FernUniversität in Hagen
Fakultät für Mathematik und Informatik
LG Kooperative Systeme

UDP Transport Options

Implementierung und Analyse von RFC 9868 in Zig

Masterarbeit

im Studiengang Master Informatik

vorgelegt von
Alan Bernstein
Matrikelnummer: 2068834

Erstprüfer: Prof. Dr. Christian Icking

Zweitprüferin: Dr. Lihong Ma

Hagen, [Datum der Abgabe]

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

APC Alternate Payload Checksum.

AUTH Authentication Option.

DTLS Datagram Transport Layer Security.

EOL End of List.

FRAG Fragmentation Option.

MDS Maximum Datagram Size.

MRDS Maximum Reassembled Datagram Size.

NOP No Operation.

OCS Option Checksum.

PMTU Path Maximum Transmission Unit.

TIME Timestamp Option.

1. Einleitung

1.1 Motivation

Das User Datagram Protocol (UDP) wurde 1980 in RFC 768 spezifiziert und ist seitdem praktisch unverändert geblieben [1]. Als minimalistisches Transportprotokoll bietet UDP lediglich Portnummern zur Demultiplexierung und eine Prüfsumme zur Fehlererkennung[1]. Diese bewusste Einfachheit macht UDP zu einem der am häufigsten genutzten Transportprotokolle im Internet.

Moderne Anwendungen nutzen UDP zunehmend für zeitkritische Kommunikation. Protokolle wie QUIC, WebRTC, DNS und zahlreiche Spiele- sowie IoT-Anwendungen setzen auf UDP, um die Latenz des TCP-Handshakes zu vermeiden und mehr Kontrolle über die Übertragungsmechanismen zu erhalten.

RFC 9868, veröffentlicht im Oktober 2025, schließt diese Lücke durch die Einführung von *Transport Options for UDP* [3]. Die Erweiterung nutzt die sogenannte *Surplus Area* – den Bereich zwischen dem Ende der UDP-Nutzdaten und dem Ende des IP-Datagramms – für optionale Transportschicht-Mechanismen. Da das UDP-Length-Feld die Länge der Nutzdaten angibt und diese von der IP-Paketlänge abweichen kann, entsteht ein bisher ungenutzter Bereich für Erweiterungen.

RFC 9868 definiert ein erweiterbares Options-Framework mit folgenden Eigenschaften:

- Optionen werden im TLV-Format (Type-Length-Value) kodiert und in der *Surplus Area* platziert
- Kategorisierung in SAFE Options (können ignoriert werden) und UNSAFE Options (modifizieren Nutzdaten)
- Unterstützung für Sicherheitsfunktionen (Prüfsummen, Integrität)
- Fragmentierung auf UDP-Ebene, unabhängig von IP-Fragmentierung

Ein entscheidender Vorteil der UDP-Options ist ihre **Abwärtskompatibilität**: Legacy-Systeme, die UDP Options nicht unterstützen, ignorieren die Surplus Area und verarbeiten lediglich die regulären UDP-Nutzdaten. Dies ermöglicht eine schrittweise Einführung der Erweiterung ohne Beeinträchtigung bestehender Infrastruktur.

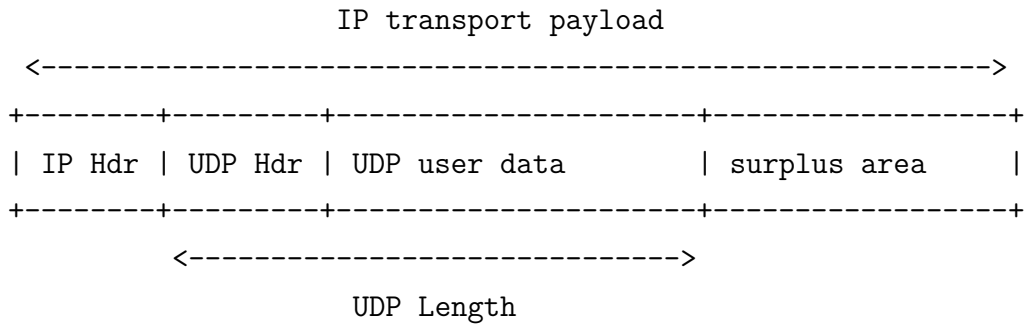
[1] Die Prüfsumme ist bei IPv4 optional [1] und bei IPv6 verpflichtend [2, Sec. 8.1].

