitung in die SEA++ Dokumentation, welche in Absatz 5.2.2 ausgeführt werden. In einigen Bereichen, wie zum Beispiel der Benennung der vorgestellten DSL oder der Notation der Felder in einem Filter, wichen die jeweiligen Veröffentlichungen voneinander ab. Insbesondere die **drop**-Anweisung war nicht in der vorgestellten Syntax nutzbar. Durch Sichtung der Implementierung konnten diese Probleme umgangen werden und die jeweiligen Angriffe in ADL-Dateien mit der ASL kurz und präzise implementiert werden.

Es sei angemerkt, dass bei der Umsetzung des Port-Scanners, welcher einen bedingten Angriff benutzte, und bei der Erweiterung von SEA++ um ARP-Datenpakete, erneut Probleme aufgetreten sind. Diese werden in Abschnitt 5.2.7 besprochen. Im Fall der ARP-Erweiterung fehlte es an Dokumentation für Adressfelder des Paketes, welche speziell behandelt werden müssen. Ein anderes Problem zeigte sich bei unbekannten Datenpaketen,

welche durch eine Filterbedingung behandelt werden und zu einem Absturz führen. Dieses Verhalten sollte in der ausgelieferten Konfiguration nicht vorhanden sein.

Die Implementierung wurde eingeschränkt durch fehlende Sprachfeatures, welche in Abschnitt 5.2.9 besprochen wurden. Im Besonderen sind hier Konstrukte zum Erstellen mehrere Datenpakete und Konditionalabfragen zu nennen. Solche Funktionalitäten würden den Umgang mit SEA++ erleichtern, waren aber bei der Definition der Angriffe nicht essenziell.

Im weiteren Verlauf zeigten sich weitere Probleme mit den Implementierungen. Die Benutzung der Knoten Id, welche von OMNeT++ als Identifikator des Knotens zugewiesen wird, erwies sich als eine problematische Abhängigkeit zur Topologie des Netzes. In Abschnitt 5.2.5 werden die daraus resultierenden Nachteile aufgezeigt. Hervorzuheben ist vor allem die fehlende Anpassbarkeit der Netzwerktopologie, bei der kleine Änderungen die Knoten Ids verschieben können.

Abschließend wurden die Ergebnisse, welche aus dem Modell gewonnen werden konnten, besprochen. Bei der Analyse sind weitere Probleme aufgetreten. Da nur die Analysemöglichkeiten vom OMNeT++ genutzt worden sind, waren feine Filterungen der Daten nicht möglich. Im Abschnitt 5.2.3 werden die Konsequenzen davon aufgeführt. Es sei insbesondere darauf hingewiesen, dass neuere OMNeT++ Versionen dieses Problem angehen.

Zuletzt wurde bei der Analyse ein ungeklärtes Problem erkannt, welches Thema des Abschnitts 5.2.1 ist. Obwohl die Zeit und der Seed der Simulation fest auf ein Datum eingeschränkt sind, geben technisch gleiche Durchläufe unterschiedliche Ergebnisse aus. Weitere Untersuchungen sind nötig, um dieses Problem abschließend zu klären.

Keines der Probleme hat die Analyse der Daten verhindert. Hauptsächlich wurde sich bei der Analyse auf den Netzwerkdurchsatz sowie die Summe der gesendeten Pakete konzentriert. Dabei konnten für alle Angriffe Ergebnisse gewonnen werden.

Die Effekte des ARP-Spoofing waren wie vermutet deutlich im Netzwerk zu erkennen. Je nach angegriffenem Bereich kann ein starker Rückgang im Netzwerkverkehr gemessen werden. Insbesondere bei dem Angriff auf den Serverbereich, ist ein Großteil der Netzwerkaktivität weggefallen. Weiterhin kann festgestellt werden, dass der Netzwerkverkehr in beide Richtungen einbricht.

Bei dem DoS-Angriff konnte kein kompletter Netzwerkausfall festgestellt werden. Obwohl die erhöhte Paketrate ab Angriffsbeginn am Router zu beobachten war, konnte kein signifikanter Rückgang im Netzwerkverkehr der Lehrstühle festgestellt werden. Aufgrund der unklarer Situation im Vergleich zum ARP-Angriff wurden der Quality-of-Service-Parameter RTT und der Pingverlust betrachtet, wobei über ersteren keine genaue Aussage ohne genauere Analyse mit erweiterten Programmen getroffen werden konnte. Ein erhöhter Pingverlust konnte beobachtet werden, sollte aber aufgrund der Problematik mit den Zufallsvariablen genauer betrachtet werden.

