Staukontrolle durch Active Queue Management

Thomas Fischer und Dominik Billing
Betreuer: Martin Metzker
Seminar Kommunikationssysteme im Sommersemester 2014
Lehrstuhl für Kommunikationssysteme und Systemprogrammierung
Ludwig-Maximilians-Universität München

***Zusammenfassung*—Bei konventioneller Staukontrolle in Routern werden Pakete nicht gleichmï¿½ï¿½ig fallengelassen, wodurch es zu einem groï¿½en Overhead kommt, der durch Flaschenhï¿½lse im Internet noch verstï¿½rkt wird. Zusï¿½tzlich kï¿½nnen Datenstrï¿½me, die besonders viele Daten ï¿½bertragen, nicht sinnvoll begrenzt werden. Wir werden als Lï¿½sung fï¿½r das Problem der Staukontrolle im Internet Active Queue Management herausarbeiten. Active Queue Management versucht Staus frï¿½hzeitig und zuverlï¿½ssig zu erkennen und durch Fallenlassen oder Markieren von Paketen (ECN – Explicit Congestion Notification) Staus zu verhindern. Dies wird von AQM-Algorithmen erreicht, indem die mittlere Pufferauslastung von Routern mï¿½glichst gering gehalten wird.**

Um einen ï¿½berblick ï¿½ber aktuelle AQM-Algorithmen zu erhalten, werden wir die AQM-Methoden RED (Random Early Detection), BLUE und AVQ (Adaptive Virtual Queue) vorstellen und untereinander vergleichen. Das Prinzip von RED ist, Pakete bereits vor dem ï¿½berlaufen der Queue mit einer bestimmten Wahrscheinlichkeit fallen zu lassen, um Staus vor dem Entstehen zu verhindern. Die Wahrscheinlichkeiten berechnen sich dabei mittels der aktuellen Queue Auslastung. Bei BLUE werden die ankommenden Pakete auch mit einer Wahrscheinlichkeit schon vorher fallen gelassen, diese berechnet sich mittels der verlorenen Pakete. Bei AVQ wird die Kapazitï¿½t einer virtuellen Queue als Entscheidung fï¿½r das Fallenlassen herangezogen.

***Schlagwï¿½rter*—Staukontrolle, AQM, ECN, RED, BLUE, AVQ**

I. Einfï¿½hrung und Motivation

Die Ende-zu-Ende (E2E) Staukontrollmechanismen von TCP sind mittlerweile ein kritischer Faktor der Robustheit des Internets. Das Internet wï¿½chst unaufhaltsam weiter, es gibt keine eng verknï¿½pfte Netzgemeinschaft und nicht jeder Endkonten verwendet die E2E Staukontrolle fï¿½r bestmï¿½glichen Datenfluss. Da auch Anwendungsentwickler sich nicht darum kï¿½mmern, E2E Staukontrolle in ihre Internet-Anwendungen zu integrieren, muss das Netz selbst seine Ressourcennutzung kontrollieren [1].

Mit der Entwicklung von immer leistungsfï¿½higeren Prozessoren, Arbeitsspeichern und Festplatten ist die knappste Ressource in Netzen aktuell die ï¿½bertragungsrate. Das bedeutet, dass nicht alle eingehenden Daten verarbeitet und weiter versendet werden kï¿½nnen. Wenn die Puffer eines Routers voll laufen, werden neu ankommende Pakete nach dem „Drop Tail“ Prinzip direkt abgewiesen (siehe Abbildung 1). Hier mï¿½ssen in Situationen hohen Datenaufkommens Pakete mehrmals versandt werden, was zu Lasten der Netzgeschwindigkeit geht. Das Transportschicht Protokoll TCP (Transmission Control Protocol) erkennt genau dann einen Stau, wenn Pakete im Fluss fehlen, also vorher schon fallen gelassen wurden [2].

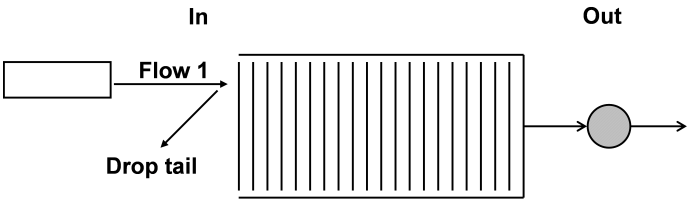

Abbildung 1. Das Prinzip von „Drop Tail“: Erreicht ein neues Paket ein System mit vollem Puffer, wird es fallengelassen [2].

Die Problemstellung ist es folglich Mechanismen zu finden, die Staus in E2E Verbindungen kontrollieren und frï¿½hzeitig identifizieren, damit Staus vermieden werden.

Um eine mï¿½glichst gute E2E Staukontrolle zu erreichen, soll die durchschnittliche Pufferausnutzung klein gehalten werden [3]. Die Herausforderung besteht darin, dass an den Flaschenhï¿½lsen der potenzielle Stau erkannt wird und das komplette Netz darauf reagiert, indem beispielsweise die Sendegeschwindigkeit des Ursprungs eines Datenstroms reduziert wird [2]. Interaktive Datenstrï¿½me wie beispielsweise telnet, Web-Browsen und der Transfer von Audio- und Video-Material kï¿½nnen sehr anfï¿½llig fï¿½r Paketverluste oder hohe Latenzen sein, die auftreten, wenn Pakete erneut verschickt werden mï¿½ssen [4].

Das folgende Kapitel II analysiert das Problem der Staukontrolle in Netzen und prï¿½sentiert Active Queue Management als Lï¿½sungsansatz fï¿½r das Problem. Zusï¿½tzlich wird mit Explicit Congestion Notification (ECN) eine Alternative zum einfachen Fallenlassen von Paketen vorgestellt. Anschlieï¿½end fï¿½hrt Kapitel III AQM ein und definiert Voraussetzungen eines funktionierenden Algorithmus. Die Active Queue Management Algorithmen RED, BLUE und AVQ werden in Kapitel IV vorgestellt und in Kapitel V miteinander verglichen. Im abschlieï¿½enden Kapitel VI erfolgt eine Zusammenfassung der Ergebnisse zusammen mit einem Ausblick auf zukï¿½nftige Entwicklungen und Forschungen sowie andere Ansï¿½tze zur Staukontrolle.

II. Staukontrolle in Netzen

Alle Staukontrollmechanismen mï¿½ssen sich mit denen von TCP, als dem wichtigsten Transportschicht Protokoll im Internet, gemessen an den Datenflï¿½ssen, vergleichen [5]. Neben [1] wird auch von Morris postuliert, dass die Effektivitï¿½t von TCP mit zunehmender Anzahl an konkurrierender

Datenstrï¿½me nachlï¿½sst [6]. Diese Probleme sind auch im Internet spï¿½rbar und es zeigt sich, dass das Internet einen gravierenden Performanceverlust dadurch erfï¿½hrt. Die einfachste Lï¿½sung fï¿½r diese Probleme wï¿½re die radikale Vergrï¿½ï¿½erung von Routerpuffern, zusammen mit einer Begrenzung der Anzahl der Pakete jedes einzelnen Datenstroms individuell. Da dies aber nicht ausfï¿½hrbar ist, mï¿½ssen Staukontrollalgorithmen implementiert werden, mit den folgenden Hauptzielen [6]:

- Es soll eine hohe Ausnutzung von Flaschenhï¿½lsen wie Routern erreicht werden.
- Der ï¿½berlauf von Flaschenhï¿½lsen und damit eine zeitliche Verzï¿½gerung durch Staus sowie ein hoher Paketverlust soll verhindert werden.
- Die zur Verfï¿½gung stehende ï¿½bertragungsrate soll gleichmï¿½ï¿½ig zwischen konkurrierenden Datenstrï¿½men aufgeteilt werden.

