

Prototypische Implementierung und Evaluation eines asynchronen, performance-orientieren SYN-Portscanners in Rust

Bachelorarbeit

von

Lennard Alexander Dubhorn

Matrikelnummer: s0592852

Fachbereich 4 – Informatik, Kommunikation und Wirtschaft –
der Hochschule für Technik und Wirtschaft Berlin

zur Erlangung des akademischen Grades
Bachelor of Engineering (B. Eng.)
im Studiengang
Wirtschaftsinformatik

Tag der Abgabe: 06.02.2025

Erstgutachten: Prof. Dr.-Ing. Alexander Stanik

Zweitgutachten: Dr.-Ing. Ingmar Poese

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation und Einführung in das Themengebiet	1
1.2 Zielsetzung und Forschungsfrage	2
2 Theoretische Grundlagen und Forschungsstand	3
2.1 Grundlagen der Netzwerkkommunikation	3
2.1.1 Ports	3
2.1.2 Transmission Control Protocol (TCP)	4
2.2 Portscanning	4
2.2.1 SYN-Scanning	5
2.3 Schnittstellen zur Paketverarbeitung unter Linux	6
2.3.1 Linux	6
2.3.2 Raw-Sockets und Address Families	7
2.3.3 Erweiterte Berkley Packet Filter (eBPF)	7
2.3.4 eXpress Data Path (XDP)	8
2.4 Abgrenzung des Themas	9
2.4.1 Smart Objects	10
2.4.2 Internet-of-Things	11
2.5 Theoretische Grundlagen	12
3 Methodik und Anforderungsanalyse	13
3.1 Entwurf des eigenen Lösungskonzepts	13
3.2 Auswahl/Entwurf der Hardware	13
3.3 Auswahl/Entwurf der Software	13
3.3.1 Entwurf von Lösungsmodulen	13
4 Konzeption und Implementierung	15
4.1 Aufbau der Implementierung	15
4.2 Contiki-Verzeichnisstruktur	15
4.2.1 Übersetzung eines Contiki-Programms	16
4.2.2 Programmieren eines Mikrocontrollers	17
5 Tests	19
5.1 Herleitung von Test-Cases	19
5.1.1 Funktionale Tests	19
5.1.2 Leistungstests/Performancetests	19

5.2	Bewertung der Ergebnisse	19
6	Evaluation	21
6.1	Herleitung von Test-Cases	21
6.1.1	Funktionale Tests	21
6.1.2	Leistungstests/Performancetests	21
6.2	Bewertung der Ergebnisse	21
7	Fazit und Ausblick	23
7.1	Analyse der Implementierung	23
7.1.1	Kostenbetrachtung	23
7.2	Anwendungsmöglichkeiten	26
7.2.1	Anwendungsbereiche von Smart Objects	26
7.2.2	Anwendungsmöglichkeiten für die Implementierung	27
7.2.3	Erweiterungsvorschläge für die Implementierung	27
7.3	Zusammenfassung	28
A	Der Blindtext	31
	Abbildungsverzeichnis	33
	Tabellenverzeichnis	35
	Quelltextverzeichnis	37
	Stichwortverzeichnis	39
	Literaturverzeichnis	41
	Eigenständigkeitserklärung	45

Kurzfassung

Diese Arbeit beschreibt die Erstellung einer internetfähigen Steuerung für elektrische Verbraucher. Anforderungen an die Steuerung werden nach dem Kano-Modell definiert. Über eine Nutzwertanalyse werden vorhandene Techniken und Standards bewertet. Exemplarisch wird die Lösung mit dem größten Nutzwert implementiert.

Ein Zigbit-Modul, bestehend aus einem AVR Mikrocontroller und einem IEEE 802.15.4 Funkchip, bildet die Basis für die Hardware. Zusammen mit einem selbst dimensioniertem Kondensatornetzteil wird das Modul in einem Steckdosengehäuse verbaut.

Um zukunftssicher zu sein, wird das Protokoll IPv6 eingesetzt. Die Adaptionsschicht übernimmt das Protokoll 6LoWPAN. Das verwendete Betriebssystem Contiki besitzt eine fertige Webserver-Applikation, die für die eigenen Zwecke angepasst wird. Das Protokoll IEC 60870-5-104 wird neu implementiert. Es basiert auf dem TCP/IP-Modell und wird vor allem im Umfeld von Energieleitsystemen eingesetzt. Es eignet sich besonders für einen automatisierten Zugriff.

Über eine öffentliche Adresse des IPv6-Tunnelbrokers SixXS ist die Steuerung weltweit erreichbar und der elektrische Verbraucher kann über einen Webbrowser oder von einem Energieleitsystem ein- und ausgeschaltet werden.

Die Anforderungen nach dem Kano-Modell wurden nahezu vollständig erfüllt. Die Implementierung eines Webservers und einer IEC 60870-5-104 Applikation ist mit den gegebenen limitierten Ressourcen möglich. Anwendungsmöglichkeiten für die Steuerung liegen im Bereich eHome und Smart Grid.

Abstract

This Master Thesis describes the implementation of a solution to control and monitor electric consumers via the Internet. Needs of this solution are defined by use of the Kano model. Existing technologies and standards are benchmarked by means of a cost-utility analysis. The solution that scores the highest value of benefit will be implemented typically.

A Zigbit Module forms the basis of the hardware. It bundles an AVR microcontroller and an IEEE 802.15.4 transceiver. Together with a self-dimensioned capacitive power supply it is mounted in a socket housing.

To be future-proof, the IPv6 protocol is used. The 6LoWPAN protocol handles the adaptation layer. Contiki is used as operating system. It is delivered with a ready-to-use web server application which is customized for the own purposes. The IEC 60870-5-104 protocol is implemented from scratch. It is based on TCP/IP and is used in the field of energy management systems. It is particularly suitable for automated access.

Via a public address given by IPv6 tunnel broker SixXS the solution is accessible worldwide. The electric consumer can be switched on and off by the means of a web browser or an energy management system.

The needs according to the Kano model are almost completely achieved. It is possible to implement a solution consisting of a web server and IEC 60870-5-104 application in ressource constraint environments. Possible applications for such a solution are in the field of home automation and smart grid.

Kapitel 1: Einleitung

1.1 Motivation und Einführung in das Themengebiet

Netzwerkscanning macht einen großen Teil des Internet Traffics im IPv4 Adressraumes aus. So ist 98 Prozent des gesamten TCP Verkehrs weltweit auf SYN-Scans zurückzuführen [1]. Bekannte Tools wie Zmap werden stetig weiterentwickelt [2] und sind seit der Entwicklung von performanten Open-Source Scannern wie Zmap oder Masscan [3][4] dazu fähig, den gesamten IPv4 Addressraum in unter einer Stunde zu scannen. Das Scannen von Netzwerken nach offenen Ports ermöglicht es Organisationen Schwachstellen ausfindig zu machen, bevor Angreifer es tun. Außerdem lassen sich durch das breitflächige Scannen von ausgewählten Addressräumen oder dem gesamten IPv4 Raum Informationen über Trends und Veränderungen dieser ableiten. Cyberangriffe haben Auswirkungen auf den Ruf und die finanzielle Stabilität von Unternehmen [5]. Die gegenwärtig hohen Angriffszahlen zum Beispiel bei Denial-of-Service Angriffen [6] unterstreichen die Wichtigkeit.

Bisherige hochleistungsscanner wie die soeben genannten, wurden überwiegend in C entwickelt [3][4][7]. C ist häufig die Standard Wahl für maschinennahe Anwendungen, da sie zum Einen ein niedriges Level an Abstraktion und zum Anderen hoch performant sein kann [TODO]. Allerdings ist C anfällig für menschengemachte Fehler **Al-Boghdady_Wassif_El-Ramly_Al-Boghdady** wie ..TODO.. [TODO], von welchen die meisten sicherheitsrelevanten aus der Fraktion der Speicherverwaltung stammen [8]. Andere Sprachen wie Go oder Python lösen einige dieser Probleme durch die Nutzung einer automatischen Speicherverwaltung [TODO]. Diese Sprachen sind allerdings im Vergleich zu Sprachen wie C weniger Performant [8].

Rust hingegen schneidet in Vergleichen bezüglich der Performance auf ähnlichem Niveau wie C ab, bringt gleichzeitig aber das höchste Sicherheitsniveau der genannten Sprachen mit [8][9]. Außerdem unterstützt Rust Konzepte von Sprachen hoher Abstraktionsebene, wie beispielsweise die der Funktionalen Programmierung oder Objektorientierung [9], während zudem in der zuletzt zitierten Untersuchung, auch die Anzahl der Zeilen niedriger als im Vergleich zu dem in C geschriebenen Code ist.

Bisher fehlt eine fundierte Untersuchung darüber, ob Rust als moderne Sprache, welche Sicherheitsgarantien, *high-level*¹ Konzepte und Performance vereint, in Kombination mit aktuellen Linux-Schnittstellen, in der Lage ist, eine konkurrenzfähige Alternative zu gängigen Hochleistungsscannern, welche überwiegend in C geschrieben sind, darzustellen. Es

¹*high-level*: Auf hoher Abstraktionsebene

ist ungeklärt, ob der potentielle Performance Unterschied gering genug ist, um durch die gewonnene Sicherheit kompensiert zu werden, weshalb diese Arbeit an diesem Punkt ansetzt.

1.2 Zielsetzung und Forschungsfrage

In dieser Arbeit wird ein prototypischer SYN-Portscanner zum breitflächigen Scannen von Netzwerken in Rust entwickelt. Der Fokus des Scanners liegt auf einer hohen Performance, weshalb die Architektur teilweise asynchron gestaltet und leistungsfähige Linux- Schnittstellen wie AF_PACKET und XDP verwendet werden. Anschließend wird dieser bezüglich ausgewählter Performance Metriken mit einer repräsentativen Auswahl an bestehenden Scannern verglichen und die Ergebnisse daraufhin evaluiert.

Es ergibt sich folgende Forschungsfrage: Inwieweit kann ein in Rust implementierter asynchroner SYN-Scanner hinsichtlich des Durchsatzes und der Ressourceneffizienz mit etablierten Hochleistungsscannern konkurrieren und durch spracheigene Sicherheitsgarantien eine tragfähige Alternative für den produktiven Einsatz darstellen?