Der Port-Scanner hat die geringsten Auswirkungen gezeigt. Wieder wurde der Netzwerkverkehr untersucht, bei dem erneut wegen dem Problem mit den Zufallsvariablen kein eindeutiger Trend zu erkennen war. Zusätzliche Parameter des TCP-Moduls konnten nicht untersucht werden, sodass kein eindeutiges Fazit gezogen werden konnte.

Zusammenfassend kann gesagt werden, dass die Analyse mit SEA++ insbesondere für Forschergruppen mit bestehenden OMNeT++-Modellen interessant ist, welche auch entsprechend weitreichende Abläufe für eine Analyse besitzen und eigene Module für aufgekommene Probleme dieser Bachelorarbeit besitzen, wie z. B. Netzwerkdatenverkehrsgeneratoren.

Insbesondere Probleme wie Abstürze durch unbekannte Pakete in der Standardkonfiguration, der eingeschränkte Sprachumfang, die Dokumentation und Probleme bei der Modularität der Topologien schränken SEA++ im Moment ein und sollten behoben werden, damit mehr Beiträge zum SEA++-Ökosystem entstehen.

Anhang A

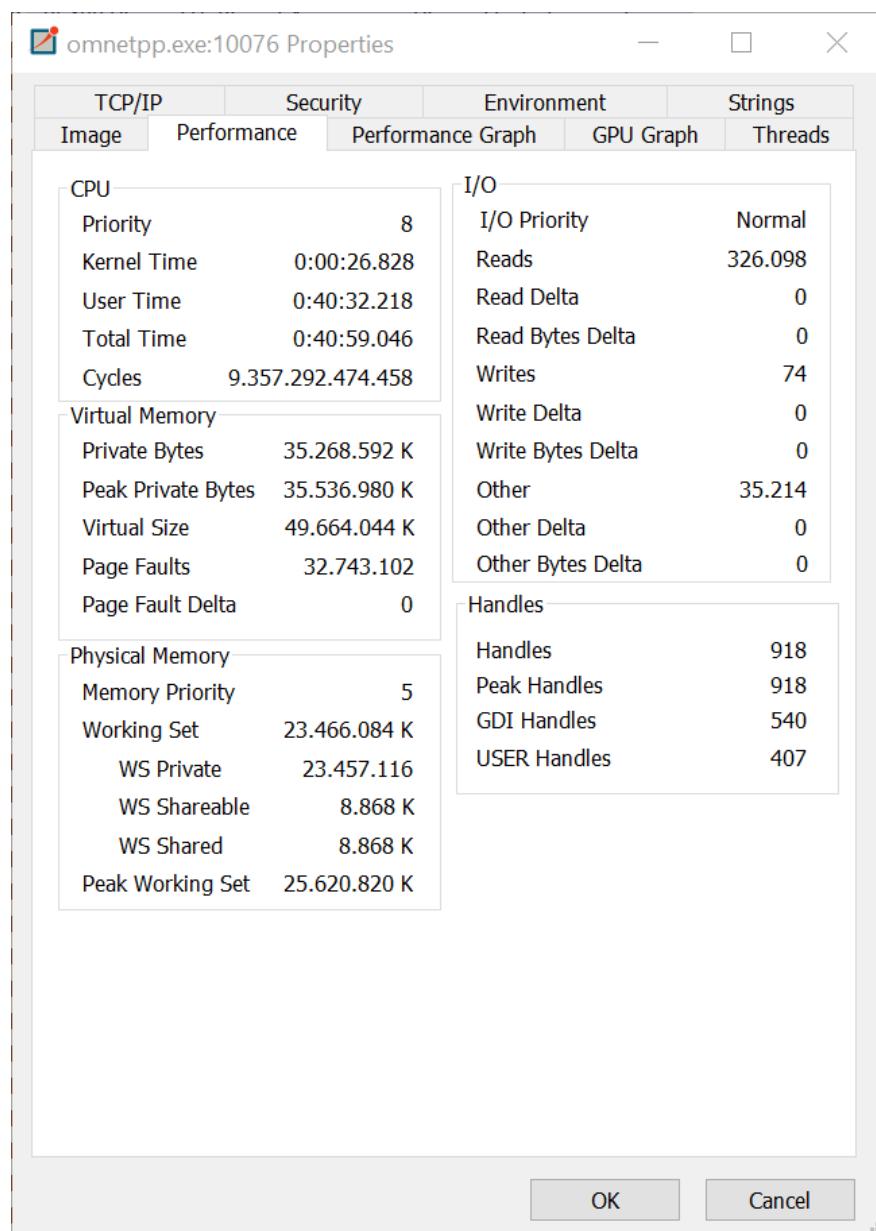
Ping-Test

Ping ausgehend von `plutonium.cs.tu-dortmund.de` zu `8.8.8.8`.

```
1 PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
2 64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=4.49 ms
3 64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=4.45 ms
4 64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=4.44 ms
5 64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=4.50 ms
6 64 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=4.45 ms
7 64 bytes from 8.8.8.8: icmp_seq=6 ttl=118 time=4.46 ms
8 64 bytes from 8.8.8.8: icmp_seq=7 ttl=118 time=4.43 ms
9 64 bytes from 8.8.8.8: icmp_seq=8 ttl=118 time=4.44 ms
10 64 bytes from 8.8.8.8: icmp_seq=9 ttl=118 time=4.47 ms
11 64 bytes from 8.8.8.8: icmp_seq=10 ttl=118 time=4.45 ms
12 64 bytes from 8.8.8.8: icmp_seq=11 ttl=118 time=4.43 ms
13 64 bytes from 8.8.8.8: icmp_seq=12 ttl=118 time=4.39 ms
14 64 bytes from 8.8.8.8: icmp_seq=13 ttl=118 time=4.48 ms
15 64 bytes from 8.8.8.8: icmp_seq=14 ttl=118 time=4.42 ms
16 64 bytes from 8.8.8.8: icmp_seq=15 ttl=118 time=4.41 ms
17 64 bytes from 8.8.8.8: icmp_seq=16 ttl=118 time=4.43 ms
18 64 bytes from 8.8.8.8: icmp_seq=17 ttl=118 time=4.41 ms
19 64 bytes from 8.8.8.8: icmp_seq=18 ttl=118 time=4.52 ms
20 64 bytes from 8.8.8.8: icmp_seq=19 ttl=118 time=4.51 ms
21 64 bytes from 8.8.8.8: icmp_seq=20 ttl=118 time=4.42 ms
22 64 bytes from 8.8.8.8: icmp_seq=21 ttl=118 time=4.43 ms
```