Um diese gleichmï¿½ï¿½ige Verteilung zu erreichen, ist TCP standardmï¿½ï¿½ig mit dem „Drop Tail“ Prinzip nicht geeignet. Deshalb mï¿½ssen Methoden entwickelt werden, die Puffer in Routern anders abarbeiten und dennoch einfach zu verwenden sind [7].

Die Internet Architektur nutzt grï¿½ï¿½tenteils zum Weiterleiten das verbindungslose IP-Protokoll, auf dem dann verbindungslose (UDP) oder verbindungsorientierte (TCP) Dienste der Transportschicht aufsetzen. Diese Architektur hat vor allem aufgrund des verbindungslosen Designs, der Flexibilitï¿½t und der Robustheit viele Vorteile, allerdings auch folgende Nachteile [8]:

- Sorgfï¿½ltiges Design ist von Nï¿½ten, um bei hoher Last einen guten Dienst zu leisten.
- Mangelnde Aufmerksamkeit auf die Dynamik der Paketweiterleitung kann dazu fï¿½hren, dass Dienste nicht mehr korrekt funktionieren und dass es schlimmstenfalls zu einem sogenannten „internet meltdown“ kommt.

Das Phï¿½nomen eines solchen Internet-Zusammenbruchs wurde wï¿½hrend der ersten Wachstumsphase des Internets in den 1980er Jahren festgestellt und wird „congestion collapse“ [9] genannt. Bereits 1986 wurden von Jacobson entwickelte Stauverhinderungsmechanismen fï¿½r Hosts entwickelt, die auch aktuell noch einen „congestion collapse“ verhindern [8].

Da das Internet seit dieser Zeit immer weiter wï¿½chst, wurde es offensichtlich, dass TCP Stauverhinderungsmechanismen [10] nicht unter allen Umstï¿½nden ausreichend gute Dienste leisten. Das Hauptproblem liegt darin, dass von den Enden der Netze nur bedingt Kontrolle ausgeï¿½bt werden kann. Deshalb mï¿½ssen auch in Routern Mechanismen angewendet werden, welche die Stauverhinderungsmechanismen der Endpunkte ergï¿½nzen. Hierbei muss man zwischen den folgenden zwei Klassen unterscheiden [8]:

Queue Management Algorithmen verwalten die Lï¿½nge von Paket-Puffern durch Fallenlassen von Paketen wenn nï¿½tig oder angemessen.

Scheduling Algorithmen legen fest, welche Pakete als nï¿½chstes gesendet werden sollen und kï¿½nnen primï¿½r dafï¿½r genutzt werden, die Zuweisung von ï¿½bertragungsraten zwischen den Datenstrï¿½men zu verwalten.

Ein guter Algorithmus vereint die Vorteile beider Arten in sich, dazu mehr in Kapitel III.

Nach Jain [11] gibt es zwei Grï¿½nde, warum das Problem der Staukontrolle in Netzen sehr schwierig ist. Erstens gibt es Voraussetzungen fï¿½r Staukontrollschemas, die es schwierig machen eine zufriedenstellende Lï¿½sung zu finden. Zweitens gibt es unzï¿½hlige Netzregeln, die das Design eines Stauschemas beeinflussen. Das sind die Grï¿½nde, warum ein Schema, das fï¿½r ein Netz entwickelt wurde, in einem anderen Netz nicht funktioniert. Grundbedingungen fï¿½r Schemata zur Staukontrolle sind:

- Ein Schema muss einen kleiner Overhead haben.
- Alle Datenstrï¿½me mï¿½ssen gleich behandelt werden.
- Es muss schnell auf andere Situationen reagiert werden kï¿½nnen.
- Es muss in schlechten Umgebungen funktionsfï¿½hig sein.
- Es muss fï¿½r alle Benutzer optimal sein.

Um E2E Staukontrolle in Routern zu betreiben, muss nicht nur jeder einzelne Router auf sich allein gestellt seinen Puffer ï¿½berwachen, sondern auch an die nï¿½chsten Router Informationen weiterleiten. Fï¿½r diese Benachrichtigung kï¿½nnten zwei reservierte Bits im IP-Header genutzt werden [2]. Mittels „Explicit Congestion Notification“ (ECN) sollen Pakete mit der Staubenachrichtigung weiterversandt werden im Gegensatz zum einfachen Fallenlassen der Pakete. Auf diese Art und Weise werden vorangehende Router darï¿½ber informiert, dass es zu einem Stau gekommen ist und mï¿½glicherweise einzelne Pakete doppelt versandt werden mï¿½ssen oder die Geschwindigkeit gedrosselt werden sollte. Router sind in der Vermittlungsschicht angesiedelt; es wird hier in der Regel das IP Protokoll verwendet, bei dem die IP-Adresse das Ziel eindeutig angibt. Im Gegensatz dazu befinden sich Switches in der Sicherungsschicht und leiten Pakete beispielsweise anhand der MAC-Adresse weiter. In kleinen Netzen ist es prinzipiell kein Problem Pakete einfach fallen zu lassen. Da im Internet E2E Verbindungen ï¿½ber sehr viele Router gehen kï¿½nnen, kann das Fallenlassen von Paketen durchaus problematisch werden. ECN ist ein zuverlï¿½ssiger Mechanismus, beinhaltet allerdings keine Methoden zur Erkennung von Staus [12]. Da es nicht vorhersehbar ist, wie hoch der Datenverkehr zukï¿½nftig sein wird, besteht das wirkliche Problem darin, Staus frï¿½hzeitig und zuverlï¿½ssig zu erkennen [13].

Als Lï¿½sung fï¿½r das Problem der Staukontrolle gehen wir auf „Active Queue Management“ (AQM) ein. AQM hat das Ziel Staus in Netzen rechtzeitig zu entdecken, bevor die Routerpuffer volllaufen. Um den Quellen von Datenstrï¿½men zu signalisieren, dass es zu Problemen gekommen ist oder kommen wird, gibt es im Prinzip die beiden Mï¿½glichkeiten Pakete fallen zu lassen oder diese zu markieren. Die erste Strategie erfordert es, dass die Endpunkte kooperieren und erzeugt durch erneuten Versandt der Pakete einen Overhead, der bezï¿½glich der verbrauchten ï¿½bertragungsrate messbar ist. Andererseits mï¿½ssen Router auf markierte Pakete reagieren, als wï¿½ren diese fallengelassen worden. Es wird hier aller-

dings kein weiterer Overhead erzeugt, da die Pakete nicht erneut gesendet werden mï¿½ssen. AQM-Algorithmen kï¿½nnen zusï¿½tzlich die ï¿½bertragungsrate von besonders gierigen Datenstrï¿½men reduzieren, indem deren Pakete hï¿½ufiger fallengelassen oder markiert werden als andere. Einen wirklichen Unterschied machen AQM-Algorithmen allerdings nur, wenn sie wirklich flï¿½chendeckend und der Situation entsprechend eingesetzt werden [2].

