DCCP - Datagram Congestion Control Protocol

Benjamin Peherstorfer
Betreuer: Andreas Müller
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: pehersto@in.tum.de

***Kurzfassung*— Für zeitkritische Anwendungen wie VoIP oder Online-Games wird immer öfter UDP eingesetzt. Diese Anwendungen sind darauf angewiesen, dass Pakete innerhalb eines gewissen Zeitfensters beim Empfänger eingehen, und damit ist eine erneute Übertragung von verlorenen Paketen meist unerwünscht. Durch den vermehrten Einsatz von UDP werden immer größere Datenmengen mit UDP übertragen, was zu Stauproblemen im Internet führen kann. Mit DCCP soll dies verhindert werden. Dieses neue Protokoll verbindet die Vorteile von UDP mit Staukontrolle. DCCP bietet eine Art Framework, auf welches man unterschiedliche Staukontroll-Mechanismen aufbauen kann. Die Möglichkeit aus verschiedenen Staukontroll-Mechanismen auszuwählen, soll Entwickler von Anwendungen den Umstieg von UDP auf DCCP erleichtern.**

***Schlüsselworte*— UDP, DCCP, Staukontrolle, TFRC, CCID, timeliness**

I. Einleitung

Der Datenverkehr im Internet wird durch zwei Protokolle auf der Transportschicht dominiert: TCP und UDP. Die früheren Protokolle benutzen UDP wegen des geringen Overheads. Bei DNS (oder auch SNMP) werden nur geringe Datenmengen verschickt und auf jede Anfrage folgt genau eine Antwort. Den teuren und langwierigen Verbindungsaufbau (*startup delay*) von TCP kann man mit UDP umgehen. Außerdem schützt die Zustandslosigkeit (*statelessness*) vor Angriffen (z.B. SYN-Flooding). Dafür handelt man sich die fehlende Zuverlässigkeit ein.

In letzter Zeit haben sich jedoch Anwendungen (z.B. VoIP, Multimedia Streaming, Online-Games) vermehrt aus anderen Gründen für UDP entschieden: Diese Anwendungen verschicken Daten, die extrem zeitkritisch sind. Kommt ein Paket innerhalb einer vorgegebenen Zeitspanne nicht beim Empfänger an, ist es nutzlos (*timeliness over reliability*). Daher braucht die Anwendung absolute Kontrolle über die versendeten Pakete, welche erst durch die Unzuverlässigkeit von UDP ermöglicht wird. Erst beim Versenden des Pakets entscheidet die Anwendung, was als Payload mitgegeben wird (z.B. die letzten Ortsinformation der Spielfiguren bei einem Online-Game). Des Weiteren spielt ein Datenverlust bei solchen Anwendungen keine große Rolle. So bevorzugen die meisten Benutzer von Videostreaming einen kurzen Qualitätsabfall gegenüber dem Anhalten des Bildes. Dies sind alles Gründe, warum TCP für solche Anwendungen nicht geeignet ist und man sich daher für UDP entschieden hat.

Von Anwendungen wie DNS und SNMP wurde UDP nur für geringe Datenmengen benutzt. Auch das hat sich geändert; zwischen einem Video in HD-Qualtität und einer DNS Abfrage ist ein großer Unterschied. Damit wird UDP auch zur Übertragung großer Datenmengen benutzt. Und das kann zu einem Stauproblem im Internet führen. Eine Staukontrolle ist daher wünschenswert. Man spricht von Stau (*congestion*), falls soviele Pakete bei einem Router ankommen, dass dieser Pakete verwerfen muss [1]. Aber auch wenn dieser Extremfall nicht eintritt, kann eine Staukontrolle hilfreich sein. Teilen sich z.B. mehrere Verbindungen einen Kanal mit fester Bandbreite und wollen alle diese Verbindungen eine konstante Übertragungsrate erreichen, so müssen sich diese selbst beschränken. Vor allem bei VoIP und Videostreaming ist eine konstante Übertragungsrate wichtig.

II. Abgrenzung und Problemformulierung

A. Anforderungen und Probleme von UDP mit Staukontrolle

Die Staukontrolle soll für Anwendungen mit einem langlebigen Datenfluss und viel Datenverkehr konzipiert werden. (Anwendungen, welche diese Annahmen nicht erfüllen (DNS, SNMP), verursachen kaum Datenverkehr und damit keinen Stau [2].) Grob kann man hier drei Typen von Anwendungen unterscheiden [3].

- *Internet-Telefonie*: Hier wird pro Zeiteinheit eine feste Anzahl an Bits übertragen. Der Benutzer merkt bereits kleinste Verzögerungen bei der Ankunft neuer Pakete. Eine Pufferung ist nicht möglich, da der Inhalt der Pakete im Voraus nicht bekannt ist. Oft werden Audio-Codecs verwendet, die eine extreme Komprimierung der Payload erlauben. Dies führt zu sehr vielen kleinen Paketen. Je kleiner die Pakete, desto kleiner sollte auch der Header sein.
  Eine schnelle Anpassung an mehr Bandbreite ist nicht nötig. Genauso soll aber auch die Begrenzung der Bandbreite während einer Stausituation nicht zu schnell geschehen. Da meist während einer Sprechpause gar nichts übertragen wird, muss die gesamte Bandbreite bei der Wiederaufnahme des Gesprächs sofort zur Verfügung stehen (also keine langen Startup-Delays).

- *Audio- und Videostreaming*: Ist ein Zeitstück des Videos beim Empfänger abgespielt worden, muss bereits das Paket mit dem nächsten Zeitstück des Videos vom Sender vorliegen. Kommt es zu einem späteren Zeitpunkt, ist es nicht mehr nützlich. Durch Pufferung, verschiedene Video-Layer (z.B. *key* und *incremental frames* bei MPEG [3]) und ähnliches, können Paketverluste ausgeglichen werden.
- *Online-Games*: Meist wird hier die Ortsinformation von den Spielfiguren in der virtuellen Welt übertragen. Dabei muss das letzte angekommene Paket den letzten Ort der Spielfigur beinhalten. Erneutes Senden (auch in sehr kurzen Zeitabständen) macht keinen Sinn. Außerdem kann erst beim Absenden des Pakets die Anwendung entscheiden, was es enthalten soll. Da es hier durchaus von Vorteil ist, wenn viel Bandbreite vorhanden ist, sollte sich die Staukontrolle schnell an neue Verhältnisse anpassen und neue Bandbreite auch sofort zur Verfügung stellen.

