

# Staukontrolle durch Active Queue Management

Thomas Fischer, Dominik Billing

*Masterseminar Kommunikationssysteme*

*Lehr- und Forschungseinheit für Kommunikationssysteme und Systemprogrammierung*

*Ludwig-Maximilians-Universität München*

**Zusammenfassung**—Dieser Artikel beschreibt die Problematik im Internet, die durch den Einsatz von konventioneller Staukontrolle in Routern und der Struktur des Internets hervorgerufen wird. Diese liegt darin, dass Pakete nicht gleichmäßig fallen gelassen werden und es auf diese Art zu großem Overhead kommt, der durch Flaschenhälse im Internet noch verstärkt wird. Zusätzlich können besonders bandbreitenhungrige Datenströme nicht sinnvoll begrenzt werden.

Wir werden als Lösung für das Problem der Staukontrolle im Internet Active Queue Management herausarbeiten. Active Queue Management versucht Staus frühzeitig und zuverlässig zu erkennen und durch Fallenlassen oder Markieren von Paketen Staus zu verhindern. Zusätzlich werden alle Datenströme gleich behandelt. Dies wird von AQM-Algorithmen erreicht, indem die mittlere Pufferauslastung von Routern möglichst gering gehalten wird.

Um einen Überblick über aktuelle AQM-Algorithmen zu erhalten, werden wir die AQM-Methoden RED, BLUE und PI vorstellen und mit untereinander und mit ECN vergleichen.

## I. EINFÜHRUNG UND MOTIVATION

Nach Floyd [1] sind Ende-zu-Ende (E2E) Staukontrollmechanismen von TCP mittlerweile ein kritischer Faktor in der Robustheit des Internets. Das Internet wächst unaufhaltsam weiter, es gibt keine eng verknüpfte Netzgemeinschaft mehr und nicht jeder Endknoten verwendet die E2E Staukontrolle für bestmöglichen Datenfluss. Entwickler kümmern sich nicht länger darum, E2E Staukontrolle in ihre Internet-Anwendungen zu integrieren. Die Konsequenz davon ist, dass das Netz selbst seine Ressourcennutzung kontrollieren muss.

Graffi [2] beschreibt, dass Bandbreite aktuell die knappste Ressource in Netzen ist. Die eingehenden und ausgehenden Bandbreiten normaler ADSL Verbindungen sind unterschiedlich. Typischerweise ist die ausgehende Bandbreite deutlich kleiner als die eingehende. Das bedeutet, dass nicht alle eingehenden Daten verarbeitet und weitergesandt werden können. Staus sind also Probleme, die hier auftreten können, wenn keine angemessenen Mechanismen angewendet werden. Das TCP-Protokoll sieht vor, Pakete nach dem "First in First out"-Prinzip fallenzulassen, wenn die Puffer voll laufen.

Die Problemstellung hier ist es also Mechanismen zu finden, Staus in E2E Verbindungen zu kontrollieren. Nach Le [3] soll die durchschnittliche Pufferauslastung klein gehalten werden, damit E2E-Staukontrolle ermöglicht wird. Die Herausforderung besteht darin, dass an den Flaschenhälsen der potenzielle Stau erkannt wird und das komplette Netz darauf reagiert, indem beispielsweise die Sendegeschwindigkeit des Ursprungs eines Datenstroms drastisch reduziert wird [2]. Viele Applikationen wie beispielsweise telnet, Web-Browser und der Transfer von Audio- und Video-Material sind sehr anfällig für

Paketverluste oder hohe Latenzen, die auftreten, wenn Pakete erneut verschickt werden müssen [4].

Das folgende Kapitel (Kapitel II) beschreibt die generellen Konzepte der Staukontrolle in Netzen und präsentiert Active Queue Management als Lösungsansatz für das Problem. In Kapitel III wird Active Queue Management definiert und einige Anwendungsfälle davon präsentiert. Die Active Queue Management Algorithmen RED, BLUE, ECN und PI werden in Kapitel IV vorgestellt und in Kapitel V miteinander verglichen. Ein Ausblick zu zukünftigen Entwicklungen und Forschungen sowie andere Ansätze zur Staukontrolle zusammen mit den zusammengefassten Ergebnissen erfolgt in Kapitel VI.