III. Definition und Anwendung von Active Queue Management

TCP ist ein E2E Protokoll, das zusammen mit dem Internet Protokoll (IP) genutzt wird, um Daten in Form von Paketen zwischen Computern ï¿½ber das Internet zu ï¿½bertragen. Wï¿½hrend IP fï¿½r die Vermittlung der Daten sorgt, gehï¿½ren Datensicherheit, Flusssteuerung und das Ergreifen von Maï¿½nahmen bei Datenverlust zu den Aufgaben von TCP. TCP implementiert einen Algorithmus zur Flusssteuerung, der „Sliding Window“ genannt wird. Das „Window“ ist die Anzahl an Bytes, die gesendet werden kï¿½nnen ohne auf Empfangsbestï¿½tigung warten zu mï¿½ssen. Der Algorithmus besteht aus folgenden Schritten [14], [15]:

1) Alle Bytes eines Windows verschicken.
2) Auf die Empfangsbestï¿½tigung warten (es kï¿½nnen mehrere Pakete auf einmal zurï¿½ck gemeldet werden).
3) Das Window bis an das Ende des lï¿½ckenlos bestï¿½tigten Bytestroms verschieben.
4) Wenn die Empfangsbestï¿½tigung eines Pakets nicht in einer angegebenen Zeit erfolgt, dann wird das Paket neu verschickt.

TCP nimmt hier keine Rï¿½cksicht auf Staus im Netz. Wenn kontinuierlich immer weiter Pakete an das Netz versandt werden, ohne dass diese vom Netz aufgenommen werden kï¿½nnen, fï¿½hrt dies zu einem Stau, der sich eventuell immer weiter aufbaut. An genau dieser Stelle muss ein Active Queue Management Algorithmus ansetzen und die Staukontrolle ï¿½bernehmen [16].

Alle Internetrouter besitzen Puffer, um auf Verzï¿½gerungen im Fluss kurzfristig reagieren zu kï¿½nnen und Daten eine kurze Zeit vorzuhalten. Die Grï¿½ï¿½e von Routerpuffern ist durch Hardware festgelegt, orientiert sich allerdings an der Dynamik des Staukontrollalgorithmus von TCP. Genauer gesagt ist es das Ziel einen Puffer nie leer laufen zu lassen. Die Grï¿½ï¿½e eines Puffers muss nach dieser Definition mit wachsender Verbindungsgeschwindigkeit ansteigen. Das Problem hierbei ist, dass auf diese Art groï¿½e und langsame Speicher verwendet werden mï¿½ssten. Durch die Anwendung von besseren Staukontrollalgorithmen wie AQM kï¿½nnen diese kostspieligen Puffer klein bleiben [17].

Die Definition von Active Queue Management lautet wie folgt:

Active Queue Management (AQM) ist das aktive Neusortieren oder Fallenlassen von Paketen innerhalb eines Puffers.

Damit sind AQM Algorithmen eine Mischung aus Scheduling und Queue Management Algorithmen, mit folgenden Zielen [18], [19]:

- Es sollen mï¿½glichst wenig Pakete fallengelassen werden.
- Daten aus einfachen Datenquellen wie beispielsweise telnet sollen mit einer sehr kurzen Verzï¿½gerung behandelt werden.
- Die ï¿½bertragungsrate soll gleichbehandelnd zwischen den unterschiedlichen Flï¿½ssen aufgeteilt werden.
- Staus sollen frï¿½hzeitig erkannt werden.
- Flï¿½sse aus nicht responsiven Quellen sollen derart behandelt werden, dass responsive Flï¿½sse davon nicht beeintrï¿½chtigt werden.
- Der Algorithmus soll einfach zu implementieren sein und schnell reagieren.

Das ist nur mï¿½glich, wenn sich alle Datenquellen untereinander derart koordinieren, dass die Queue-Grï¿½ï¿½e unter Kontrolle gehalten wird. Das Problem hierbei besteht darin, dass ï¿½bertragungsrate, Schnelligkeit und Pufferplatz unabhï¿½ngig von einander zugeordnet werden mï¿½ssen[20].

Selbst wenn AQM Mechanismen verwendet werden, ist es oft sehr sinnvoll zusï¿½tzlich auch ECN zu verwenden, um einen unnï¿½tigen Overhead zu vermeiden. Bei ECN wird durch das Markieren eines Pakets die Kommunikation zwischen einzelnen Routern im Netz ermï¿½glicht. Zusammen haben AQM und ECN das Potenzial den Effekt von Verlusten in latenz-sensitiven Flï¿½ssen zu reduzieren [4]. Aus diesem Grund hï¿½ngt der Erfolg jedes AQM Machanismus stark damit zusammen, ob er mit ECN verbunden werden kann und wie gut diese Verbindung erreicht wird [21].

Im Laufe der Zeit ist sehr viel Aufwand in unterschiedliche AQM-Methoden und deren Feintuning gesteckt worden. Viele dieser Arbeiten basieren auf Heuristiken und Simulationen und nicht auf einem systematischen Ansatz. Ihr gemeinsames Problem ist, dass jede vorgeschlagene Konfiguration nur fï¿½r ein spezielles Datenverkehrsaufkommen geeignet ist, aber nachteilige Effekte aufweist, wenn sie in einer anderen Umgebung angewendet wird. Entsprechend mï¿½ssen AQM-Methoden entwickelt und ihre Parameter so eingestellt werden, dass sie in einer Vielzahl von unterschiedlichen Umgebungen einsetzbar sind und zumindest keine schlechteren Ergebnisse liefern als TCP selbst [22]. Es existieren sehr viele unterschiedliche AQM-Methoden, die komplett unterschiedliche Ansï¿½tze verfolgen. Manche Algorithmen berechnen beispielsweise Wahrscheinlichkeiten, um Pakete fallen zu lassen, andere nutzen Verfahren aus der mathematischen Optimierung aus, um die bestmï¿½gliche Staukontrolle zu erreichen. Alle Algorithmen unterscheiden sich in der Anzahl an mï¿½glichen Einsatzgebieten, Komplexitï¿½t und Qualitï¿½t [21].

Im nï¿½chsten Kapitel stellen wir drei der bekanntesten Algorithmen vor.

