

Alternative Transport Layer Protokolle

Christian Mehlis

Zusammenfassung

In IP-basierten Netzen sind TCP und UDP im Vergleich zu deren Alternativen sehr dominant. In dieser Arbeit beschäftige ich mich mit den wichtigsten Alternativen zu TCP und UDP, die Vor- und Nachteile dieser und deren Probleme in realen Netzen. Einführend werden die Grundlagen der geschichteten Struktur des Internets erklärt und dann auf die einzelnen Protokolle näher eingegangen. Im Fazit folgt die Auseinandersetzung mit den Problemen von SCTP, DCCP und UDPLite.

I. EINORDNUNG

Die Transportschicht des TCP/IP-Referenzmodells dient primär der Adressierung von Prozessen auf einem Host. Die Transportschicht befindet sich zwischen der Internetschicht, welche mit Hilfe von IP-Adressen für das Routing und der Anwendungsschicht, welche für den Austausch anwendungsspezifischer Daten, zuständig ist. Im ISO/OSI-Modell existiert die Transportschicht mit der gleichen Funktionalität; die beiden Modelle sind hier äquivalent.

Die Adressierung erfolgt in der Transportschicht mit Hilfe von Ports. Dies ist ein numerischer Wert zwischen 0 und $2^{16} - 1$. Ports werden von der Internet Assigned Numbers Authority (IANA) zentral und weltweit eindeutig für alle Transportprotokolle vergeben und sind nicht an ein einzelnes Schicht-vier-Protokoll gebunden. Jedoch ist für die Registrierung ein Antrag bei der IANA notwendig. Die meisten Registrierungen sind für TCP und UDP, für SCTP sind z. B. HTTP oder SSH registriert.

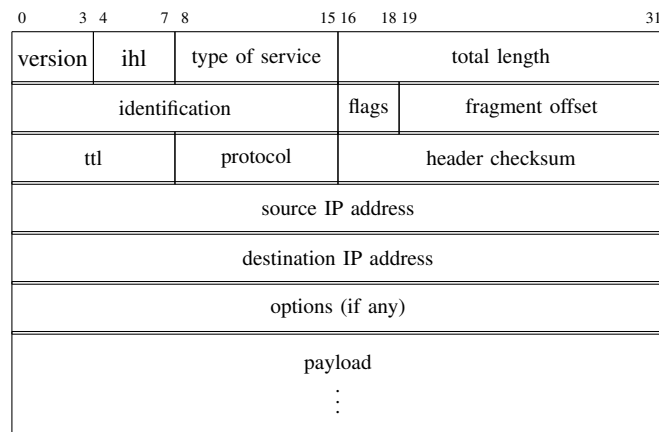


Abbildung 1. IP-Header

A. Transmission Control Protocol

Das wohl bekannteste und auch mit weitem Abstand verbreitetste Transportschichtprotokoll ist das Transmission Control Protocol[15]. TCP garantiert eine „Ende-zu-Ende“-Semantik, d. h., es wird ein virtueller Tunnel zwischen den beiden beteiligten Hosts aufgebaut, der alle Daten, die der Absender verschickt, garantiert ankommen lässt.

Ein TCP-Segment bezeichnet ein Paket, welches einen definierten Bereich des Stroms transportiert. TCP garantiert nicht nur, dass keine Segmente der Übertragung verloren gehen, sondern auch, dass keine Segmente mehrfach oder zwei Segmente in der falschen Reihenfolge an die Anwendungsschicht zugestellt werden. Natürlich können Pakete bei der Übertragung verloren gehen; diese werden dann jedoch von TCP erneut übertragen. TCP folgt somit einer strikten „exactly-once-Semantik“ [19]. TCP verwendet zum Verbindungsaufbau einen 3-Wege-Handshake, welcher beiden Hosts garantiert, dass der jeweils andere nun für eine Kommunikation bereit ist. Außerdem werden über den Handshake Informationen wie aktuelle Sequenznummern und Buffergrößen ausgetauscht.

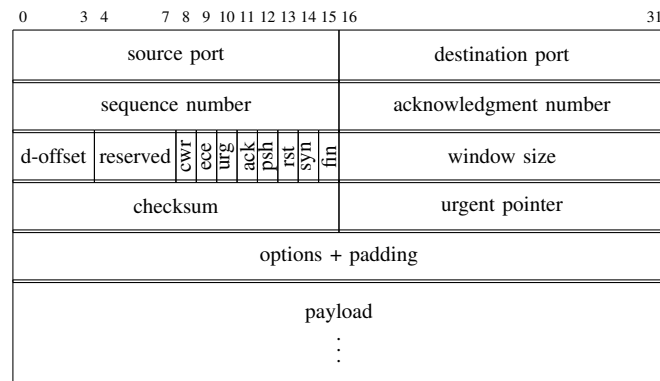


Abbildung 2. TCP-Header

Um zu garantieren, dass alle Pakete auch wirklich beim Zielhost ankommen, spezifiziert TCP eine Reihe von Methoden zur Bestätigung von Paketen, sowie eine Stau- und Flusskontrolle. Die Staukontrolle[15] (engl. congestion control) ist ein Mechanismus, um auf eine Überlast in einem Netzwerkteil geeignet zu reagieren. Die Flusskontrolle[1] regelt die Senderate in Bezug auf den Buffer des Empfängers. Sobald der Empfänger einen Verlust von Paketen feststellt, teilt er dies dem Sender mit und dieser reagiert, indem er fortan die Anzahl der Pakete, die er pro Zeiteinheit sendet, reduziert. In TCP existieren verschiedene Implementierungen der Staukontrolle, die alle untereinander kompatibel sind.