1.2 Problemstellung

Die Implementierung von UDP Options gemäß RFC 9868 wirft mehrere technische Herausforderungen auf, die in dieser Arbeit adressiert werden.

1.2.1 Das Surplus-Area-Konzept

Anders als TCP, dessen Header ein explizites Optionsfeld vorsieht, bietet der ursprüngliche UDP-Header keinen Raum für Erweiterungen. RFC 9868 löst dieses Problem durch Nutzung der *Surplus Area*: Da das UDP-Length-Feld die Länge des UDP-Headers plus Nutzdaten angibt, kann diese kleiner sein als die vom IP-Header implizierte Transportnutzlast. Der entstehende Bereich wird für UDP Options verwendet:



Die Surplus Area beginnt mit einem 2-Byte Option Checksum (OCS), gefolgt von den eigentlichen Optionen im TLV-Format (Type-Length-Value). Diese Struktur ermöglicht variable Optionslängen und zukünftige Erweiterungen.

1.2.2 Unidirektionalität und Statelessness

RFC 9868 stellt explizit klar: *"UDP is unidirectional; UDP Options do not change that fact."* [3, Sec. 6]. Diese Design-Entscheidung hat weitreichende Implikationen:

- **Keine Option erfordert eine Antwort:** Keine der in RFC 9868 definierten Optionen verpflichtet den Empfänger zu einer Reaktion. Selbst REQ (Echo Request) erzeugt keine automatische Antwort auf Protokollebene.
- **Antworten sind Anwendungslogik:** Wenn ein Empfänger auf eine REQ-Option mit einer RES-Option (Echo Response) antwortet, geschieht dies durch eine bewusste Entscheidung der Anwendung, nicht durch einen Protokollmechanismus. Die Anwendung entscheidet, *wann* und *ob* eine Antwort gesendet wird.

Für Anwendungsfälle, die bidirektionale Kommunikation oder garantierte Antworten erfordern, verweist RFC 9868 auf ein separates Dokument, das solche Mechanismen spezifizieren soll. Dies unterstreicht, dass UDP Options bewusst auf der unidirektionalen Natur von UDP aufbauen.

Für die Implementierung bedeutet dies, dass beide Kommunikationspartner als symmetrische Peers agieren: Jeder Peer muss sowohl senden als auch empfangen können, ohne dass das Protokoll eine Client-Server-Hierarchie vorschreibt.

1.2.3 Implementierungsherausforderungen

Die zentrale Herausforderung bei der Implementierung von UDP Options liegt im Zugriff auf die Surplus Area. Existierende Betriebssystem-Stacks liefern über Standard-Socket-APIs ausschließlich die durch das UDP-Length-Feld begrenzten Nutzdaten an die Anwendungsschicht; die Surplus Area wird stillschweigend verworfen [3, Sec. 18]. Da die ursprüngliche UDP-Spezifikation [1] keinen Mechanismus für den Zugriff auf Daten jenseits der UDP-Länge vorsieht, erfordert eine Implementierung im Userspace den Einsatz von Raw Sockets, die vollständige Kontrolle über die Paketstruktur ermöglichen, jedoch erhöhte Systemberechtigungen voraussetzen.

Eine weitere Herausforderung stellt die Kompatibilität mit Middleboxes dar. Während NAT-Geräte UDP-Pakete mit Surplus Area typischerweise korrekt weiterleiten – sie modifizieren lediglich Header-Felder und ignorieren den übrigen IP-Payload – interpretieren bestimmte Intrusion Detection Systeme und Firewalls die Diskrepanz zwischen UDP-Length und IP-Length als Angriffsindikator [3, Sec. 18]. RFC 9868 adressiert dieses Problem durch die Option Checksum (OCS).

1.3 Zielsetzung

Ziel dieser Arbeit ist die Implementierung einer RFC 9868-konformen Bibliothek für UDP Transport Options in der Programmiersprache Zig. Die Implementierung fokussiert auf eine Userspace-Lösung mittels Raw Sockets und umfasst Parser sowie Serializer für TLV-kodierte UDP Options, OCS-Validierung gemäß Section 9 des RFC 9868, Unterstützung der Must-Support Options (EOL, NOP, APC, FRAG, MDS, MRDS, REQ, RES) sowie Raw-Socket-Programmierung für den Zugriff auf die Surplus Area.

