MPTCP

Dominik Weidemann

10. Juni 2011

Zusammenfassung Die Struktur des Internets hat sich über die Jahre von Singlelinks zu Multilinks verändert, u.a. durch Mobil-Telefone, die sowohl eine WLAN-, als auch eine 3G-Schnittstelle haben. Eine Bündelung dieser Schnittstellen bringt Vorteile im Bezug auf Stabilität und Geschwindigkeit. Das Multipath-TCP (MPTCP) ist ein neues IETF-Protokoll, das Funktionen für die Verbesserung der Geschwindigkeit und Stabilität von Verbindungen mittels transparenter Mehrwegeunterstützung bereitstellt. In dieser Arbeit werden die aktuellen Probleme genauer erläutert, Verbesserungsansätze vorgestellt und schließlich bewertet, ob MPTCP eine Lösung für die identifizierten Probleme sein kann.

1 Einleitung

Der Grossteil des Internet-Verkehrs (mehr als 90%) verwendet auf der Transportschicht das Transmission-Control-Protocol (TCP) [7]. Es wurde vor ca. 30 Jahren [10] definiert und entwickelt. Seitdem haben sich die Anforderungen an das Internet geändert. Viele Server sind heute mit mehreren Netzwerk-Schnittstellen ausgestattet (dies wird *Multihoming* genannt), um Redundanz oder Lastverteilung zu ermöglichen. Es wäre wünschenswert, wenn diese physischen Verbindungen zu einer für den Anwender transparenten logischen Verbindung verschmolzen werden könnten, ohne dabei Kompatibilität zu bereits bestehenden Anwendungen und Technologien zu verlieren. Damit ließen sich für Anwendungen sowohl transparente Ausfallsicherung bzw. ein nahtloser Wechsel zwischen Pfaden, als auch Geschwindigkeitsvorteile erzielen.

TCP erlaubt pro Socket nur einen logischen Fluss [10]. Daher wurden einige Alternativen auf verschiedenen Ebenen des OSI-Modells entwickelt, z.B. STCP (Transportschicht) [12], welches Garantien wie TCP zusichert (Verlässlichkeit, Sequenzialität, Stau-Kontrolle), aber eben auch Multihoming erlaubt. Auf der Sicherungsschicht gibt es mehrere Technologien, die unter dem Begriff "Link Aggregation" geführt werden. Sie erlauben es, mehrere direkte Verbindungen zwischen zwei Geräten zu bündeln und so Redundanz und Geschwindigkeitsvorteile zu ermöglichen.
Von bisherigen Techniken konnte sich aber bis jetzt keine im Internet großflächig durchsetzen, weil z.B. keine Abwärtskompatibilität besteht. Die grundlegende Idee von MPTCP ist nun, TCP zu erweitern und damit neue Features, wie oben erwähntes Bündeln von Kanälen, bessere Stau-Kontrolle, Redundanz in der Verbindung zu einem Host etc. hinzuzufügen und sich dabei aber nahtlos in die vorhandene Struktur einzugliedern.

Die weiteren Abschnitte der Arbeit gliedern sich wie folgt:
Als nächstes wird in Kapitel 2 die Problemstellung herausgearbeitet und dabei auf die Struktur des Internets und die Rolle von TCP eingegangen. Auch werden Grenzen von TCP im Bezug auf Stau-Kontrolle und Multihoming aufgezeigt.
Abschnitt 3 gibt es einen Überblick über MPTCP und die Unterschiede zu TCP. Hier werden neben dem speziellen Gebiet der Pfadwahl und der Stau-Kontrolle ebenfalls die Vor- und Nachteile einer Transportschicht-Implementierung von transparenten Multilinks diskutiert. Das Kapitel schließt mit einer kritischen Diskussion über MPTCP hinsichtlich Design/Implementierung, wirtschaftlichen Aspekten und der großflächigen Inbetriebnahme im Internet ab.
Der Schluss (Kapitel 4) gibt es einen Ausblick auf die Zukunft von MPTCP und es wird erklärt, wie das Protokoll es schaffen könnte, die Struktur des Internets nachhaltig zu vereinfachen und zu verbessern.