Anhang B

OMNeT++-Auslastung



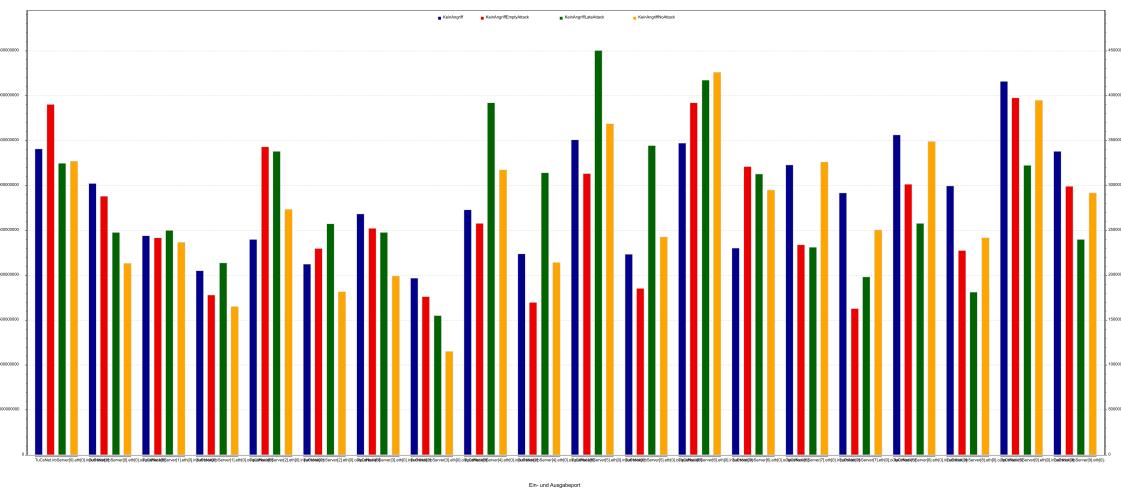
Anhang C

OMNeT++-Zufallsverteilung

C.1 Konfiguration

```
1 [Config KeinAngriffNoAttack]
2 extends = KeinAngriff
3 description = "Simulation ohne Angriff"
4
5 [Config KeinAngriffEmptyAttack]
6 extends = KeinAngriff
7 description = "Simulation ohne Angriff"
8 **.attackConfigurationFile = "attacks/empty.xml"
9
10 [Config KeinAngriffLateAttack]
11 extends = KeinAngriff
12 description = "Simulation ohne Angriff"
13 **.attackConfigurationFile = "attacks/late.xml"
```

C.2 Ergebnis



Anhang D

ASL-Message-Typen

Die hier gelisteten Messagetypen wurde aus einem Enum im Quellcode entnommen. Es müssen nicht zwingend alle Typen vollständig implementiert sein.

D.1 Anwendungsschicht

App.Type	C++ Packet Object	C++ controlInfo Object
APP.0000	cPacket("CreatedPacket-Layer5", 0)	UDPSendCommand()
APP.0001	cPacket("CreatedPacket-Layer5", 0)	UDPBindCommand()
APP.0002	cPacket("CreatedPacket-Layer5", 0)	UDPConnectCommand()
APP.0003	cPacket("CreatedPacket-Layer5", 0)	UDPCloseCommand()
APP.0004	cPacket("CreatedPacket-Layer5", 0)	UDPDataIndication()
APP.0005	cPacket("CreatedPacket-Layer5", 0)	UDPErrorIndication()
APP.0100	cPacket("CreatedPacket-Layer5", 0)	TCPSendCommand()
APP.0101	cPacket("CreatedPacket-Layer5", 0)	TCPOpenCommand()
APP.0102	cPacket("CreatedPacket-Layer5", 0)	TCPConnectInfo()
APP.0103	cPacket("CreatedPacket-Layer5", 0)	TCPErrorInfo()
APP.0104	cPacket("CreatedPacket-Layer5", 0)	TCPCommand()
APP.0200	TrafficLightStatus() ¹	UDPSendCommand()
APP.0201	TrafficLightStatus() ¹	UDPDataIndication()
APP.0300	TrafficLightCmd() ¹	DPDataIndication()
APP.0301	TrafficLightCmd() ¹	UDPSendCommand()
APP.1000	ApplicationPacket()	UDPDataIndication()
APP.1001	SendApplicationPacket()	UDPSendCommand()

D.2 Transportschicht

TRA.Type	C++ Packet Object	C++ controlInfo Object
TRA.0000	UDPPacket("CreatedPacket-UDP", 0)	IPv4ControlInfo()
TRA.0010	TCPSegment("CreatedSegment-TCP", 0)	IPv4ControlInfo()

D.3 Vermittlungsschicht

NET.Type	C++ PacketObject	C++ controlInfo Object
NET.0000	IPv4Datagram("Created packet-IPv4Datagram", 0)	IPv4Datagram()
NET.0001	IPv4Datagram("Created packet-IPv4Datagram", 0)	Ieee802Ctrl()

D.4 Sicherungsschicht

MAC.Type	C++ Packet Object	C++ controlInfo Object
MAC.0000	PPPFrame("CreatedPacket-PPP", 0)	null
MAC.0010	EthernetIIFrame("CreatedPacket-EthernetIIFrame", 0)	null
MAC.0020	IdealAirFrame("CreatedPacket-IdealAirFrame", 0)	null
MAC.0030	AirFrame("CreatedPacket-AirFrame", 0)	null