## II. STAUKONTROLLE IN NETZEN

Neben [1] wird auch von Morris [5] postuliert, dass die Effektivität von TCP mit zunehmender Anzahl an konkurrierenden Datenströme nachlässt. Dieser Effekt beginnt sich zu zeigen, sobald mehr Datenströme als Pakete die vorhandene Bandbreite verwenden. Diese Probleme sind auch im Internet spürbar und es zeigt sich, dass das Internet einen gravierenden Performanceverlust dadurch erfährt. Die einfachste Lösung für diese Probleme wäre die radikale Vergrößerung von Routerpuffern zusammen mit einer Begrenzung der Anzahl der Pakete jedes einzelnen Datenstroms individuell. Da dies aber nicht ausführbar ist, müssen Staukontrollalgorithmen implementiert werden, mit den folgenden Hauptzielen. Es soll eine hohe Ausnutzung von Flaschenhälsen wie Routern erreicht werden. Der Überlauf von Flaschenhälsen und damit eine zeitliche Verzögerung durch Staus sowie ein hoher Paketverlust soll verhindert werden. Zusätzlich soll die zur Verfügung stehende Bandbreite gleichmäßig zwischen konkurrierenden Datenströmen aufgeteilt werden [5]. Um diese gleichmäßige Verteilung zu erreichen, ist TCP standardmäßig mit dem „First Come First Serve“-Prinzip nicht geeignet. Deshalb müssen hierfür andere Methoden entwickelt werden, die den Puffer in Routern anders abarbeiten und dennoch einfach zu verwenden sind [6].

Die Internet Protokoll (IP) Architektur basiert auf einem verbindungslosen E2E Paketdienst, der TCP benutzt. Die vielen Vorteile dieses verbindungslosen Designs, der Flexibilität und Robustheit wurden schon oft beschrieben. Allerdings kommen diese Vorteile zu einem Preis. Sorgfältiges Design ist benötigt, um einen guten Dienst zu leisten bei hoher Last. Fehlende Aufmerksamkeit bei den Dynamiken des Paketweiterleitens kann in ernsthafter Dienstdegradation enden oder „Internet Zusammenbruch“ [7]. Dieses Phänomen wurde während der ersten Wachstumsphase des Internets in

den 1980er Jahren festgestellt und wird „congestion collapse“ [8] genannt. Bereits 1986 wurden von Jacobson entwickelte Stauverhinderungsmechanismen für Hosts entwickelt, die auch aktuell einen „congestion collapse“ verhindern [7].

Da das Internet seit dieser Zeit immer weiter wächst, wurde es offensichtlich, dass TCP Stauverhinderungsmechanismen [9], die absolut wichtig, nötig und mächtig sind, nicht unter allen Umständen ausreichend gute Dienste leisten. Das Hauptproblem liegt darin, dass von den Enden der Netze nur bedingt Kontrolle ausgeübt werden kann. Deshalb müssen auch in Routern Mechanismen angewendet werden, die die Stauverhinderungsmechanismen der Endpunkte ergänzen. Hierbei muss man zwischen den beiden Klassen „Queue Management“ und „Scheduling“ von Router Algorithmen unterscheiden. Queue management Algorithmen verwalten die Länge von Paket-Puffern durch Fallenlassen von Paketen wenn nötig oder angemessen. Scheduling Algorithmen legen fest, welche Pakete als nächstes gesendet werden sollen und können primär dafür genutzt werden, die Zuweisung von Bandbreite zwischen den Datenströmen zu verwalten [7].

Nach Jain [10] gibt es zwei Gründe, warum das Problem der Staukontrolle in Netzen sehr schwierig ist. Erstens gibt es Voraussetzungen für Staukontroll Schemas, die es schwierig machen eine zufriedenstellende Lösung zu finden. Zweitens gibt es unzählige Netzregeln, die das Design eines Schemas beeinflussen. Das sind die Gründe, warum ein Schema, das für ein Netz entwickelt wurde, in einem anderen Netz nicht funktioniert. Grundbedingung für Schemas zur Staukontrolle sind: Das Schema muss einen kleinen Overhead haben, alle Datenströme gleichbehandeln, schnell auf andere Situationen reagieren können, in schlechten Umgebungen funktionsfähig sein und für alle Benutzer optimal sein.

Um E2E-Staukontrolle in Routern zu betreiben, muss nicht nur jeder einzelne Router auf sich allein gestellt seinen Puffer überwachen, sondern auch an die nächsten Router Informationen weiterleiten. Für diese Benachrichtigung könnten zwei reservierte Bits im TCP/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 darüber informiert, dass es zu einem Stau gekommen ist und möglicherweise einzelne Pakete doppelt versandt werden müssen oder die Geschwindigkeit gedrosselt werden sollte. Prinzipiell ist es in Netzen, in denen es die Hauptaufgabe von Routern ist, Pakete an den Output-Port weiterzuleiten, kein Problem Pakete einfach fallen zulassen. Dies ist allerdings im Internet heute nicht mehr der Fall, da E2E-Verbindungen über sehr viele Router gehen können. ECN ist ein zuverlässiger Mechanismus, beinhaltet allerdings keine Methoden zur Erkennung von Staus [11]. Da es nicht vorhersehbar ist, wie hoch der Datenverkehr zukünftig sein wird, besteht das wirkliche Problem also darin, Staus frühzeitig und zuverlässig zu erkennen [12].