IV. Drei Beispiele fï¿½r Active Queue Management Algorithmen

Seitdem die ersten Ideen fï¿½r Active Queue Management vorgestellt wurden und der Einfï¿½hrung von ECN in TCP/IP wurden viele verschiedene AQM Algorithmen entwickelt. Es wï¿½rde weit ï¿½ber den Rahmen dieser Arbeit hinausgehen, sie alle zu erwï¿½hnen, weshalb hier die gï¿½ngigsten drei

RED, BLUE und AVQ vorgestellt werden. Es wurden diese drei Algorithmen ausgewï¿½hlt, da zu diesen sehr viele Arbeiten existieren und sie bei vielen anderen Algorithmen als Referenz dienen.

A. RED: Random Early Detection

Der erste Algorithmus, der prï¿½sentiert wird, ist „Random Early Detection“ (RED). Er war einer der ersten AQM Algorithmen und viele andere Arbeiten entwickelten diesen weiter, z.B. in [23] oder [24]. Algorithmen, die anders ablaufen, werden oft mit RED verglichen.

Floyd und Van Jacobson haben RED 1993 vorgestellt [25]. In ihrer Arbeit wird die Funktionsweise des Algorithmus dargestellt. Um die Sender ï¿½ber einen Stau zu informieren kann RED entweder Pakete fallen lassen oder das ECN-Bit im Header setzen, je nachdem ob der Router ECN-fï¿½hig ist. Im Folgenden werden wir nur die Mï¿½glichkeit des Markierens (ECN-Bit Setzen) betrachten, was auf den eigentlichen Algorithmus keinerlei Auswirkungen hat.

Das Prinzip von RED ist es Pakete mit einer Wahrscheinlichkeit zu markieren, die sich proportional zum Anteil der ï¿½bertragungsrate verhï¿½lt, den diese Verbindung belegt. Auf diese Weise versucht der Algorithmus die Ressourcenzuteilung fair zu machen.

Als Messgrï¿½ï¿½e wird die durchschnittliche Queue Lï¿½nge benutzt. Die durchschnittliche Lï¿½nge Q_{avq} wird fï¿½r jedes eintreffende Paket mittels der aktuellen Lï¿½nge der Queue q und dem Gewicht der Queue w_q folgendermaï¿½en neu berechnet:

$$Q_{avg} = (1 - w_q)Q_{avg} + w_q q$$

Dieser Wert wird mit zwei Parametern verglichen, der minimalen Queuelï¿½nge Q_{min} und der maximalen Queuelï¿½nge Q_{max}. Ist $Q_{min} > Q_{avg}$, so wird nichts unternommen. Wird aber $Q_{min} < Q_{avg} < Q_{max}$, so wird das Paket mit einer Markierungswahrscheinlichkeit p_a markiert, und sobald $Q_{avg} > Q_{max}$ wird jedes Paket markiert.

Fï¿½r die Berechnung der finalen Markierungswahrscheinlichkeit p_a wird die Markierungswahrscheinlichkeit p_b benï¿½tigt. Diese berechnet sich aus der minimalen und maximalen Queuelï¿½nge Q_{min} und Q_{max}, der Durchschnittslï¿½nge Q_{avg} und dem Maximum fï¿½r p_b, max_b, wie folgt:

$$p_b = max_b \frac{Q_{avg} - Q_{min}}{Q_{max} - Q_{min}}$$

p_b steigt folglich linear von 0 bis zum Wert max_b an. Die finale Markierungswahrscheinlichkeit wird mittels p_b und eines Zï¿½hlers z berechnet:

$$p_a = \frac{p_b}{1 - zp_b}$$

Der Zï¿½hler wird fï¿½r jedes einkommende Paket inkrementiert. Ein Paket wird mit der Wahrscheinlichkeit p_a markiert. Sobald ein Paket tatsï¿½chlich markiert wird, wird der Zï¿½hler z wieder zurï¿½ck auf 0 gesetzt. Mithilfe des Zï¿½hlers steigt die Wahrscheinlichkeit somit fï¿½r jedes weitere Paket an.

for *jedes ankommende Paket* **do**
  Berechne Q_{avg} **if** $Q_{min} < Q_{avg} < Q_{max}$ **then**
    Berechne die Wahrscheinlichkeit $p_a = \frac{p_b}{1-zp_b}$;
      Mit der Wahrscheinlichkeit p_a: Markiere das ankommende Paket ;
  else
    if $Q_{max} < Q_{avg}$ **then**
      Markiere das ankommende Paket ;
    end
  end
end

Algorithmus 1 : RED

Damit ergibt sich als Algorithmus fï¿½r RED:

Neben der Queuelï¿½nge in Paketen kï¿½nnte RED auch die Bytelï¿½nge und somit die Anzahl an Bytes eines Pakets zur Bewertung heranziehen. Diese Information wird dann in die Markierungswahrscheinlichkeit p_b mit einbezogen. Nach der Berechnung ï¿½ndert sich der Wert dann in

$$p_b = p_b \frac{Paketbytes}{maximale\ Paketbytes}$$

Dadurch werden groï¿½e Pakete mit einer hï¿½heren Wahrscheinlichkeit markiert als kleine Pakete. An der Beschreibung des Algorithmus wird ersichtlich, dass RED viele Parameter benï¿½tigt, die vorab festgelegt werden mï¿½ssen. Fï¿½r jede Situation mï¿½ssen die richtigen Werte gefunden werden, damit RED gute Ergebnisse liefert. Das stellt ein nicht zu vernachlï¿½ssigendes Problem des Algorithmus dar. Mehr dazu in Kapitel V.

B. BLUE

Der BLUE Algorithmus wurde 1999 von Feng et.al. an der University of Michigan in Zusammenarbeit mit IBM vorgestellt [26]. BLUE wurde entwickelt, um einige Schwachstellen von RED zu verbessern, ist aber ein vï¿½llig neuer Ansatz. RED verlï¿½sst sich auf die Queuelï¿½nge, um Staus zu erkennen, und benï¿½tigt viele Parameter, die konfiguriert werden mï¿½ssen. Die Autoren von BLUE argumentieren, dass RED nur wenn diese richtig konfiguriert sind und wenn ausreichend Pufferplatz zur Verfï¿½gung steht optimal lï¿½uft. BLUE dagegen verlï¿½sst sich auf den Paketverlust und die Verbindungsauslastung um seine Markierungswahrscheinlichkeit zu berechnen. Genauso wie RED kann BLUE dann entweder Pakete fallen lassen oder mittels ECN markieren.