Es gilt, die Vorteile der unzuverlässigen Datenübertragung von UDP zu erhalten (*timeliness over reliabilty*), aber trotzdem eine effektive Staukontrolle darauf aufzubauen. Dies sind völlig widersprüchliche Ziele: zum einen will die Anwendung die absolute Kontrolle des Datenverkehrs haben, zum anderen will die Staukontrolle den Datenverkehr begrenzen, falls es zu einer Stausituation kommt.
Man hat sich in [2] daher für einen Kompromiss entschieden: Damit die Staukontrolle überhaupt funktionieren kann, entscheidet sie, wann ein Paket verschickt wird (sie kann also den Datenverkehr begrenzen), die Anwendung entscheidet aber nach wie vor, welche Daten in diesem Paket stehen. (Das ist bei TCP nicht möglich![1])

Verschiedene Anwendungen haben verschiedene Anforderungen an die Staukontrolle, d.h. wie sich der Staukontroll-Mechanismus bei Stau verhält. Die Anwendung soll daher die Wahl zwischen verschiedenen Staukontroll-Mechanismen für jeden Datenfluss haben. Des Weiteren soll die Hinzunahme von neuen Algorithmen wenig Änderungen an der Anwendung oder dem Protokoll erfordern, so dass zukünftige Algorithmen schnell einfließen könnnen.
Bereits um die Auswahl des Algorithmus zu ermöglichen, muss eine Möglichkeit geboten werden, um Features des derzeitigen Datenflusses auszuhandeln. Dafür führt man *Feature Negotiation* ein. Diese ermöglicht es zuverlässig verschiedenste Parameter auszuhandeln, wie z.B. Senderate der Acknowledgements.

Explicit Congestion Notification (ECN) soll unterstützt werden. Mit ECN ist es möglich, dass Router Stau signalisieren, ohne Pakete zu verwerfen. Beim Senden eines Pakets, setzt der Sender zufällig eines von zwei Bits im IP Header. Erreicht das Paket einen Router in einer Stausituation oder am Beginn einer Stausituation, so setzt dieser beide Bits auf Eins. Der Empfänger reicht schließlich diese Information an den Sender weiter (z.B. beim Acknowledgement hinzugefügt). [5] Der Sender kann damit die Stausituation im Netz abschätzen. Das zufällige Setzen von einem der zwei Bits, verhindert das Fälschen der Information durch den Empfänger (ECN Nonces). [6]

Weitere typische Anforderungen sind z.B. den Header-Overhead möglichst gering halten, da Anwendungen dieser Art oft sehr viele kleine Pakete senden und daher ein großer Header besonders schwer wiegt (z.B. Verwenden von verkürzten Sequenznummern). Aber auch das Zusammenspiel mit Firewalls und NAPTs soll besser sein als bei UDP, wo dies auf die Anwendungsschicht verlagert wurde, da UDP verbindungslos ist.

B. Staukontrolle für ein unzuverlässiges Transportprotokoll

Um Staukontrolle und UDP zu kombinieren, gibt es drei Möglichkeiten: (1) man integriert die Staukontrolle in die Anwendung (also über UDP), (2) die Staukontrolle arbeitet unter UDP, am besten auf einer neuen Schicht zwischen IP und UDP, oder (3) man führt auf der Transportschicht ein neues Protokoll ein, welches UDP ersetzt und die oben angeführten Forderungen erfüllt.

1) Staukontrolle in der Anwendung: Die Staukontrolle ist direkt in der Anwendung und kann daher genau auf den Aufgabenbereich angepasst werden. In [2] wird jedoch davon ausgegangen, dass die meisten Entwickler auch mit einer Standardlösung zufrieden sind, falls diese die Möglichkeit bietet, aus verschiedenen Staukontroll-Mechanismen zu wählen. Der große Nachteil der Staukontrolle in der Anwendung ist, dass für jede Anwendung die Staukontrolle neu implementiert werden muss. Die Staukontrolle ist aber ein sehr komplexes und fehleranfälliges System, wie man an den zahlreichen fehlerhaften Implementierungen von TCP gesehen hat [2], [7]. Weiter wird in [2] argumentiert, dass man eher dazu geneigt ist, die Staukontrolle zu umgehen, wenn sie in der Anwendung integriert ist, als wenn sie im Netzwerk-Stack/Betriebssystem liegt.
Ein technisches Problem ergibt sich außerdem noch mit ECN. Um die ECN Fähigkeit zu nutzen, müsste der Anwendung direkter Zugriff auf den IP Header gegeben werden. Selbst wenn ECN auf einem Netzwerkpfad nicht verfügbar ist, bieten die ECN Nonces immer noch eine Möglichkeit zu überprüfen, ob der Empfänger auch alle Paketverluste korrekt mitteilt. Dies müsste man ebenfalls aufgeben. Weitehin ist damit das Firewall-Problem von UDP nicht gelöst; wieder müssten Firewalls ein Protokoll der Anwendungsschicht ansehen, um Verbindungen erkennen zu können.
Von derzeitigen Anwendungen wird meist eine sehr grobe Staukontrolle direkt in der Anwendung benutzt. So wechselt z.B. Skype die Codecs, falls die Bandbreite kleiner wird [8].

[1]TCP_NODELAY muss sich der Staukontrolle unterordnen, siehe z.B. Abschnitt 4.2.3.4 in [4].

2) Staukontrolle unterhalb von UDP: Eine Grundvoraussetzung für jede Staukontrolle ist ein Feedback-Mechanismus (z.B. Acknowledgements). Hierfür werden Sequenznummern benötigt. Will man die Staukontrolle unter UDP einordnen, so muss der Feedback-Mechanismus entweder in der Anwendung stattfinden und die Informationen an einen *Congestion Manager* weitergeben, oder man verlegt auch den Feedback-Mechanismus selbst unter UDP. Ein Feedback-Mechanismus in der Anwendung hat die gleichen Probleme als würde man die Staukontrolle in die Anwendung integrieren. Verlegt man den Feedback-Mechanismus unter UDP, so muss man irgendwo die Sequenznummern unterbringen. Dazu braucht man einen zusätzlichen Header. Man müsste dann auch eine Protokollnummer im IP Header dafür reservieren und hätte damit schon fast ein neues Protokoll eingeführt. Lösungen um den Header im Optionen-Feld von IP zu verstecken, führen vermutlich zu Problemen bei verschiedenen IP Implementierungen, siehe [2].