2 Problemstellung

2.1 TCP und Multipath

Als TCP entwickelt wurde, existierten noch keine Technologien, die mehrere Pfade von einem Recher aus erlauben und deswegen gab es auch keinen Grund, Möglichkeiten für das Benutzen mehrerer Pfade in das Protokoll mit aufzunehmen[10]. Wie schon erwähnt ist TCP sehr verbreitet und die Technologien der Hosts haben sich auch weiter entwickelt.
In dem Artikel von Handley[6] wird beschrieben, dass der Aufwand für den Wechsel eines bereits etablierten Protokolls hoch ist, und dieser nur betrieben wird, wenn der Bedarf nach Erneuerung groß genug ist (vgl. DNS).
Es wurden viele Erweiterungen für TCP entworfen, z.B. Stau-Kontroll-Mechanismen (slow start, fast recovery und fast retransmit) [1] oder der Nagle-Algorithmus [9]. Jetzt ist aber ein Punkt erreicht an dem TCP einfach nicht mehr zeitgemäß ist: Viele Hosts haben heute mehrere Netzwerk-Schnittstellen - sei es aus Redundanzgründen oder, wie bei Mobil-Telefonen, wegen der verschiedenen Verbindungs-Technologien. Allerdings wurde dies bei der Entwicklung von TCP nicht mit bedacht. So kann im Moment nur eine Verbindung von einem Endpunkt zu einem anderen Punkt aufgebaut - nicht aber mehrere physische Verbindungen zu einer logischen zusammengefasst werden.
Aktuell behilft man sich mit properitären Protokollen auf anderen Ebenen, wie z.B. (split)-multi-link-trunks von Nortel oder das Dynamic Trunking Protocol von Cisco (beide auf Layer 2 des OSI-Modells). Dieses Vorgehen hat aber Nachteile, wenn Hardware von verschiedenen Herstellern kombiniert werden soll: Eine Kompatibilität ist oft nicht gegeben und die Verbesserungen sind nur lokal im eigenen Netzwerk - die Benutzer von WLAN und mobilem Internet mit einem Smartphone erleben weiterhin Verbindungsabbrüche, wenn der Kanal, über den die aktive Verbindung geht, gestört wird.

2.2 Stau-Kontrolle

Neben dem Problem mit Multilinks gibt es einen weiteren wichtigen Themenbereich - die Stau-Kontrolle oder auch Congestion-Control. Ein De-

finitionsversuch wäre: “Ein Netzwerk wird aus Sicht des Users als verstopft (congested) angesehen, wenn es eine spürbare Qualitätsminderung des Services aufgrund erhöhter Netzwerklast gibt” (frei übersetzt von [13])
Vor einigen Jahren war Bandbreite knapp und das Ziel war daher, die vorhandenen Kanäle möglichst wenig zu verstopfen. Mittlerweile hat sich das Problem ein wenig verlagert: Es ist viel günstiger schnellere Hardware und damit mehr Bandbreite zu kaufen als sich eine bessere Politik zur Stau-Vermeidung zu überlegen. Dabei wird eine neue, sehr ähnliche Frage aufgeworfen: Wie kann man die vohandenen Leitungen effektiv nutzen (und trotzdem Lastspitzen abfangen)?
Eine wichtige Rolle spielt hier die Transportschicht des OSI-Modells: Es gibt eine Vielzahl von Algorithmen, die es in Verbindung mit TCP ermöglichen, Staus zu vermeiden - eine grundlegende Konfiguration ist TCP Reno: Den Anfang bildet der *slow-start* - dabei wird mit einem sehr kleinen Sendefenster begonnen, welches sich dann bei jeder Runde exponentiell erhöht, bis eine Schwelle überschritten wird [1]. Dann tritt die *congestion-avoidance*-Phase ein, in welcher sich das Fenster nur noch linear erhöht. Wird nun irgendwann ein Verlust durch den Retransmission-Timer festgestellt, so halbiert sich die Schwelle und das Sendefenster wird wieder auf 1 Paket gesetzt. Damit fängt der Sender wieder mit Slow-Start an.
Ein weiterer Algorithmus ist *fast retransmit*. Bekommt TCP hier drei dublizierte ACKs, so wird das verloren gegangene Paket direkt losgeschickt ohne auf ein Timeout zu warten. Ebenfalls wie bei einem Timeout wird jedoch der Schwellwert reduziert. Dann startet *fast recovery*, was bedeutet, dass das Sendefenster, nicht wie bei *slow start* auf maximal 4 Pakete, sondern einen Wert knapp über dem Schwellwert gesetzt wird. Damit kann sich die Verbindung schneller wieder erholen, da das Fenster bereits deutlich größer ist als beim Start. Der Grund für dieses Vorgehen ist, dass die dublizierten ACKs signalisieren, dass weitere schon verschickte Pakete bereits das Ziel erreicht haben und so die Verbindung nicht komplett überlastet ist.