Zur Flusskontrolle wird ein Verfahren namens „sliding window“ benutzt. Je nach Endgerät besitzt der TCP-Stack unterschiedlich viel Speicher, um Pakete zu buffern, deren Vorgänger verloren gegangen sind. Das Buffern ist notwendig, da die Pakete in der richtigen Reihenfolge an die Anwendungsschicht übergeben werden müssen. Wenn nun das neu übertragene Paket zu lange braucht, kann der Buffer beim Empfänger vollaufen und keine neuen Pakete mehr aufnehmen.

Der Empfänger bestätigt immer das letzte in Reihenfolge erhaltene Byte. Bei einem Paketverlust bemerkt nun der Sender über ein Timeout den Verlust und sendet den Strom ab dem nächsten Byte erneut.

Über die „Fast-Retransmit“ und „Fast Recovery“ Funktionen bemerkt der Sender den Verlust eines Pakets über die duplizierten Bestätigungen.

Bei der Verwendung der Option „Selective Acknowledgements“ [12] kann der Empfänger dem Sender den Verlust explizit mitteilen, damit dieser möglichst schnell reagieren kann.

B. User Datagram Protocol

Das User Datagram Protocol[14] bietet im Vergleich zu TCP keine Garantien in Bezug auf tatsächliches Ankommen von Paketen und bietet keine Möglichkeiten der Stau- und Flusskontrolle. UDP fügt dem IP-Header lediglich

Absender- und Empfängerports, sowie ein Längenfeld und eine Prüfsumme hinzu.

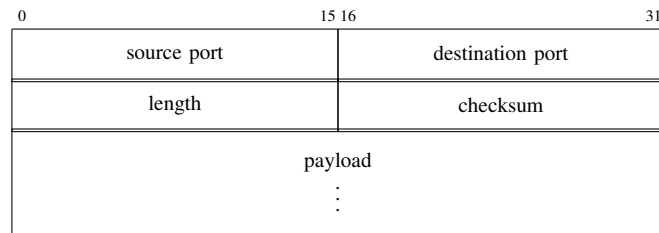


Abbildung 3. UDP-Header

Die Prüfsumme bezieht sich auf das gesamte Paket mit IP-Pseudo-Header, UDP-Header und Nutzlast des UDP-Pakets. Sie ist in IPv4 optional und in IPv6 verpflichtend, da IPv6 keine eigene Prüfsumme besitzt. Soll keine Prüfung der Integrität beim Empfänger stattfinden, wird das Prüfsummenfeld auf null gesetzt.

II. NACHTEILE VON TCP UND UDP

Welches Transportprotokoll nun in einer Anwendung benutzt wird, hängt primär von den Anforderungen ab, die ein Programm an die darunter liegenden Schichten stellt. Anwendungen, die Nutzdaten fehlerfrei übertragen wollen, wählen traditionell TCP als Transportprotokoll. Anders sieht es bei Liveübertragungen aus. Diese verlangen andere Garantien von der Transportschicht. So ist es bei einer Medienübertragung wichtiger, dass keine Zeitverzögerung entsteht und die Inhalte in Echtgeschwindigkeit ausgegeben werden können. Das Verlieren eines einzelnen Pakets, wird zugunsten der Zeitverzögerung, die bei einem Neusenden des Pakets entstehen würde, meist bevorzugt. Auch das Kippen einzelner Bits in den Mediendaten kann durch einen fehlertoleranten Video- und/oder Audiodekoder gut verarbeitet werden. So sind Fehler in den Mediendaten meist dem Verlust des gesamten Pakets, wegen eines Bitfehlers, vor zu ziehen. Auch gibt es bei Mediendaten häufig eine Priorisierung bestimmter Frames, d. h., es gibt Teile des Medienstromes, die eine höhere Priorität haben als andere. Dies können z. B. vollständige Frames eines MPEG-Datenstromes sein (I-Frames). Jedoch ist trotzdem häufig eine Staukontrolle auf Transportschicht erwünscht, da es beim Verbindungsaufbau nicht absehbar ist, ob sich Verbindungseigenschaften, wie Durchsatz oder Fehlerrate, verschlechtern.

III. ALTERNATIVE TRANSPORTPROTOKOLLE

IPv4 wie auch IPv6 besitzen ein Protokoll-Feld in ihren Headern, welches das verwendete Schicht-vier-Protokoll anzeigt. Dieses ist genau acht Bits breit und begrenzt die Zahl der möglichen Transportprotokolle auf maximal 256. Eine wichtige Eigenschaft für neue Transportprotokolle ist der faire Umgang mit TCP (TCP-Fairness). TCP-Fairness bedeutet, dass ein Protokoll eine Methode zur Staukontrolle (congestion control) beinhaltet, welches TCP-Verbindungen nicht verdrängt[23].

Im Folgenden werden die wichtigsten Alternativen zu TCP und UDP vorgestellt, die jeweils auf die Vielfalt von Nutzdaten optimiert sind.

A. Stream Control Transmission Protocol

Das Stream Control Transmission Protocol [20][13][16][21][6] versucht als Transportprotokoll die wichtigsten Features aus den Transportprotokollen TCP und UDP wiederzuverwenden und weitere Eigenschaften und Funktionen in einem Protokoll zu vereinen. SCTP ist verbindungsorientiert, mit den drei Zuständen Setup, Kommunikation

und Verbindungsabbau und kann wie TCP einen zuverlässigen Transport garantieren (reliability). Außerdem ist es wie bei TCP möglich eine Verbindung mit Stau- und Flusskontrolle zu erzeugen. Die Flusskontrolle ist jedoch in SCTP optional. SCTP kann aber auch benutzt werden, um einen UDP-artigen Transport zu leisten, der keinerlei Garantien über Reihenfolge von Paketen macht und selbst das Ankommen von Paketen nicht garantiert.

Zusätzlich bietet SCTP mehr Protokolleigenschaften, die den Transport in Szenarien wie mobile Internetnutzung oder maximale Zuverlässigkeit und damit verbundenes Multihoming ermöglichen. Multihoming bedeutet, dass ein Host über mehr als eine Adresse erreichbar ist, z. B. wenn ein Host an mehreren Providernetzen angeschlossen ist, um die Redundanz und somit die Ausfallsicherheit zu erhöhen.