Kapitel 2: Theoretische Grundlagen und Forschungsstand

In diesem Kapitel werden die nötigen Grundlagen zum Verständnis des Portscanning in Form von SYN-Scans, sowie das nötige Wissen über Netzwerkkommunikation, die genutzten Technologien und Linux-Schnittstellen vermittelt. Des Weiteren wird auf Asynchrone Programmierung eingegangen, sodass ein Verständnis für das nachfolgende Konzept der Implementierung gegeben ist.

Anschließend werden die zum Vergleich genutzten Scanner vorgestellt und eingeordnet. Auch Rust und dessen Besonderheiten wird genauer vorgestellt.

2.1 Grundlagen der Netzwerkkommunikation

Bei der Kommunikation in TCP/IP¹ Netzwerken dient das IP-Protokoll und die IP-Adressen zur Identifikation der Maschine im Netzwerk, während die genaue Adressierung der spezifischen Anwendungen durch sogenannte Ports bzw. der sogenannten Portnummer bestimmt wird [10] (TODO SEITE?). Die Portnummer ist ein 16-Bit-Wert und kann somit zwischen jeweils einschließlich 0 und 65535 liegen [Computernetzwerke S107]. Einige Portnummern sind fest vergeben oder für bestimmte Anwendungen registriert [11], was es ermöglicht, gezielt nach bestimmten Anwendungen zu scannen.

2.1.1 Ports

Ports können in verschiedene Zustände eingeordnet werden. Für diese Arbeit ist nur die Unterscheidung zwischen offen und geschlossen/gefiltert relevant.

Offen: Eine Anwendung lauscht auf dem Port und akzeptiert eingehende valide TCP oder UDP Anfragen [10]. Geschlossen / Gefiltert: Der mit dem Port verbundene Service ist zwar ansprechbar, aber akzeptiert keine eingehenden Verbindungen / Es gibt lediglich eine ICMP (Fehler) Antwort oder gar keine, da beispielsweise kein Service für diesen Port existiert [10].

Ein IP-Port Paar z.B. 192.178.0.42:8080 wird Socket bzw. Socket Paar genannt.

¹Eine grundlegende Kenntnis über das TCP/IP Modell wird angenommen

mglw. Irrelevant: Ein Verbindung zweier Rechner über einen spezifischen Dienst ist durch die beiden Socket Paare eindeutig identifizierbar [Computernetzwerke S108].

2.1.2 Transmission Control Protocol (TCP)

Das Transmission Control Protocol operiert in der Transportschicht des TCP/IP Modells und ist eines der meistgenutzten Transportprotokoll des Internets [12] TODO S71. Es gewährleistet eine zuverlässige, verbindungsorientierte Datenübertragung zwischen den Prozessen der Hosts. Die ursprüngliche Spezifikation erfolgte im RFC 793 [13], welches durch RFC 9293 [14] konsolidiert wurde. Für die Entwicklung eines SYN-Portscanners sind insbesondere der Aufbau des TCP-Headers und der Mechanismus des Verbindungsauftbaues entscheidend.

Da das TCP Protokoll Daten als Datenstrom (Stream) statt einzeln (Message) versendet, wird vorher eine Verbindung in einem sogenannten *Three-Way-Handshake* aufgebaut [12] S71/72. Bei diesem werden TCP Pakete mit jeweils unterschiedlichen Werten in den *Control Bits (Flags)* des TCP-Headers nach folgendem Muster ausgetauscht:

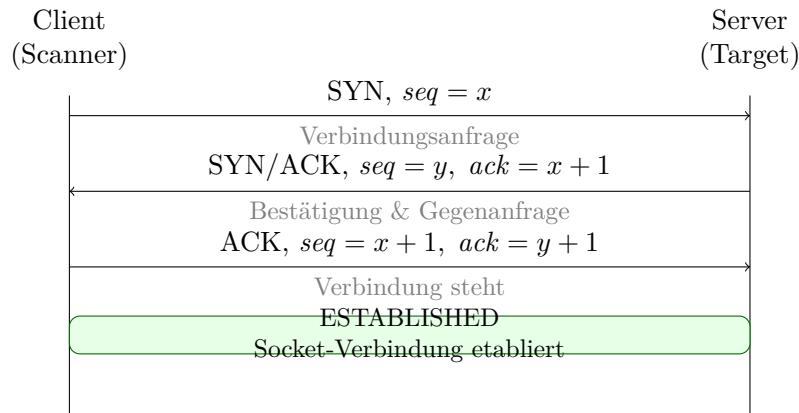


Abbildung 2.1: Der TCP Three-Way-Handshake. Entscheidend für die Zuordnung ist, dass die *Acknowledgment Number* im zweiten Schritt dem Wert der initialen *Sequence Number* + 1 entspricht ($x + 1$).

Da das TCP Protokoll Daten als Datenstrom (Stream) statt einzeln (Message) versendet, wird vorher eine Verbindung in einem sogenannten *Three-Way-Handshake* aufgebaut [12] S71/72. Bei diesem werden TCP Pakete mit jeweils unterschiedlichen Werten in den *Control Bits (Flags)* des TCP-Headers nach folgendem Muster ausgetauscht:

2.2 Portscanning

Portscanning, als Art des Netzwerkscannings, ist eines der fundamentalen Verfahren in der Netzwerksicherheit zum Auffinden von potentiellen Schwachstellen [10]. Ein PortsScanner verschickt Pakete an ein Zielsystem und zieht anhand der Antworten, oder auch

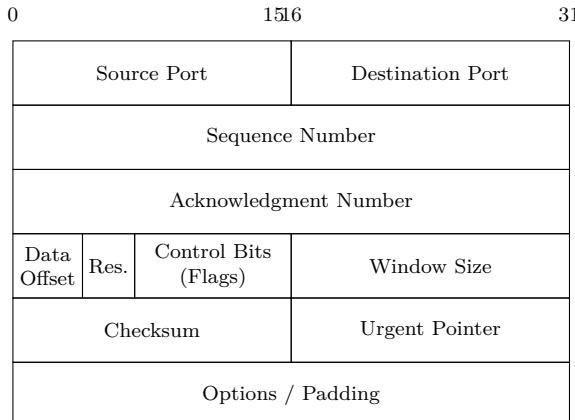


Abbildung 2.2: Aufbau des TCP-Headers (ohne Nutzdaten). Für SYN-Scans sind insbesondere die *Control Bits* (z. B. SYN, RST, ACK) relevant.

ausbleibenden Antworten, Rückschlüsse auf den Zustand des Systems. Das Ziel ist die Identifikation von offenen Ports bzw. aktiven Diensten, was als erster Schritt für weiterführende Sicherheitsanalysen oder aber auch Angriffe dienen kann. [TODO NIST S4-3]

Beim Scannen von Ports können grundsätzlich zwei strategische Ausrichtungen unterschieden werden: - Vertikales Scannen: Hierbei wird ein einzelner Ziel-Host (TODO Host erklären) auf eine Vielzahl von Ports (oft alle 65535) gescannt, um ein möglichst vollständiges Profil möglicher Schwachstellen des Zielsystems zu erlangen und somit eher für das Penetrationtesting geeignet ist. - Horizontales Scannen: Ein sehr großer Addressbereich, beispielsweise das komplette IPv4-Internet wird gescannt. Dafür ist die Anzahl der zu scannenden Ports sehr klein oder auf einen einzigen beschränkt. Dies bietet die Möglichkeit wertvolle Daten über Trends oder die Verbreitung von Schwachstellen zu untersuchen [3].

2.2.1 SYN-Scanning

Für einen *half-open* SYN-Scan wie er in dieser Arbeit behandelt wird, sind lediglich die ersten beiden Schritte des Verbindungsablaufes relevant. Es wird davon gebraucht gemacht, dass bereits und ausschließlich eine SYN/ACK Antwort den Port als offen klassifiziert, was den weiteren Verbindungsaufbau irrelevant macht [10]. Um den Scan zu verschleiern und den Verbindungsversuch trotz dessen sauber abzuschließen, kann anschließend noch ein Paket, bei welchem die RST Flag der Control Bits gesetzt ist gesendet werden [10].

Um antwortende Hosts effizient zu identifizieren, wird das Prinzip des SYN-Cookies adaptiert, welches ursprünglich als Abwehrmechanismus gegen Denial-of-Service-Angriffe spezifiziert wurde [15, S. 8]. Dieses Verfahren bietet den Vorteil, dass der Scanner keinen Zustand für die versendeten Pakete im Arbeitsspeicher halten muss. Stattdessen werden verbindungsspezifische Informationen unter Verwendung eines Hash-Algorithmus (z. B. keyed SipHash) kodiert und als *Sequence Number* in den TCP-Header des ausgehenden SYN-Pakets eingetragen. Antwortet ein Ziel-Host mit einem SYN-ACK-Paket, so enthält

dessen Acknowledgment Number gemäß TCP-Spezifikation den inkrementierten Wert der ursprünglichen *Sequence Number*. Die Validierung lässt sich abstrahiert wie folgt beschreiben:

```
is_valid = hash(value_0, value_1, ..., secret) == answer.ack_num - 1
```

Entscheidend für den Scanner sind also folgende Header Felder:

Tabelle 2.1: TCP Header Felder

Header Feld	Beschreibung
Source Port	Beschreibt den genutzten Port des Ausgangsdienstes.
Destination Port	Beschreibt den zu scannenden Port des Zielsystems.
Sequence Number	Wird zur Speicherung des SYN-Cookies genutzt.
Acknowledgement Number	Wird zum Abrufen des SYN-Cookies genutzt.
Control Bits (Flags)	Wird für die verschiedenen Phasen des Verbindungsaufbaues angepasst oder ausgelesen.

2.3 Schnittstellen zur Paketverarbeitung unter Linux

Um einen performanten Scanner zu bauen, müssen die genutzten Technologien zum einen für die Netzwerkprogrammierung geeignet und zum anderen hohe Sende- und Empfangsraten zulassen, während möglichst wenig Rechenressourcen verbraucht werden.