Eine mögliche Lösung für das Redundanz-Problem ist das Ressource-Pooling in der Transport-Schicht. Das bedeutet, dass mehrere Pfade zwischen zwei Endpunkten zusammengefasst werden können und für die Anwendung nur als ein Pfad sichtbar sind. Allerdings wird diese logische Verbindung, wie z.B. in 2.2 über mehrere Routen realisiert. ISP1 kann nun dem Fluss von

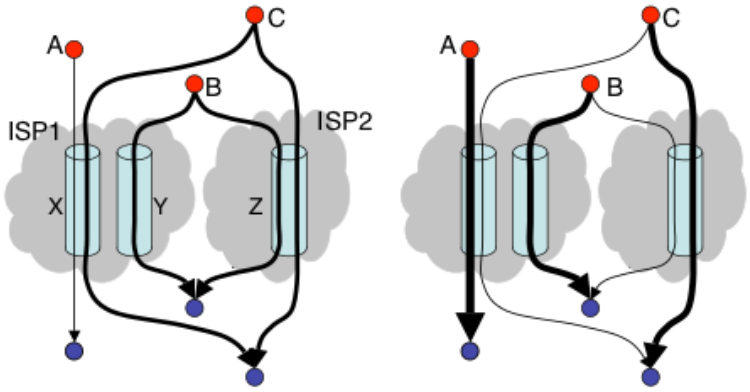

Abbildung 1: Stau-Kontrolle ISP-übergreifend mit Multilinks [14]

C über X bescheid geben, dass Route X überlastet ist. Damit kann auch der Kanal von A Daten schicken. C wird jetzt natürlich mehr Daten über Z schicken, wodurch B merkt, dass der Weg überlastet ist und dementsprechend auf Y ausweicht. Diese ISP-übergreifende Möglichkeit der Stau-Kontrolle gibt es mit den traditionellen Ansätzen nicht. Mit dem Ressource-Pooling kann also die Fragestellung von weiter oben geklärt werden, denn die gesamte Bandbreite wird ISP-übergreifend optimal ausgenutzt. Die Idee des Ressource-Poolings greift MPTCP auf.

3 MPTCP

3.1 Überblick

MPTCP ist eine tiefgehende Erweiterung von TCP. Während andere Erweiterungen nur das Verhalten von Hosts modifizieren, erlaubt MPTCP das Benutzen mehrere Pfade um Daten einer einzelnen Verbindung zu übertragen. Mit Hilfe des Ressource-Pooling-Prinzips [14] ist es MPTCP möglich, diese grundlegende Änderung in der Schnittstelle zur Anwendung zu verbergen und nach außen wie normales TCP zu wirken. Für Hosts mit Multihoming bietet es auch Redundanz: Fällt eine Netzwerk-Schnittstelle aus, so wird der Datenstrom über einen anderen Subflow geleitet, ohne, dass es zu einem Verbindungsabbruch kommt.

3.2 Anforderungen

Hiermit ergeben sich für die Entwicklung von MPTCP folgende Anforderungen:

- Mit am wichtigsten ist wohl die Möglichkeit, die **Inbetriebnahme** von MPTCP inkrementell durchzuführen um Interoperabilität zu gewährleisten.
- Der vorherige Punkt impliziert damit auch die **Abwärtskompatibilität** des Protokolls: MPTCP sollte sich wie normales TCP verhalten, wenn die Gegenstelle kein MPTCP spricht.
- Wünschenswert ist eine starke **Modularisierung**, sodass Teile, wie die Stau-Kontroll-Algorithmen oder die Pfadwahl, ohne Probleme ausgetauscht werden können.
- Jeder Fluss einer Ende-zu-Ende-Verbindung sollte von den anderen **unabhängig** sein - das bedeutet, dass es zu keinem Verbindungs-Abbruch kommen darf, wenn ein einzelner Fluss abbricht. (Es darf also insbesondere keinen "Master-Fluss" geben)

[3]