Die Verifikation der Implementierung erfolgt durch Unit Tests für die einzelnen Komponenten sowie Integrationstests über das Loopback-Interface. Zusätzlich wird das Verhalten der UDP Options bei der Übertragung über das Internet untersucht, um die Kompatibilität mit bestehender Netzwerkinfrastruktur zu evaluieren.

Explizit außerhalb des Umfangs dieser Arbeit liegen der REQ/RES-Anwendungsfall

für Path MTU Discovery, welcher in RFC 9869 (DPLPMTUD for UDP Options) definiert ist [4], die TIME Option, deren Anwendungsfall im RFC nicht spezifiziert ist, sowie die Optionen AUTH, UCMP und UENC, die laut Section 11.9 des RFC noch nicht vollständig spezifiziert sind. Die Implementierung erfolgt ausschließlich im Userspace; Kernel-Module werden nicht entwickelt.

1.4 Eingesetzte Technologien

Als zentrale Programmiersprache wurde **Zig** (Version 0.15.2) gewählt. Zig ist eine relative neue Systemprogrammiersprache mit manueller Speicherverwaltung ohne versteckte Allokationen – ein entscheidender Vorteil für die byte-genaue Konstruktion von Netzwerk-Headern. Die nahtlose C-Interoperabilität ermöglicht die direkte Nutzung von POSIX-Schnittstellen für Raw Sockets. Zusätzlich bieten integrierte Laufzeitprüfungen einen wirksamen Schutz gegen Speicherfehler beim Parsen von Netzwerkpaketen.

Für die Analyse des Netzwerkverkehrs und die Verifikation der generierten UDP-Pakete sind **Wireshark** und **tcpdump** unverzichtbar, da sie eine detaillierte Inspektion auf Bitebene erlauben.

Ergänzend wird ein transparenter, KI-gestützter Ansatz verfolgt. Large Language Models (LLMs) wie **Claude** (Anthropic) und **Codex** (OpenAI) dienen als Assistenzsysteme für Code-Reviews, die Analyse komplexer RFC-Spezifikationen, LaTeX-Formatierung sowie zur sprachlichen Glättung von Texten.

2. Grundlagen

2.1 Das User Datagram Protocol (UDP)

Das User Datagram Protocol (UDP) ist neben dem Transmission Control Protocol (TCP) eines der beiden zentralen Transportprotokolle der TCP/IP-Protokollfamilie. Im Gegensatz zu TCP verfolgt UDP ein bewusst minimalistisches Design: Es sendet Datagramme ohne vorherigen Handshake oder Zustandsabgleich mit dem Empfänger, bietet keine Garantien für Zustellung, Reihenfolge oder Duplikaterkennung und verzichtet auf jeglichen Flusskontroll- oder Staukontrollmechanismus [1]. Mit der Protokollnummer 17 im IP-Header registriert, wurde UDP in RFC 768 spezifiziert und trägt die Bezeichnung STD 6 als Internet-Standard. Bemerkenswert ist, dass der Protokollstandard RFC 768 seit seiner Veröffentlichung im August 1980 nie direkt aktualisiert wurde – eine Stabilität von über 45 Jahren, die erst mit RFC 9868 durchbrochen wird [3].

2.1.1 Historische Entwicklung

Die Entstehung von UDP ist untrennbar mit der Aufspaltung des ursprünglichen *Transmission Control Program* verbunden. In RFC 675 beschrieben Cerf, Dalal und Sunshine im Dezember 1974 ein monolithisches Protokoll, das sowohl Netzwerk- als auch Transportfunktionen in einem einzigen Protokoll vereinte [5]. Ab 1978 wurde dieses Konzept schrittweise in separate Schichten aufgeteilt: Das Internet Protocol (IP) übernahm die Netzwerkfunktionen, während TCP die zuverlässige Transportschicht bildete. Die Erkenntnis, dass nicht alle Anwendungen die Zuverlässigkeitsgarantien und den damit verbundenen Overhead von TCP benötigen, führte zur Entwicklung eines leichtgewichtigen Alternativprotokolls – UDP.