Router sollten den Ursprüngen eines Datenstrom signalisieren, dass die Sendegeschwindigkeit reduziert werden muss, um Staus zu vermeiden. Zusätzlich kann so eine gleichmäßige Verteilung der Bandbreite erfolgen, wenn besonders gierige Datenströme vorhanden sind. „Active Queue Management“

(AQM) hat das Ziel Staus in Netzen rechtzeitig zu entdecken, bevor die Routerpuffer volllaufen. Es gibt im Prinzip zwei Möglichkeiten andere Router darüber zu informieren, dass ein Stau bevorsteht: Pakete können fallengelassen oder markiert werden. Die erste Strategie erfordert, dass die Endpunkte kooperieren und erzeugt einen Overhead, weil Pakete anschließend erneut gesendet werden müssen. Endpunkte müssen auf markierte Pakete reagieren, als wären sie fallengelassen worden, allerdings wird hier kein weiterer Overhead erzeugt, da die Pakete nicht erneut gesendet werden müssen. AQM-Algorithmen können zusätzlich die Bandbreite von besonders gierigen Datenströmen reduzieren, indem deren Pakete häufiger fallengelassen werden [2].

### III. DEFINITION UND ANWENDUNG VON ACTIVE QUEUE MANAGEMENT

TCP ist eine Menge von Regeln (Protokollen), die zusammen mit dem Internet Protokoll (IP) genutzt werden, um Daten in Form von Paketen zwischen Computern über das Internet zu übertragen. Während IP dafür sorgt, dass Daten tatsächlich zugestellt werden, kümmert sich TCP darum, die einzelnen Datenpakete nachzuverfolgen, in die Daten für ein effizientes Routen durch das Internet aufgeteilt werden. TCP implementiert einen Algorithmus zur Flusskontrolle, der „Sliding Window“ genannt wird. Das „Window“ ist die maximale Anzahl an Paketen, die gesendet werden können ohne auf Rückmeldung warten zu müssen. Der Algorithmus besteht also aus folgenden Schritten [13]:

- 1) Alle neuen Segmente eines Windows verschicken.
- 2) Auf die Rückmeldung warten (es können mehrere Pakete auf einmal zurück gemeldet werden).
- 3) Das Window zur angegebenen Position verschieben und die Windowgröße auf der Wert aus der Rückmeldung setzen.

Wenn die Rückmeldung eines Pakets nicht in einer angegebenen Zeit erfolgt, dann wird das Paket neu verschickt. TCP nimmt hier keine Rücksicht auf Staus im Netz. Wenn also kontinuierlich immer weiter Pakete an das Netz versandt werden, ohne dass diese vom Netz aufgenommen werden können, führt dies zu einem Stau, der sich eventuell immer weiter aufbaut. An genau dieser Stelle muss ein Active Queue Management Algorithmus ansetzen und die Staukontrolle übernehmen [13].

Es existieren die beiden unterschiedlichen Routeralgorithmen zeitplan-orientierte und queue-management-orientierte Algorithmen. Zeitplan-orientierte Routeralgorithmen bieten eine gleichbehandelnde Bandbreitenverwendung, sind aber oft zu komplex für highspeed Implementierungen und skalieren nicht gut bei einer hohen Anzahl an unterschiedlichen Flüssen. Queue-Management Routeralgorithmen sind sehr einfach zu implementieren, versagen allerdings bei der Gleichbehandlung von Flüssen. Ein wirklich guter Active Queue Management Algorithmus muss also ein Hybrid aus beiden möglichen Algorithmen sein und damit eine Gleichbehandlung von Flüssen gewährleisten und gleichzeitig relativ einfach zu implementieren sein [14] [15].

Alle Internetrouter besitzen Puffer, um Pakete während eines Staus vorzuhalten. Heute wird die Größe von Routerpuffern von der Dynamik des Staukontrollalgorithmus von

TCP festgelegt. Genauer gesagt ist es das Ziel einen Puffer nie leer laufen zu lassen. Die Größe eines Puffers muss nach dieser Definition mit wachsender Verbindungsgeschwindigkeit ansteigen. Das Problem hierbei ist, dass auf diese Art große und langsame Speicher verwendet werden müssten. Durch die Anwendung von besseren Staukontrollalgorithmen wie AQM können diese kostspieligen Puffer klein bleiben [16].