So ermöglicht SCTP eine Teilung von Daten über mehrere Streams, die in einer SCTP Verbindung existieren. In einer einzigen SCTP-Verbindung können bis zu 256 Ströme existieren, die jeweils eine eigene Stau- und Flusskontrolle besitzen.

Ein über zwei Streams verteilter Videostream mit Bild und Ton kann so dem Ton eine andere Priorität als dem Bild geben und so ein Blockieren einer Datenart durch eine andere verhindert werden. Ein Stream ist immer nur eine unidirektionale Verbindung. Wenn eine bidirektionale Verbindung erwünscht ist, muss dies über zwei unidirektionale Verbindungen geschehen, welche in jede Richtung wieder unterschiedliche Verbindungsparameter besitzen können. Diese logische Trennung der Richtungen wurde jedoch zur API abstrahiert und ist dem Programmierer nicht mehr ersichtlich.

Eine TCP-Verbindung wird durch ein Paar aus IP-Adressen und Ports identifiziert, was eine redundante Kommunikation über mehrere Netzwerkschnittstellen nicht ermöglicht. Eine SCTP-Verbindung besteht zwischen zwei Hosts, die jeweils eine Menge von Adressen besitzen. So kann die Verbindung ohne Probleme auch beim Ausfall eines ganzen Uplinks bestehen bleiben, da die Pakete dann über die andere Verbindung geleitet werden können.

SCTP bietet auch Verbesserung in der Sicherheit gegen SYN-Flooding-Attacken[2]. Hierzu verwendet SCTP einen 4-Wege-Handshake. Die Pakete für einen Verbindungsaufbau sind INIT, INIT-ACK, COOKIE-ECHO, COOKIE-ACK. Solange die Verbindung nicht vollständig aufgebaut ist, ist die Serverseite zustandslos, d. h., ein Angreifer kann durch vorgetäuschte Verbindungen keine Ressourcen auf dem Server binden. Auf jedes INIT sendet der Server immer ein INIT-ACK, öffnet jedoch keine halboffene Verbindung. Erst nach einem COOKIE-ECHO besteht für den Server eine Verbindung zu einem Client.

Um die Flexibilität gegenüber TCP weiter zu steigern, ist es in SCTP dem Programmierer möglich, Eigenschaften von SCTP Verbindungen zu manipulieren, die sich in TCP nicht zur Laufzeit des Kernels realisieren lassen. So kann der Programmierer für jedes Socket Verbindungsparameter wie Timeout-Schranken oder Details zum Congestion Control setzen.

Über das Prinzip der Ströme lassen sich in einer SCTP-Verbindung Teilverbindungen mit unterschiedlichen Verbindungsparametern herstellen. So kann ein Strom in einer SCTP-Verbindung zuverlässigen Transport garantieren und ein anderer, in der selben SCTP-Verbindung garantiert dies nicht. Congestion Control leistet SCTP jedoch immer.

SCTP bietet jedoch keine byteorientierte Übertragung wie TCP. Nach jedem write() auf einem Socket wird eine Marke eingefügt, die dem Empfänger das Ende einer Nachricht zeigt. SCTP ist somit wie auch UDP nachrichtenorientiert. Eine Verbindung ohne Staukontrolle über SCTP zu betreiben, ist, genauso wie das Versenden von Nachrichten, ohne vorher eine Verbindung erfolgreich aufgebaut zu haben, nicht möglich.

Um eine Anwendung von TCP auf SCTP zu portieren genügt es, im Programm ein SCTP Socket zu benutzen.

Natürlich werden so die meisten Features aus SCTP, die man noch explizit einstellen muss, nicht verwendet.

Die IP-Pakete tragen bei SCTP im Protokollfeld die Nummer 132 für SCTP statt 6 für TCP. Zum Verbindungsaufbau wird nun ein 4-Wege-Handshake benutzt. Die beiden Hosts der Verbindung tauschen Informationen zu ihren Interfaces aus, um ein Multihoming zu erkennen und dies auch zu nutzen. Jeder Aufruf von read() auf dem SCTP-Socket an einem Ende der Verbindung gehört genau zu einem Aufruf von write() an dem anderen Ende. Hier lässt sich die Nachrichtenorientierung von SCTP erkennen.

Natürlich lassen sich so nicht alle Vorteile von SCTP nutzen. Die Multi-Homing Unterstützung wird jedoch so direkt aktiviert und wird ohne weitere Änderungen im Programmcode genutzt. Auch der Sicherheitsgewinn durch den 4-Wege-Handshake wird so direkt aktiviert.

Wenn auf einem SCTP Socket ein Event auftritt, kann dies der Programmierer abfangen und darauf geeignet reagieren. So gibt es z. B. ein SCTP_PEER_ADDR_CHANGE-Event, welches eine Veränderung an den möglichen Adressen des Kommunikationspartners anzeigt. Über dieses Event kann ein Host seinem Kommunikationspartner mitteilen, dass nun eine weitere IP-Adresse zur Kommunikation zur Verfügung steht (Multi-Homing). Um auch bei einer nicht benutzten Verbindung festzustellen, ob diese noch aktiv, d. h., der Kommunikationspartner noch antwortet, wird ein HEARTBEAT Chunk gesendet und mit einem HEARTBEAT-ACK bestätigt.

Jedes SCTP-Paket verfügt, wie auch IPv6-Pakete, über einen kleinen Protokollheader und wird dann für jeden Stream mit einem weiteren Header ausgestattet.