Am 28. August 1980 veröffentlichte Jon Postel am Information Sciences Institute (ISI) der University of Southern California RFC 768, die Spezifikation des User Datagram Protocol [1]. Mit nur drei Seiten ist RFC 768 eines der kürzesten RFC-Dokumente überhaupt[1]. Diese Kürze spiegelt das Designziel wider: UDP sollte ein minimalistisches, transaktionsorientiertes Protokoll sein, das lediglich Portnummern zur Demultiplexierung und eine optionale Prüfsumme zur Fehlererkennung bereitstellt. Als erste Anwendungsfälle nennt RFC 768 explizit den Internet Name Server[2] sowie das Trivial File Transfer Protocol (TFTP, beschrieben in IEN 133).

[1] Zum Vergleich: RFC 793 (TCP) umfasst 85 Seiten, RFC 9868 (UDP Options) 51 Seiten.
[2] Ein Vorläufer des heutigen DNS, das diesen frühen Namensdienst später ablöste.

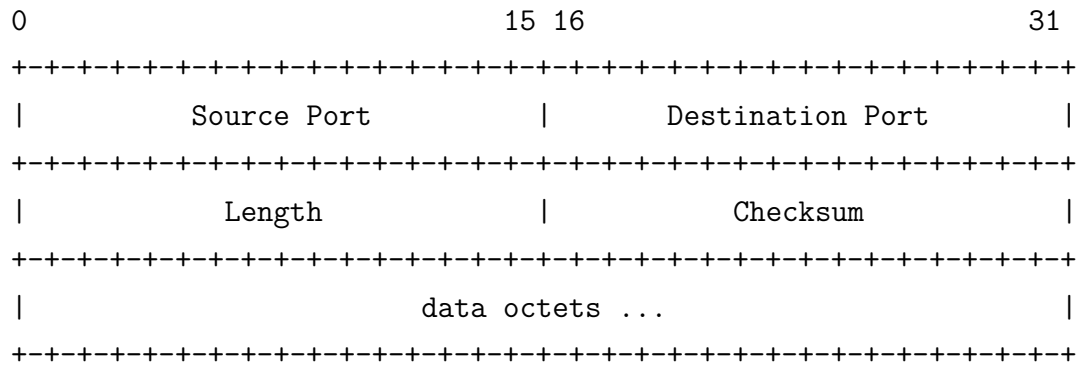
Die endgültige Formalisierung des TCP/IP-Stacks erfolgte im September 1981 mit der Veröffentlichung von RFC 791 (IPv4) [6] und RFC 793 (TCP) [7]. Gemeinsam mit UDP bildeten diese drei Protokolle die Grundpfeiler der Internet-Protokollsuite. Eine weitere wichtige Ergänzung stellte RFC 1122 dar, in dem Braden 1989 die Implementierungsanforderungen für Internet-Hosts präzisierte [8]. Dieses Dokument konkretisierte unter anderem die Anforderungen an UDP-Implementierungen, etwa die Behandlung von Paketen mit ungültiger Prüfsumme und das Verhalten bei nicht erreichbaren Zielports.

Seit seiner Veröffentlichung 1980 ist der Protokollstandard RFC 768 im Kern unverändert geblieben. Zu den wenigen ergänzenden Anpassungen zählen die Prüfsummenbehandlung und die Unterstützung von Jumbogrammen: Während die Prüfsumme bei IPv4 optional ist, schreibt RFC 8200 sie für IPv6 grundsätzlich als verpflichtend vor [2, Sec. 8.1] – mit der Ausnahme, dass RFC 6936 für bestimmte Tunnelprotokolle einen Zero-Checksum-Modus erlaubt [9]. RFC 2675 erweiterte UDP zudem um die Unterstützung von IPv6-Jumbogrammen, bei denen das UDP-Längenfeld auf null gesetzt wird [10]. RFC 8085 fasst als BCP 145 die aktuellen Richtlinien für die Nutzung von UDP zusammen und adressiert insbesondere Fragen der Staukontrolle bei UDP-basierten Anwendungen [11].