Ein gleichbehandelnder Queue-Algorithmus sollte die folgenden Bedingungen erfüllen. Erstens sollen Daten aus einfachen Datenquellen wie beispielsweise Telnet mit einer sehr kurzen Verzögerung behandelt werden. Zweites soll das Fallenlassen von Paketen vermieden werden. Das ist allerdings nur möglich, wenn sich alle Datenquellen untereinander derart koordinieren, dass die Queue-Größe unter Kontrolle gehalten wird. Das Problem hierbei besteht darin, dass Bandbreite, Schnelligkeit und Pufferplatz unabhängig von einander zugeordnet werden müssen. Die Auswirkung von Flüssen aus nicht responsiven Quellen sollen möglichst gering gehalten werden und die anderen responsiven Flüsse nicht beeinträchtigen [17].

Zusätzlich zu Active Queue Management (AQM) Mechanismen ist es sehr sinnvoll Explicit Congestion Notification (ECN) zu nutzen, da hier anstatt Pakete fallen zu lassen Informationen im TCP-Header weitergegeben werden. Auf diese Art werden auch Router, die vor oder nach dem potenziellen Stau liegen über Probleme informiert und können entsprechend reagieren. Zusammen haben AQM und ECN das Potenzial den Effekt von Verlusten in latenz-sensitiven Flüssen zu reduzieren [4]. Der Erfolg jedes AQM Mechanismus hängt stark mit der möglichen Verbindung des Algorithmus zu ECN zusammen [18].

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 für ein spezielles Datenverkehrsaufkommen geeignet ist, aber nachteilige Effekte aufweist, wenn sie in einer anderen Umgebung angewendet wird. Entsprechend müssen AQM-Methoden entwickelt und ihre Parameter so eingestellt werden, dass sie für eine Vielzahl von unterschiedlichen Umgebungen verwendbar sind und zumindest keine schlechteren Ergebnisse liefern als TCP selbst [19]. Es existieren sehr viele unterschiedliche AQM-Methoden, die komplett unterschiedliche Ansätze verfolgen. Manche Algorithmen berechnen beispielsweise Wahrscheinlichkeiten, um Pakete fallen zu lassen, andere nutzen Verfahren aus der mathematischen Optimierung aus, um die bestmögliche Staukontrolle zu erreichen. Alle Algorithmen unterscheiden sich in der Anzahl an möglichen Einsatzgebieten, Komplexität und Qualität [18].

Im nächsten Kapitel stellen wir drei der bekanntesten Algorithmen vor.

#### IV. DIE GÄNGIGSTEN ACTIVE QUEUE MANAGEMENT ALGORITHMEN

Seitdem die ersten Ideen für Active Queue Management vorgestellt wurden und mit der Einführung von ECN in IP/TCP wurden viele verschiedene AQM Algorithmen entwickelt. Es

würde weit über den Rahmen dieser Arbeit hinausgehen, sie alle zu erwähnen, weshalb hier nur drei vorgestellt werden.

##### A. RED: Random Early Detection

Der erste Algorithmus, der präsentiert wird, ist Random Early Detection (RED). Er war einer der ersten AQM Algorithmen und viele andere Arbeiten entwickelten diesen weiter wie z.B. in [20] oder [21]. Andererseits werden Algorithmen, die anders ablaufen, oft mit RED verglichen.

Floyd und Van Jacobson haben RED 1993 vorgestellt [22]. In ihrer Arbeit wird die Funktionsweise des Algorithmus dargestellt. Um die Sender über die Verstopfung zu informieren kann RED entweder Pakete fallen lassen oder das ECN-Bit im Header setzen, je nach Router. Im folgenden werden wir nur die Möglichkeit des Markierens betrachten, was auf den eigentlichen Algorithmus keinerlei Auswirkungen hat. Als Messgröße wird die durchschnittliche Queue Länge benutzt. Die durchschnittliche Länge  $Q_{avg}$  wird für jedes eintreffende Paket mittels der aktuellen Länge der Queue  $q$  und dem Gewicht der Queue  $w_q$  folgendermaßen neu berechnet:  $Q_{avg} = (1 - w_q)Q_{avg} + w_q q$ . Dieser Wert wird mit zwei Parametern verglichen, der minimalen Queue Länge  $Q_{min}$  und der maximalen Queue Länge  $Q_{max}$ . 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.