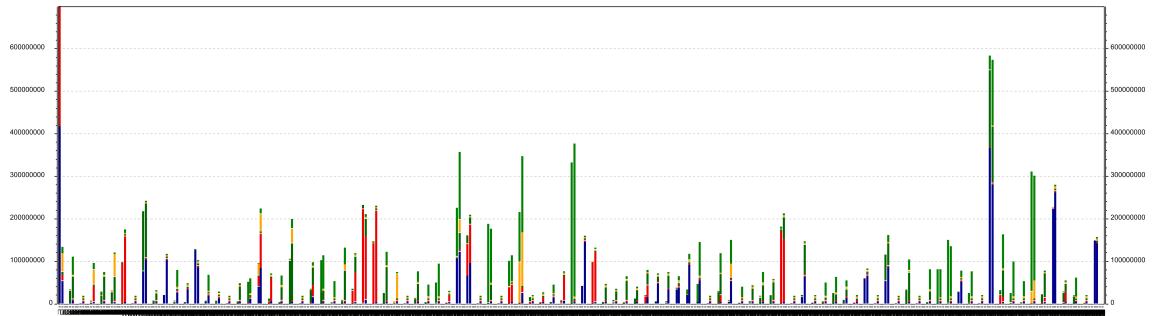
¹Zugehörig zu dem Traffic-Light-Beispiel, mit dem SEA++ vorgestellt wurde.