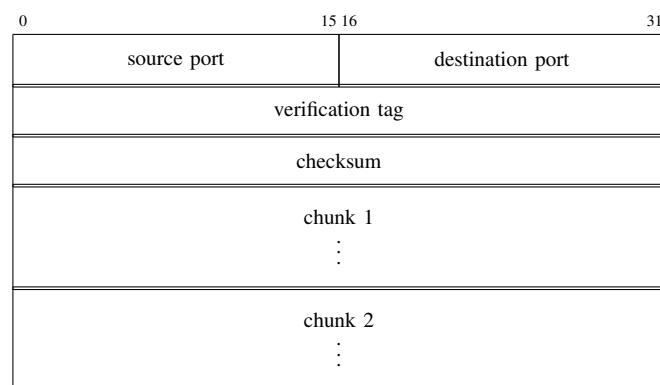


Abbildung 4. SCTP-Header

Die 2-Byte-breiten Ports behalten die Semantik aller Transportprotokolle bei. Der 4-Byte-verification-Tag ist ein Zufallswert, der eine SCTP-Verbindung eindeutig identifiziert. Er dient außerdem dazu, Attacken auf den Paketfluss zu unterbinden, bei denen dem Angreifer nur die IPs und Ports bekannt sind. Die 4-Byte-breite CRC-Prüfsumme[22] deckt das komplette SCTP-Paket mit allen Headern und der Nutzlast ab.

Der Header eines jeden SCTP Stroms beinhaltet den Typ des Streams (wie z. B. DATA, INIT oder HEARTBEAT), gefolgt von Flags, die je nach Strom Typ eine andere Semantik besitzen, und die Länge der Nutzlast dieses Teilpaketes. Gefolgt wird dieser Header von weiteren, für den spezifischen Stream benötigte Header-Feldern.

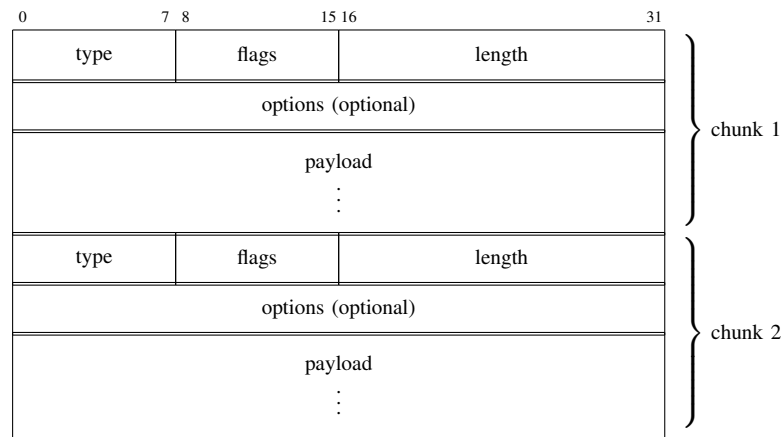


Abbildung 5. SCTP-Stream (Chunk) Header

B. Datagram Congestion Control Protocol

Datagram Congestion Control Protocol[8][9][3] (DCCP) ist ein Transportprotokoll, welches auf der Paketsemantik von UDP aufbaut und dieser zusätzliche Eigenschaften wie z. B. Staukontrolle verleiht.

Bei dem Transport von Medieninhalten wird meist TCP oder UDP benutzt, da diese beiden die größte Verbreitung besitzen und so eine nahezu hundertprozentige Verfügbarkeit auf allen Hosts garantieren. TCP und UDP besitzen jedoch Eigenschaften, die einen optimalen Transport von Livedaten nicht ermöglichen. TCP benutzt Retransmissions um verlorene Pakete erneut zuzustellen. Dies verzögert jedoch die Zustellung. UDP bietet der Anwendungsschicht keine Features wie Verbindungsaufbau oder Staukontrolle, was typische Aufgaben der Transportschicht sind. Wenn nun UDP in einer Anwendung benutzt werden soll, muss die Anwendung diese Anforderungen implementieren. Auch ist es erwünscht, TCP gegenüber fair zu agieren, was jedoch UDP nicht leistet.

DCCP fügt nun UDP einige Schlüsseleigenschaften hinzu, die den TCP-Traffic im Netzwerk nicht stören, dem Benutzer einen an seine Bandbreite angepassten Strom liefert und nicht vom Programmierer der Anwendung programmiert werden muss.

So liefern moderne Audiocodecs, die bei VoIP eingesetzt werden, eine variable Bitrate, um die Qualität konstant zu halten. Gerade bei VoIP spielt die Verzögerung, die durch ein neues Anfordern eines verlorenen Pakets entsteht, eine entscheidende Rolle. Da die Verzögerung bei Echtzeitanwendungen minimiert werden muss, kommt ein nochmals angefordertes Paket häufig zu spät an und kann nur noch verworfen werden. Die Neuansforderung war somit sinnlos.

DCCP wurde unter Berücksichtigung von verschiedenen Zielen entwickelt.

1) *Minimalismus*: Im Datagram Congestion Control Protokoll wurde darauf geachtet, nur solche Dinge zu spezifizieren, die auch von hohem Interesse der Programmierer sind. Alle Dinge, die nicht zu den direkten Eigenschaften gehören, sollen in höheren Schichten realisiert werden.

Da gerade VoIP bei der Protokollspezifikation als mögliches Hauptanwendungsgebiet von DCCP angesehen wurde, haben die Entwickler gerade auf die Anforderung von VoIP hin optimiert. So ist der Header möglichst klein gehalten, um den Overhead vieler kleiner VoIP Pakete zu minimieren. Außerdem wurde auf eine Headerkomprimierung verzichtet, um die Rechenintensivität zu minimieren. Weniger Overhead reduziert auch die Transportkosten auf Seiten des Providers, der z. B. ein Mobilfunk-Netz betreibt und dort nur begrenzte Netzkapazitäten besitzt.