Für die Berechnung der finalen Markierungswahrscheinlichkeit  $p_a$  wird die Markierungswahrscheinlichkeit  $p_b$  benötigt. Diese berechnet sich wie folgt aus der minimalen und maximalen Queue Länge  $Q_{min}$  und  $Q_{max}$ , der Durchschnittslänge  $Q_{avg}$  und dem Maximum für  $p_b$ ,  $max_b$ , wie folgt:  $p_b = \max_p \frac{Q_{avg} - Q_{min}}{Q_{max} - Q_{min}}$ .  $p_b$  steigt folglich linear von 0 bis zum Wert  $max_b$  an. Die finale Markierungswahrscheinlichkeit wird mittels  $p_b$  und eines Zählers  $z$  berechnet:  $p_a = \frac{p_b}{1 - zp_b}$ . Der Zähler wird für jedes einkommende Paket inkrementiert. Ein Paket wird mit der Wahrscheinlichkeit  $p_a$  markiert. Ist dies der Fall, so wird der  $z$  auf 0 gesetzt. Mithilfe des Zählers steigt die Wahrscheinlichkeit somit für jedes weitere Paket an.

Die Markierungswahrscheinlichkeit für ein Paket einer Verbindung verhält sich also in etwa proportional zum Anteil der Bandbreite, den diese Verbindung belegt. Auf diese Weise versucht der Algorithmus die Ressourcenzuteilung fair zu machen. Neben der Queue Länge in Paketen könnte RED auch die Bytelänge und somit die Anzahl an Bytes eines Pakets zur Bewertung heranziehen. Diese Information wird dann in den Markierungswahrscheinlichkeit  $p_b$  mit einbezogen. Nach der Berechnung ändert sich der Wert dann in  $p_b = \frac{p_b \cdot \text{Paketbytes}}{\text{maximalePaketbytes}}$ . Dadurch werden große Pakete mit einer höheren Wahrscheinlichkeit markiert als kleine Pakete. An der Beschreibung des Algorithmus wird ersichtlich, dass RED viele Parameter benötigt, die vorab festgelegt werden müssen. Damit RED gut läuft, müssen für jeden Router die richtigen Werte gefunden werden, was ein nicht zu vernachlässigendes Problem des Algorithmus darstellt. Mehr dazu in Abschnitt V.

##### B. BLUE

Der BLUE Algorithmus wurde 1999 von Feng et.al. an der University of Michigan in Zusammenarbeit mit IBM vorge-

stellt [23]. BLUE wurde entwickelt, um einige Schwachstellen von RED zu verbessern, ist aber ein völlig neuer Ansatz. Wie bereits erläutert verlässt sich RED auf die Queuelänge, um Verstopfungen anzuzeigen, und benötigt viele Parameter, die konfiguriert werden müssen. Die Autoren von BLUE argumentieren, dass RED nur wenn diese richtig konfiguriert sind und wenn ausreichend Pufferplatz zur Verfügung steht optimal läuft. BLUE dagegen verlässt sich auf den Paketverlust und die Verbindungsauslastung um seine Markierungswahrscheinlichkeiten zu berechnen. Genaue 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 für die Erhöhung oder Erniedrigung dieser Wahrscheinlichkeit wird auf Basis der verlorenen Pakete beziehungsweise auf Basis der ungenutzten Verbindungen getroffen: Erhält der Router die Information, dass ein Paket verloren gegangen ist, wird  $p_m$  um den Wert  $d_1$  erhöht. 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 geändert wird. Formal läuft der Algorithmus folgendermaßen ab:

```

if(Paketverlust  $\wedge$  (now - last_update) > freeze_time) then
     $p_m = p_m + d_1$ 
    last_update = now
if(Verbindung frei  $\wedge$  (now - last_update) > freeze_time)) then
     $p_m = p_m - d_2$ 
    last_update = now

```

Die beiden Parameter  $d_1$  und  $d_2$  geben an, um wie viel  $p_m$  zu erhöhen beziehungsweise zu reduzieren ist.  $d_1$  sollte deutlich größer sein als  $d_2$ , da BLUE somit auf Verstopfungen sehr viel schneller reagieren kann. Die Autoren geben weiterhin an, dass in ihren Experimenten die minimale Zeitspanne zwischen Änderungen an  $p_m$ , die *freeze\_time*, konstant gehalten wurde. Sie sagen aber auch, dass in einem Netz dieser Parameter zufällig für jeden Router gewählt werden sollte, um globale Synchronisation zu vermeiden. Dieser Algorithmus passt sich somit selbstständig an den aktuellen Bedarf des Netzes an und benötigt keine Router-abhängigen 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 [24]. Wie bereits der Name andeutet basiert der Algorithmus auf einer virtuellen Queue. Für die Markierung beziehungsweise das Fallenlassen von Paketen wird die Kapazität dieser virtuellen Queue und keine Markierungswahrscheinlichkeit zu Rate gezogen.