Der BLUE Algorithmus kennt nur eine Markierungswahrscheinlichkeit p_m; jedes einzureihende Paket wird mit dieser Wahrscheinlichkeit markiert. Die Entscheidung fï¿½r die Erhï¿½hung oder Erniedrigung dieser Wahrscheinlichkeit wird auf Basis der verlorenen Pakete beziehungsweise auf Basis der ungenutzten Verbindungen getroffen: Erhï¿½lt der Router die Information, dass ein Paket verloren gegangen ist, wird p_m um den Wert d_1 erhï¿½ht. Erkennt er eine ungenutzte Verbindung, wird p_m um d_2 reduziert. Ein weiterer Parameter ist hierbei noch wichtig: die *freeze_time*. Damit das Netz und die Sender Zeit haben, auf die Aktion des Routers zu reagieren, muss mindestens dieses Zeitintervall vergangen sein, bis die

Markierungswahrscheinlichkeit wieder geï¿½ndert wird. Formal lï¿½uft ein Schritt des Algorithmus folgendermaï¿½en ab:

for *jedes ankommende Paket* **do**
  if *Paketverlust* $\wedge$ *(now - last_update)* $<$ *freeze_time* **then**
    $p_m = p_m + d_1$;
    $last\_update = now$;
  end
  if *Verbindung frei* $\wedge$ *(now - last_update)* $<$ *freeze_time* **then**
    $p_m = p_m - d_2$;
    $last\_update = now$;
  end
end

Algorithmus 2 : BLUE

Die beiden Parameter d_1 und d_2 geben an, um wie viel p_m zu erhï¿½hen beziehungsweise zu reduzieren ist. d_1 sollte deutlich grï¿½ï¿½er sein als d_2, da BLUE somit auf Staus sehr viel schneller reagieren kann. Die Autoren geben weiterhin an, dass in ihren Experimenten die *freeze_time* konstant gehalten wurde. Sie sagen aber auch, dass in einem Netz dieser Parameter zufï¿½llig fï¿½r jeden Router gewï¿½hlt werden sollte, um globale Synchronisation zu vermeiden. Dieser Algorithmus passt sich somit selbstständig an den aktuellen Bedarf des Netzes an und benï¿½tigt keine Router-abhï¿½ngigen Parameter zur Konfiguration.

C. AVQ: Adaptive Virtual Queue

Ein weiterer AQM-Algorithmus, der eine andere Idee verfolgt, ist der Adaptive Virtual Queue (AVQ) Algorithmus. Er wurde 2001 von Kunniyur und Srikant vorgestellt [27]. Wie bereits der Name andeutet, verwendet der Algorithmus eine virtuelle Queue. Fï¿½r die Markierung beziehungsweise das Fallenlassen von Paketen wird die Kapazitï¿½t dieser virtuellen Queue und keine Markierungswahrscheinlichkeit zu Rate gezogen.

Bei AVQ verwaltet der Router neben der echten Queue eine virtuelle Queue mit der Kapazitï¿½t $C_v \leq C$, wobei C die Kapazitï¿½t der Verbindung und C_v die der virtuellen ist. Zu Beginn ist $C_v = C$. Bei jedem ankommenden Paket wird ï¿½berprï¿½ft, ob der Puffer der virtuellen Queue das Paket aufnehmen kï¿½nnte. Ist dem so, wird das echte Paket in die tatsï¿½chliche Queue eingereiht, ansonsten wird es markiert oder fallen gelassen. Die Kapazitï¿½t der virtuellen Queue wird ebenfalls bei jedem ankommenden Paket angepasst gemï¿½ï¿½ der Differentialgleichung (siehe auch Abbildung 2)

$$\dot{C}_v = \alpha(\gamma C - \lambda)$$

Hierbei ist α ein Glï¿½ttungsparameter, γ die angestrebte Auslastung der Verbindung und λ die Ankunftsrate der Verbindung. Das Markieren passiert auf diese Weise aggressiver, also hï¿½ufiger, wenn die Verbindung ihre gewï¿½nschte ï¿½bertragungsrate ï¿½berschreitet und weniger aggressiv, wenn nicht.

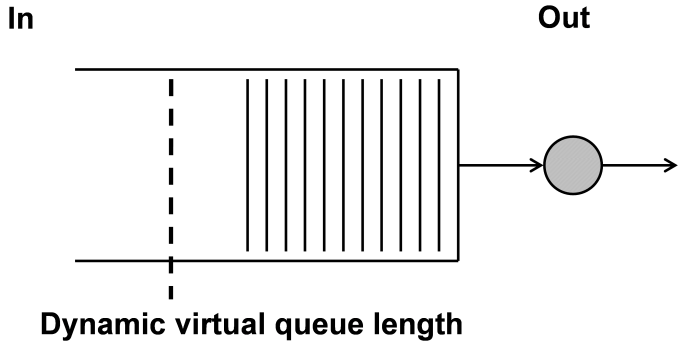

Abbildung 2. Das Prinzip von *AVQ*: Die Grï¿½ï¿½e der virtuellen Queue wird dynamisch angepasst, um bessere Leistungsgrenzen zu erhalten [2].

Es ist klar, dass in der virtuellen Queue keine Pakete eingereiht werden mï¿½ssen, lediglich die Lï¿½nge, also die Kapazitï¿½t der virtuellen Queue muss ermittelt werden. Die Autoren geben in ihrer Arbeit auch an, wie dieses Verfahren implementiert wird:

for *jedes ankommende Paket* **do**
  if $VQ = max(VQ - C_v(t-s), 0)$ **then**
    Paket markieren ;
  else
    $VQ = VQ + b$;
  end
  $C_v = max(min(C_v + \alpha \cdot \gamma \cdot C(t-s), C) - \alpha \cdot b, 0)$;
  $s = t$;
end

Algorithmus 3 : AVQ

Hierbei ist B die Puffergrï¿½ï¿½e in Bytes, s die Ankunftszeit des letzten Pakets, t die aktuelle Zeit, b die Paketgrï¿½ï¿½e des aktuellen Pakets in Bytes und VQ die Anzahl an Bytes, die aktuell in der virtuellen Queue sind.

Die Autoren erlï¿½utern, dass die algorithmische Komplexitï¿½t von AVQ in etwa der von RED entspricht. Anstelle der Lï¿½nge der Queue wird bei AVQ jedoch die Ausnutzung der Queue als Entscheidungskriterium fï¿½r das Markieren von Paketen verwendet. Fï¿½r die Verwendung von AVQ mï¿½ssen der Glï¿½ttungsparameter α und die gewï¿½nschte Ausnutzung γ vorab angegeben werden. Diese dienen der Stabilitï¿½t des Verfahrens. Weiter geben die Autoren an, dass γ es den ISP's erlaubt einen Ausgleich zwischen hoher ï¿½bertragungsratenausnutzung und kleinen Puffern vorzunehmen. Eine Regel, wie diese Parameter gesetzt werden sollten, befindet sich in der Originalarbeit in [27].

V. Vergleich der vorgestellten Algorithmen

Nachdem wir im vorigen Kapitel IV die AQM Algorithmen RED, BLUE und AVQ vorgestellt haben, werden wir anschlieï¿½end diese untereinander vergleichen.

In der zur Verfï¿½gung stehenden Literatur lies sich kein Vergleich zwischen den Algorithmen BLUE und AVQ finden. Deshalb kann auch an dieser Stelle kein solcher Vergleich vorgenommen werden.