2.3.1 Linux

Linux ist ein Open-Source Betriebssystem-Kernel, welcher aufgrund neuartiger Subsysteme (wie eBPF [später] und XDP [später]), eine programmierbare Paketverarbeitung nahe an der Hardware ermöglicht [16]. Dies ist für die Entwicklung eines Hochleistungs-Scanners von großem Vorteil.

Ein zentrales Konzept zum Verständnis der Performance-Grenzen ist die Unterscheidung zwischen *User Space* und *Kernel Space* [16] S23:

- **Kernel Space:** Hier läuft der Kern des Betriebssystems (Ring 0) mit vollem Zugriff auf die Hardware und den Speicher. Treiber und der Netzwerk-Stack operieren auf dieser Ebene.
- **User Space:** Hier laufen reguläre Anwendungen (Ring 3) in isolierten Speicherbereichen.

Die Kommunikation zwischen diesen Ebenen erfolgt über *System Calls*. Jeder Wechsel (*Context Switch*) zwischen User und Kernel Space, sowie das Kopieren von Daten zwischen diesen Speicherbereichen, erzeugt Overhead. Beim Versenden und Empfangen sehr vieler Pakete summiert sich dieser Overhead, da jedes Paket im Normalfall sowohl Kernel Space, als auch User Space, durchschreitet [TODO]. Dies belastet die CPU und wird für den Durchsatz zum Flaschenhals.

2.3.2 Raw-Sockets und Address Families

Als Endpunkt für die Kommunikation werden *Sockets* genutzt [17]. Die traditionelle Netzwerkprogrammierung unter Linux abstrahiert die Komplexität der Netzwerkprotokolle wie TCP. So übernimmt der Kernel dabei vollständig den *Three-Way-Handshake* und das Zustandsverwaltung [TODO]. Für einen SYN-Scanner ist dies ungeeignet, da der Scanner lediglich das initiale SYN-Paket senden und die Antwort registrieren will, ohne eine vollwertige Verbindung aufzubauen, welche Ressourcen im Kernel binden würde.

RAW Sockets erlauben der Anwendung, Netzwerkpakete unter Umgehung bestimmter Layer des Kernel-Stacks zu senden und zu empfangen [18]. Der Entwickler muss die Protokoll-Header selbst konstruieren. Dies ist für half-open Port-Scanner essenziell, um individuelle Pakete zu generieren, ohne dass der Kernel automatisch in den Verbindungsauflauf eingreift.

Die Adress-Familien definieren dabei die Interpretation der Adressen und die Ebene des Zugriffs [19]. Hier ein paar Beispiele: TODO Besser formulieren

- **AF_INET (Standard):** AF_INET Operiert auf Layer 3 (IP Ebene), deshalb fügt der Kernel den IP-Header hinzu. Auch das Routing wird (TODO vollständig?) durch den Kernel vorgenommen.
- **AF_PACKET (Performant):** Diese Familie ermöglicht direkten Zugriff auf Layer 2 (Ethernet-Ebene). Anwendungen können rohe Ethernet-Frames lesen und schreiben. Dies bietet die vollständige Kontrolle über die Paketerstellung. TODO wie weit durch Kernel geschickt?
- **AF_XDP (Sehr Performant):** Hierbei handelt es sich um eine für Hochleistungspaketverarbeitung optimierte Address-Familie. Sie ermöglicht das Senden und Empfangen von Paketen unter fast vollständiger Umgehung des Kernel-Stacks.

(TOOD -> Abbildung Netzwerkstack bzw Paketverlauf)

2.3.3 Erweiterte Berkley Packet Filter (eBPF)

Ursprünglich als *Berkeley Packet Filter* (BPF) für Werkzeuge wie tcpdump entwickelt, um Pakete effizient zu filtern [20], wurde die Technologie zum erweitert, sodass grundlegend neue Möglichkeiten erschlossen wurden.

eBPF ist eine im Linux-Kernel integrierte virtuelle Maschine (VM), die es erlaubt, benutzerdefinierten Bytecode sicher und effizient im Kernel-Kontext auszuführen, ohne Kernel-Module schreiben oder den Kernel neu kompilieren zu müssen [21] S207. eBPF-Programme werden zur Laufzeit durch einen JIT-Compiler (Just-In-Time) in native Maschinensprache übersetzt. Ein *Verifier* stellt vor der Ausführung sicher, dass der Code sicher ist TODO cite. So werden Fehler wie beispielsweise Endlosschleifen oder falsche Speicherzugriffe vermieden.

Für einen SYN-Scanner ist eBPF nützlich, da es ermöglicht, eingehende Antwortpakete (SYN-ACK) extrem früh zu filtern und an den User Space weiterzuleiten, bevor teure Speicherstrukturen des Kernels angelegt werden. So werden nur relevante Daten an den User Space weitergereicht.

2.3.4 eXpress Data Path (XDP)

XDP definiert eine limitierte Ausführungsumgebung für eBPF-Programme, die direkt im Kontext des Netzwerktreibers ausgeführt werden. Dies ermöglicht eine programmierbare Hochleistungspaketverarbeitung direkt im Betriebssystemkern. Im Gegensatz zu früheren Ansätzen, die den Kernel vollständig umgehen (z.B. DPDK), integriert sich XDP kooperativ in den bestehenden Stack [22].

Ein XDP-Programm kann Pakete verwerfen (XDP_DROP), an den regulären Netzwerkstack weiterleiten (XDP_PASS), über dieselbe Schnittstelle zurücksenden (XDP_TX) oder an eine andere CPU bzw. einen Userspace-Socket umleiten (XDP_REDIRECT) [22][23].

Die Effizienz von XDP resultiert aus der Positionierung im Datenpfad. In herkömmlichen Linux-Netzwerkarchitekturen durchläuft ein Paket nach dem Empfang durch die Netzwerkkarte (NIC) den gesamten Netzwerk-Stack. Erst danach erreichen die Daten den User Space. Dies erfordert CPU und Speicher- aufwendige Kontextwechsel (Context Switches) zwischen Kernel- und User-Mode, sowie die Allokation komplexer Metadatenstrukturen (sk_buff) (TODO siehe Abbildung vorher) [22][24].

XDP greift vor dieser Allokation ein (TODO siehe Abbildung). Tests zeigen, dass XDP auf einem einzelnen CPU-Kern bis zu fünfmal mehr Pakete pro Sekunde verarbeiten kann als der Standard Linux-Stack [22].

-> Abbildung XDP einfügen

Die Performance und Verfügbarkeit von XDP hängen vom verwendeten Betriebsmodus ab. Nach Zhang et al. [24] und Vieira et al. [23] lassen sich drei Modi unterscheiden:

- **Native Mode (Driver Mode):** Dies ist der Standardmodus für Hochleistungsanwendungen. Das XDP-Programm wird direkt im Netzwerkkartentreiber ausgeführt. Die Verarbeitung erfolgt nach dem DMA-Transfer (Direct Memory Access) in den Ring-Buffer, aber vor der sk_buff-Allokation. Dies erfordert explizite Unterstützung durch den Treiber der Netzwerkkarte.
- **Offloaded Mode (Hardware Mode):** Hierbei wird das eBPF-Programm vom Kernel auf die Netzwerkkarte ausgelagert und direkt auf der Hardware ausgeführt. Dies bietet die höchste Performance, da die Host-CPU vollständig von der Paketverarbeitung entlastet wird, setzt aber die Nutzung einer sogenannten *Smart NIC* Netzwerkkarte voraus.

- **Generic Mode (SKB Mode):** Dieser Modus dient der Kompatibilität. Wenn ein Treiber XDP nicht nativ unterstützt, führt der Kernel das XDP-Programm an einer späteren Stelle im Netzwerkstack aus. Zwar gehen hier die massiven Performance-Vorteile der Speicherersparnis verloren, jedoch wird sichergestellt, dass XDP-Anwendungen auf jeder Hardware funktionsfähig bleiben. Trotzdem profitiert es von effizienteren, sperrfreien Ring-Buffer-Strukturen (<- TODO bessere Argumente)

2.4 Abgrenzung des Themas

. Außerdem ist es das Ziel, horizontal (TODO siehe nächstes Kapitel) zu scannen. Anders als beispielsweise der reguläre SYN-Scan des Tools NMap [10], welcher in der Regel vertikal erfolgt.

Mechanismen zur Erkennungsvermeidung des Scans werden in dieser Implementierung nur rudimentär behandelt, da der Fokus klar auf der Performance und Ressourceneffizienz der Anwendung liegt. Außerdem beschränkt sich diese Arbeit auf den IPv4 Addressraum, da dies genügt, um der Forschungsfrage nachzugehen.

Tabelle 2.2: OSI-7-Schichtenmodell

Schicht			Beispielprotokoll	TCP/IP-Modell
7	Anwendungsschicht	Application Layer		
6	Darstellungsschicht	Presentation Layer	HTTP, FTP, SNMP	Application Layer
5	Sitzungsschicht	Session Layer		
4	Transportschicht	Transport Layer	TCP, UDP	Transport Layer
3	Vermittlungsschicht	Networking Layer	IPv4, IPv6, ICMP	Internet Layer
2	Sicherungsschicht	Data Link Layer	Ethernet, WLAN, ISDN	Link Layer
1	Physikalische Schicht	Physical Layer		

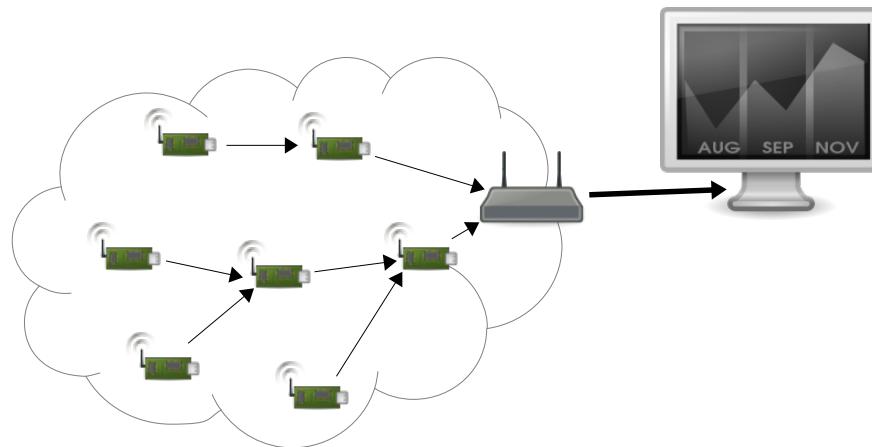


Abbildung 2.3: Aufbau eines drahtlosen Sensornetzwerkes

Bei drahtlosen Sensornetzen, international auch Wireless Sensor Networks (WSN) genannt, handelt es sich um Netzwerke aus kleinen Sensoren, die ihre Messergebnisse zu einer zentralen Station senden. Die Sensoren helfen dabei einander, die Informationen weiterzureichen (siehe Abbildung 2.3). So ein drahtloses Sensornetz kann zum Beispiel in einem Gebäude Temperatur- und Luftfeuchtigkeitswerte an eine zentrale Station liefern, die dann die Klimaanlage entsprechend ansteuert. Eine komplette Verkabelung jedes einzelnen Sensors kann dabei je nach Anzahl sehr aufwendig und kostspielig sein. Wenn die Sensoren dazu noch mobil sein sollen, empfiehlt sich eine drahtlose Kommunikation.