Trotz dieser ergänzenden Spezifikationen hat die Bedeutung von UDP in den vergangenen Jahren massiv zugenommen: Protokolle wie QUIC [12], das HTTP/3 als Transportschicht dient [13], WebRTC, DNS und zahlreiche IoT-Anwendungen setzen auf UDP als Basis. Durch die Verbreitung von QUIC und HTTP/3 wird mittlerweile ein erheblicher und wachsender Anteil des Web-Verkehrs über UDP abgewickelt. Gerade diese weitgehende Stabilität von RFC 768 bei gleichzeitig wachsender Bedeutung von UDP motiviert RFC 9868 als erste direkte Erweiterung des Protokollstandards seit über 45 Jahren.

2.1.2 Protokollstruktur

Ein UDP-Datagramm besteht aus einem Header mit fester Länge von 8 Byte und den darauf folgenden Nutzdaten. RFC 768 definiert den Header wie folgt [1]:

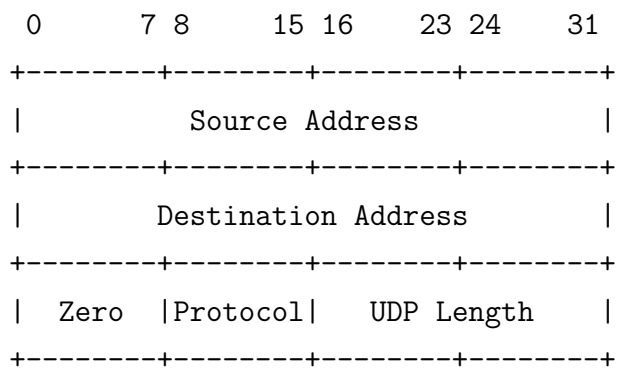


Die vier Felder des Headers haben jeweils eine Breite von 16 Bit:

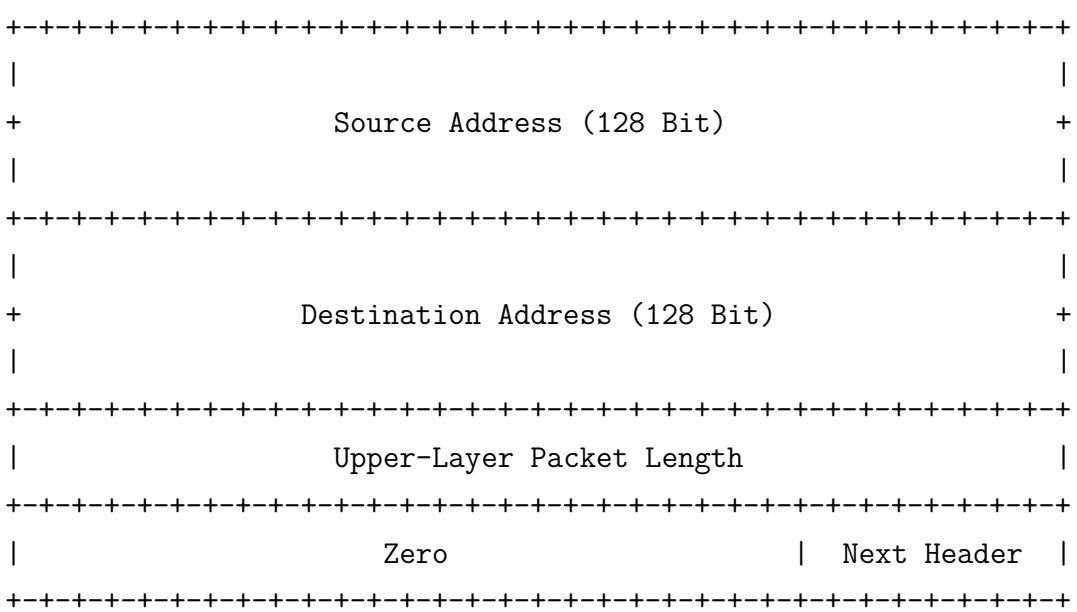
- `Source Port` – Die Portnummer des sendenden Prozesses. Dieses Feld ist optional; wird es nicht benötigt, wird es auf null gesetzt [1]. In der Praxis nutzen Anwendungen den *Source Port* zur Identifikation des Absenders, damit der Empfänger Antworten an den korrekten Port zurücksenden kann.
- `Destination Port` – Die Portnummer des Zielprozesses auf dem empfangenden Host. Zusammen mit der IP-Zieladresse ermöglicht dieses Feld die *Demultiplexierung* eingehender Datagramme an die korrekte Anwendung.
- `Length` – Die Gesamtlänge des UDP-Datagramms in Byte, bestehend aus Header *und* Nutzdaten. Der Minimalwert beträgt 8, was einem Datagramm ohne Nutzdaten entspricht [1]. Da das Feld 16 Bit breit ist, ergibt sich eine theoretische Maximalgröße von 65 535 Byte[3].
- `Checksum` – Eine Prüfsumme über einen Pseudo-Header, den UDP-Header und die Nutzdaten. Die Berechnung verwendet das *Einerkomplement*-Verfahren: Alle 16-Bit-Wörter werden addiert und das Einerkomplement der Summe bildet den Prüfsummenwert [14]. Bei IPv4 ist die Prüfsumme optional – wird sie nicht berechnet, wird das Feld auf null gesetzt [1]. Bei IPv6 hingegen ist die Prüfsumme verpflichtend, da IPv6 selbst keine Header-Prüfsumme besitzt [2, Sec. 8.1][4].