3.3 Funktionsweise

Verbindungs-Aufbau Wie bei TCP beginnt der Verbindungs-Aufbau mit dem Drei-Wege-Handschlag. Dabei wird bei jedem der drei Pakete die Option MP_CAPABLE mit verschickt. Im SYN-Paket schickt der Client seinen Schlüssel. Die Gegenstelle antwortet mit SYN/ACK und dem eigenen Schlüssel. Dann schickt der Client das ACK mit beiden Schlüsseln. Dieses vorgehen bietet dem Server die Möglichkeit, vor dem Empfangen des ACK mit beiden Schlüsseln, zustandslos zu bleiben [5]. Als letzten Schritt muss der Server den Erhalt des ACK mit MP_CAPABLE und beiden Schlüsseln mit einem ACK (ohne der MP_CAPABLE Option, dafür optional bereits mit Nutzdaten) bestätigen. Damit ist die Verbindung etabliert.

3.4 Pfadwahl

Mit der Option ADD_ADDRESS können der Gegenstelle weitere Adressen mitgeteilt werden, die zu dem entsprechenden Endpunkt gehören. Will eine Seite zu einer weiteren Adresse (oder zu der gleichen Adresse, aber zu einem anderen Port) einen weiteren Subflow aufbauen, so muss er ein Token mit versenden und kann dann eine Verbindung herstellen [3]. Aufgrund der Struktur der anderen Protokolle ist es MPTCP nicht möglich, sicherzustellen, dass die Pfade der Subflows genügend verschieden sind um eine Verbesserung gegenüber TCP zu erreichen, allerdings wird dies erwartet [4].

Hinzufügen eines Subflows Wurden die Adressen der Endpunkte ausgetauscht, können weitere Subflows initiiert werden. Dazu wird an eine weitere Adresse der Gegenstelle ein SYN-Paket mit der Option MP_JOIN und einem Hash des oben genannten Schlüssels (genannt Token) geschickt. So erkennt der Netzwerk-Stack der Gegenseite, dass ein neuer Subflow für eine bereits aktive Verbindung erstellt werden soll, ohne, auf einem Port lauschen zu müssen. Zusätzlich wird eine Zufallszahl mitgeschickt, die mit einem MAC (Message Authentication Code) die Integritäts- und Echtheits-Überprüfung der beiden Teilnehmer ermöglicht. Um den Verbindungsaufbau herzustellen ist, wie oben, ein Drei-Wege-Handschlag nötig. Bei MP_JOIN kann ausgewählt werden, ob der Pfad nur als Backup oder als aktiver Subflow benutzt werden soll [5].

Daten-Austausch Der Austausch von Daten geschieht auf sehr ähnliche Weise, wie bei TCP, der Unterschied ist, dass es jetzt 2 Ebenen gibt: Die Daten-Ebene, die für den ganzen Socket gilt und die Subflow-Ebene für jeden einzelnen Pfad zur Gegenstelle. Es wird ein verbindungs-globales Sende- und Empfangs-Fenster benutzt. Damit können Subflows, die Kapazität frei haben, senden, solange der Empfänger noch Speicher frei hat - es kommt nicht dazu, dass Subflows aufeinander warten müssen [5].

3.5 Stau-Kontrolle

Wichtig ist auch, wie MPTCP die Stau-Kontrolle händelt: Würden zwei TCP-Verbindungen (A und B) über eine Route verlaufen, so teilen sich beide die vorhandene Bandbreite 50:50. Wäre aber jetzt A vom Typ MPTCP, und hätte 9 Verbindungen über diese Route öffnet, dann würde jeder SubFlow sich wie eine normale TCP-Verbindung verhalten und das Ergebnis wäre, dass B nur ein Zehntel der Bandbreite bekommt, obwohl es nur 2 logische Verbindungen gibt und die Kapazität der Route daher wie vorher fair aufgeteilt werden sollte.
Die Autoren von [15] kommen nach theoretischen Überlegungen und Experimenten zu folgender Variante: Es wurden zwei Fairness-Ziele aufgestellt

- Der Datendurchsatz von MPTCP sollte mindestens genauso groß sein, wie von TCP auf dem besten von MPTCP benutzen Pfad.
- MPTCP sollte sich fair verhalten (höchstens so viel Bandbreite verbrauchen, wie ein normaler TCP-Stream), wenn es sich mit anderen TCP-Verbindungen eine gemeinsame Engstelle teilt. Gibt es keine Engstelle, so darf es mehr Bandbreite beanspruchen.