- 2) *Robustheit*: DCCP soll ohne die Verwendung von rechenintensiver Verschlüsselung grundlegenden Schutz gegen typische Angriffe liefern. So wird, wie auch in TCP und SCTP, mit einer zufällig gewählten Sequenznummer verhindert, dass ein Angreifer Daten in die Verbindung injiziert, oder diese von außen beenden kann. Außerdem wurde bei der Protokollspezifikation darauf geachtet, aktuelle Probleme in Bezug auf typische Heimnetzwerk-Soft- und Hardware, so wie schlecht konfigurierte Firewalls und NAT zu berücksichtigen.
- 3) *Erweiterbarkeit*: DCCP bringt nicht nur einen Congestion-Control-Algorithmus mit, sondern stellt ein Framework zur Integration verschiedener Congestion-Control-Mechanismen bereit. So existieren TCP-artige bzw. TCP-friendly Congestion-Control-Implementierungen, sowie auf besondere äußere Umstände angepasste, wie auf geringe Bandbreite (TFRC[5]) oder extrem hohe Bandbreite (XCP[7]). Durch diese Erweiterbarkeit ist es z. B. möglich, den vermuteten Grund eines Paketverlustes mitzuteilen und so zielgerichteter darauf zu reagieren.

Zur Steigerung der Portabilität wird bei DCCP eine bidirektionale Verbindung als zwei unidirektionale Verbindungen angesehen, die jeweils andere Verbindungsparameter, wie z. B. einen anderen Congestion-Control-Algorithmus benutzen können.

- 4) *Effizienz*: DCCP stellt der Anwendungsschicht eine UDP-artige API zur Verfügung, jedoch sollen die Basis-Features von DCCP, wie bei TCP, ohne Mitwirken der Anwendung funktionieren.
- 5) *Unterstützung für zeitkritischen Transport*: Auch wenn DCCP seine Funktionen wie automatisches Congestion Control vor der Anwendungsschicht verstecken kann, ist es auch möglich, aus der Anwendungsschicht heraus diese zu steuern, um eine maximale Portabilität zu erhalten.

Um das Datagram Congestion Control Protokoll nicht zu komplex zu gestalten, und so Implementierung und Nutzung kompliziert zu machen, wurden auch Design-Entscheidungen getroffen, welche zu einem Ausschluss von Features führten.

- 1) *Flow-Control*: Da DCCP nicht den Anspruch hat zuverlässig (reliable) zu sein, ist es ein Problem des Empfängers, wenn dort Pakete wegen eines übergelaufenen Buffers verworfen werden. Gerade in Medienströmen ist hier jedoch das neue Paket einem alten vorzuziehen, und so ist das Fehlen dieses Features bei dem Transport von Mediendaten nicht problematisch.
- 2) *Bestimmbare Zuverlässigkeit*: Auch wenn DCCP keinen generellen zuverlässigen Transport garantiert, wäre es möglich einzelne Pakete, welche die Anwendungsschicht als solche markiert, zuverlässig zu übertragen. Diese Funktionalität würde jedoch das Vorhalten dieser Pakete für den Fall des erneuten Sendens erforderlich machen, was wiederum wesentlich mehr Aufwand für die Implementierung von DCCP bedeutet hätte.
- 3) *Streams*: In DCCP wurden Streams nicht eingebaut, da sich dieses Feature sehr leicht auf der nächsten Protokollschicht über DCCP implementieren lässt.
- 4) *Multicast*: Da DCCP verbindungsorientiert spezifiziert wurde und auch eine Kommunikation in beide Richtungen für das Congestion Control stattfinden muss, schließt sich eine Nutzung von DCCP für Multicast aus.

Das DCC-Protokoll garantiert eine Unicast-Verbindung, welche verbindungsorientiert arbeitet und einen bidirektionalen Datenfluss ermöglicht. Diese bidirektionale Verbindung ist jedoch in zwei unidirektionale Verbindungen aufgeteilt, welche jeweils über andere Verbindungsparameter verfügen können. So ist es möglich, zwei verschiedene Congestion-Control-Algorithmen für eine bidirektionale Verbindung zu verwenden, um so die spezifischen Eigenschaften, die diese Verbindung besitzt, besser im Netz abzubilden. Verbindungen starten und enden mit einem

3-Wege-Handshake wie in TCP.

DCCP-Pakete haben einen 16-Byte-großen Header, der jedoch durch Optionen erweitert werden kann.

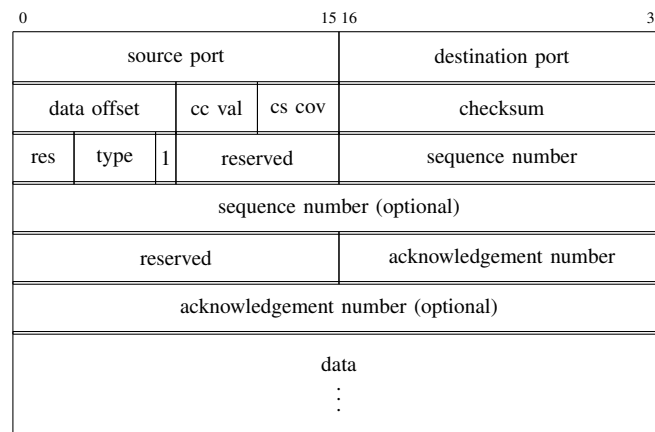


Abbildung 6. DCCP-Header

Die Portfelder gleichen denen in TCP und UDP. Der Datenoffset gibt an, ab welcher Stelle Nutzdaten im Paket kommen, bzw. wie viele Bytes zum Header gehören, da sich dieser durch zusätzliche Optionen erweitern lässt. Das Typenfeld gibt an, um welchen Typ eines DCCP Pakets es sich handelt, wie z. B. Data, oder DataAck. Die Sequenznummer ist die fortlaufende Nummer des Pakets, die bei jedem Paket (auch solchen ohne Nutzdaten) inkrementiert wird. Die 48 Bits sollten mit 2^{48} möglichen Zuständen auch für lange Übertragungen ausreichen. Bei zwanzig Paketen die Sekunde für über 400.000 Jahre.