[3] Für IPv6-Jumbogramme wird das `Length`-Feld auf null gesetzt; die tatsächliche Länge ergibt sich aus dem Jumbo Payload Option Header [10].
[4] RFC 6936 definiert eine Ausnahme für bestimmte Tunnelprotokolle, bei denen ein Zero-Checksum-Modus erlaubt ist [9].

Pseudo-Header. Die UDP-Prüfsumme schützt nicht nur die Transportschichtdaten, sondern bezieht über einen *Pseudo-Header* auch Informationen der Vermittlungsschicht ein. Dieses Konzept stellt sicher, dass ein Datagramm, das durch einen IP-Fehler an den falschen Host oder das falsche Protokoll zugestellt wird, erkannt und verworfen werden kann. Für IPv4 definiert RFC 768 den folgenden Pseudo-Header [1]:



Bei IPv6 erweitert sich der Pseudo-Header auf die 128-Bit-Adressen. Anstelle des 8-Bit-Protokollfeldes verwendet IPv6 ein 8-Bit-`Next Header`-Feld (Wert 17 für UDP) sowie ein 32-Bit-Feld für die Paketlänge der Transportschicht. Die verbleibenden 24 Bit werden mit Nullen aufgefüllt [2, Sec. 8.1]:



2.1.3 Eigenschaften

Die in Abschnitt 2.1.2 beschriebene Header-Struktur verdeutlicht bereits das minimalistische Design von UDP. Aus diesem Designziel leiten sich eine Reihe funktionaler Eigenschaften ab, die maßgeblich beeinflussen, für welche Anwendungsfälle das Protokoll geeignet ist.

Nachrichtenorientierung. UDP ist ein *nachrichtenorientiertes* Protokoll: Jede Sendeoperation erzeugt genau ein Datagramm, und der Empfänger erhält jede Nachricht als atomare Einheit. RFC 768 beschreibt UDP als Verfahren, mit dem Programme Nachrichten an andere Programme senden können („a procedure to send [...] messages to other programs“) [1].

Zustandslosigkeit. UDP verwaltet keinen protokollspezifischen Zustand für einzelne Kommunikationsbeziehungen und hält keinerlei Datenstrukturen wie Sequenznummern, Fenstergrößen oder Retransmission-Timer vor [3, Sec. 6].

Minimale Latenz. Da UDP verbindungslos arbeitet, entfällt jeglicher Handshake vor der Datenübertragung. Die erste Nachricht kann unmittelbar gesendet werden, ohne dass ein vorheriger Verbindungsaufbau abgewartet werden muss [11, Sec. 1]. Ebenso existiert kein Verbindungsabbau, der zusätzliche Latenz verursachen könnte.

Unabhängigkeit der Datagramme. Jedes UDP-Datagramm ist eine eigenständige, von allen anderen Datagrammen unabhängige Einheit. Der Verlust eines Datagramms hat daher keine Auswirkung auf die Verarbeitung nachfolgender Datagramme. Diese Eigenschaft vermeidet das sogenannte *Head-of-Line-Blocking* (HoL-Blocking), bei dem verlorene Pakete die Auslieferung nachfolgender Daten verzögern. Diese Eigenschaft von UDP war eine der Motivationen für die Entwicklung von QUIC: QUIC nutzt UDP als Transportschicht und realisiert auf Anwendungsebene ein Stream-Multiplexing, bei dem der Verlust eines Pakets nur den betroffenen Stream blockiert, nicht jedoch andere parallele Streams [12, Sec. 1].