Ein Sensor in einem drahtlosen Sensornetz ist ausgestattet mit einer Kommunikationseinheit, die sich selbstständig innerhalb des Netzes konfiguriert und darüber die eingelesenen Messgrößen zu einer zentralen Stelle transportiert.

2.4.1 Smart Objects

Diese Definition von Smart Objects beruht auf **vasseur10interconnecting**. Dabei handelt es sich um kleine Objekte, die mit folgenden Einheiten ausgestattet sind:

- eine Form von Sensor und/oder Aktor
- ein kleiner Mikrocontroller
- eine Kommunikationseinheit
- eine Energieversorgung

Ein Sensor und/oder Aktor wird benötigt, um mit der Umwelt interagieren zu können. Ein Sensor kann bestimmte Messgrößen erfassen, die dann verarbeitet werden können. Ein Aktor ist das Gegenstück zu einem Sensor. Über ihn kann aktiv die Umwelt beeinflusst werden. Die Kommunikationseinheit befähigt das Smart Object über ein Netzwerk kommunizieren zu können. Es ist dabei möglich mit anderen Smart Objects zu kommunizieren als auch mit weit entfernten Geräten, im Falle des Internets weltweit. Ein kleiner Mikrocontroller übernimmt die Informationsverarbeitung. Dieser nimmt Daten von Sensoren entgegen bzw. steuert den Aktor. Die Kommunikation wird ebenso von ihm geregelt, von einem Sensor gemessene Werte können über das Netzwerk weitergegeben werden. Der Mikrocontroller kann als Kern des Smart Object angesehen werden. Die Energieversorgung ist notwendig, um alle Einheiten ausreichend mit elektrischer Energie zu versorgen.

Alle diese technischen Eigenschaften machen ein Objekt aber noch nicht zu einem Smart Object. Erst die Anwendung und sein Verhalten machen aus einem Objekt mit den obigen Eigenschaften ein Smart Object. Allerdings ist es sehr schwierig, dieses Verhalten zu definieren. Die Anwendungen sind sehr unterschiedlich und es ist völlig unbekannt, wie Smart Objects in der Zukunft eingesetzt werden. Gemeinsam ist allen Anwendungen allerdings, dass Smart Objects mit der physikalischen Umwelt interagieren und über ein Netzwerk

kommunizieren. Dieses Verhalten und die obigen technischen Eigenschaften machen ein Smart Object aus.

Von der Kommunikationstechnologie her ähnelt ein Smart Object einem Sensor in einem Sensornetz. Allerdings ist im Gegensatz zu einem Sensornetz ein Netz aus Smart Objects nicht allein auf die Datenübermittlung fokussiert. Bei einem Sensornetz ist der Datenfluss immer vom Sensor ins Netzwerk, wohingegen der Datenfluss bei einem Smart Object bidirektional ist. Der Fokus liegt dabei auf einer Vielzahl von Funktionen insbesondere der Steuerung und Überwachung [vasseur10interconnecting]. Es ist also die Anwendung, die ein Gerät zu einem Smart Object macht. Aber Entwicklungen, hauptsächlich in der Energieversorgung und Kommunikationstechnik, kommen beiden Forschungsgebieten zugute.

2.4.2 Internet-of-Things

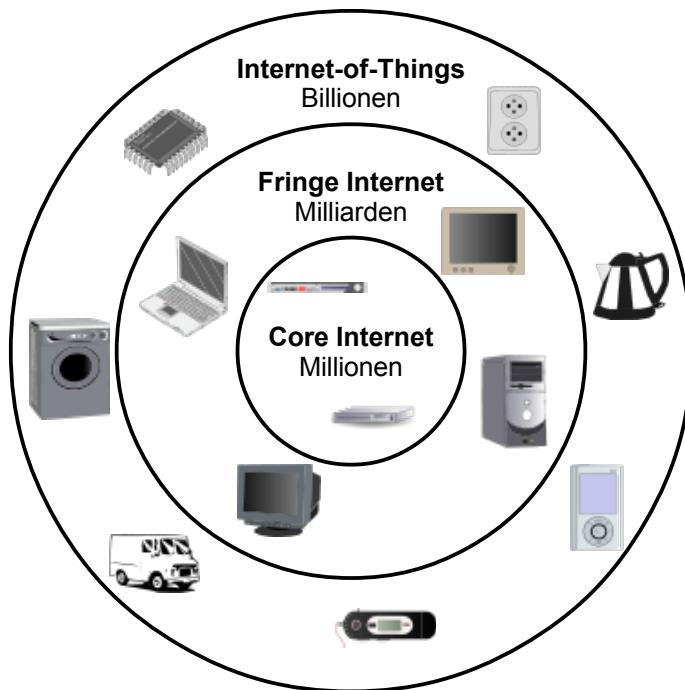


Abbildung 2.4: Die Internet-of-Things Vision **Bormann:6LoWPAN**

Das Internet-of-Things, auch Internet der Dinge genannt, ist eine Vision, wie sich das Internet in Zukunft entwickeln kann. Es soll aus der Vernetzung vieler Kleinstgeräte – auch oder vielleicht hauptsächlich Smart Objects – bestehen. **Bormann:6LoWPAN** betrachten das Internet-of-Things als eine weitere Schale des Internets (Abbildung 2.4), die gerade anfängt sich heraus zu bilden. Der Kern des Internets (Core Internet) besteht heute aus vielen Routern und Servern, die zusammen Millionen von Teilnehmern ausmachen. Um diesen Kern herum liegt das sogenannte Fringe Internet, was beispielsweise aus privaten Rechnern oder Laptops besteht, aber auch aus lokalen Netzwerken, die zum Internet

verbunden werden. Die Teilnehmer des Fringe Internet gehen in die Milliarden. Um dieses Fringe Internet herum beginnt sich eine weitere Schale zu bilden, das sogenannte Internet-of-Things. Auf lange Sicht kann hier mit Billionen von Teilnehmern gerechnet werden. Hauptsächlich bestehen diese Teilnehmer aus IP-fähigen eingebetteten Systemen (embedded systems). Eine so große Zahl an Teilnehmern kann nur mithilfe des Internet Protokolls der Version 6 ans Internet angeschlossen werden, da das bisherige Internet Protokoll der Version 4 keine freien Adressräume mehr zur Verfügung hat.

2.5 Theoretische Grundlagen

...

Kapitel 3: Methodik und Anforderungsanalyse

Da die verschiedenen Technologien aufeinander aufbauen, manchmal zueinander kompatibel sind, manchmal inkompatibel sind, werden nachfolgend mögliche Komplettlösungen hergeleitet. Sie werden zuerst kurz vorgestellt, miteinander verglichen und bewertet. Um einen systematischen Entscheidungsprozess zu gewährleisten, werden die möglichen Komplettlösungen mithilfe der Nutzwertanalyse verglichen und bewertet. Der Vorteil der Nutzwertanalyse ist es, dass die Bewertung nachvollziehbar und überprüfbar ist. Nach der Auswertung wird die Lösung ausgesucht, die implementiert werden soll.

3.1 Entwurf des eigenen Lösungskonzepts

- Darstellung des eigenen Ansatzes, ggf. Abgrenzung von anderen Arbeiten
- Vor-/Nachteile
- ...

3.2 Auswahl/Entwurf der Hardware

...

3.3 Auswahl/Entwurf der Software

3.3.1 Entwurf von Lösungsmodulen

...

Kapitel 4: Konzeption und Implementierung

4.1 Aufbau der Implementierung

Die Implementierung der Lösung erfolgte in zwei Teilen. Es wurde die Hardware erstellt und zeitlich parallel dazu wurde die Software entwickelt. Beide Teile konnten relativ unabhängig voneinander implementiert werden. Die Schnittstelle von Hardware und Software ist das Zigbit-Modul ATZB-24-A2. Auf diesem Modul wird die fertige Software einprogrammiert. Die Hardware versorgt das Modul mit Energie und schaltet nach Softwarevorgaben die Steckdose ein und aus.

4.2 Contiki-Verzeichnisstruktur

Die Contiki-Verzeichnisstruktur besteht aus verschiedenen Unterverzeichnissen, die verschiedene Bedeutungen haben.

Codeauszug 4.1: Contiki-Verzeichnisstruktur mit ausgewählten Unterverzeichnissen

```
1 /apps
2   /ftp
3   /ping6
4   /telnet
5   /telnetd
6   /twitter
7   /webserver
8   /webserver-nano
9   ...
10 /core
11   /lib
12   /net
13   /sys
14   ...
15 /cpu
16   /arm
17   /avr
18   ...
```

```
19 /doc
20 /examples
21   /webserver-ipv6-raven
22   ...
23 /platform
24   /avr-raven
25   /avr-zigbit
26   ...
27 /tools
```

Das Verzeichnis `/core` beinhaltet das Kernstück von Contiki. Hier wird zum Beispiel das System im Unterverzeichnis `/sys` definiert. Das beinhaltet das Prozesshandling und Protothreads sowie verschiedene Timer. Ebenfalls befindet sich hier im Unterverzeichnis `/net` der Netzwerk-Stack aufbauend auf uIP. Zum Netzwerk-Stack gehört auch IEEE 802.15.4 und 6LoWPAN. Im Unterverzeichnis `/lib` stehen verschiedene libraries zur Verfügung.

