FernUniversität in Hagen
Fakultät für Mathematik und Informatik
LG Kooperative Systeme

UDP Transport Options (RFC 9868)

Implementierung und Analyse 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 optionale Prüfsumme zur Fehlererkennung. 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. Dabei müssen Funktionen, die TCP auf Transportebene bereitstellt – wie Fragmentierung, Zeitstempel oder erweiterte Prüfsummen – in der Anwendungsschicht reimplementiert werden.

RFC 9868, veröffentlicht im Oktober 2025, schließt diese Lücke durch die Einführung von *Transport Options for UDP* [2]. 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

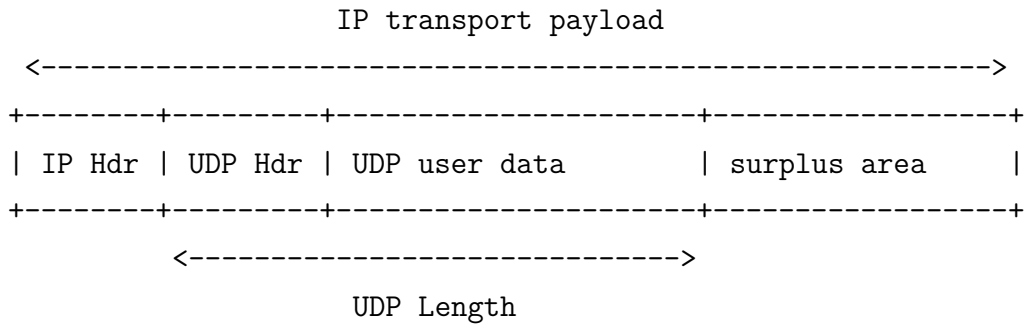
Ein entscheidender Vorteil der UDP-Options ist ihre **Abwärtskompatibilität**: Legacy-Systeme, die UDP Options nicht unterstützen, ignorieren die Surplus Area und verarbeiten lediglich die regulären UDP-Nutzdaten. Dies ermöglicht eine schrittweise Einführung der Erweiterung ohne Beeinträchtigung bestehender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

1.2 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."* [2, 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 [2, 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 [2, 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 [3], 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 **Gemini** (Google) 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)

2.1.1 Historische Entwicklung

2.1.2 Protokollstruktur

2.1.3 Eigenschaften

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

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] 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.

[3] 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.

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