Bei Anwendungen, die keine 48-Bit-breiten Paketnummern benötigen, kann DCCP auch 24-Bit-breite IDs verwenden, so dass man pro Paket acht Byte an Overhead einsparen. Ein größerer Raum für Sequenznummer erhöht die Sicherheit gegen Injection-Angriffe[9], da mit 48 Bits ein viel größerer Zustandsraum erzeugt wird als mit 32 Bit in TCP. Ziel der Protokolldesigner war es, DCCP minimal die Sicherheit von TCP zu verleihen, was sie selbst als eher einfach ansahen.

Die Acknowledgement-Nummer bestätigt das Eintreffen von Paketen am Host, jedoch ist nicht garantiert, dass durch eine Überlast beim Empfänger das Paket doch noch verworfen werden muss, da DCCP keine Staukontrolle besitzt. Wie in UDPLite wird in DCCP eine variable Checksummenbreite benutzt (CsCoverage). Diese gibt wie in UDPLite den Bereich an, der durch eine Prüfsumme geschützt werden soll. Dieses Feld ist 4-Bit-breit und gibt die Prüfsummenbreite in 32-Bit-Worten an.

C. Lightweight User Datagram Protocol (UDPLite)

UDPLite[11][10] ist ein minimalistisches Transportprotokoll, welches in Anlehnung an UDP dem IP-Paket nur zwei Ports und eine partielle Prüfsumme hinzufügt. Die partielle Prüfsumme stellt im Unterschied zu UDP das Kern-Feature von UDPLite dar, denn häufig werden bei Medienübertragungen einzelne Bit-Fehlern dem Verlust des ganzen Paketes vorgezogen. Ein Paket, welches Medieninhalte transportiert, wird meist in einen kritischen und in einen unkritischen Teil aufgeteilt. In dem kritischen Teil des Pakets liegen die UDPLite-Header und Metainformationen zu dem Strom selbst, z. B. der Index des Pakets. In dem unkritischen Teil liegen die Video oder Audiodaten.

Am Anfang des UDPLite-Pakets befinden sich die obligatorischen 16-Bit-breiten Portnummern des Absenders und des Ziels. Diese werden von einem 8-Bit-breitem Offset-indikator und der 8-Bit-breiten Checksum gefolgt. Im

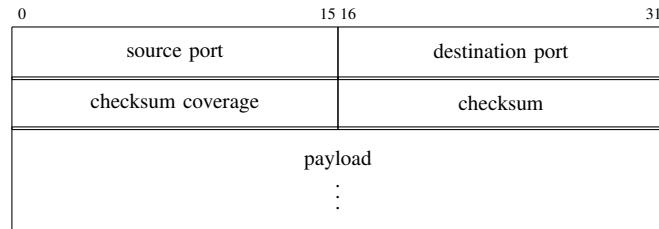


Abbildung 7. UDPLite-Header

Unterschied zu UDP liefert ein UDPLite-Paket seine Größe nicht selbst in einem eigenen Headerfeld mit. Jedoch war diese Information bereits in UDP redundant, wenn man bedenkt, dass der IP-Header bereits ein Längenfeld enthält und die Längeninformation von UDP nur die IP-Länge abzüglich der IP-Header-Breite ist.

Das in UDP für die Länge genutzte Feld wird in UDPLite benutzt, um dem Empfänger mitzuteilen, worauf sich die nachfolgende Checksum bezieht. Die Besonderheit im Unterschied zu UDP ist, dass der Sender auf Anwendungsschicht entscheidet, über welche Daten die Prüfsumme gebildet werden soll. Wenn z. B. in den ersten acht Byte der Nutzdaten im UDPLite-Paket Metainformationen eines Medienstreams sind, ist es von großer Wichtigkeit, dass diese auch unbeschadet vom Empfänger gelesen werden können. Die weiteren Nutzerdaten, z. B. die Mediendaten selbst, sollen auch bei einzelnen Bitfehlern an die Anwendungsschicht weitergereicht werden, um so möglichst viele Informationen, nutzen zu können. Tatsächlich werden Bitfehler in dem nicht überwachten Paketbereichen nicht von der Trasportschicht erkannt. Diese müssen von der Anwendungsschicht erkannt und behandelt werden. Bei dem typischem Anwendungsfall von Echtzeit-Mediendaten, ist es meist unerheblich, wenn einzelne Bits im Medienstrom fehlerhaft sind. Diese machen sich nur durch minimale Bildstörungen bemerkbar. Jedoch wären viele, aufgrund von Fehlern in den Mediendaten, verworfene Pakete für den Betrachter eines Filmes durch Bildaussetzer sichtbar und als nachteilig zu betrachten. Die Prüfsumme selbst ist ein Einerkomplement der Daten, die am Headeranfang startet und am gegebenen Offset endet.

Die UDPLite-Entwickler empfehlen darüber hinaus, die Prüfsummen auf den anderen, unterhalb von der Transportschicht liegenden Schichten soweit wie möglich zu ignorieren, da sonst der Vorteil der partiellen Prüfsummen durch unterliegende Protokolle aufgehoben werden. Wenn z. B. ein Bitfehler in den Nutzdaten auftritt und UDPLite als Transportprotokoll benutzt wird, ignoriert die Transportschicht diesen Fehler und leitet die Daten an die Anwendungsschicht weiter. Wenn jedoch schon die Bitsicherungsschicht Fehler erkennt, dann wird der Frame dort sofort verworfen und erreicht die Transportschicht erst gar nicht.