Damit wird erreicht, dass zum einen MPTCP überhaupt einen Vorteil gegenüber TCP hat und zum anderen wird es so eingeschränkt, dass es nicht alle TCP-Verbindungen verdrängt und damit Routen verstopft, die von Verbindungen genutzt werden, welche ihren Pfad nicht wechseln können.
Mit diesen Zielen lässt sich nun ein Algorithmus entwickeln.
Ein erster Ansatz war, dass der Datenstrom komplett auf den am wenigsten verstopften Pfad verlagert wird. Damit ist MPTCP aber auf diesem Pfad "gefangen", da jetzt keine Pakete mehr über die anderen Pfade geschickt werden und so nicht festgestellt werden kann, ob sich die Stau-Situation dort verbessert hat. Verschlechtert sich die Situation auf dem benutzten Weg, kann MPTCP nicht mehr zurück wechseln.
Ebenso ist es nicht effektiv, den Datenstrom gleichmäßig auf die verschiedenen Subflows zu verteilen - es gibt Szenarien, wo die Aufteilung dann dann weit von der optimalen Lösung entfernt ist.
Am Schluss ist eine Mischung aus beiden Ansätzen entstanden: Es wird

über jeden Subflow noch so viel Traffic geleitet, dass MPTCP erkennen kann, wenn Bandbreite in einem Kanal frei wird. Der am wenigsten verstopfte Pfad wird am meisten genutzt. Dabei muss die beschriebene Fairness zu TCP beachtet werden. Insgesamt ist die Stau-Kontrolle sehr ähnlich zu TCP New Reno [11].

3.6 Probleme mit MPTCP

Wie jede Veränderung bestehender Strukturen, muss auch MPTCP vielen verschiedenen Problemen begegnen. Sie reichen von wirtschaftlichen Aspekten über Implementations-Fragen bis hin zu Technischen Schwierigkeiten bei der Inbetriebnahme.

Wirtschaft Das Deployment kostet Geld. Endnutzer benötigen mindestens zwei separate Anschlüsse an das Internet um den Redundanz-Aspekt voll ausschöpfen zu können. Ebenso ist es gerade in der Anfangs-Phase von MPTCP mit hohem Aufwand (und evtl. auch Geld) für die User verbunden, die Software zu installieren.
Für ISPs es anders: Die primären Kosten sind nicht hoch, da die Netzwerk-Infrastruktur nicht verändert werden muss. Allerdings bringen verbesserte Redundanz der Verbindungen von Endusern weniger Geld für Support ein. Ebenso wird für den Kunden die Netzwerk-Effizienz transparenter und damit der Wettbewerb gesteigert. Möglicherweise sehen ISPs auch eine Gefahr in MPTCP für ihre bestehenden Trafic-Engineering-Politiken und versuchen deswegen eine Einführung zu verhindern [8].

Implementierung Bei der Referenz-Implementierung der Université catholique de Louvain für den Linux-Stack sind die Entwickler auf diverse Schwierigkeiten gestoßen. Es mussten viele Entscheidungen gefällt werden, z.B. wo Pakete für die Subflows geschedult werden: Sollte es Buffer für jeden Subflow geben oder eine globalen Warteschlange die von allen Subflows benutzt wird? Probleme mit der Gleitkomma-Arithmetik mussten gelöst werden: Linux erlaubt aus Performance-Gründen keine Gleitkomma-Operationen im Linux-Kernel. Also musste sich mit While-Schleifen, ex-

tra Zählern und anderen “Tricks” geholfen werden. Außerdem gibt es bei Multipath-Verbindungen Schwierigkeiten, die richtige Empfangs-Buffergröße zu wählen, da es vorkommen kann, dass der Puffer eines schnellen Subflows überläuft, weil Pakete auf einem langsamen Pfad noch nicht angekommen sind, aber noch vor den Daten des schnellen Stroms benötigt werden [2].

Inbetriebnahme Wird MPTCP im Internet eingesetzt, so muss sichergestellt werden, dass Middleboxes die Funktionalität nicht so beeinträchtigen, sodass keine Kommunikation mehr möglich ist. Dementsprechend musste bei der Spezifikation von MPTCP sehr genau darauf geachtet werden, dass es zu jedem Zeitpunkt im Verbindungs-Aufbau oder generell bei dem Benutzen von TCP-untypischen Optionen ein Fallback zu normaler TCP-Kommunikation gibt [11].