3) Staukontrolle auf der Transportschicht: Insgesamt spricht daher alles für eine Implementierung der Staukontrolle auf der Transportschicht. Anwendungen müssten wenig an ihrem Code ändern, um das neue Protokoll zu verwenden. Auch an der Struktur unter der Transportschicht müsste nichts geändert werden, d.h. einer schnellen Ausbreitung des Protokolls stehen keine langwierigen Änderungen an Routern usw. bevor.

C. Staukontrolle auf der Transportschicht

Auch wenn nun entschieden ist, dass die Staukontrolle auf der Transportschicht stattfinden soll, so bleibt immer noch die Frage offen, ob man ein neues Protokoll entwickelt, oder ein bekanntes modifiziert. Zwei Protokolle bieten sich hierfür an: TCP und SCTP.

Bei TCP sind sämtliche Mechanismen (Flusskontrolle, Staukontrolle, Zuverlässigkeit usw.) ineinander verwoben. Lässt man etwas davon weg, funktionieren alle anderen nicht mehr. Außerdem benutzt TCP Sequenznummern, welche Bytes im Stream zählen, hier will man aber Datagramme. Man will vielleicht sogar erlauben, dass Datagramme in der falschen Reihenfolge beim Empfänger ankommen (*out-of-order delivery*). Zusätzlich setzen sich Änderungen von TCP nur sehr langsam durch. In [2] wird daher empfohlen, TCP so zu lassen wie es ist.

Auch gegen SCTP finden sich genügend Argumente. Da SCTP das Bündeln von mehreren Streams erlaubt, beinhaltet der Header viel Information (viel Overhead), die hier nicht gebraucht wird. Auch die Wahl von verschiedenen Staukontroll-Mechanismen wird von SCTP nicht unterstützt. Größere Änderungen an SCTP könnten außerdem vorhandene Implementierungen durcheinander bringen.

Nach diesen Überlegungen kommt man zu dem Schluss, dass die Entwicklung eines neuen Protokolls die beste Lösung ist: Das Datagram Congestion Control Protocol (DCCP), welches unzuverlässige Datenübertragung mit Staukontrolle (und verschiedenen Staukontroll-Mechanismen) bietet, sowie ECN benutzt und darüber hinaus noch geringen Overhead hat.

III. Datagram Congestion Control Protocol (DCCP)

A. Überblick und Funktionsweise

DCCP wird in RFC 4340 [6] vorgestellt und definiert. Man unterscheidet die DCCP Kernfunktionalitäten, welche in RFC 4340 beschrieben sind und den Staukontroll-Mechanismus oder -Algorithmus. Der Staukontroll-Mechanismus kann je nach Anwendung beliebig ausgetauscht werden und baut auf den Kernfunktionalitäten wie z.B. Feedback-Mechanismus, Verbindungsaufbau (Handshake) oder Sequenznummern auf. So können neue Staukontroll-Algorithmen entwickelt werden, ohne die Kernfunktionalitäten von DCCP zu ändern.

Sollen Daten mit DCCP übertragen werden, wird eine Verbindung aufgebaut. Ein Acknowledgement Framework informiert den Sender, wieviele und warum Daten/Datagramme verloren gingen. Es werden also Pakete nicht einfach nur bestätigt, sondern es wird zusätzliche Information an den Sender geleitet, so dass beim Sender der Staukontroll-Mechanismus genug Information hat, um seine Staukontrolle korrekt durchzusetzen. DCCP unterscheidet zwischen verschiedenen Arten von Paketverlusten: Daten korrupt, Protokoll Spezifikation verletzt, Puffer voll usw. Eine erneute Übertragung eines verloren gegangen Pakets muss jedoch die Anwendung selbstständig durchführen.

B. Pakettypen und Header

Bei DCCP wird zwischen zehn verschiedenen Pakettypen unterschieden, deren Header sich jeweils unterscheiden. So werden z.B. verschiedene Pakettypen für den Verbindungsaufbau und für die Datenübertragung verwendet. Der Vorteil hiervon ist, dass die nötigen Felder im Header für den Verbindungsaufbau nicht auch bei den Paketen während der Datenübertragung mitgeschleppt werden müssen (Overhead minimieren).
Es gibt die Pakettypen DCCP-Request und DCCP-Response um eine Verbindung aufzubauen, für die Datenübertragung die Pakettypen DCCP-Data, DCCP-Ack und DCCP-DataAck und bei der Terminierung die Pakettypen DCCP-CloseReq, DCCP-Close und DCCP-Reset. Um eine Verbindung zu synchronisieren, wurden noch die Pakettypen DCCP-Sync und DCCP-SyncAck eingeführt. (Warum es nötig sein kann eine Verbindung zu synchronisieren, wird in Abschnitt III-C beschrieben.)

Wie eine Datenübertragung mit Verbindungsauf- und abbau aussieht, ist in Abb. 1 dargestellt. In Teil (1) wird die Verbindung aufgebaut. Dabei initiiert der Client die Verbindung mit einem DCCP-Request Paket. Der Server bestätigt die Verbindung mit DCCP-Response, welches der

Client mit DCCP-Ack bestätigt. Damit ist die Verbindung aufgebaut, d.h. alle Parameter sind ausgehandelt, sowie die Sequenznummern initialisiert (siehe unten). Nach der Datenübertragung in (2) will der Server in (3) die Verbindung mit DCCP-CloseReq trennen. Der Client bestätigt dies mit DCCP-Close. Der Server signalisiert dann mit seinem DCCP-Reset Paket, dass er keine weiteren Daten mehr von dieser Verbindung empfangen wird.