A. Vergleich BLUE und RED

In ihrer Arbeit zeigen Feng et al anhand einiger Versuche, dass BLUE im Vergleich zu RED deutlich weniger Pakete fallen lï¿½sst [26]. Fiï¿½r ihren Versuch haben sie ein Netz mithilfe des LBNL Network Simulators [28] simuliert. Fiï¿½r alle Quellen wurde ECN aktiviert, was bedeutet, dass jedes verlorene Paket einen Pufferiï¿½berlauf der Queue darstellt. Die Sender wurden zufiï¿½llig innerhalb der ersten Sekunde gestartet. Die Paketverluste wurden nach 100 Sekunden Simulation und weiteren 100 Sekunden, also insgesamt 200 Sekunden, fiï¿½r die gesamte Simulationszeit gemessen. Die Parameter fiï¿½r RED wurden experimentell bestimmt, Q_{min} betrug immer 20% der Queueliï¿½nge, Q_{max} 80%. Fiï¿½r BLUE wurde d_1 um eine Griï¿½iï¿½enordnung hiï¿½her als d_2 gesetzt. Insgesamt wurden vier Konfigurationen fiï¿½r RED und vier Konfigurationen fiï¿½r den BLUE Algorithmus angewandt und die Ergebnisse verglichen.

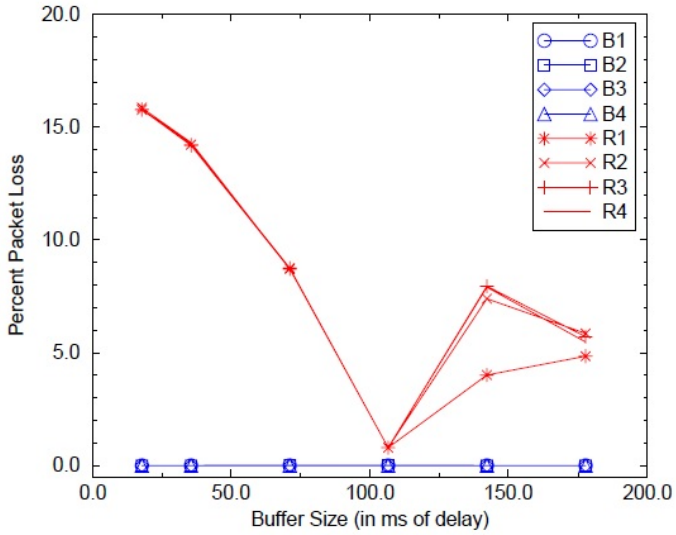

Abbildung 3. Vergleich der Verlustraten von RED und BLUE bei 1000 Quellen [26].

Zwischen den Ergebnissen der einzelnen Konfigurationen sowohl bei RED als auch bei BLUE gab es nur minimale Unterschiede. Die Queueliï¿½nge fiï¿½r eine Engstelle des Netzes wurde von 100KB bis 1000KB gesetzt, was einer Verziï¿½gerung zwischen 17,8ms und 178ms entspricht. Bei 1000 Quellen, die gleichzeitig senden, war sowohl bei RED als auch bei BLUE die Verbindung fiï¿½r alle Queueliï¿½ngen zu 100% ausgelastet. Die Paketverlustraten sind in Abbildung 3 dargestellt. Bei BLUE betrug sie fiï¿½r alle Verziï¿½gerungen 0%. Der RED Algorithmus dagegen zeigt bei niedrigen Verziï¿½gerungen eine Paketverlusterate von bis zu iï¿½ber 15%. Diese fiï¿½llt mit griï¿½iï¿½er werdenden Puffern auf knapp iï¿½ber 0% bei einer Verziï¿½gerung von etwa 100ms, steigt dann aber wieder an auf etwa 5%. Fiï¿½r das gleiche Experiment mit 4000 Quellen war die Auslastung ebenfalls bei beiden stets bei 100%. Die Paketverlusterate von RED lag diesmal bei kleinen Verziï¿½gerungen bei iï¿½ber 30%, was mit zunehmenden Queueliï¿½ngen auf etwa 25% abfiel. Bei BLUE lag die Verlustrate dagegen zu Beginn bei etwa 15%

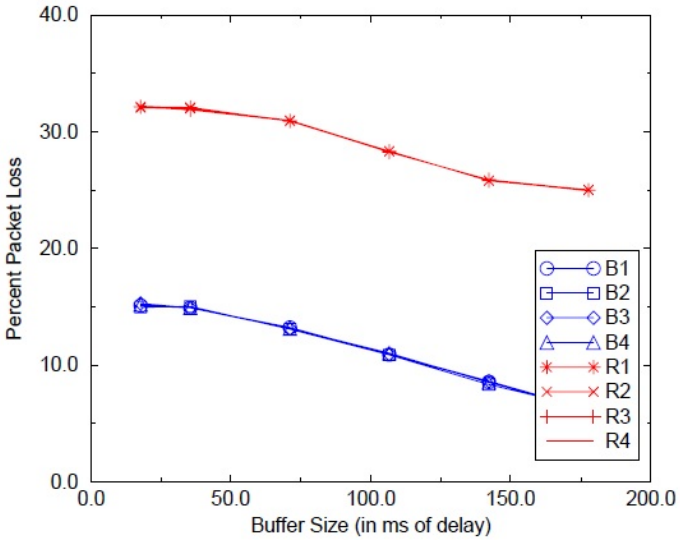

Abbildung 4. Vergleich der Verlustraten von RED und BLUE bei 4000 Quellen [26].

und sank auf unter 10% mit griï¿½iï¿½er werdenden Queues (siehe Abbildung 4).

B. Vergleich AVQ und RED

Analog zu BLUE hat auch Kunniyar in seiner Arbeit zu AVQ Vergleiche des Verfahrens zu anderen AQM Algorithmen gemacht [27]. Neben RED wurde der Algorithmus auch mit Random Early Marking (REM), dem PI Controller und GKVQ verglichen. Diese werden hier jedoch vernachlïï¿½ssigt, da auch die Algorithmen nicht priï¿½sentiert worden sind, allerdings wurde unter den vorgegebenen Randparametern mit AVQ ein besseres Ergebnis als mit allen anderen Algorithmen erzielt.

Die beschriebenen Versuche wurden mit dem Simulator *ns-2* durchgefiï¿½hrt. Der Parameter γ, die angestrebte Auslastung, wurde fiï¿½r AVQ auf den Wert 0,98 gesetzt; der Gliï¿½ttungsparameter α wurde auf 0,15 gesetzt. Die Konfiguration fiï¿½r RED wurde gemï¿½iï¿½ den Empfehlungen aus [29] vorgenommen. Die Queueliï¿½nge an der engsten und somit relevanten Stelle der Verbindung betrug 100 Pakete bei einer Paketgriï¿½iï¿½e von 1000 bytes. Fiï¿½r das erste Experiment wurde bei den TCP Verbindungen ECN aktiviert. Jedes verlorene Paket ist folglich ein Zeichen des Pufferiï¿½berlaufs. Die Grenzwerte fiï¿½r RED wurden auf $Q_{min} = 0,37$ und $Q_{max} = 0,75$ gesetzt. Die durchschnittliche Verziï¿½gerung betrug zwischen 30ms und 60ms. Beim Experiment wurde die Paketverlustrate sowie die Auslastung der Verbindung bei unterschiedlicher Anzahl von FTP Verbindungen gemessen, die Anzahl der Verbindungen lag zwischen 20 und 180.