4 Zusammenfassung und Ausblick

Betrachtet man die heutige Struktur des Internets, so stellt man fest, dass das am meisten genutzte Transportschicht-Protokoll TCP veraltet ist. Es kann nur einen Pfad von einem Endpunkt zu einem anderen benutzen und hat auch Mängel in der Stau-Kontrolle. Ebenso fehlen Möglichkeiten zur Redundanz. Die Entwicklung geht in die Richtung, dass Redundanz auf anderen Schichten des OSI-Modells erreicht wird. Seitdem eine Arbeitsgruppe von der IETF für MPTCP eingerichtet wurde, hat sich viel getan: Es wurde ein Protokoll entwickelt, was sich abwärts-kompatibel verhält und viele Vorteile von anderen Protkoll-Erweiterungen von TCP vereinigt. Im Gegensatz zu SCTP [2] muss bei MPTCP keine Anwendung umgeschrieben werden um die Vorteile nutzen zu können.
Die Spezifikation ist mittlerweile fast komplett und es gibt eine rudimentär funktionierende Implementierung für Linux, Nokia-Handys und das Nexus One. Damit ist MPTCP auf einem sehr guten Weg.
Neben kleinen Experimenten, die bis jetzt durchgeführt wurden und erfolgreich waren, müssen größer angelegte Feldtests zeigen, ob sich MPTCP wirklich nahtlos in die bestehende Internet-Struktur einfügen lässt und ob es zu Geschwindigkeits-Verbesserungen gegenüber TCP kommt.

Ob MPTCP jemals großflächtig ausgerollt wird, ist fraglich, allerdings sind die Vorraussetzungen gut. Letztendlich wird es aber auch zum großen Teil an den Benutzern liegen, die genügend Interesse zeigen müssen - die ISPs und Netzwerk-Hardware-Hersteller haben möglicherweise kein Interesse daran, dass MPTCP Verbreitung findet [8]. Die Entwicklung des Internets geht aber in Richtung von Multihoming und Ressource-Pooling, weshalb MPTCP, gerade auch im Hinblick auf die einfachere und effizientere Stau-Kontrolle, TCP ablösen sollte.
In der Zukunft wird MPTCP auf jeden Fall ein Bestandteil in dem Projekt SKIMS sein, einem verteilten Immunsystem für Mobile Knoten. Hier soll es mit einer modifizierten Pfadauswahl auf Android-Geräten dafür sorgen, dass der Anwender ohne Verbindungs-Abbruch weiter arbeiten kann, wenn ein Pfad als unsicher eingestuft und damit abgeschaltet wird.

Literatur

[1] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.

[2] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. Multipath tcp: From theory to practice. In *IFIP Networking, Valencia*, May 2011.

[3] A. Ford C. Raiciu, M. Handley. Multipath tcp design decisions - working in progress. July 2009.

[4] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182 (Informational), March 2011.

[5] A. Ford, Roke Manor Research, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses, 2011.

[6] M Handley. Why the internet only just works. 2006.

[7] Wolfgang John and Sven Tafvelin. Analysis of internet backbone traffic and header anomalies observed. In *Proceedings of the 7th ACM*

SIGCOMM conference on Internet measurement, IMC '07, pages 111–116, New York, NY, USA, 2007. ACM.

[8] Tapio Levä, Henna Warma, Alan Ford, and Alexandros Kostopoulos. Business aspects of multipath tcp adoption. *Towards the Future Internet*, pages 21–30, 2010.

[9] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, January 1984.

[10] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093.

[11] C. Raiciu, M. Handley, and D. Wischik. Coupled Congestion Control for Multipath Transport Protocols, 2011.

[12] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFC 6096.

[13] Michael Welzl. *Network Congestion Control: Managing Internet Traffic (Wiley Series on Communications Networking & Distributed Systems)*. John Wiley & Sons, 2005.

[14] Damon Wischik, Mark Handley, and Marcelo Bagnulo Braun. The resource pooling principle. *ACM CCR*, 2008.

[15] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 8–8, Berkeley, CA, USA, 2011. USENIX Association.