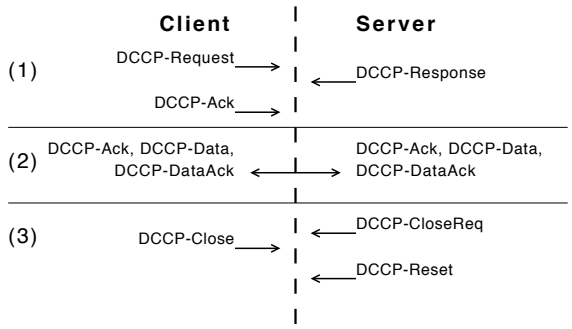

Abbildung 1. Ablauf einer Datenübertragung mit DCCP.

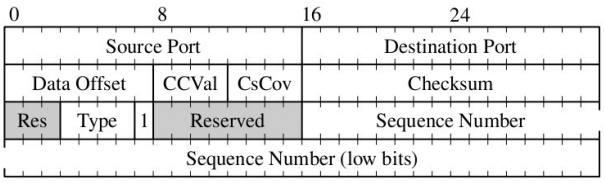

Abbildung 2. Generic-Header eines DCCP Pakets. Quelle: [3].

Ein DCCP-Header besteht aus zwei Teilen. Den *Generic-Header* trägt jedes Paket, der *Packet-Header* ist für jeden Pakettyp spezifisch. In Abb. 2 ist der Generic-Header dargestellt:

- *Source und Destination Ports* (jeweils 16 Bits): Sind wie bei TCP und UDP zu verstehen.
- *Data Offset* (8 Bits): Gibt den Offset vom Anfang des DCCP Headers bis zum Start der Payload (in 32 Bit Worten) an.
- *CCVal* (4 Bits): Wird vom Staukontroll-Mechanismus verwendet.
- *CsCov* (4 Bits): Hier kann angegeben werden, was alles von der Checksumme geprüft wird (nur Header oder auch Payload). Eventuell Performancegewinn möglich.
- *Checksum* (16 Bits): Checksumme über Header bzw. auch Daten.
- *Type* (4 Bits): Gibt den Pakettyp an, d.h. welcher Packet-Header diesem Generic-Header folgt.
- *1* (1 Bit): Dieses Bit gibt an, ob lange oder kurze Sequenznummern verwendet werden. Wie man an Abb. 2 sofort sieht, nimmt die Sequenznummer den Großteil des Platzes ein. Daher kann man zwischen kurzen und langen Sequenznummern wählen (siehe Abschnitt III-C).
- *Sequence Number* (48 oder 24 Bits): Die Sequenznummer dieses Packets.

C. Sequenznummern

Eine der wichtigsten Kernfunktionalitäten von DCCP sind Sequenznummern. Nahezu alle anderen Funktionalitäten (Verbindungsaufbau, Feature Negotiation usw.) werden erst durch Sequenznummern möglich. Zusätzlich werden Sequenznummern z.B. benötigt um Duplikate zu erkennen, verlorene Pakete zu identifizieren, oder Angriffen (*packet injection*) entgegen zu wirken.

Sequenznummern von DCCP zählen Datagramme und keine Bytes wie bei TCP. Jedes Datagramm, auch Acknowledgements, erhöhen die Sequenznummer. DCCP will Staukontrolle auch auf Acknowledgements anwenden und Feature Negotiation (z.B. von Staukontroll-Mechanismen) während der Verbindung erlauben. Dies geschieht aber durch Pakete, welche keine Payload tragen und daher müssen auch diese eine Sequenznummer haben, um sie bestätigen zu können [3].

Wie schon oben bei Abb. 2 angesprochen, bietet DCCP die Möglichkeit, kurze Sequenznummern (*short sequence numbers*) zu verwenden. Eine (lange) Sequenznummer von DCCP besteht aus 48 Bits. Um Overhead zu sparen, können z.B. DCCP-Data Pakete während der Datenübertragung die oberen 24 Bits weglassen und nur noch die unteren 24 Bits übertragen. Die Implementierung auf der anderen Seite erweitert dann diese 24 Bits wieder auf 48 Bits (Pseudocode in [6]). Es wird jedoch empfohlen, dieses Feature mit Sorgfalt zu verwenden, da hierdurch Angriffe leichter möglich sind, siehe [6].

Bei TCP werden *cumulative acknowledgements* verwendet, d.h. die ACK-Nr. bestätigt den Empfang der Pakete/Bytes bis zu dieser ACK-Nr., wobei die nächste erwartete Sequenznummer genau die ACK-Nr. ist. Damit ist der Status der aktuellen Verbindung mit dieser Acknowledgement Nummer beschrieben. Die Teilnehmer wissen wieviel bereits angekommen ist und damit wieviel verloren ging. Für DCCP ist diese Art der Bestätigung nicht sinnvoll, da DCCP niemals erneute Übertragung von Paketen durchführt. Daher hat man sich bei DCCP dazu entschlossen, dass eine ACK-Nr. der Sequenznummer des letzten angekommenen Pakets/Datagramms entspricht. Das hat aber zur Folge, dass aus der ACK-Nr. nicht mehr geschlossen werden kann, wieviele Pakete verloren gingen (*connection history*), was für die Staukontrolle entscheidend ist. Daher gibt es bei DCCP weitere Mechanismen, die genau spezifizieren, was mit den einzelnen Paketen geschehen ist, siehe dazu Abschnitt III-D.

