ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Netzwerkstaukontrolle und die Arbeitsweise verschiedener Algorithmen in TCP

von

Michał Roziel

Matrikelnummer : 5012845

Ein wissenschaftlicher Bericht im Rahmen der Vorlesung

„Wissenschaftliches Arbeiten“

an der htw saar im Studiengang Informatik

Saarbrücken, den 29. August 2025

Abstract

Dieser Bericht zielt darauf ab, aktuell benutzte Algorithmen der Netzwerkstaukontrolle in Computernetzwerken zu vergleichen.

Die Algorithmen der Staukontrolle, welche in diesen Vergleich einfließen sind : TCP BBR, TCP NewReno, TCP Cubic, und TCP Vegas.

Mittels des Open-Source Netzwerksimulators *NS-3* [1] wird ein virtuelles Netzwerk mit einer 10Mbit/s Engstelle zwischen zwei Endpunkten aufgestellt. In diesem Netzwerk wird zunächst ein künstlicher Datenverkehr erzeugt. Der genannte Datenverkehr wird anschließend aufgezeichnet und das dabei entstehende *Congestion Window*(CW) dient als Basis für den Vergleich. Die unterschiedlichen CW's werden schließlich analysiert und in Bezug auf das gegebene Szenario evaluiert. Dieser Bericht bietet dem Leser einen Überblick über einige CC-Algorithmen und ein grundlegendes Verständnis darüber, wie Computernetzwerke auf Datenstau und Paketverlust reagieren.

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Insbesondere habe ich alle KI-basierten Werkzeuge angegeben, die ich bei der Erstellung, Übersetzung oder Überarbeitung des Textes verwendet habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“ zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, den 29. August 2025

Unterschrift Michał Roziel

Inhaltsverzeichnis

1. Einleitung

Network congestion control (**CC**) ist ein essentieller Bestandteil der meisten modernen Computernetzwerke. Wenn eine Netzwerkschnittstelle zu einem Zeitpunkt versucht eine zu große Menge an Datenpaketen aufzunehmen, kommt es zu Stau von Datenverkehr und zu einem potentiellen Verlust von Datenpaketen. Aufgrund diesem Vorkommen werden Algorithmen innerhalb von Netzwerkprotokollen verwendet, diese erkennen den Anstau von Datenverkehr im Netzwerk, und helfen den Fluss von Datenpaketen zu steuern. Neben dem effizienten Durchfluss von Informationen ist zeitgleich auch die Fairness, was die Verteilung von Ressourcen einer Netzwerkschnittstelle an ihre Hosts angeht, wichtig. Auch dies wird von CC-Algorithmen gewährleistet. In dem heutigen Stand von Rechnernetzen werden verschiedene Protokolle zur Netzwerkstaukontrolle verwendet, Ich werde mich im Rahmen dieses Berichts allerdings auf das Transmission Control Protocoll beschränken, da dies das am meisten verbreitete ist.

Mit stets weiterentwickelten Rechnernetzen, und einem jährlich zunehmenden Datenverkehr, wie auch in der Deutschen Internetschnittstelle *DE-CIX* [2] in Frankfurt gewinnen diese Algorithmen an Bedeutung.

1.1. Grundlagen und Begriffe

Um die Funktionsweise und den Unterschied zwischen den hier behandelten CC-Algorithmen verstehen zu können, muss der Leser erst ein Grundverständnis über die Kommunukation innerhalb von Computernetzwerken haben. Grundsätzlich wird heutzutage eins von zwei am verbreitesten Protokollen verwendet, entweder TCP oder UDP.

1.1.1. Transmission Control Protocoll

Das Transmission Control Protocoll (**TCP**) ist ein weit verbreitetes Netzwerkprotokoll. TCP lässt sich in zu der Schicht 4 (Transportschicht) in dem OSI-Modell einordnen, hierbei liegt es zwichen der Vermittlungs- und Kommunikationsschicht. TCP wurde 1981 unter RFC 793 erstmals standarisiert. [3]
Die Aufgabes des TCP Protokolls ist es, zwischen zwei Hosts eine Verbindung aufzubauen, welche anschließend dazu genutzt wird, Nachrichten in zu verschicken und zu empfangen.

> "transport-layer protocol [...] from an application's perspective, it is as if the hosts running the processes were directly connected;" [4, 241].

Dies Bedeutet, dass TCP ebenfalls eine gewisses Abstraktionsniveau des Nachrichtenaustausches abnimmt.

Ein klares Unterscheidungsmerkmal des TCP von dem ebenfalls bekannten User Datagramm Protocoll (UDP) ist, dass TCP verbindungsorientiert arbeitet, während UDP als verbindungslos gilt. Zudem werden bei TCP Daten in Form von Packets

verschickt, während man bei UDP von Datagrammen spricht.
Bei TCP wird zu Beginn ein *Three-Way Handshake* wie folgt durchgeführt :

Der Sender schickt zunächst eine Nachricht mit einem Verbindungswunsch an den Empfänger und setzt das Flag *Synchronize,* ***SYN***.
Der Empfänger antwortet mit einer Bestätigung die erste Nachrichte erhalten zu haben, und schickt ebenfalls einen Verbindungswunsch. Es werden die Flags ***SYN-ACK*** (Synchronise-Acknowledge) gesetzt.