IV. BEWERTUNG

Ein anderer Weg, die Anforderung moderner und zukünftiger Netzarchitekturen in der Transportschicht zu realisieren, ist das bestehende TCP anzupassen[17][18][4]. Das zweite große Anwendungsfeld für alternative Transportprotokolle ist, neben dem Live-Medientransport, der Transport über ein kabelloses Netzwerk. Hier sind die Umstände warum es zu Paketverlusten kommt andere als in Kabelbebandenen Netzwerken. In Ethernet-Netzwerken geschieht ein Paketverlust meist durch eine Überlast in einem Switch oder Router. In einem WLAN ist der letzte Hop der größte Produzent von verlorenen Paketen. Jedoch soll der Transport unabhängig von der Bitsicherungsschicht sein und über alle Schicht-zwei-Technologien funktionieren. Da TCP jedoch bei einem Paketverlust eine Überlast annimmt und die Senderate reduziert, kommt es zu einer geringeren Auslastung des Netzwerks und damit verbundenen

zu einer geringeren Datenrate. Um dem Problem dieser Fehleinschätzung von TCP zu begegnen, gibt es Ansätze TCP zu modifizieren, jedoch die Kompatibilität mit dem bestehenden TCP-Implementierungen beizubehalten. Diese Überlegungen werden unter dem Begriff „Mobile-TCP“ zusammengefasst.

Die Anpassungen können entweder im Netzwerkstack des mobilen Hosts stattfinden oder in einer Komponente, die zwischen dem mobilen und dem stationären Host liegt. Ersteres versucht das Verhalten von TCP auf die Bedingungen eines mobilen Hosts anzupassen ohne den TCP Standard zu verletzen oder zu erweitern.

Die andere Möglichkeit, die Transportschicht an einen mobilen Host anzupassen, ist, einen Proxy im Netzwerk zwischen den beiden Kommunikationspartner zu benutzen. Dieser ist typischerweise sehr nahe am mobilen Host, meist an der Schnittstelle zwischen kabelgebundenem und kabellosem Netz. Dies erfordert keine Änderungen an Server- oder Clientsoftware. Der Proxy agiert als Vermittler zwischen den an der TCP-Verbindung beteiligten Hosts und versucht mit verschiedenen Techniken die TCP-Verbindung zu optimieren. So werden Pakete, die zum mobilen Host gehen, immer im Proxy gespeichert, bis der mobile Host den Empfang dieser bestätigt hat. Wenn nun der mobile Host ein Paket vom Server nochmals anfordert, dann reicht der Proxy diese Anfrage nicht weiter, sondern antwortet selbst mit dem zuvor gespeicherten Paket. Mit diesem Buffern kann die Paketlaufzeit sehr weit reduziert werden. Außerdem werden Kontrollpakete die vom mobilen Host stammen und einen Stau im Netzwerk anzeigen – dies vermutet der mobile Host, da er Pakete verliert – verworfen und so dem Server nie mitgeteilt. Dieser wird in Folge dessen auch seine Sendeleistung nicht reduzieren und die Datengeschwindigkeit beibehalten.

A. Evaluierung der alternativen Transportprotokolle

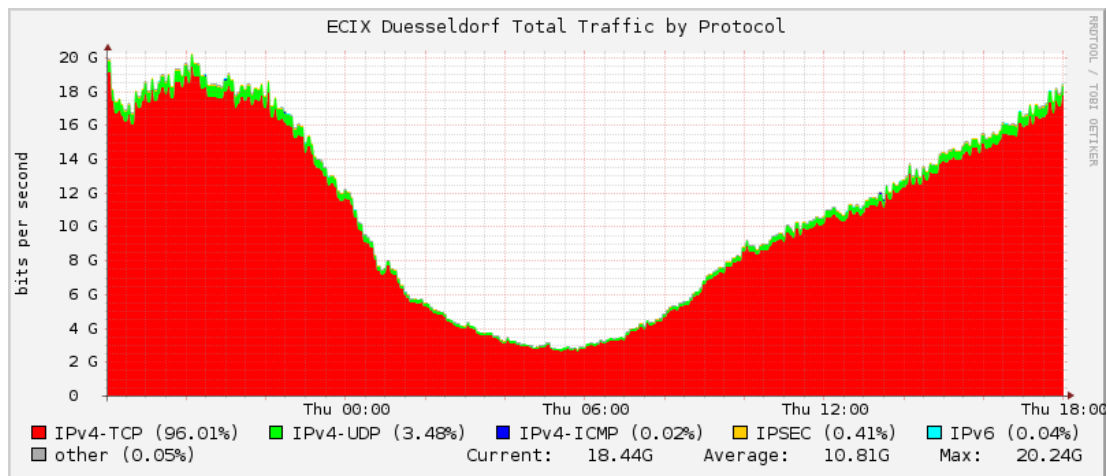


Abbildung 8. Anteil der Transportprotokolle am Datenaufkommen im Internet — Die Grafik zeigt anhand eines Internetknotens, beispielhaft wie sich der Traffic im Internet zusammensetzt. Die Daten wurden am ecix über einen Tag gesammelt und repräsentieren den Traffic von 59 autonomen Systemen (AS), darunter Telefonica, Vodafone, Akamai, Google, Hansenet, Netkologne und Cogent.

Die Gründe, warum sich DCCP und die anderen Alternativen zu TCP und UDP (noch) nicht durchgesetzt haben, sind in sehr unterschiedlichen Bereichen zu finden.

Für oder gegen den Erfolg eines Standards spielen viele Faktoren eine Rolle. Essentiell ist die Unterstützung der Transportprotokolle im Betriebssystem der Anwender. Hier wird dem Programmierer erst ermöglicht, ein Socket zu erstellen und so ein bestimmtes Transportprotokoll zu nutzen. Damit sich SCTP und DCCP wirklich im Endverbrauchermarkt durchsetzen zu können, bedarf es jedoch Veränderung an vielen Home-Routern. Diese

sind meist nur mit grundlegenden Funktionalitäten zum Routing von Paketen ausgestattet und implementieren nicht alle denkbaren Szenarien.