Anhang E

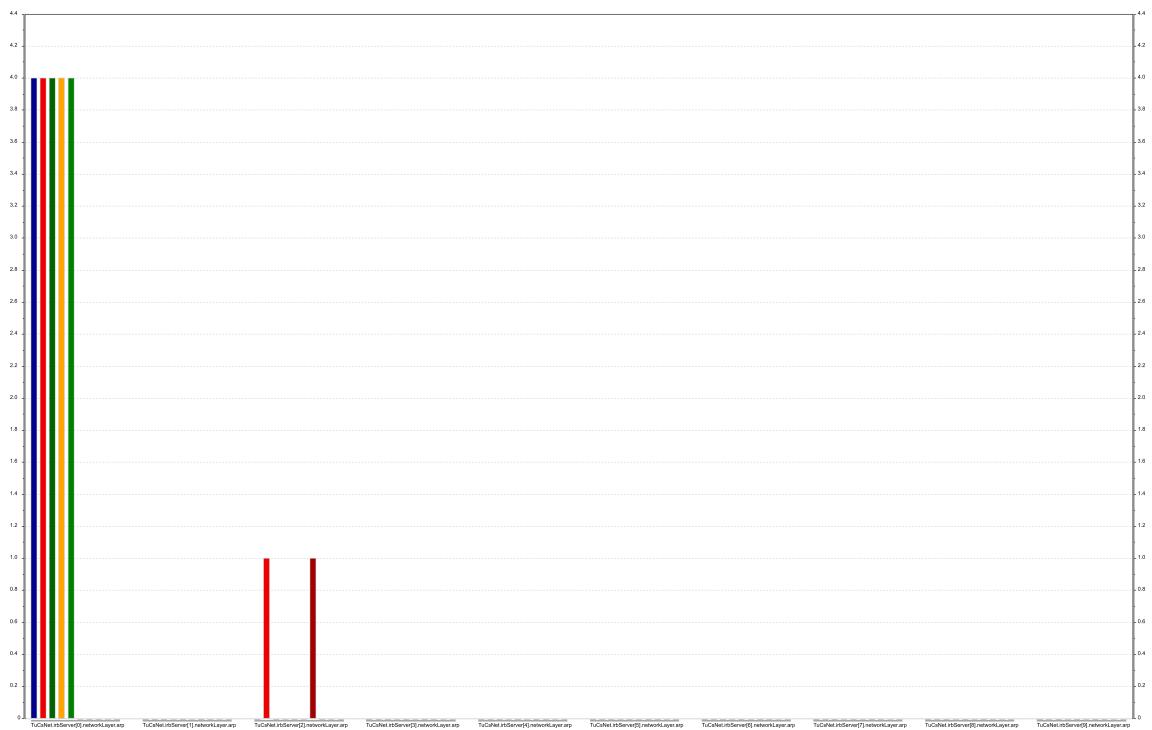
Auswertungen

E.1 ARP

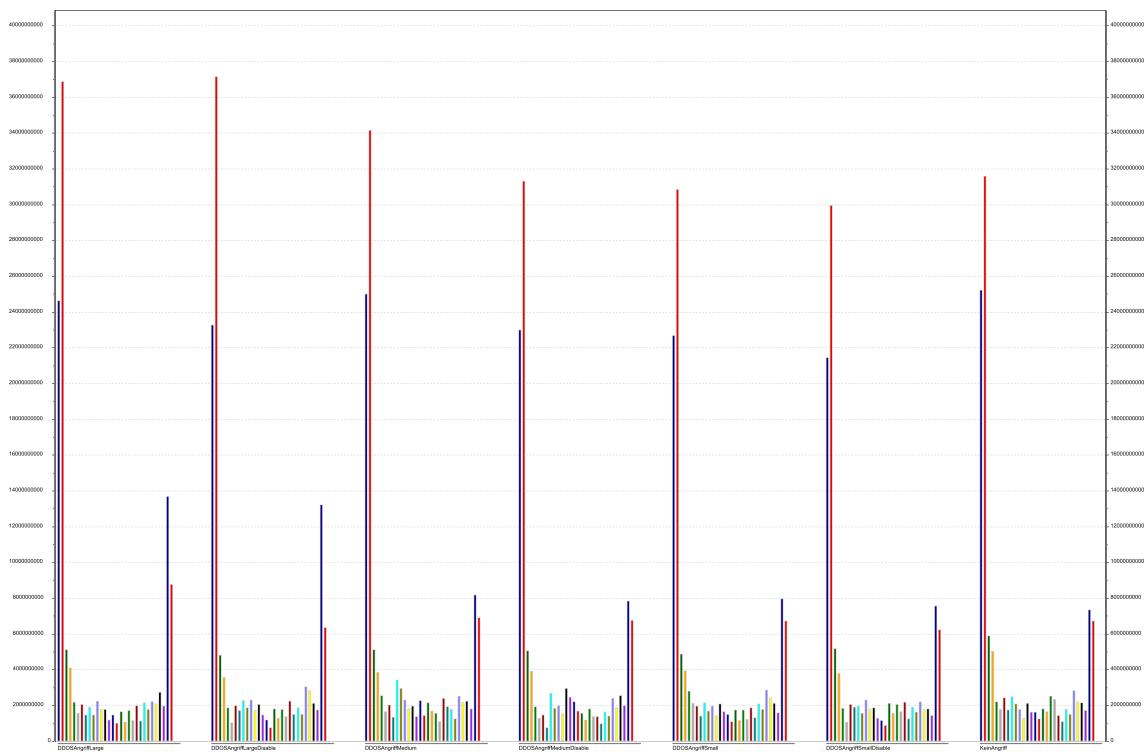
E.1.1 Summe der Bits im studentischen Bereich



E.1.2 Angefangene ARP-Auslösungen



E.2 DoS



Literatur

- [1] Tim Berners-Lee, Roy T. Fielding und Henrik Frystyk Nielsen. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. <http://www.rfc-editor.org/rfc/rfc1945.txt>. RFC Editor, Mai 1996. URL: <http://www.rfc-editor.org/rfc/rfc1945.txt>.
- [2] T. Bradley, C. Brown und A. Malis. *Inverse Address Resolution Protocol*. RFC 2390. RFC Editor, Sep. 1998.
- [3] *Cybercrime, Bundeslagebild 2019*. 2019. URL: https://www.bka.de/DE/AktuelleInformationen/StatistikenLagebilder/Lagebilder/Cybercrime/cybercrime_node.html.
- [4] *Die Lage der IT-Sicherheit in Deutschland 2019*. 2019. URL: https://www.bsi.bund.de/DE/Publikationen/Lageberichte/lageberichte_node.html.
- [5] *Glossar der Cyber-Sicherheit*. 2020. URL: https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/Empfehlungen/cyberglossar/Functions/glossar.html?cms_lv2=9817302.
- [6] *Google Public DNS*. 2020. URL: <https://developers.google.com/speed/public-dns>.