Bei AVQ verwaltet der Router neben der echten Queue eine virtuelle Queue mit der Kapazität  $C_v \leq C$ , wobei  $C$

die Kapazität der Verbindung ist. Zu Beginn ist  $C_v = C$ . Bei jedem ankommenden Paket wird überprüft, ob der Puffer der virtuellen Queue das Paket aufnehmen könnte. Ist dem so, wird das echte Paket in die tatsächliche Queue eingereiht, ansonsten wird es markiert oder fallen gelassen. Die Kapazität der virtuellen Queue wird ebenfalls bei jedem ankommenden Paket angepasst gemäß der Differentialgleichung  $\dot{C}_v = \alpha(\gamma C - \lambda)$  wobei  $\alpha$  ein Glättungsparameter,  $\gamma$  die angestrebte Auslastung der Verbindung und  $\lambda$  die Ankunftsrate der Verbindung ist. Das Markieren passiert auf diese Weise aggressiver, also häufiger, wenn die Verbindung ihre gewünschte Bandbreite überschreitet und weniger aggressiv, wenn nicht.

Es ist klar, dass in der virtuellen Queue keine Pakete eingereiht werden müssen, lediglich die Länge, also die Kapazität der virtuellen Queue muss ermittelt werden. Die Autoren geben in ihrer Arbeit auch an, wie dieses Verfahren als Algorithmus konkret implementiert werden kann:

Für jedes ankommende Paket DO

$VQ = \max(VQ - C_v(t - s), 0)$

if( $VQ + b > B$ ) then

    Paket markieren

else

$VQ = VQ + b$

$C_v = \max(\min(C_v + \alpha \cdot \gamma \cdot C(t - s), C) - \alpha \cdot b, 0)$

$s = t$

wobei  $B$  die Puffergröße in Bytes,  $s$  die Ankunftszeit des letzten Pakets,  $t$  die aktuelle Zeit,  $b$  die Paketgröße des aktuellen Pakets in Bytes und  $VQ$  die Anzahl an Bytes, die aktuell in der virtuellen Queue sind, ist.

Die Autoren erläutern, dass die algorithmische Komplexität von AVQ in etwa der von RED entspricht. Anstelle der Länge der Queue wird bei AVQ jedoch die Ausnutzung der Queue als Entscheidungskriterium für das Markieren von Paketen verwendet. Für die Verwendung von AVQ müssen der Glättungsparameter  $\alpha$  und die gewünschte Ausnutzung  $\gamma$  vorab angegeben werden. Diese dienen der Stabilität des Verfahrens. Weiter geben die Autoren an, dass  $\gamma$  es den ISP's erlaubt einen Ausgleich zwischen hoher Bandbreitenausnutzung und kleinen Puffern vorzunehmen. Eine Regel, wie diese Parameter gesetzt werden sollten, befindet sich in der Originalarbeit in [24].

## V. VERGLEICH DER VORGESTELLTEN ALGORITHMEN

### 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 lässt [23]. Für ihren Versuch haben sie ein Netz mithilfe des LBNL Network Simulators [25] simuliert. Für alle Quellen wurde ECN aktiviert, was bedeutet, dass jedes verlorene Paket einen Pufferüberlauf der Queue darstellt. Die Sender wurden zufällig innerhalb der ersten Sekunde gestartet. Die Paketverluste wurden nach 100 Sekunden Simulation und weiteren 100 Sekunden, also insgesamt 200 Sekunden, für die gesamte Simulationszeit gemessen. Die Parameter für RED wurden experimentell bestimmt,  $Q_{min}$  betrug immer 20% der Queuelänge,  $Q_{max}$  80%. Für BLUE wurde  $d_1$  um eine Größenordnung höher als  $d_2$  gesetzt. Insgesamt wurden vier

Konfigurationen für RED und vier Konfigurationen für den BLUE Algorithmus angewandt und die Ergebnisse verglichen.

Zwischen den Ergebnissen der einzelnen Konfigurationen sowohl bei RED als auch bei BLUE gab es nur minimale Unterschiede, weshalb diese hier zusammengefasst werden. Die Queuelänge für Engstelle des Netzes wurde von 100KB bis 1000KB gesetzt, was einer Verzögerung zwischen 17,8ms und 178ms entspricht. Bei 1000 Quellen die gleichzeitig senden, war sowohl bei RED als auch bei BLUE die Verbindung für alle Queueängen zu 100% ausgelastet. Die Paketverlustrate betrug bei BLUE für alle Verzögerungen 0%. Der RED Algorithmus dagegen zeigt bei niedrigen Verzögerungen eine Paketverlustrate von bis zu über 15%. Diese fällt mit größer werdenden Puffern auf knapp über 0% bei einer Verzögerung von etwa 100ms, steigt dann aber wieder an auf etwa 5%. Für das gleiche Experiment mit 4000 Quellen war die Auslastung ebenfalls bei beiden stets bei 100%. Die Paketverlustrate von RED lag diesmal bei kleinen Verzögerungen bei über 30%, was mit zunehmenden Queueängen auf etwa 25% abfiel. Bei BLUE lag die Verlustrate dagegen zu Beginn bei etwa 15% und sank auf unter 10% mit größer werdenden Queues.