Um Angriffe (wie *packet injection*) zu erkennen, benutzt DCCP (sowie TCP) *Sequence* und *Acknowledgement Number Windows*, welche festlegen, in welchem Bereich die nächsten

Sequenznummern und ACK-Nr. liegen sollen. Kommt ein Paket beim Empfänger an, prüft DCCP die Sequenznummer anhand von Regeln, welche in [6] beschrieben sind. Dabei gilt im Groben, dass eine Sequenznummer gültig ist, falls sie im gültigen Bereich des Fensters liegt.
Kommt es während einer Datenübertragung zu einem Ausfall und gehen dadurch viele aufeinanderfolgende Pakete verloren, so kann es vorkommen, dass die Pakete, welche schließlich wieder bis zum Empfänger durchdringen, bereits eine Sequenznummer außerhalb des gültigen Bereichs haben. (Bei TCP würden nun einfach die verlorenen Pakete durch *cumulative acknowledgements* erkannt und neu übertragen oder überhaupt die Verbindung terminiert.) DCCP weiß in so einem Fall jedoch nicht, ob jemand die Pakete eingeschleust hat, ob sie von einer alten Verbindung stammen, oder ob sie wirklich zur aktuellen gültigen Verbindung gehören. Daher versucht DCCP, die Verbindung wieder zu synchronisieren. Dabei schickt der Empfänger an den Sender ein DCCP-Sync Paket, welches die eingegangene Sequenznummer enthält. Der Partner der Verbindung empfängt das DCCP-Sync und antwortet mit einem DCCP-SyncAck falls er die Sequenznummer als gültig betrachtet. Dann aktualisiert der Empfänger sein Fenster und die Verbindung ist wieder synchron, siehe Abb. 3. Obwohl dieser Ansatz recht einfach und übersichtlich klingt, gibt es doch einige Probleme und Sonderfälle. Wie mit diesen umzugehen ist, wird in [6] besprochen.

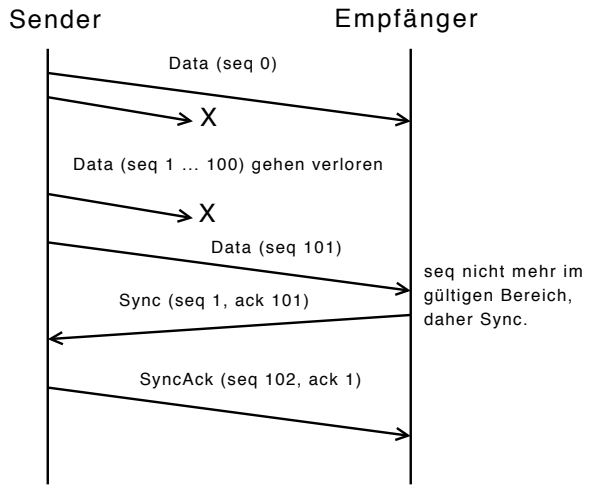

Abbildung 3. Synchronisation einer Verbindung. Nach [6].

D. Acknowledgements

Acknowledgements haben bei DCCP eine andere Bedeutung als bei TCP. Wird bei DCCP ein Datagramm durch ein ACK bestätigt, so heißt dies nur, dass der Header des Pakets erfolgreich bearbeitet wurde (Checksumme stimmt, Optionen wurden bearbeitet usw.), aber nicht, dass das Paket auch bei der Anwendung ankommt. Warten z.B. mehrere Pakete in einem Empfangspuffer auf eine Anwendung, so ist vielleicht nur das letzte empfangene Paket von Bedeutung (z.B. Ortsinformation bei Online-Games), der Rest wird verworfen. Acknowledgements sind bei DCCP hauptsächlich für die Staukontrolle eingeführt worden und die Staukontrolle interessiert nur was an Verlusten wegen der Situation im Netz stattgefunden hat. Daher ist dieser Ansatz vernünftig.

(a)	0010011?	Länge	SSLLLLLL	SSLLLLLL	...
(b)	0010011?	Länge ...	00000000 00000011	11000000 01000000	... 00000101

Tabelle I
ACK VECTOR OPTION

DCCP bietet über Optionen/Features einige Mechanismen an, die zusätzliche Information über die Pakete der aktuellen Verbindung beinhalten. Der eingesetzte Staukontroll-Mechanismus entscheidet dabei, welche Optionen aktiv sind und benutzt werden. So legt z.B. das *Ack Ratio Feature* die Rate fest, mit welcher der Empfänger die DCCP-Acks versenden muss. Damit lässt sich bei Stau eine optimale Rate für den Acknowledgement-Verkehr einstellen. Wieder wird über den Staukontroll-Mechanismus festgelegt, welcher Wert hier genau eingestellt werden soll.

Die wichtigsten Informationen für die Staukontrolle werden jedoch über die *Ack Vector Option* geliefert. Dies ist eine Option im DCCP-Header und beschreibt den Status der letzten Pakete. Der Aufbau eines solchen Ack Vektors ist in Tabelle I angegeben.

Das erste Byte des Vektors spezifiert die Option-Nr., das zweite Byte die Länge des Vektors. Danach folgen Felder (je ein Byte) mit dem Aufbau “SSLLLLLL”. Dabei stehen die ersten beiden Bits “SS” für einen Status (Received (00), Received ECN Marked (01), Not Yet Received (11)) und die folgenden 6 Bits “LLLLLL” geben die Anzahl der aufeinanderfolgenden Pakete mit diesem Status an. Dabei ist das erste Byte in einem Ack Vektor auf das Paket mit der ACK-Nr. bezogen, welche ebenfalls in diesem Paket geschickt wird (und damit ist ein Paket mit Ack Vektor aber ohne ACK-Nr. ungültig). Betrachtet man das Beispiel von Tabelle I in Zeile (b) und geht davon aus, dass die ACK-Nr. des Pakets 100 ist, so bedeuten die fünf SSLLLLLL-Komponenten des Vektors [6]

1) Paket 100 ist angekommen,
2) Paket 99 ist nicht angekommen,
3) Pakete 98 bis 95 sind angekommen,
4) Paket 94 ist mit Status “Received ECN marked” angekommen, und
5) Pakete 93 bis 88 sind angekommen.

Die Information über ein Paket A wird solange in den Ack Vektoren mitgeschickt, bis ein Paket mit Ack Vektor, welches Paket A behandelt, vom Gegenüber bestätigt wird. Dadurch kann es vorkommen, dass eine Sequenznummer/Datagramm in zwei Ack Vektoren behandelt wird, und dass sich diese Informationen widersprechen! Der Empfänger kann z.B. direkt nach dem Abschicken eines Pakets mit Ack Vektor ein noch fehlendes Paket erhalten und die Ack Vektor Information

im nächsten Datagramm dann aktualisieren. Der Empfänger erhält so zwei Pakete mit Ack Vektoren, welche sich bei einem Datagramm unterscheiden. (Zuerst “Not Yet Received” und dann “Received”.) Ein andere Möglichkeit ist, dass im Netz die Acknowledgements einfach umgedreht werden. Was bei solchen Inkonsistenzen zu tun ist, wird in [6] genau festgelegt (die Ack Vektoren werden miteinander kombiniert und das Ergebnis anhand einer Tabelle des RFCs abgelesen).