In den beiden Verzeichnissen `/cpu` und `/platform` ist die unterschiedliche Hardware beschrieben, die von Contiki unterstützt wird. Ein Programm, das auf einem Zigbit-Modul vom Hersteller Atmel geladen werden soll, verwendet die Verzeichnisse `/platform/avr-zigbit` und `/cpu/avr`.

Das Verzeichnis `/examples` beinhaltet Beispielprojekte. Hier gibt es ein Projekt `webserver-ipv6-raven`. Anhand dieses Beispiels wird deutlich, was notwendig ist, um eine Webserver-Applikation für das Ravenboard zu kompilieren.

Im Hauptverzeichnis gibt es eine Datei `Makefile.include`. Diese Datei ist Teil des Contiki-`Makefile`-Systems. Sie wird innerhalb eines Contiki-Projektes aufgerufen und bindet, abhängig von der Konfiguration des Projektes, die richtigen Dateien des Contiki-Systems ein.

4.2.1 Übersetzung eines Contiki-Programms

Ein Programm wird in Contiki mithilfe des Befehls „make“ übersetzt. Am Beispiel des Projekts `webserver-ipv6-raven` soll exemplarisch gezeigt werden, wie das `Makefile` System funktioniert. Durch den Befehl „make“ wird die Datei `Makefile` im gleichen Verzeichnis abgearbeitet.

Codeauszug 4.2: Auszug aus `examples/webserver-ipv6-raven/Makefile`

```
1 ifndef TARGET
2   TARGET=avr-raven
3   MCU=atmega1284p
4 endif
5 all:
6   ${MAKE} -f Makefile.webserver TARGET=$(TARGET) NOAVRSIZE=1 webserver6.elf
```

Hier wird das TARGET, also die Plattform, und die MCU, die Microcontroller Unit, gesetzt und dann die Datei Makefile.webserver aufgerufen. Dabei wird der Parameter NOAVRSIZE gesetzt, um beim Übersetzen eine zusätzliche Ausgabe zur Speicherbelegung (avr-size) zu unterdrücken. Das Programm wird als Datei webserver6.elf erstellt.

Codeauszug 4.3: Auszug aus examples/webserver-ipv6-raven/Makefile.webserver

```
1 all: webserver6
2 APPS=raven-webserver raven-lcd-interface
3 UIP_CONF_IPV6=1
4 CONTIKI = ../../
5 include $(CONTIKI)/Makefile.include
```

In der Datei Makefile.webserver werden die Applikationen raven-webserver und raven-lcd-interface über die Variable APPS eingebunden. Die Compiler Variable UIP_CONF_IPV6 wird gesetzt, um IPv6 zu aktivieren, weiterhin wird das Haupt-Makefile **Makefile.include** eingebunden.

Codeauszug 4.4: Auszug aus examples/webserver-ipv6-raven/webserver6.c

```
1 #include "webserver-nogui.h"
2 /*-----*/
3 AUTOSTART_PROCESSES(&webserver_nogui_process);
4 /*-----*/
```

In der Datei webserver6.c wird ausgewählt, welche Contiki-Prozesse automatisch gestartet werden sollen. In unserem Beispiel ist das der Prozess „webserver_nogui_process“. Dieser Prozess ist Teil der Applikation raven-webserver.

4.2.2 Programmieren eines Mikrocontrollers

Durch Übersetzung des Contiki-Beispielprojektes webserver-ipv6-raven wird eine Datei „webserver6.elf“ im ELF Format erzeugt. Hiervon wird eine Kopie „webserver6-avr-raven.elf“ erstellt. Diese Datei enthält Informationen darüber, was in den Flash und in den EEPROM des AVR Mikrocontrollers geladen werden muss. Ebenfalls beinhaltet es Informationen über das Setzen der Fuse Bits. Fuse Bits sind Einstellungen des Mikrocontrollers, die nicht von der Software geändert werden können. Sie schalten gewisse Funktionen, zum Beispiel woher die Taktfrequenz bezogen wird, ein oder aus.

Codeauszug 4.5: Auszug aus examples/webserver-ipv6-raven/Makefile

```
1 TARGET=avr-raven
2 MCU=atmega1284p
3 OUTFILE=webserver6-$(TARGET)
4 avr-objcopy -O ihex -R .eeprom -R .fuse -R .signature \
5             $(OUTFILE).elf $(OUTFILE).hex
6 avr-size -C --mcu=$(MCU) $(OUTFILE).elf
```

Durch zusätzliche Befehle im Makefile wird eine Datei „webserver6-avr-raven.hex“ erzeugt. Diese Datei enthält den Inhalt, der in den Flash geschrieben werden soll, im Intel-HEX-Format. Mit dem „avr-size“-Befehl wird die Anzahl der Bytes angezeigt, die im Flash, im RAM und im EEPROM benötigt werden. Dies kann mit dem zur Verfügung stehenden Speicher verglichen werden.

Codeauszug 4.6: Ausgabe vom Befehl „avr-size“

```
1 AVR Memory Usage
2 -----
3 Device: atmega1284p
4
5 Program: 70874 bytes (54.1% Full)
6 (.text + .data + .bootloader)
7
8 Data: 13013 bytes (79.4% Full)
9 (.data + .bss + .noinit)
10
11 EEPROM: 63 bytes (1.5% Full)
12 (.eeprom)
```

Durch Hinzufügen eines zusätzlichen Befehls ins Makefile kann eine Datei „webserver6-avr-raven_eeprom.hex“ erzeugt werden. Diese Datei enthält dann den Inhalt im Intel-HEX-Format, der in den EEPROM geschrieben werden soll. Mit den beiden Dateien im Intel-HEX-Format ist es möglich, den Mikrocontroller mit einem Programmiergerät zu beschreiben, das kein ELF Format lesen kann.

Codeauszug 4.7: Befehl um eeprom.hex zu erzeugen

```
1 avr-objcopy -O ihex -j .eeprom \
2   -set-section-flags=.eeprom="alloc,load" --change-section-lma \
3   .eeprom=0 $(OUTFILE).elf $(OUTFILE)_eeprom.hex
```

Mittels des Programms Atmel Studio 6 werden wahlweise die erzeugten Dateien im Intel-HEX-Format oder die erzeugte Datei im ELF Format in den Flash und in den EEPROM des Mikrocontrollers programmiert. Zu Beginn wurde das Beispielprojekt webserver-ipv6-raven auf den Mikrocontroller AT-Mega1284P eines Atmel Ravenboard programmiert. Dabei wurde die Programmierschnittstelle ISP und das Programmiergerät Atmel STK500 verwendet.

Später bei der entwickelten Hardware wurde die Programmierschnittstelle JTAG und das Programmiergerät Atmel JTAGICE3 verwendet.

Kapitel 5: Tests

In diesem Kapitel wird die erstellte Lösung getestet und die Entwicklung validiert. Dazu ist die Testumgebung zu beschreiben.

Ziel ist der glaubhafte Nachweis der Funktionsfähigkeit, bzw. die quantitative Bewertung der erstellten Lösung.

Ein wichtiges Kriterium ist die Nachvollziehbarkeit der Tests. Das heißt, die Tests müssen so beschrieben sein, dass der Leser der Arbeit die Ergebnisse eigenständig wiederholen und validieren kann.

5.1 Herleitung von Test-Cases

Herleitung der Testmethodik.

5.1.1 Funktionale Tests

...

5.1.2 Leistungstests/Performancetests

...

5.2 Bewertung der Ergebnisse

...

Kapitel 6: Evaluation

In diesem Kapitel wird die erstellte Lösung getestet und die Entwicklung validiert. Dazu ist die Testumgebung zu beschreiben.

Ziel ist der glaubhafte Nachweis der Funktionsfähigkeit, bzw. die quantitative Bewertung der erstellten Lösung.

Ein wichtiges Kriterium ist die Nachvollziehbarkeit der Tests. Das heißt, die Tests müssen so beschrieben sein, dass der Leser der Arbeit die Ergebnisse eigenständig wiederholen und validieren kann.

6.1 Herleitung von Test-Cases

Herleitung der Testmethodik.

6.1.1 Funktionale Tests

...

6.1.2 Leistungstests/Performancetests

...

6.2 Bewertung der Ergebnisse

...

Kapitel 7: Fazit und Ausblick

In diesem Kapitel wird die erstellte Lösung begutachtet. Es werden Anwendungsmöglichkeiten, allgemein von Smart Objects und speziell für die Implementierung, behandelt. Mögliche Erweiterungen werden vorgeschlagen und ein Fazit der Arbeit wird gezogen.

7.1 Analyse der Implementierung

Die erstellte Lösung wird kritisch begutachtet. Dafür werden zuerst Kostenbetrachtungen bezogen auf die Hardware und auf die Software durchgeführt. Die Ergebnisse fließen in die abschließende Bewertung mit ein.

7.1.1 Kostenbetrachtung

Um in der Bewertung der Implementierung die Kosten einschätzen zu können, sollen hier zum einen die Hardwarekosten vorgestellt werden. Dazu sollen die einzelnen Materialkosten der verwendeten Bauteile zusammengefasst werden. Um die gesamten Hardwarekosten ermitteln zu können, müssen auch Entwicklungskosten und Fertigungskosten berücksichtigt werden. Fertigungskosten beinhalten dabei die eigentlichen Arbeitsstunden, die für die Fertigung aufgebracht werden müssen, als auch die Investitionskosten für den Fertigungsarbeitsplatz. Weil das sehr abhängig von der Art der Fertigung ist, sei es eine Einzelstückfertigung oder eine Serienfertigung, werden diese Kosten hier nicht näher betrachtet. Die Hardwarekosten werden rein durch die Materialkosten repräsentiert.

Zum anderen sollen die Softwarekosten abgeschätzt werden. Dies wird durch die Ermittlung der Entwicklungskosten für die Contiki-Applikation iec104 erreicht. Anhand eines ange nommenen Stundensatzes für einen Softwareentwickler und den aufgewendeten Stunden werden die Kosten für die Entwicklung berechnet.