Die Auslastung der Verbindung lag bei RED bei 20 FTP Verbindungen bei knapp unter 90%. Mit steigender Anzahl an Verbindungen sank diese leicht ab, bewegte sich aber stets zwischen 85% und 90%. Fiï¿½r AVQ lag die Auslastung bei 20 Verbindungen bei 95%. Diese stieg kontinuierlich mit steigenden Verbindungen an und lag bei 180 Verbindungen etwa bei 98%, der vorab festgelegten angestrebten Auslastung. Die Paketverluste von RED lagen bei 20 Verbindungen etwa bei 0.

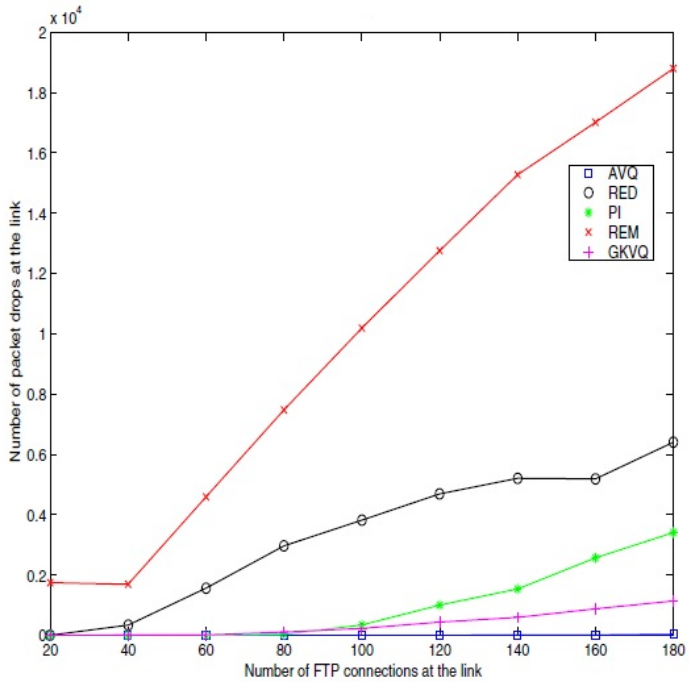

Abbildung 5. Absolute Anzahl an Paketverlusten von verschiedenen AQM Verfahren bei steigender Anzahl an FTP-Verbindungen [27].

Mit steigenden FTP Verbindungen stiegen diese in etwa linear auf ï¿½ber 6.000 verlorene Pakete bei 180 Verbindungen (siehe Abbildung 5, schwarz). Bei AVQ lagen die Paketverluste fï¿½r jede Anzahl an Verbindungen bei 0 (siehe Abbildung 5, blau). Der AVQ Algorithmus zeigt fï¿½r diesen Aufbau eine deutlich besser Leistung mit hï¿½herer Auslastung und ohne Paketverluste.

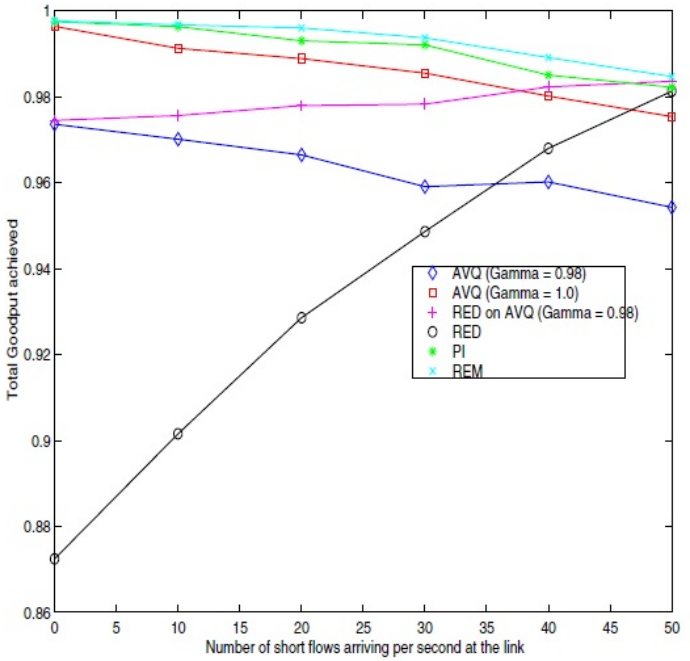

Abbildung 6. Anteil korrekt zugestellter Pakete von verschiedenen AQM Verfahren bei steigender Anzahl an short flows [27].

In einem zweiten Experiment wurde ECN deaktiviert. Das heiï¿½t, dass sowohl RED als auch AVQ Pakete fallen lassen, um auf einen Stau hinzuweisen. Der Versuchsaufbau ist gleich dem vorherigen mit der Ausnahme, dass fï¿½r AVQ die angestrebte Auslastung γ auf den Wert 1 gesetzt wurde, die Adaptive Virtual Queue also voll genutzt werden soll. Die Anzahl an FTP Verbindungen wurde fest auf 40 gesetzt; dafï¿½r wurden TCP short-flows mit 20 Paketen verwendet, deren Ankunftszeit an der kritischen Verbindungsstelle langsam gesteigert wurde. Die Auslastung der Verbindung fï¿½r AVQ lag konstant bei den angestrebten 100%. Bei RED lag diese bei 10 ankommenden short-flows pro Sekunde bei etwa 94% und stieg auf ï¿½ber 99% bei 50 ankommenden short-flows pro Sekunde. Der Anteil an korrekt zugestellten Paketen, ohne diese einmal fallen zu lassen, lag bei AVQ bei wenigen short-flows bei knapp unter 100% und sank bis zu knapp unter 98% bei 50 ankommenden short-flows pro Sekunde (siehe Abbildung 6, rot). Fï¿½r RED lag diese Quote anfangs nur bei 87%, stieg dann aber stark an auf 98% bei 50 short-flows pro Sekunde und lag damit hï¿½her als bei AVQ (siehe Abbildung 6, schwarz).

C. Zusammenfassung

Zu Beginn dieser Arbeit wurde die Notwendigkeit fï¿½r AQM Algorithmen erlï¿½utert. Sie sind notwendig, um aktiv einen Pufferï¿½berlauf der Router in groï¿½en Netzen zu verhindern. RED war einer der ersten Algorithmen, der dies zum Ziel hatte. Bereits 1998 wurde von der Network Working Group in RFC 2309 empfohlen, einen AQM Mechanismus zu implementieren, wobei explizit RED als Verfahren vorgeschlagen wurde [8]. Ebenfalls wurde empfohlen, weiter Forschung in diesem Bereich zu betreiben. Hervorzuheben ist auch, dass die Empfehlung fï¿½r RED vor der Empfehlung fï¿½r ECN ausgegeben wurde, welche 1999 in RFC 2481 gegeben wurde [30].

Aus dieser Forschung sind zahlreiche weitere AQM Algorithmen hervorgegangen. Hier wurden davon die Verfahren BLUE und AVQ vorgestellt und mit RED verglichen. Die Autoren von BLUE zeigen mit ihren Versuchen, dass BLUE eine deutlich geringere Paketverlustrate als RED aufweiï¿½t und besonders mit kleinen Puffern noch eine gute Performance zeigt. In den Experimenten zur Leistung von AVQ wurde gezeigt, dass dieses Verfahren eine bessere Auslastung der Verbindung und geringere Paketverluste als RED aufweist. Es ist jedoch zu beachten, dass diese Versuche in kontrollierten Umgebungen durchgefï¿½hrt wurden mit dem Ziel zu zeigen, dass das jeweilige Verfahren das beste ist.