Wie oben erwähnt, werden Pakete in einem Ack Vektor solange behandelt, bis dies bestätigt wurde. Stellt man sich nun die Verbindung eines Videostreams vor, so sendet ausschließlich der Server Daten an den Client. Der Client bestätigt auch die Daten des Servers mit DCCP-Acks und schickt darin immer den Ack Vektor mit. Wird jedoch ein DCCP-Ack des Clients nicht vom Server bestätigt, so schickt der Client sämtliche Informationen aller Pakete der bisherigen Verbindung in allen DCCP-Ack Paketen mit. Daher muss der Sender einer solchen einseitigen Verbindung Bestätigungen von Acknowledgements verschicken (*acks of acks*). (Dabei sind *acks-of-acks-of-acks* hierbei nicht nötig!) Bei Verbindungen, bei denen Daten in beiden Richtungen fließen, werden diese Acknowledgements in DCCP-DataAck Paketen verschickt. (Wieder kontrolliert der Staukontroll-Mechanismus, wann Ack Vektoren bestätigt werden.)

Zwei weitere Optionen sollen hier kurz beschrieben werden: *NDP Count Option* und *Data Dropped Option*. Da Sequenznummern auch bei DCCP-Acks erhöht werden, kann ohne zusätzliche Information nicht entschieden werden, ob wirkliche Daten der Anwendung verloren gingen. Hier hilft die *NDP Count Option*, welche die Länge von aufeinanderfolgenden Nicht-Daten-Paketen angibt (*non-data-packet*). Ist in einem Paket die NDP Count Option auf n gesetzt, so heißt dies, dass die letzten n Pakete Nicht-Daten-Pakete waren. Gehen z.B. während einer Verbindung 10 Pakete verloren und die Option ist bei dem nächsten eintreffenden Paket gesetzt und hat einen Wert ≥ 10, dann sind keine Anwendungsdaten verloren gegangen.
Erreichen Pakete/Daten nicht die Anwendung, so bietet die *Data Dropped Option* die Möglichkeit, zusätzliche Information an den Sender zu schicken, warum die Daten nicht weitergereicht wurden. Der Aufbau der Data Dropped Option ist ähnlich zum Ack Vektor. Mögliche Gründe warum Daten nicht weitergereicht wurden, sind z.B. “Protocol Constraints” (000), “Corrupt” (011) oder “Application Not Listening” (010).

E. Feature Negotiation

Während und am Anfang einer Verbindung mit DCCP müssen einige Parameter ausgehandelt werden (Staukontroll-Algorithmus, Ack Vector verwenden, NDP Count Option verwenden usw.). Bei DCCP hat man sich für eine sehr klare Lösung entschieden. Es gibt einen allgemeinen Mechanismus, die *Feature Negotiation*, welche es erlaubt, solche Parameter auszuhandeln. Die einzelnen Parameter werden dabei über Codes (*feature number*) identifiziert.

Es wurden vier Optionen eingeführt: *Change L*, *Confirm L*, *Change R* und *Confirm R*. Eine Feature Negotiation wird durch das Setzen einer Change Option eingeleitet und durch eine Confirm Option abgeschlossen. Dabei unterscheiden “L” und “R” ob das Feature bei sich selbst oder beim Partner geändert werden soll. Will z.B. Teilnehmer A ein Feature F bei Teilnehmer B ändern (z.B. Teilnehmer B soll keinen Ack Vektor mehr schicken), dann schickt Teilnehmer A ein Change R an B. Will jedoch Teilnehmer A ein Feature F von sich selbst ändern, so schickt er ein Change L. Entsprechend wird mit Confirm L und Confirm R geantwortet. Wieder können hier einige Sonderfälle auftreten (Change geht verloren, Unbekannte Option usw.), die in [6] besprochen werden.

Damit zwei Partner sich einigen können, gibt es Regeln (*Reconciliation Rules*) die Prioritäten festlegen. Bis jetzt sind zwei Regeln spezifiziert: *Server-Priority* und *Non-Negotiable*. Bei Server-Priority besteht das Feature aus einer Liste von Möglichkeiten. Sendet A die Liste (3,2,1) (von z.B. verschiedenen Staukontroll-Mechanismen) an B, so wählt dieser etwas davon aus und schickt die Antwort zurück. Bei Non-Negotiable handelt es sich nur um einen Wert (z.B. Ack Ratio). Dieser Wert muss akzeptiert werden, falls er gültig ist.

Insgesamt erlaubt diese allgemeine Methode das Hinzufügen von neuen Features ohne große Änderungen am Protokoll.

IV. Staukontroll-Mechanismen

Wie schon oben erwähnt, ist DCCP lediglich ein Framework für die Staukontrolle [3]. Aufbauend auf dieses Framework können verschiedene Staukontroll-Mechanismen und -Algorithmen eingesetzt werden. Bei DCCP wird der Staukontroll-Mechanismus am Anfang einer Verbindung ausgewählt. Jedem Mechanismus ist eine Nummer zugeordnet, der *congestion control identifier* (CCID). Die Funktionsweise eines solchen Staukontroll-Mechanismus wird durch das Staukontroll-Profil (*congestion control profile*) festgelegt. Im Folgenden sollen drei Staukontroll-Mechanismen etwas genauer besprochen werden.

A. TCP-like Congestion Control

Der *TCP-like congestion control* Mechanismus versucht die TCP Staukontrolle basierend auf SACK [9] möglichst genau nachzubilden. Als CCID hat man 2 gewählt. Das zugehörige Staukontroll-Profil findet man in RFC 4341 [10].

CCID 2 passt sich sehr schnell an die verfügbare Bandbreite an und versucht damit auf lange Sicht soviel Bandbreite wie möglich zu erlangen. Dieses schnelle Anpassen an neue Situationen führt jedoch zu großen

und abrupten Schwankungen in der Bandbreite (*additive increase/multiplicative decrease* (AIMD)). Daher wird CCID 2 eingesetzt, falls die Anwendung mit solchen Schwankungen umgehen kann. Grob kann man sagen, dass Anwendungen bei denen eine Pufferung möglich ist, für CCID 2 geeignet sind (z.B. Multimedia Streaming). Anwendung bei denen dies nicht möglich ist (z.B. VoIP), sollten auf andere Mechanismen ausweichen.