Materialkosten

In Tabelle 7.1 sind die Bauteile aufgelistet, die zum Erstellen der Steckdose verwendet wurden. Sie benennt das Bauteil und zeigt die Anzahl an, die verbaut wurden. Die Bauteile wurden – bis auf die Tchibo Steckdose – vom Handelsunternehmen Mouser Electronics bezogen. In der dritten Spalte ist die entsprechende Artikelnummer aufgelistet. Die vierte

Tabelle 7.1: Materialkosten Steckdose

#	Bauteil	Artikelnummer	Einzel-preis in €	Massen-preis in €
Zigbit-Platine				
1	ATZB-24-A2	556-ATZB-24-A2	25,43	14,42
1	Widerstand 100kOhm	71-CMF60100K00FKEB	0,206	0,099
1	Widerstand 1kOhm	71-CMF551K0000FHEK	0,116	0,005
1	Kondensator 3,3uF	667-EEU-HD1H3R3	0,165	0,104
1	Spannungsregler LP2950CZ-3.0	926-2950CZ-3.0/NOPB	0,676	0,27
1	Steckverbinder FFC 6 Pin	538-52271-0679	1,30	0,583
2	Steckverbinder FFC 18 Pin	538-52271-1879	1,71	0,94
Steckdosen-Platine				
1	Kondensator X2 275V	80-R46KN368050M2M	0,66	0,263
1	Widerstand 560Ohm, 5Watt	594-AC05W560R0J	0,38	0,198
2	Widerstand 1MOhm	594-MRS251M1%TR	0,074	0,038
2	Gleichrichterdiode	512-1N4004	0,076	0,025
2	Zener-Diode 24V	512-1N4749ATR	0,203	0,036
1	Kondensator 470uF	667-EEU-FR1E471YB	0,248	0,152
1	NPN-Transistor BC547B	512-BC547B	0,186	0,05
1	Widerstand 220kOhm	271-220K-RC	0,125	0,012
Steckdosen-Gehäuse				
1	Tchibo Digitale Zeitschaltuhr	4 043002 669758	5,99	5,99
Gesamtpreis				38,61
				24,27

Spalte zeigt den Preis bei dem Bezug von nur einem Bauteil. Bei einer Massenfertigung der Steckdose werden andere Preise angeboten. Der Stückpreis bei einer Bestellung von 1000 Fertigungssätzen ist in der letzten Spalte aufgelistet. Die Preise wurden am 06.05.2013 abgefragt und sind inklusive Mehrwertsteuer angegeben.

Die Gesamtmaterialkosten der verwendeten Bauteile betragen 38,61€. Nicht berücksichtigt sind Verbrauchsmaterialien. Dazu zählen die Leiterplatte für die Zigbit-Platine, die aus einer vorhandenen Lochrasterplatine ausgesägt wurde, Leitungen für die Verkabelung und Lötzinn. Diese Kosten sollen hier vernachlässigt werden. Bei einer Kalkulation für eine Fertigung in einem Unternehmen müssen diese als Materialgemeinkostenzuschlag zu den Materialkosten hinzugefügt werden. Zusammen mit den Fertigungskosten, also die Arbeitskosten, die zum Fertigen der Steckdose notwendig sind, und den Verwaltungs- und Vertriebsgemeinkostenzuschlag ergeben sich die Selbstkosten je Stück.

Entwicklungskosten iec104

Um den Aufwand von einer Contiki-Applikation abzuschätzen, sollen beispielhaft die Entwicklungskosten der implementierten Applikation iec104 ermittelt werden. Zum Ermitteln

der Kosten, muss der Stundensatz des Entwicklers mit der Zeit in Stunden multipliziert werden, die dafür notwendig ist, eine solche Applikation zu programmieren. Dazu gehört eine Designphase, die eigentliche Programmierung und eventuelle Fehlerkorrekturen. Weil die Kosten proportional zur Zeit sind, soll nur diese betrachtet werden. Es wird geschätzt, dass für die Entwicklung der Applikation iec104 etwa drei bis vier Mannwochen benötigt wurden. Die Schätzung wird dadurch erschwert, dass nicht kontinuierlich an der Entwicklung gearbeitet wurde. Die Arbeit daran wurde öfter durch andere notwendige Arbeiten unterbrochen.

Deswegen soll noch eine quantitative Bewertung anhand der Anzahl geschriebener Zeilen Quellcode LOC (Lines Of Code) stattfinden. In der Praxis ist solch eine Bewertung, also der Rückschluss von den LOC auf den Aufwand, allerdings problematisch. Die Zeit, die für die Designphase verwendet wird, wird nicht berücksichtigt. Eine überlegte und optimierte Programmierung benötigt tendenziell weniger LOC. Die Qualität der Programmierung wird auch nicht bewertet. Verschiedene Programmiersprachen und individuelle Programmierstile wirken sich ebenso auf die LOC aus.

Trotzdem gibt es Faustformeln, die einen Zusammenhang zwischen der Anzahl geschriebener Zeilen Quellcode LOC und der eingesetzten Entwicklungszeit sehen. Nach **Ludewig:SoftwareEngineering** wird in einem Softwareprojekt nach n Stunden Aufwand – wobei hier der Gesamtaufwand gemeint ist, nicht die reine Programmierungszeit – ein System mit zwei mal n LOC erzeugt. Ein studentisches Projekt, das nicht kommerziell vertrieben werden soll, kann in der gleichen Zeit in etwa die dreifache Anzahl an LOC liefern. Dort können demnach mit jeder Stunde Aufwand in etwa sechs LOC entwickelt werden.

Um nach dieser Faustformel herauszufinden, wie viele Stunden in die Entwicklung der Contiki-Applikation iec104 eingeflossen sind, werden die Anzahl an Zeilen Quellcode ermittelt. Die Contiki-Applikation iec104 besteht aus 14 Dateien:

- 1 Makefile
- 7 Header-Dateien
- 6 C-Dateien

Insgesamt enthalten diese Dateien 1011 Zeilen. Wenn die 211 Leerzeilen und 112 Kommentarzeilen abgezogen werden, bleiben 688 Zeilen Quellcode. Nach der Faustformel oben ergibt sich eine Entwicklungszeit von etwa 115 Stunden. Bei der Annahme von einem Stundensatz von 100€ ergeben sich Entwicklungskosten von 11.500€. Bei einer Wochenarbeitszeit von 40 Stunden, ergeben sich ungefähr 3 Mannwochen, die notwendig sind, um die Contiki-Applikation iec104 zu entwickeln. Das deckt sich in etwa mit den Erfahrungen aus dieser Arbeit.

7.2 Anwendungsmöglichkeiten

Dieses Kapitel stellt zuerst allgemein verschiedene Anwendungsbereiche von Smart Objects vor. Darauffolgend wird erörtert, in welchen Anwendungsbereichen die erstellte Lösung eingesetzt werden könnte. Anhand der Erfahrung, die bei der Implementierung gemacht wurden, werden mögliche Erweiterungen behandelt.

7.2.1 Anwendungsbereiche von Smart Objects

Smart Objects können in einer Vielzahl von Anwendungsbereichen eingesetzt werden, da sie sehr flexibel in ihrer Beschaffenheit sind. Die Programmierung kann individuell angepasst werden. Durch die Verwendung der richtigen Sensoren und Aktoren sind sie für verschiedenste Einsatzbereiche verwendbar. Nachfolgend sollen folgende fünf mögliche Anwendungsbereiche kurz beschrieben werden:

- eHome-Bereich
- Gebäudeautomation
- Industriearmatisierung
- Logistik
- Smart Grid

Der eHome-Bereich ist ein Bereich in dem sich mehrere proprietäre Lösungen verbreitet haben, die oft nicht untereinander kompatibel sind. Dies kann auch ein Mitgrund dafür sein, dass sich die Heiamtatisierung bisher nicht so entwickelt hat wie vor etwa 7-9 Jahren prognostiziert [vasseur10interconnecting]. Wichtig ist auch, gerade im eHome-Bereich, eine einfache Installation der Geräte. Nur wenn ein Endnutzer ohne spezielles Expertenwissen Geräte in Betrieb nehmen und verwenden kann, wird sich eine Lösung durchsetzen können. Mögliche Einsatzgebiete von Smart Objects im eHome-Bereich sind zum Beispiel Steuerungen von Licht, der Heizung, von Fenster, von Rollläden und von Türschlössern.

Der eHome-Bereich wird für private Wohnhäuser eingesetzt. Im Gegensatz dazu zielt die Gebäudeautomation eher auf den professionellen Einsatz in großen Gebäuden meist im Zusammenspiel mit einem visualisierten Gebäudemanagementsystem. **Vermesan:TheInternetOfThings** beschreiben unter dem Titel „Smart IPv6 Building“ verschiedene Forschungsprojekte, die die Verwendung von IPv6 in der Gebäudeautomation untersuchen. Unter anderem wird auch das Hobnet-Projekt erwähnt, das spezielle Anwendungsfälle von Smart Objects in der Gebäudeautomation untersucht. Hier kommt auch 6LoWPAN zum Einsatz. Allgemein sind die Ziele einer Gebäudeautomation Energieeinsparungen, Sicherheit und Komfortgewinn.

Der Bereich Smart Grid wird einer der größten Anwendungsbereiche für Smart Objects werden [**Hersent:TheInternetOfThings**]. Seit mehreren Jahren geht der Trend in der

Stromerzeugung immer mehr in Richtung erneuerbaren Energiequellen wie Windkraftanlagen oder Photovoltaikanlagen. Diese sind aber im Gegensatz zu klassischen fossilen Kraftwerken dezentral aufgestellt. Das führt zu einem erheblichen Umbruch in der Stromnetzführung. Die Energie wird nicht allein an wenigen zentralen Orten durch große Kraftwerke, sondern auch dezentral durch kleine teils privat teils kommerziell geführten Energiequellen erzeugt. Das erschwert die Kraftwerksregelung und die Leitung des Lastflusses. Um trotzdem eine gute Netzstabilität zu gewährleisten, muss das Stromnetz intelligenter und vielfältiger überwacht und gesteuert werden.