Meist werden Systeme mit Internetzugang über eine Firewall gegen Angriffe aus dem Internet gesichert. Dies ist jedoch für alternative Transportprotokolle eines der größten Hindernisse auf technischer Seite, da eine sinnvolle Firewall auf einer Whitelist aufbaut, welche meist sehr strikt, nur bekannten Transportprotokollen den Zugang gewährt und alles Unbekannte zurückweist. In einer solchen Netzwerkumgebung ist z. B. der Aufbau einer SCTP Verbindung nicht möglich. Auch das häufig verwendete NAT ist in einfachen Routern nur mit TCP und UDP möglich. So muss der Router das spezielle Schicht-vier-Protokoll kennen, da diese eine Prüfsumme besitzen können, die sich auch über den IP-Header erstreckt. Wenn nun der Router mit NAT die IP-Adresse der Quelle oder des Ziels im Paket ändert, muss die Prüfsumme der Transportschicht erneut berechnet werden. Hierfür ist jedoch eine Kenniss über den Aufbau des Pakets und dessen Prüfsummenalgorithmus erforderlich. Wenn nun der Router nur TCP und UDP kennt, kann er keine weiteren Transportprotokolle über die NAT ins Internet routen.

Protokoll	Betriebssystem	NAT	Standardisierung
TCP	Windows/Linux/BSD	OK	1981-1999 (RFC0793-RFC2581)
UDP	Windows/Linux/BSD	OK	1980 (RFC0768)
SCTP	Linux/BSD	1	2000-2007 (RFC2960-RFC4960)
DCCP	Linux/BSD	1	2006 (RFC4340)
UDP Lite	Linux/BSD	1	2004 (RFC3828)

Abbildung 9. ¹⁾ Ob ein bestimmtes Transportprotokoll mit einem konkreten Router funktioniert, hängt von dem verwendeten Routerbetriebssystem ab. Router die auf Linux basieren haben meist eine Unterstützung für alle bekannten Transportprotokolle integriert.

V. ZUSAMMENFASSUNG

Die Verwendung alternativer Transportprotokolle kann die Sicherheit vor Angriffen erhöhen, mit Congestion Control den TCP-Traffic nicht benachteiligen und mit partiellen Prüfsummen die Qualität von Liveübertragungen verbessern. Jedoch sind heutige private Netzwerke an die Nutzung von NAT gebunden, was zu unvorhersehbaren Netzwerkproblemen führt. Mit den heutigen Netzwerken ist lediglich die Nutzung von TCP und UDP ohne Probleme möglich. Dies hindert natürlich die Verbreitung von SCTP, DCCP und UDP-Lite, da ein Netzwerkfehler nur sehr schwer zu lokalisieren und noch schwieriger zu beheben ist. Die alternativen Transportprotokolle sind in bestimmten Situation klar TCP und UDP überlegen, jedoch nicht auf dem typischen Anwender-PC verfügbar, weshalb TCP und UDP noch eine lange Zeit dominant bleiben werden. Womöglich könnte mit der Umstellung von IPv4 auf IPv6 eine Verbreitung in den Massenmarkt stattfinden, da hierfür sehr viel Hard- und Software ausgetauscht werden muss und dabei auf eine breite Unterstützung aller Transportprotokolle geachtet werden kann.

LITERATUR

- [1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.
- [2] CERT. *CERT® Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks*.
- [3] G. Fairhurst and A. Sathiseelan. Quick-Start for the Datagram Congestion Control Protocol (DCCP). RFC 5634 (Experimental), August 2009.
- [4] A. Fieger and M. Zitterbart. Zuverlässige transportdienste für mobile computing. *Informatik - Forschung und Entwicklung*, pages 179–188, 2001.
- [5] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 43–56, New York, NY, USA, 2000. ACM.
- [6] Andreas Jungmaier, Erwin P. Rathgeb, Michael Schopp, and Michael Tüxen. Sctp - a multi-link end-to-end protocol for ip-based networks. *AEU - International Journal of Electronics and Communications*, 55(1):46 – 54, 2001.
- [7] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):89–102, 2002.
- [8] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006. Updated by RFCs 5595, 5596.
- [9] Eddie Kohler, Mark Handley, and Sally Floyd. Designing dccp: congestion control without reliability. *SIGCOMM Comput. Commun. Rev.*, 36(4):27–38, 2006.
- [10] Stephen Pink Lars-Ake Larzon, Mikael Degermark. Udp lite for real time multimedia applications. published via Hewlett-Packard, 1999.
- [11] L-A. Larzon, M. Degermark, S. Pink, L-E. Jonsson, and G. Fairhurst. The Lightweight User Datagram Protocol (UDP-Lite). RFC 3828 (Proposed Standard), July 2004.
- [12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), October 1996.
- [13] L. Ong and J. Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May 2002.
- [14] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.
- [15] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.
- [16] Phill Conrad Randall Stewart. Sctp: An overview. In *Stream Control Transmission Protocols (SCTP): A Reference Guide*, 2004.
- [17] Karunaharan Ratnam and Ibrahim Matta. Wtcp: An efficient mechanism for improving wireless access to tcp services. *International journal of communication systems*, 2002.
- [18] Prasun Sinha, Thyagarajan Nandagopal, Narayanan Venkitaraman, Raghupathy Sivakumar, and Vaduvur Bharghavan. Wtcp: A reliable transport protocol for wireless wide-area networks. *Wirel. Netw.*, 8(2/3):301–316, 2002.
- [19] Jennifer L. Welch Soma Chaudhuri, Brian A. Coan. Using adaptive timeouts to achieve at-most-once message delivery. *Distributed Computing*, 9:109 – 117, 1995.

- [20] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007.
- [21] Randall Stewart and Chris Metz. Sctp: New transport protocol for tcp/ip. *IEEE Internet Computing*, 5:64–69, 2001.
- [22] J. Stone, R. Stewart, and D. Otis. Stream Control Transmission Protocol (SCTP) Checksum Change. RFC 3309 (Proposed Standard), September 2002. Obsoleted by RFC 4960.
- [23] Jörg Widmer, Robert Denda, and Martin Mauve. A survey on tcp-friendly congestion control. *IEEE Network*, 15:28–37, 2001.