- [7] M. Handley und E. Rescorla and. *Internet Denial-of-Service Considerations*. RFC 4732. RFC Editor, Dez. 2006.
- [8] Hubert Hundt. *Cyber-Angriff auf die Ruhr-Universität Bochum*. Ruhr-Universität Bochum. 7. Mai 2020. URL: <https://news.rub.de/presseinformationen/servicemeldungen/2020-05-07-digitale-lehre-laeuft-weiter-cyber-angriff-auf-die-ruhr-universitaet-bochum>.
- [9] *INET Framework for OMNeT++*. 6. Nov. 2014. URL: <https://github.com/inet-framework/inet/tree/v2.6.0>.
- [10] *inet-framework/inet*. 2020. URL: <https://github.com/inet-framework/inet/releases>.
- [11] OpenSim Limited. *OMNeT++ - Simulation Manual*. *OMNeT++ version 5.6.1*. URL: <https://doc.omnetpp.org/omnetpp/manual/>.
- [12] OpenSim Limited. *What is OMNeT++?* 2020. URL: <https://omnetpp.org/intro/>.
- [13] OpenSim Limited. „Working with OMNeT++: Bird’s-eye view“. In: (22. Aug. 2020).
- [14] M. Mathis u. a. *TCP Selective Acknowledgment Options*. RFC 2018. RFC Editor, Okt. 1996.
- [15] Matthew Monte. *Network Attacks & Exploitation*. John Wiley & Sons, Inc, Feb. 2015. DOI: 10.1002/9781119183440. URL: <https://doi.org/10.1002/9781119183440>.