Eine Maßnahme dabei ist das Smart Metering. Stromverbrauchszähler liefern über eine Kommunikationsschnittstelle den aktuellen Energieverbrauch an das Energieversorgungsunternehmen. Solche Art intelligente Zähler existieren schon länger für große Energieverbraucher wie Produktionsbetriebe, seit einigen Jahren werden Smart Meter auch für Privathaushalte angeboten.

7.2.2 Anwendungsmöglichkeiten für die Implementierung

Das Protokoll IEC104 ist ein Fernwirkprotokoll aus dem Bereich der Energieautomatisierung. Der Bereich Smart Grid ist also ein klassischer Anwendungsfall für dieses Protokoll. An ein zentrales Leitsystem werden Informationen übertragen und Steuerbefehle entgegen genommen. Allerdings wird der Zugriff auf eine Steuerung von einem elektrischen Verbraucher im Privathaushalt für ein Energieversorgungsunternehmen nicht von Bedeutung sein. Ein möglicher Anwendungsfall liegt eher in einem Smart Meter. Hier könnten mittels IEC104 Verbrauchsstände übertragen werden. Zusätzlich wäre in Absprache mit dem Privathaushalt eine Steuerung möglich. Genaue Anwendungsfälle müssen noch erprobt werden. Denkbar wäre eine Notabschaltung einer lokalen Photovoltaikanlage, wenn mehr Energie ins Stromnetz eingespeist wird als dort verbraucht wird bzw. in andere Stromnetzregionen abgeführt werden kann. Ebenfalls denkbar ist eine gezielte Steuerung von größeren Energieverbrauchern, bei denen der Einsatz zeitlich flexibel ist (Demand Side Management). Beispiele dafür sind die Waschmaschine oder die Batterie eines Elektrofahrzeugs. Bei einer großer Netzauslastung können diese steuerbaren Verbraucher zurückgefahren werden. Das Energieversorgungsunternehmen muss dann weniger Reserven für Spitzenlast vorhalten. So hilft eine intelligente Netzführung dabei, Kosten in der Energieerzeugung zu reduzieren.

7.2.3 Erweiterungsvorschläge für die Implementierung

Die erstellte Lösung ist ein Prototyp, der beispielhaft implementiert worden ist. Während der Arbeit sind mehrere Ideen entstanden, wie die Implementierung verbessert oder erweitert werden kann. Zuerst werden Vorschläge für die IEC104-Slave-Applikation aufgelistet.

- Informationsmeldungen mit Zeitstempel

Die Statusänderung der Steckdose erfolgt aktuell über den Datentyp M_SP_NA_1

(Type Ident 1): Einzelbitmeldung ohne Zeitstempel. Eine Verbesserung wäre die Verwendung des Datentyps M_SP_TB_1 (Type Ident 30): Einzelbitmeldung mit dem Zeitstempel CP56Time2a. Hier wird zusätzlich ein Zeitstempel mit Jahr, Monat, Tag, Stunde, Minute, Sekunde und Millisekunde übertragen. So ist der exakte Zeitpunkt der Statusänderung bekannt. Voraussetzung dafür ist aber, dass die Steuerung mit der aktuellen Zeit synchronisiert ist. Über IEC104 existiert dafür eine Möglichkeit mit dem Datentyp C_CS_NA_1 (Type Ident 103): Zeitsynchronisationsbefehl. Ein Zeitstempel wird vom IEC104-Master zum IEC104-Slave gesendet und dort übernommen. Weil die Übertragungszeit dabei aber nicht berücksichtigt wird, ist die Verwendung vom IEC-Standard nur bedingt empfohlen. In der Praxis wird oft das Protokoll NTP (Network Time Protocol) verwendet. Eine Implementierung von NTP im Betriebssystem Contiki existiert bereits **Contiki-syslog**.

- interne Statusinformationen des Contiki Betriebssystems

Das Modul iec104-para kann mit zusätzlichen Informationsobjekten erweitert werden. Zusätzliche interne Statusinformationen wie die Anzahl der laufenden Prozesse, TCP/IP-Verbindungen oder die Betriebszeit könnten übertragen werden. Für Messwerte gibt es bereits den Datentyp M_ME_NA_1 (Type Ident 9): Messwert, normalisiert und ohne Zeitstempel. Weitere Datentypen können implementiert werden.

Eine weitere Verbesserungsmöglichkeit betrifft nicht die Implementierung selbst sondern dem Testaufbau. Als 6LoWPAN-Router wurde ein handelsüblicher PC mit einer Linux-Installation verwendet. Als 6LoWPAN-Schnittstelle wurde ein USB-Stick RZUSBSTICK von Atmel verwendet, der auf dem PC eine Ethernet-Schnittstelle simuliert. Unter anderem hat das den Nachteil, dass auf dem PC der 6LoWPAN-Netzwerkverkehr nicht beobachtet werden kann, weil nur der simulierte Ethernet-Verkehr sichtbar ist. Es wird aber aktuell daran gearbeitet, 6LoWPAN direkt im Linux-Kernel zu implementieren [**Ott2012**]. Eine eingeschränkte Variante wird schon seit der Kernel Version 3.2.46 unterstützt und laufend erweitert. Als Funkchips werden der MRF24J40 von Microchip und der AT86RF230 von Atmel, der auch im Zigbit-Modul verbaut ist, unterstützt. **Ott2012** stellt eine Hardware vor bestehend aus einem BeagleBone und einem MRF24J40MA Funkchip. Damit kann 6LoWPAN nativ unter Linux verwendet werden ohne zusätzliche Hardware und ohne ein zusätzliches Betriebssystem. Es ist allerdings nicht bekannt, ob es Interoperabilitätsprobleme zwischen der 6LoWPAN-Implementierung im Linux-Kernel und im Contiki-Betriebssystem gibt.

7.3 Zusammenfassung

Es konnte gezeigt werden, dass eine internetfähige Steuerung mithilfe eines 8-bit Mikrocontrollers implementiert werden kann. Das Betriebssystem Contiki stellt dazu den notwendigen TCP/IP-Stack und andere Werkzeuge und Hilfsmittel zur Verfügung. Eine Webserver-Anwendung und eine Vielzahl anderer Anwendungen sind Teil des Betriebssystems. Mit der

Contiki-Applikation iec104 wurde gezeigt, dass die Implementierung des Protokolls IEC104 in einer limitierten Umgebung durchaus möglich ist. Auch die Verwendung von IEC104 im Zusammenspiel mit IPv6 hat funktioniert. Bisher war keine Implementierung von IEC104 über IPv6 bekannt. Generell zeigt die Contiki-Applikation iec104, dass die Implementierung neuer internetfähiger Anwendungen möglich und nicht aufwendiger als bei anderen Betriebssystemen ist. Die geringen Ressourcen müssen allerdings bei der Programmierung berücksichtigt werden.

Bei der Implementierung ist der 8KByte große RAM-Speicher die markanteste Ressourcen-grenze. Die Auslastung beträgt 88,9%. Das Hinzufügen von zusätzlichen Contiki-Applika-tionen oder das Erweitern der Applikation iec104 ist deswegen nicht ohne Weiteres mög-lich. Durch den Einsatz von anderer Hardware mit einem größeren RAM-Speicher kann dieses Problem umgangen werden. So steht bei dem AVR Ravenboard mit 16KByte doppelt so viel RAM-Speicher zur Verfügung. Der Hersteller Redwire bietet mit dem Econotag ein fertiges Modul mit USB-Schnittstelle an. Verwendet wird der Freescale MC13224V, der einen 32-bit ARM7 Mikrocontroller mit einer IEEE-802.15.4-Schnittstelle kombiniert. Er verfügt über 128KByte Flash- und 96KByte RAM-Speicher. Das Betriebssystem Contiki unter-stützt diese Modul über die Platform redbee-econotag. Beide, das AVR Ravenboard und das Econotag, sind aber für den Einbau in die verwendete Steckdose zu groß.

Als Kommunikationstechnologie wurde 6LoWPAN verwendet. Es scheint ideal für den Einsatz bei ressourcenbeschränkten Geräten zu sein. Anfang dieses Jahres hat die ZigBee Alliance die Spezifikation ZigBee IP veröffentlicht [[zigbee-ip](#)]. Damit werden unter der Verwendung von 6LoWPAN vermaschte drahtlose IPv6-Netzwerke unterstützt. Dies ist ein Beispiel dafür, dass die Verwendung von 6LoWPAN in vielen Bereichen voranschreitet. Eine weitere Entwicklung, die die Verbreitung von 6LoWPAN unterstützt, ist die Spezifizierung des Protokolls RPL [[RFC6550](#)]. RPL ist ein Routing-Protokoll, das für verlustbehaftete Sensornetze entwickelt worden ist. Die speziellen Anforderungen konnten von existierenden Routing-Protokollen wie OSPF oder RIP nicht erfüllt werden. Die Implementierung von RPL im Betriebssystem Contiki wird ContikiRPL genannt [[tsiftes10rpl](#)].

Sehr wichtig für die weitere Verbreitung von 6LoWPAN ist die Interoperabilität von ver-schiedenen Implementierungen. **ko11beyond** haben zwei unabhängige Implementierungen, Contiki und TinyOS, zusammen getestet. Die Interoperabilität zwischen beiden war gege-ben, allerdings hatten kleine Unterschiede im jeweiligen Protokollstack einen Einfluss auf die Gesamtsystemleistung. Neben der Interoperabilität ist die Leistungsfähigkeit bei dem Zusammenspiel verschiedener Implementierungen von Bedeutung.

Als weitere Schwierigkeit kommt hinzu, dass viele Implementierungen den Funkempfänger so oft wie möglich ausschalten. Im Betriebssystem Contiki steht diese Funktionalität unter dem Namen ContikiMAC zur Verfügung. So kann der Energieverbrauch um bis zu 80% re-duziert werden [[dunkels11contikimac](#)]. Das gesteuerte Ausschalten des Funkempfängers hat allerdings einen markanten Einfluss auf das Kommunikationsverhalten im Netzwerk. Weil keine Spezifikationen oder Standards für diese Funktionalität existieren, verhalten

sich Implementierungen sehr verschieden. **dunkels11adhoc** beschreiben, dass das Zusammenspiel von Kommunikationstechnologie und Ausschaltverhalten des Funkempfängers ein wichtiges Forschungsgebiet für das Internet-of-Things ist.