### B. Vergleich AVQ und RED

Analog zu BLUE hat auch Kunniyar in seiner Arbeit zu AVQ Vergleiche des Verfahrens zu anderen AQM Algorithmen gemacht [24]. Neben RED wurde der Algorithmus auch mit Random Early Marking (REM), dem PI Controller und GKVQ verglichen. Diese werden hier jedoch vernachlässigt, da auch die Algorithmen nicht präsentiert worden sind.

Die beschriebenen Versuchen wurden mit dem Simulator *ns-2* durchgeführt. Der Parameter  $\gamma$ , die angestrebte Auslastung, wurde für AVQ auf den Wert 0,98 gesetzt; der Glättungsparameter  $\alpha$  wurde auf 0,15 gesetzt. Die Konfiguration für RED wurde gemäß den Empfehlungen aus <http://www.aciri.org/floyd/REDparameters.txt> vorgenommen. Die Queuelänge an der engsten und somit relevanten Stelle der Verbindung betrug 100 Pakete bei einer Paketgröße von 1000 bytes. Für das erste Experiment wurde bei den TCP Verbindungen ECN aktiviert. Jedes verlorene Paket ist folglich ein Zeichen des Pufferüberlaufs. Die Grenzwerte für RED wurden auf  $Q_{min} = 0,37$  und  $Q_{max} = 0,75$  gesetzt. Die durchschnittliche Verzögerung 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%. Für 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. Mit steigenden FTP Verbindungen stiegen diese in etwa linear auf über 6.000 verlorene Pakete bei 180 Verbindungen. Bei AVQ lagen die Paketverluste für jede Anzahl an Verbindungen

bei 0. Der AVQ Algorithmus zeigt für diesen Aufbau eine deutlich besser Leistung mit höherer Auslastung und ohne Paketverluste.

In einem zweiten Experiment wurde ECN deaktiviert. Das heißt, dass sowohl RED als auch AVQ Pakete fallen lassen, um auf Verstopfung hinzuweisen. Der Versuchsaufbau ist gleich dem vorherigen mit der Ausnahme, dass für AVQ die angestrebte Auslastung  $\gamma$  auf den Wert 1 gesetzt wurde, die AVQ also voll genutzt werden soll. Die Anzahl an FTP Verbindungen wurde fest auf 40 gesetzt; dafür wurden TCP short-flows mit 20 Paketen verwendet, deren Ankunftszeit an der kritischen Verbindungsstelle langsam gesteigert wurde. Die Auslastung der Verbindung für AVQ lag konstant bei den angestrebten 100%. Bei RED lag diese bei 10 ankommenden short-flows pro Sekunde bei etwa 94% und stieg auf über 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. Für RED lag diese Quote anfangs nur bei 87%, stieg dann aber stark an auf 98% bei 50 short-flows pro Sekunde und lag damit höher als bei AVQ.

### C. Zusammenfassung

Zu Beginn dieser Arbeit wurde die Notwendigkeit für AQM Algorithmen erläutert. Sie sind notwendig, um aktiv einen Pufferüberlauf der Router in großen 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 [7]. Ebenfalls wurde empfohlen, weiter Forschung in diesem Bereich zu betreiben. Hervorzuheben ist auch, dass die Empfehlung für RED vor der Empfehlung für ECN ausgegeben wurde, welche 1999 in RFC 2481 gegeben wurde [26].

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 aufweist 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 durchgeführt wurden mit dem Ziel zu zeigen, dass das jeweilige Verfahren das beste ist.

## VI. AUSBLICK UND ANDERE ANSÄTZE

Neben dem hier vorgestellten Ansatz des Active Queue Managements gibt es auch andere Verfahren, die zum Ziel haben, Überlastungen in Netzen zu vermeiden. In [27] werden hierzu zum Beispiel Zugangssteuerung und Routing unter Verkehrsberücksichtigung genannt. Bei der Zugangssteuerung werden nur dann neue, virtuelle Verbindungen aufgebaut, wenn das Netz diese auch verkraften kann. Beim Routing unter

Verkehrsberücksichtigung werden Algorithmen angewandt, die im Netz nach möglichst wenig ausgelasteten Pfaden suchen. Diese Ansätze helfen somit auch, die Queues der Router zu entlasten.