Multicast und Broadcast. UDP ist verbindungslos und wird häufig als Transport für IP-Multicast und (unter IPv4) IP-Broadcast verwendet. Ein UDP-Datagramm kann an eine Multicast-Gruppe oder eine Broadcast-Adresse gesendet und von mehreren Empfängern empfangen werden, sofern Netzwerk und Hosts entsprechend konfiguriert sind. Dabei gelten die üblichen Eigenschaften von UDP: keine Garantie für Zustellung, Reihenfolge oder Duplikaterkennung. Diese Kombination eignet sich besonders für lokale Mechanismen wie Service Discovery sowie für bestimmte Echtzeitanwendungen in kontrollierten Netzen.

Diese Eigenschaften machen UDP zu einem attraktiven Transportprotokoll für latenzempfindliche, verlusttolerante oder gruppenbasierte Anwendungen. Allerdings fehlt UDP jede Form der protokolleigenen Erweiterbarkeit. Genau diese Lücke adressiert RFC 9868 durch die Nutzung der *Surplus Area* (siehe Abschnitt 2.4.2), um Optionen zu transportieren, ohne den bestehenden UDP-Header zu verändern.

2.2 Internet Protocol (IP)

2.2.1 IPv4

2.2.2 IPv6

2.3 Middleboxes und Protokoll-Ossifikation

2.3.1 Internet-Ossifikation

2.3.2 Auswirkungen auf Protokollerweiterungen

2.4 RFC 9868: Transport Options for UDP

2.4.1 Überblick

2.4.2 Der Surplus Area

2.4.3 Options-Struktur

2.4.4 Definierte Options-Typen

2.4.5 SAFE und UNSAFE Options

2.5 Checksum Offloading

3. Analyse

3.1 Anforderungsanalyse

3.1.1 Funktionale Anforderungen

3.1.2 Nicht-funktionale Anforderungen

3.1.3 Speichereffizienz und Zero-Copy

3.2 Existierende Implementierungen

3.2.1 Linux Kernel

3.2.2 Userspace-Implementierungen

3.3 Herausforderungen

3.3.1 Zugriff auf den Surplus Area

3.3.2 NAT-Traversal

3.3.3 Interoperabilität

3.3.4 Alternative Implementierungsansätze

3.4 Technologieauswahl

3.4.1 Begründung für Zig

3.4.2 Architekturentscheidungen

4. Konzeption

4.1 Architekturübersicht

4.2 Datenstrukturen

4.2.1 IP-Header

4.2.2 UDP-Header

4.2.3 UDP Options

4.3 Options-Verarbeitung

4.3.1 Parsing-Algorithmus

4.3.2 Serialisierungs-Algorithmus

4.4 Checksum-Berechnung

4.4.1 Option Checksum (OCS)

4.5 Fragmentierung

4.5.1 FRAG Option-Konzept

4.5.2 Reassembly-Strategie

4.6 API-Design

4.6.1 High-Level API

4.6.2 Low-Level API

4.7 Fehlerbehandlung

4.8 State Management

4.8.1 Zustandsspeicherung für Fragmentierung

4.8.2 Timeout-Handling und Garbage Collection

4.9 Sicherheitsüberlegungen

4.9.1 Denial-of-Service-Schutz

4.9.2 Behandlung malformierter Options

5. Realisierung

5.1 Die Programmiersprache Zig

5.1.1 Überblick

5.1.2 Relevante Sprachfeatures

5.1.3 Raw Sockets in Zig

5.2 Projektstruktur

5.3 Raw Socket-Implementierung

5.3.1 Socket-Erstellung

5.3.2 Paketempfang

5.3.3 Plattform-Spezifika

5.4 Options-Parser

5.4.1 Hauptparser

5.4.2 Spezifische Parser

5.5 Options-Serialisierer

5.5.1 Serialisierung

5.6 OCS-Implementierung

5.6.1 Checksum-Berechnung

5.6.2 OCS-Validierung

5.7 Fragmentierung

5.7.1 Fragment-Cache

5.8 Build-Konfiguration

6. Evaluation

6.1 Testkonzept

6.2 Unit-Tests

6.2.1 Parser-Tests

6.2.2 OCS-Tests

6.2.3 Serialisierer-Tests

6.3 Integrationstests

6.3.1 End-to-End-Tests

6.3.2 Fragmentierungs-Tests

6.4 Performanz-Evaluation