- [16] T. Narten, E. Nordmark und W. Simpson. *Neighbor Discovery for IP Version 6 (IPv6)*. RFC 2461. RFC Editor, Dez. 1998.
- [17] NETA: A NETwork Attacks Framework. 2013. URL: <https://nesg.ugr.es/index.php/en/neta-2>.
- [18] OMNeT++ Installation Guide Version 4.6. 2014. URL: <https://doc.omnetpp.org/omnetpp4/InstallGuide.pdf>.
- [19] Jianli Pan und Raj Jain. *A Survey of Network Simulation Tools: Current Status and Future Developments*. 2008. URL: <https://www.cse.wustl.edu/~jain/cse567-08/ftp/simtools/index.html>.
- [20] David C. Plummer. *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*. STD 37. <http://www.rfc-editor.org/rfc/rfc826.txt>. RFC Editor, Nov. 1982. URL: <http://www.rfc-editor.org/rfc/rfc826.txt>.
- [21] Jon Postel. *Transmission Control Protocol*. STD 7. <http://www.rfc-editor.org/rfc/rfc793.txt>. RFC Editor, Sep. 1981. URL: <http://www.rfc-editor.org/rfc/rfc793.txt>.
- [22] Francesco Racciatti u. a. *SEA++; user manual; SEA++ with SDN support INET-based*. 7. Jan. 2017. URL: https://github.com/seapp/seapp_stable/blob/master/seapp-manual/seapp-manual.pdf.
- [23] K. Ramakrishnan, S. Floyd und D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. <http://www.rfc-editor.org/rfc/rfc3168.txt>. RFC Editor, Sep. 2001. URL: <http://www.rfc-editor.org/rfc/rfc3168.txt>.
- [24] J. Reynolds. *Assigned Numbers: RFC 1700 is Replaced by an On-line Database*. RFC 3232. RFC Editor, Jan. 2002.
- [25] J. Reynolds und J. Postel. *Assigned Numbers*. RFC 1700. RFC Editor, Okt. 1994.
- [26] Keith W. Ross und James F. Kurose. *Computernetze ein Top-Down-Ansatz mit Schwerpunkt Internet*. 2002.
- [27] Theodore John Socolofsky und Claudia Jeanne Kale. *TCP/IP tutorial*. RFC 1180. <http://www.rfc-editor.org/rfc/rfc1180.txt>. RFC Editor, Jan. 1991. URL: <http://www.rfc-editor.org/rfc/rfc1180.txt>.
- [28] *tcp(7) - Linux man page*. 2020. URL: <https://linux.die.net/man/7/tcp>.
- [29] Marco Tiloca, Francesco Racciatti und Gianluca Dini. „Simulative evaluation of security attacks in networked critical infrastructures“. In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2014, S. 314–323.
- [30] Marco Tiloca u. a. „SEA++: A Framework for Evaluating the Impact of Security Attacks in OMNeT++/INET“. In: *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*. Hrsg. von Antonio Virdis und Michael Kirsche. Cham: Springer International Publishing, 2019, S. 253–278. ISBN: 978-3-030-12842-5. DOI: 10.1007/978-3-030-12842-5_7. URL: https://doi.org/10.1007/978-3-030-12842-5_7.
- [31] J. Touch. *Recommendations on Using Assigned Transport Port Numbers*. BCP 165. RFC Editor, Aug. 2015.
- [32] *Web Almanac 2019*. 2019. URL: <https://almanac.httparchive.org/en/2019/page-weight>.

- [33] Omer Yoachimik und Vivek Ganti. *Network-Layer DDoS Attack Trends for Q2 2020*. 5. Aug. 2020. URL: <https://blog.cloudflare.com/network-layer-ddos-attack-trends-for-q2-2020/>.