Ein weiterer Ansatz, der den AQM Verfahren schon recht ähnlich ist, wird auch in [27] erwähnt: 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 über die anderen Router an den Sender zurück. 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.

Wie bereits angedeutet sind AQM Algorithmen 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 für die Nutzer wäre jedoch, dass AQM Algorithmen auch wirklich auf allen Routern implementiert werden. Wie in dieser Arbeit gezeigt wäre das eine Verbesserung hin zur bestmöglichen 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] 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.
- [6] 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, Mär. 1998, pp. 299–306vol.1.
- [7] 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.
- [8] J. Nagle, "Congestion control in ip/tcp internetworks," *SIGCOMM Comput. Commun. Rev.*, vol. 14, Nr. 4, pp. 11–17, Okt. 1984. [Online]. Available: <http://doi.acm.org/10.1145/1024908.1024910>
- [9] W. Stevens, "Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," United States, 1997.
- [10] R. Jain, "Congestion control in computer networks: issues and trends," *Network, IEEE*, vol. 4, Nr. 3, pp. 24–30, Mai 1990.
- [11] S. Floyd, "Tcp and explicit congestion notification," *SIGCOMM Comput. Commun. Rev.*, vol. 24, Nr. 5, pp. 8–23, Okt. 1994.
- [12] R. Jain, "Congestion control and traffic management in atm networks: Recent advances and a survey," *Comput. Netw. ISDN Syst.*, vol. 28, Nr. 13, pp. 1723–1738, Okt. 1996. [Online]. Available: [http://dx.doi.org/10.1016/0169-7552\(96\)00012-8](http://dx.doi.org/10.1016/0169-7552(96)00012-8)
- [13] S. Oldak, W. Gong, C. Hollo, D. Towsley, V. Misra, und Y. Chait, "Active queue management for differentiated services," August 2006. [Online]. Available: <http://www.freepatentsonline.com/7085236.html>
- [14] B. Suter, T. Lakshman, D. Stiliadis, und A. Choudhury, "Efficient active queue management for internet routers," in *Proceedings of INTEROP, Engineering Conference*, 1998.
- [15] R. Pan, B. Prabhakar, und K. Psounis, "Active queue management toward fair bandwidth allocation," Januar 2008. [Online]. Available: <http://www.google.de/patents/US7324442>
- [16] G. Appenzeller, I. Keslassy, und N. McKeown, "Sizing router buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, Nr. 4, pp. 281–292, Aug. 2004.
- [17] A. Demers, S. Keshav, und S. Shenker, "Analysis and simulation of a fair queueing algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 19, Nr. 4, pp. 1–12, Aug. 1989.
- [18] C. Hollo, 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, pp. 1726–1734vol.3.
- [19] 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, Mär. 2000, pp. 1435–1444vol.3.
- [20] 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.
- [21] 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, pp. 942–951vol.2.
- [22] S. Floyd und V. Jacobson, "Random early detection gateways for congestion avoidance," *Networking, IEEE/ACM Transactions on*, vol. 1, Nr. 4, pp. 397–413, Aug. 1993.
- [23] W. Feng, K. Shin, D. Kandlur, und D. Saha, "Blue: A new class of active queue management algorithms," 1999. [Online]. Available: <http://www.eecs.umich.edu/techreports/cse/99/CSE-TR-387-99.pdf>
- [24] 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, pp. 123–134, Aug. 2001.
- [25] S. McCanne, S. Floayd, und K. Fall, "ns version 1 - lbnl network simulator." [Online]. Available: <http://ee.lbl.gov/ns/>
- [26] K. Ramakrishnan, A. L. Research, und S. Floyd, "A proposal to add explicit congestion notification (ecn) to ip," United States, 1999.
- [27] A. Tannenbaum und D. Wetherall, *Computernetzwerke*. Pearson Deutschland GmbH, 2012.
- [28] S. Athuraliya, S. Low, V. Li, und Q. Yin, "Rem: active queue management," *Network, IEEE*, vol. 15, Nr. 3, pp. 48–53, Mai 2001.
- [29] W. Feng, K. Shin, D. Kandlur, und D. Saha, "The blue active queue management algorithms," *IEEE/ACM Trans. Netw.*, vol. 10, Nr. 4, pp. 513–528, Aug. 2002.
- [30] R. Wolff, *Stochastic modeling and the theory of queues*. Prentice Hall, 1998.