VI. Ausblick und andere Ansï¿½tze

Neben dem hier vorgestellten Ansatz des Active Queue Managements gibt es auch andere Verfahren, die zum Ziel haben, ï¿½berlastungen in Netzen zu vermeiden. In [14] werden hierzu zum Beispiel Zugangssteuerung und Routing unter Verkehrsberï¿½cksichtigung genannt. Bei der Zugangssteuerung werden nur dann neue, virtuelle Verbindungen aufgebaut, wenn das Netz diese auch verkraften kann. Beim Routing unter Verkehrsberï¿½cksichtigung werden Algorithmen angewandt,

die im Netz nach mï¿½glichst wenig ausgelasteten Pfaden suchen. Diese Ansï¿½tze helfen somit auch, die Queues der Router zu entlasten.

Ein weiterer Ansatz, der den AQM Verfahren ï¿½hnlich ist, wird auch in [14] erwï¿½hnt: sogenannte Drosselpakete. Hier drosselt der Router den Verkehr nicht, indem er mit den Sendern durch das Markieren oder Fallenlassen von Paketen kommuniziert, sondern er gibt diese Nachricht ï¿½ber die anderen Router an den Sender zurï¿½ck. Wenn der Router ausgelastet ist, schickt er ein Drosselpaket an den Router, von dem er das letzte Paket erhalten hat. Dieser reagiert sofort darauf, indem er weniger Pakete an diesen Router weiterleitet und seinerseits ein Drosselpaket an den Router vor ihm schickt. So kommt die Nachricht schneller beim Sender an und der Router wird schneller entlastet.

AQM Algorithmen sind ein aktuelles Thema der Forschung. Auch in naher Zukunft wird es wohl noch weitere neue Algorithmen geben, die bessere Eigenschaften als die bisherigen aufweisen. Aktuell wichtiger fï¿½r die Nutzer wï¿½re jedoch, dass AQM Algorithmen auch wirklich auf allen Routern implementiert werden. Wie in dieser Arbeit gezeigt wï¿½re das eine Verbesserung hin zur bestmï¿½glichen Ausnutzung der heute angebotenen Ressourcen.

LITERATUR

[1] S. Floyd und K. Fall, "Router mechanisms to support end-to-end congestion control," Lawrence Berkeley National Laboratory, Berkeley CA, Tech. Rep., 1997.

[2] K. Graffi, K. Pussep, N. Liebau, und R. Steinmetz, "Taxonomy of active queue management strategies in context of peer-to-peer scenarios," Technische Universität Darmstadt, Tech. Rep., 2007.

[3] L. Le, J. Aikat, K. Jeffay, und F. Smith, "The effects of active queue management on web performance," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Ser. SIGCOMM '03. ACM, 2003.

[4] K. Ramakrishnan, S. Floyd, und D. Black, "The addition of explicit congestion notification (ecn) to ip," United States, 2001.

[5] J. Crowcroft und P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing tcp," *SIGCOMM Comput. Commun. Rev.*, vol. 28, Nr. 3, 1998.

[6] R. Morris, "Tcp behavior with many flows," in *Proceedings of the 1997 International Conference on Network Protocols (ICNP '97)*, Ser. ICNP '97. IEEE Computer Society, 1997.

[7] B. Suter, T. Lakshman, D. Stiliadis, und A. Choudhury, "Design considerations for supporting tcp with per-flow queueing," in *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, 1998.

[8] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, und L. Zhang, "Recommendations on queue management and congestion avoidance in the internet," United States, 1998.

[9] J. Nagle, "Congestion control in ip/tcp internetworks," *SIGCOMM Comput. Commun. Rev.*, vol. 14, Nr. 4, 1984.

[10] W. Stevens, "Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," United States, 1997.

[11] R. Jain, "Congestion control in computer networks: issues and trends," *Network, IEEE*, vol. 4, Nr. 3, 1990.

[12] S. Floyd, "Tcp and explicit congestion notification," *SIGCOMM Comput. Commun. Rev.*, vol. 24, Nr. 5, 1994.

[13] R. Jain, "Congestion control and traffic management in atm networks: Recent advances and a survey," *Comput. Netw. ISDN Syst.*, vol. 28, Nr. 13, 1996.

[14] A. Tannenbaum und D. Wetherall, *Computernetzwerke*. Pearson Deutschland GmbH, 2012.

[15] T. Socolofsky und C. Kale, "Tcp/ip tutorial," RFC 1180 (Informational), Internet Engineering Task Force, 1991.

[16] S. Oldak, W. Gong, C. Hollot, D. Towsley, V. Misra, und Y. Chait, "Active queue management for differentiated services," 2006.

[17] G. Appenzeller, I. Keslassy, und N. McKeown, "Sizing router buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, Nr. 4, 2004.

[18] B. Suter, T. Lakshman, D. Stiliadis, und A. Choudhury, "Efficient active queue management for internet routers," in *Proceedings of INTEROP, Engineering Conference*, 1998.

[19] R. Pan, B. Prabhakar, und K. Psounis, "Active queue management toward fair bandwidth allocation," 2008.

[20] A. Demers, S. Keshav, und S. Shenker, "Analysis and simulation of a fair queueing algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 19, Nr. 4, 1989.

[21] C. Hollot, V. Misra, D. Towsley, und W.-B. Gong, "On designing improved controllers for aqm routers supporting tcp flows," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2001.

[22] V. Firoiu und M. Borden, "A study of active queue management for congestion control," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2000.

[23] S. Floyd, R. Gummadi, und S. Shenker, "Adaptive red: An algorithm for increasing the robustness of red's active queue management," AT&T Center for Internet Research at ICSI, Tech. Rep., 2001.

[24] R. Pan, B. Prabhakar, und K. Psounis, "Choke - a stateless active queue management scheme for approximating fair bandwidth allocation," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2000.

[25] S. Floyd und V. Jacobson, "Random early detection gateways for congestion avoidance," *Networking, IEEE/ACM Transactions on*, vol. 1, Nr. 4, 1993.

[26] W. Feng, K. Shin, D. Kandlur, und D. Saha, "Blue: A new class of active queue management algorithms," 1999.

[27] S. Kunniyur und R. Srikant, "Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management," *SIGCOMM Comput. Commun. Rev.*, vol. 31, Nr. 4, 2001.

[28] S. McCanne, S. Floayd, und K. Fall. ns version 1 - lbnl network simulator. [Online]. Available: http://ee.lbl.gov/ns/

[29] S. Floyd. (1997) Red: Discussions of setting parameters. [Online]. Available: http://www.aciri.org/floyd/REDparameters.txt

[30] K. Ramakrishnan, A. L. Research, und S. Floyd, "A proposal to add explicit congestion notification (ecn) to ip," United States, 1999.