Anhang A: Der Blindtext

Weit hinten, hinter den Wortbergen, fern der Laender Vokalien und Konsonantien leben die Blindtexte. Abgeschieden wohnen Sie in Buchstabhausen an der Kueste des Semantik, eines grossen Sprachozeans. Ein kleines Baechlein namens Duden fliest durch ihren Ort und versorgt sie mit den noetigen Regelialien. Es ist ein paradiesmatisches Land, in dem einem gebratene Satzteile in den Mund fliegen. Nicht einmal von der allmaechtigen Interpunktion werden die Blindtexte beherrscht - ein geradezu unorthographisches Leben.

Eines Tages aber beschloss eine kleine Zeile Blindtext, ihr Name war Lorem Ipsum, hinaus zu gehen in die weite Grammatik. Der grosse Oxmox riet ihr davon ab, da es dort wimmele von boesen Kommata, wilden Fragezeichen und hinterhaeltigen Semikoli, doch das Blindtextchen liess sich nicht beirren. Es packte seine sieben Versalien, schob sich sein Initial in den Guertel und machte sich auf den Weg. Als es die ersten Huegel des Kursivgebirges erklommen hatte, warf es einen letzten Blick zurueck auf die Skyline seiner Heimatstadt Buchstabhausen, die Headline von Alphabetdorf und die Subline seiner eigenen Strasse, der Zeilengasse. Wehmuetig lief ihm eine rhetorische Frage ueber die Wange, dann setzte es seinen Weg fort. Unterwegs traf es eine Copy. Die Copy warnte das Blindtextchen, da, wo sie herkaeme waere sie zigmał umgeschrieben worden und alles, was von ihrem Ursprung noch uebrig waere, sei das Wort und und das Blindtextchen solle umkehren und wieder in sein eigenes, sicheres Land zurueckkehren. Doch alles Gutzureden konnte es nicht ueberzeugen und so dauerte es nicht lange, bis ihm ein paar heimtueckische Werbetexter auflauerten, es mit Longe und Parole betrunken machten und es dann in ihre Agentur schleppten, wo sie es fuer ihre Projekte wieder und wieder missbrauchten. Und wenn es nicht umgeschrieben wurde, dann benutzen Sie es immernoch.

Abbildungsverzeichnis

2.1	Der TCP Three-Way-Handshake. Entscheidend für die Zuordnung ist, dass die <i>Acknowledgment Number</i> im zweiten Schritt dem Wert der initialen <i>Sequence Number</i> + 1 entspricht ($x + 1$).	4
2.2	Aufbau des TCP-Headers (ohne Nutzdaten). Für SYN-Scans sind insbesondere die <i>Control Bits</i> (z. B. SYN, RST, ACK) relevant.	5
2.3	Aufbau eines drahtlosen Sensornetzwerkes	9
2.4	Die Internet-of-Things Vision Bormann:6LoWPAN	11

Tabellenverzeichnis

2.1	TCP Header Felder	6
2.2	OSI-7-Schichtenmodell	9
7.1	Materialkosten Steckdose	24

Quelltextverzeichnis

4.1	Contiki-Verzeichnisstruktur mit ausgewählten Unterverzeichnissen	15
4.2	Auszug aus examples/webserver-ipv6-raven/Makefile	16
4.3	Auszug aus examples/webserver-ipv6-raven/Makefile.webserver	17
4.4	Auszug aus examples/webserver-ipv6-raven/webserver6.c	17
4.5	Auszug aus examples/webserver-ipv6-raven/Makefile	17
4.6	Ausgabe vom Befehl „avr-size“	18
4.7	Befehl um eeprom.hex zu erzeugen	18

Stichwortverzeichnis

S

Smart Object 10

Literaturverzeichnis

- [1] H. Griffioen, G. Koursounis, G. Smaragdakis und C. Doerr, „Have you syn me? characterizing ten years of internet scanning,“ in *Proceedings of the 2024 ACM on Internet Measurement Conference*, 2024, S. 149–164.
- [2] Z. Durumeric, D. Adrian, P. Stephens, E. Wustrow und J. A. Halderman, „Ten Years of ZMap,“ en, in *Proceedings of the 2024 ACM on Internet Measurement Conference*, Madrid Spain: ACM, Nov. 2024, S. 139–148, ISBN: 979-8-4007-0592-2. DOI: 10.1145/3646547.3689012 Adresse: <https://dl.acm.org/doi/10.1145/3646547.3689012>
- [3] Z. Durumeric, E. Wustrow und J. A. Halderman, „ZMap: Fast Internet-wide Scanning and Its Security Applications,“ en,
- [4] R. D. Graham, *robertdavidgraham/masscan*, C, Jan. 2026. Adresse: <https://github.com/robertdavidgraham/masscan>
- [5] S. Rudnev, A. Zolkin, N. Artemyev und A. Tychkov, „THE ECONOMIC IMPORTANCE OF CYBERSECURITY FOR ENTERPRISES IN THE CONTEXT OF DIGITAL TRANSFORMATION,“ *EKONOMIKA I UPRAVLENIE: PROBLEMY, RSHENIYA*, Jg. 11/2, S. 46–55, Jan. 2024. DOI: 10.36871/ek.up.p.r.2024.11.02.006
- [6] O. I. Falowo, I. Okpala, E. Kojo, S. Azumah und C. Li, „Exploration of Various Machine Learning Techniques for Identifying and Mitigating DDoS Attacks,“ in *2023 20th Annual International Conference on Privacy, Security and Trust (PST)*, Aug. 2023, S. 1–7. DOI: 10.1109/PST58708.2023.10320151 Adresse: <https://ieeexplore.ieee.org/document/10320151/>
- [7] X. Li, *idealeer/xmap*, C, Jan. 2026. Adresse: <https://github.com/idealeer/xmap>
- [8] W. Bugden und A. Alahmar, „The safety and performance of prominent programming languages,“ *International Journal of Software Engineering and Knowledge Engineering*, Jg. 32, Nr. 05, S. 713–744, 2022.
- [9] M. Costanzo, E. Rucci, M. Naiouf und A. D. Giusti, „Performance vs Programming Effort between Rust and C on Multicore Architectures: Case Study in N-Body,“ Nr. arXiv:2107.11912, Okt. 2021, arXiv:2107.11912 [cs]. DOI: 10.48550/arXiv.2107.11912 Adresse: <http://arxiv.org/abs/2107.11912>
- [10] G. Lyon, *Nmap network scanning: official Nmap project guide to network discovery and security scanning*, eng, Zero-day release: May 2008. Sunnyvale, CA: Insecure.Com LLC, 2010, ISBN: 978-0-9799587-1-7.

- [11] IANA, *Service Name and Transport Protocol Port Number Registry*. Adresse: <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>
- [12] S. Wendzel, *IT-Sicherheit für TCP/IP- und IoT-Netzwerke: Grundlagen, Konzepte, Protokolle, Härtung* (Springer eBook Collection), ger, 2., aktualisierte und erweiterte Auflage. Wiesbaden: Springer Vieweg, 2021, ISBN: 978-3-658-33422-2. DOI: 10.1007/978-3-658-33423-9
- [13] J. Postel, *Transmission Control Protocol*, en. 1981, RFC0793. DOI: 10.17487/rfc0793 Adresse: <https://www.rfc-editor.org/info/rfc0793>
- [14] W. Eddy, *Transmission Control Protocol (TCP)*. Aug. 2022. DOI: 10.17487/RFC9293 Adresse: <https://datatracker.ietf.org/doc/rfc9293>
- [15] W. Eddy, *TCP SYN Flooding Attacks and Common Mitigations*. Aug. 2007. DOI: 10.17487/RFC4987 Adresse: <https://datatracker.ietf.org/doc/rfc4987>
- [16] M. Kerrisk, *The Linux programming interface: a Linux und UNIX system programming handbook*, eng, Ninth printing. San Francisco, CA: No Starch Press, 2018, ISBN: 978-1-59327-220-3.
- [17] *socket(2) - Linux manual page*, man7.org, Accessed: 2026-01-11. Adresse: <https://man7.org/linux/man-pages/man2/socket.2.html>
- [18] *raw(7) - Linux manual page*, man7.org, Accessed: 2026-01-11. Adresse: <https://man7.org/linux/man-pages/man7/raw.7.html>
- [19] *address_families(7) - Linux manual page*, man7.org, Accessed: 2026-01-11. Adresse: https://man7.org/linux/man-pages/man7/address_families.7.html
- [20] S. McCanne und V. Jacobson, „The BSD Packet Filter: A New Architecture for User-level Packet Capture,“ in *USENIX Winter 1993 Conference (USENIX Winter 1993 Conference)*, San Diego, CA: USENIX Association, Jan. 1993. Adresse: <https://www.usenix.org/conference/usenix-winter-1993-conference/bsd-packet-filter-new-architecture-user-level-packet>
- [21] N. R. Pinnapareddy, „eBPF for high-performance networking and security in cloud-native environments,“ *International Journal of Science and Research Archive*, Jg. 15, Nr. 2, S. 207–225, Mai 2025, ISSN: 25828185. DOI: 10.30574/ijjsra.2025.15.2.1264
- [22] T. Høiland-Jørgensen u. a., „The eXpress data path: fast programmable packet processing in the operating system kernel,“ en, in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, Heraklion Greece: ACM, Dez. 2018, S. 54–66, ISBN: 978-1-4503-6080-7. DOI: 10.1145/3281411.3281443 Adresse: <https://dl.acm.org/doi/10.1145/3281411.3281443>
- [23] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior und L. F. M. Vieira, „Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications,“ en, *ACM Computing Surveys*, Jg. 53, Nr. 1, S. 1–36, Jan. 2021, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3371038

-
- [24] X. Zhang, X. Shu, L. Chen und R. Xie, „High-Performance Network Firewall Based on XDP,“ in *2024 20th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, Guangzhou, China: IEEE, 2024, S. 1–6, ISBN: 979-8-3503-5632-8. DOI: 10.1109/ICNC-FSKD64080.2024.10702282
Adresse: <https://ieeexplore.ieee.org/document/10702282/>

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 06.02.2025

Lennard Alexander Dubhorn