SACK TCP Staukontrolle basiert auf *Slow Start* und *Congestion Avoidance*. Zusätzlich werden *Fast Retransmit* und *Fast Recovery* verwendet [9]. Am Anfang einer Verbindung wird während der Slow-Start-Phase für jedes eintreffende ACK das Staufenster (*congestion window*) verdoppelt. Dauert das Eintreffen eines ACKs zu lange, so wird in die Congestion-Avoidance Phase gewechselt. Dort wird nun das Staufenster halbiert und nur noch linear erhöht [11], [12].
Um die Performance dieses Algorithmus zu verbessern, wurden noch weitere Algorithmen entwickelt [12]:

- *Fast Retransmit*: Treffen beim Sender dreimal ACKs mit der gleichen Acknowledgement Nummer ein, wird das Paket bereits als verloren angenommen. (Man wartet also nicht bis ein Timer abläuft.)
- *Fast Recovery*: Beim Verlust eines Pakets wird nicht wieder in die Slow-Start-Phase gewechselt, sondern in die Congestion-Avoidance-Phase.

Mit der Ack Vector Option bietet DCCP einen Mechanismus wie SACK von TCP an. Damit kann der Algorithmus in CCID 2 dann das AIMD Verhalten der SACK TCP Staukontrolle simulieren [3].

B. TCP-friendly Rate Control (TFRC)

TFRC wurde bereits 2003 unabhängig von DCCP vorgestellt und wird in RFC 5348 [13] beschrieben. Bei DCCP hat TFRC CCID 3 und das Profil wird in RFC 4342 [14] festgelegt. Ziel ist es, die abrupten Schwankungen von CCID 2 zu vermeiden, aber trotzdem eine faire Aufteilung der Bandbreite zu ermöglichen.

Der Ablauf einer Anpassung der Bandbreite durch TFRC lässt sich in vier Schritte einteilen [13]:

1) Der Empfänger misst die *Loss-Event-Rate* und schickt diese mit Zeitstempel an den Sender zurück. Die Loss-Event-Rate entspricht in etwa der Anzahl der verlorenen Pakete durch die Anzahl der übertragenen Pakete in einem gewissen Zeitintervall.
2) Der Sender kann mit der Loss-Event-Rate und dem Zeitstempel die RTT berechnen.
3) Die Loss-Event-Rate, RTT sowie Paketgröße und andere Parameter werden in die TCP-Durchsatzgleichung eingesetzt und man erhält die zu benutzende Bandbreite.
4) Der Sender aktualisiert seine Senderate entsprechend.

Schickt der Empfänger wirklich die Loss-Event-Rate an den Empfänger, so kann dieser nicht kontrollieren ob sie richtig ist. D.h. der Empfänger kann sich so mehr Bandbreite erschleichen, als ihm eigentlich zusteht. Darum fordert CCID 3, dass nicht die Loss-Event-Rate an den Sender geschickt wird, sondern *Loss Intervals*. Dabei handelt es sich um eine Option welche angibt, was mit den letzten Paketen passiert ist. Abgesichert wird diese Information über die ECN Nonces. Daraus kann dann der Sender die Loss-Event-Rate berechnen.

C. TCP-Friendly Rate Control for Small Packets (TFRC-SP)

Der *TCP-Friendly Rate Control for Small Packets* Mechanismus ist in RFC 4828 [15] spezifiziert und wird noch als experimentell angesehen. Die CCID ist 4 und das Profil wird in RFC 5622 [16] beschrieben.

Bei Anwendungen wie VoIP werden viele kleine Pakete verschickt. Dies hat zwei Gründe. Die Anwendung kann nicht lange auf neue Daten warten, sondern muss dafür sorgen, dass der Empfänger möglichst aktuelle Daten hat (zeitkritisch) und neue Codecs sind so gut, dass für kleine Audioabschnitte weniger als 20 Bytes Daten anfallen [3]. Man hat nun festgestellt, dass sich TFRC für solche Datenströme nicht eignet.
Während TCP solche Daten einfach in großen Segmenten verschickt, hat DCCP keinen Einfluss darauf, welche Pakete von der Anwendung kommen. Da bei TFRC die Paketgröße in die Durchsatzgleichung einfließt, verhält sich TFRC wie ein TCP-Fluss, der sehr kleine Segmente verschickt. Solch ein TCP-Fluss erzielt aber eine kleinere Durchsatzrate (in Bytes pro Sekunde) als ein TCP-Fluss mit großen Segmenten. Darum treten solche TCP-Flüsse praktisch nie auf [3]. Benutzen ein TCP-Fluss mit großen Segmenten und ein TFRC-Fluss mit kleinen Paketen einen Durchsatzknoten gemeinsam, reduziert (bei gleicher Paketverlust-Rate) TFRC die Senderate (in Pakete pro Sekunden) immer weiter, obwohl der Durchsatz (in Bytes pro Sekunde) des TFRC-Flusses eigentlich viel kleiner ist, als von dem TCP-Fluss. In Abb. 4 ist dies gut zu erkennen. Der TFRC-Fluss reduziert die Senderate immer weiter, obwohl er viel weniger Durchsatz erzeugt als der TCP-Fluss.
Bei TFRC-SP wird in der TCP-Durchsatzgleichung einfach eine andere Paketgröße eingesetzt und man erhält damit einen höheren Durchsatz und trotzdem verhält sich der Algorithmus noch fair gegenüber einem TCP-Fluss [16], siehe Abb. 4.

Diese Änderung zieht jedoch einige Probleme nach, weswegen TFRC-SP auch noch als experimentell angesehen wird. Verwirft man z.B. keine Pakete sondern nur Bytes (und sieht ein Paket als verloren an, falls ein Byte verloren wurde), so verhält sich TFRC-SP unfair gegenüber TCP, wie in [3] gezeigt wird.