6.4.1 Messaufbau

6.4.2 Parsing-Performanz

6.4.3 Vergleich mit Standard-UDP

6.4.4 Baseline-Definition

6.4.5 CPU-Overhead-Analyse

6.5 Interoperabilitätstests

6.5.1 Testpartner

6.5.2 Testergebnisse

6.6 Erfüllung der Anforderungen

7. Fazit

7.1 Zusammenfassung

7.2 Erkenntnisse

7.2.1 Technische Erkenntnisse

7.2.2 Methodische Erkenntnisse

7.3 Limitationen

7.4 Ausblick

7.4.1 Kurzfristige Erweiterungen

7.4.2 Langfristige Perspektiven

7.4.3 Forschungsfragen

7.5 Schlusswort

Literaturverzeichnis

[1] J. Postel. *User Datagram Protocol*. RFC 768. Internet Engineering Task Force, Aug. 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768.

[2] S. Deering und R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200. Internet Engineering Task Force, Juli 2017. DOI: 10.17487/RFC8200. URL: https://www.rfc-editor.org/info/rfc8200.

[3] J. Touch. *Transport Options for UDP*. RFC 9868. Internet Engineering Task Force, Okt. 2025. DOI: 10.17487/RFC9868. URL: https://www.rfc-editor.org/info/rfc9868.

[4] J. Touch. *Datagram Packetization Layer Path MTU Discovery for UDP Options*. RFC 9869. Internet Engineering Task Force, Okt. 2025. DOI: 10.17487/RFC9869. URL: https://www.rfc-editor.org/info/rfc9869.

[5] V. Cerf, Y. Dalal und C. Sunshine. *Specification of Internet Transmission Control Program*. RFC 675. Internet Engineering Task Force, Dez. 1974. DOI: 10.17487/RFC0675. URL: https://www.rfc-editor.org/info/rfc675.

[6] J. Postel. *Internet Protocol*. RFC 791. Internet Engineering Task Force, Sep. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791.

[7] J. Postel. *Transmission Control Protocol*. RFC 793. Internet Engineering Task Force, Sep. 1981. DOI: 10.17487/RFC0793. URL: https://www.rfc-editor.org/info/rfc793.

[8] R. Braden. *Requirements for Internet Hosts – Communication Layers*. RFC 1122. Internet Engineering Task Force, Okt. 1989. DOI: 10.17487/RFC1122. URL: https://www.rfc-editor.org/info/rfc1122.

[9] G. Fairhurst und M. Westerlund. *Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums*. RFC 6936. Internet Engineering Task Force, Apr. 2013. DOI: 10.17487/RFC6936. URL: https://www.rfc-editor.org/info/rfc6936.

[10] D. Borman, S. Deering und R. Hinden. *IPv6 Jumbograms*. RFC 2675. Internet Engineering Task Force, Aug. 1999. DOI: 10.17487/RFC2675. URL: https://www.rfc-editor.org/info/rfc2675.

[11] L. Eggert, G. Fairhurst und G. Shepherd. *UDP Usage Guidelines*. RFC 8085. Internet Engineering Task Force, März 2017. DOI: 10.17487/RFC8085. URL: https://www.rfc-editor.org/info/rfc8085.

[12] J. Iyengar und M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. Internet Engineering Task Force, Mai 2021. DOI: 10.17487/RFC9000. URL: https://www.rfc-editor.org/info/rfc9000.

[13] M. Bishop. *HTTP/3*. RFC 9114. Internet Engineering Task Force, Juni 2022. DOI: 10.17487/RFC9114. URL: https://www.rfc-editor.org/info/rfc9114.

[14] R. Braden, D. Borman und C. Partridge. *Computing the Internet Checksum*. RFC 1071. Internet Engineering Task Force, Sep. 1988. DOI: 10.17487/RFC1071. URL: https://www.rfc-editor.org/info/rfc1071.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt.

.....	.....
Ort, Datum	Unterschrift