V. Ausblick

Derzeit ist DCCP noch nicht weit verbreitet. Es gibt erste Implementierungen von DCCP unter Linux, FreeBSD und

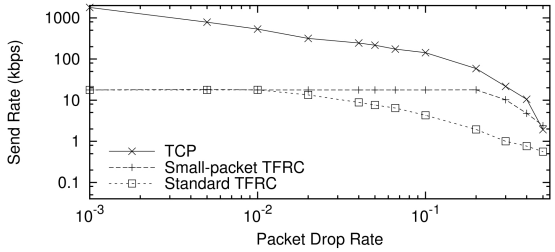

Abbildung 4. Vergleich von TCP (1460-Byte Segmente), TFRC (14-Byte Segmente, Anwendung sendet maximal 12 kbps (inkl. Header)) und TFRC-SP bei vorgegebene Paketverlust-Rate. Quelle [3].

NetBSD, aber noch keine Anwendung die DCCP als (Haupt-) Protokoll verwenden. Dies hat mehrere Gründe. Firewalls und NAPTs verstehen DCCP meist noch nicht. Anwendungen setzen oft auf eigene Mechanismen um Schwankungen in der Bandbreite auszugleichen, wie am Beispiel von Skype in [8] gezeigt wird. Weiter hoffen die Entwickler von DCCP auf mehr Feedback von den Anwendungsentwicklern, da die Staukontroll-Mechanismen noch einige Probleme haben.
Es gibt bereits zahlreiche Untersuchungen, wie sich DCCP mit verschiedenen Staukontroll-Mechanismen bei unterschiedlichen Anwendungen verhält. In [17] wird die Qualität von Videostreams über DCCP untersucht. Dabei kommt TFRC (CCID 3) zum Einsatz. Verschiedene Szenen in einem Video brauchen verschiedene Bitraten, um die Videoqualität gleich zu halten. Da TFRC diese Unterschiede aber nicht beachtet, kommt es zu teilweise starken Schwankungen der Qualität. Eine konstante Qualität während der gesamten Übertragung wäre wünschenswert. In [18] wird die Übertragung von VoIP mit TFRC (CCID 3) in einem echten Netzwerk (und keinem Simulator) untersucht. Dabei zeigt sich, dass TFRC bei langen Pausen (*idle period*) die Senderate zu stark begrenzt, was zu einem Qualitätsverlust führt. Weiter ist z.B. noch nicht klar, wie mit Datenflüssen umgegangen werden soll, welche sehr schnell zwischen einer (hohen) Datenübertragung und einer Pause wechseln.

VI. Zusammenfassung

Mit DCCP wurde ein Protokoll entwickelt, welches Staukontrolle auch für unzuverlässigen Datenfluss ermöglicht. Besonders der modulare Aufbau von DCCP und der Staukontrolle ist essentiell. Wie das Beispiel von VoIP und TFRC-SP gezeigt hat, führen kleine Unterschiede bei Anwendungen bereits zu völlig anderen Anforderungen an die Staukontrolle. Bei einem Protokoll können solche Änderungen an der Staukontrolle nicht so schnell einfließen, daher ist die Entkoppelung von Protokoll und Staukontrolle ein vernünftiger Schritt. Da die Staukontrolle aber auf Feedback-Mechanismen angewiesen ist, mussten diese bei DCCP recht allgemein gehalten werden.

Insgesamt bietet DCCP mit dem modularen Aufbau den richtigen Ansatz, um verschiedenste Staukontroll-Mechanismen zu unterstützen. Auch der Bedarf an einem unzuverlässigen Protokoll mit Staukontrolle ist vorhanden. Jedoch lässt sich nicht sagen, ob und wann sich DCCP wirklich durchsetzt.

Literatur

[1] L. Mamatas, T. Harks, and V. Tsaoussidis, "Approaches to Congestion Control in Packet Networks," *Journal of Internet Engineering*, January 2007.

[2] S. Floyd, M. Handley, and E. Kohler, "Problem Statement for the Datagram Congestion Control Protocol (DCCP)," RFC 4336 (Informational), Internet Engineering Task Force, Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4336.txt

[3] E. Kohler, M. Handley, and S. Floyd, "Designing dccp: congestion control without reliability," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2006, pp. 27–38.

[4] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122 (Standard), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 4379. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[5] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt

[6] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340 (Proposed Standard), Internet Engineering Task Force, Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[7] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz, "Known TCP Implementation Problems," RFC 2525 (Informational), Internet Engineering Task Force, Mar. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2525.txt

[8] L. De Cicco, S. Mascolo, and V. Palmisano, "Skype video responsiveness to bandwidth variations," in *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. New York, NY, USA: ACM, 2008, pp. 81–86.

[9] E. Blanton, M. Allman, K. Fall, and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP," RFC 3517 (Proposed Standard), Internet Engineering Task Force, Apr. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3517.txt

[10] S. Floyd and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control," RFC 4341 (Proposed Standard), Internet Engineering Task Force, Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4341.txt

[11] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, August 1988.

[12] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001 (Proposed Standard), Internet Engineering Task Force, Jan. 1997, obsoleted by RFC 2581. [Online]. Available: http://www.ietf.org/rfc/rfc2001.txt

[13] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348 (Proposed Standard), Internet Engineering Task Force, Sep. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5348.txt

[14] S. Floyd, E. Kohler, and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)," RFC 4342 (Proposed Standard), Internet Engineering Task Force, Mar. 2006, updated by RFC 5348. [Online]. Available: http://www.ietf.org/rfc/rfc4342.txt

[15] S. Floyd and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant," RFC 4828 (Experimental), Internet Engineering Task Force, Apr. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4828.txt

[16] E. Kohler and S. Floyd, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)," RFC 5622 (Experimental), Internet Engineering Task Force, Aug. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5622.txt

[17] J. V. Velthoven, K. Spaey, and C. Blondia, "Performance of constant quality video applications using the dccp transport protocol," in *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN'06)*, Nov. 2006, pp. 511–512.

[18] H. Hwang, X. Yin, Z. Wang, and H. Wang, "The internet measurement of voip on different transport layer rotocols," in *Information Networking, 2009. ICOIN 2009. International Conference on*, Jan. 2009, pp. 1–3.