Als letztes schickt der Sender eine Bestätigung, dass die Nachricht des Empfängers angekommen ist, dies geschieht wieder mit dem Flag ***ACK***. Nun ist die Verbindung bereit, Daten in beide Richtungen zu übertragen. Nach jedem gesendeten Datenpaket folgt eine ***ACK*** Bestätigung.
Die Abbildung 3 im Anhang visualisiert den Three-Way Handshake.

1.2. Algorithmen der Staukontrolle

Algorithmen der Staukontrolle kommen zum Einsatz, wenn in einem Netzwerk ein potentieller Datenstau erkannt wird. Grundsätzlich starten die Algorithmen in dem Arbeitsmodus *slow start*. Falls Datenstau erkannt wird, treten diese Algorithmen in die Phase *congestion avoidance*.

Hiermit wird mit verschiedenen Vorgehensweisen gegen die Netzwerküberlastung gesteuert.
Es existieren unterschiedliche Typen von CC-Algorithmen, die jeweils ihre eigenen Vor- und Nachteile haben. In diesem Bericht wird jeweils ein Algorithmus pro Typ behandelt.

1.2.1. TCP BBR

Der *Bottleneck Bandwith and Round-Trip Time* (**BBR**) Algorithmus wurde erstmals in dem von der *Association for Computing Machinery* (**ACM**) veröffentlichten Magazin ACM vorgestellt. [5]

Seit 2016 wurden 3 Versionen entwickelt. Diese werden jeweils mit $BBRv\_x$ gekennzeichnet, wobei x die entsprechende Version beschreibt. BBR ist ein modellbasiertes Staukontrollverfahren, dessen Arbeitsweise in 4 Phasen unterteilt werden kann :

1. $STARTUP$(Startphase)
2. $DRAIN$(Abbau des Staus)
3. $ProbeBW$(Messen der Bandbreite)
4. $ProbeRTT$(Messen der Round-Trip-time)

Während andere CC-Algorithmen wie etwa NewReno oder CUBIC auf Paketverlust reagieren, misst BBR die vorliegende Engpassbreite sowie RTT, und bestimmt daraus die Sende-Rate, welche anschließend in dem Netzwerk eingesetzt wird. Da BBR nicht direkt auf Paketverlust reagiert, kommt es mit gut mit zufälligen Paketverlust

zurecht, hat aber zeitgleich Schwierigkeiten mit der Fairness was die Verteilung von Ressourcen angeht :

> "Unfairness happens [...], which relates to different settings of round-trip times and buffer sizes. Essentially, the reason for the unstable or unequal unfairness is the absence of responding mechanisms for bandwidth convergence." [6]

Die Fairness kann hierbei mit dem zwischen 0 und 1 liegenden Jain's Fairness (**JFI**) Index gemessen werden. [7]

1.2.2. TCP NewReno

Der TCP **NewReno** Algorithmus wurde erstmal 1999 mit dem RFC 2582 eingeführt. [8] NewReno ist ein *Loss-Based* Algorithmus, das bedeutet der Arbeitsmodus *congestion avoidance* tritt ein, falls ein Paketverlust erkannt wird. Das Wachstum des `cwnd` (vgl.1.3.1) ist hierbei linear gegenüber der Anzahl der Pakete.

1.2.3. TCP Cubic

Cubic ist ebenfalls ein *Loss-Based* Algorithmus.

Die Wachstumsfunktion, welche anschließend die neue Größe des *Congestion Window* beschreibt, wird definiert als : $W(t) = C(t-K)^3 + W_{max}$. Hierbei ist W die größe des Congestion Windows, C eine Cubic-Konstante, t die Zeit seit dem letzten Paketverlust, und K die Zeit, welche *TCP Cubic* benötigt um wieder *Wmax* zu erreichen.

Mit der Sensitivität auf Paketverlust β kann K wie folgt berechnet werden [9] :

$$K = \sqrt[3]{\frac{W_{\max} \cdot \beta}{C}}$$

1.2.4. TCP Vegas

Vegas ist ein Avoidance-basierter CC-Algorithmus, welcher 1994 vorgestellt wurde. Im Kapitel 5. von dem Buch *TCP Congestion Control : A Systems Approach* wird beschrieben, dass Vegas den gemessenen Durchsatz von Paketen mit dem erwarteten Wert vegleicht. [10] Auf Basis dieses Vergleiches wird dann bestimmt, wie viele Pakete ohne Verluste verschickt werden können.

Der erwartete Wert ist hier der Idealwert von dem Durchsatz, welcher über das Netzwerk geesendet werden könnte, falls keine Warteschlange vorliegen würde. Dieser kann wie folgt bestimmt werden : $ExpectedRate = \frac{\texttt{cwnd}}{BaseRTT}$ [10] wobei $BaseRTT$ die minimale RTT (vgl. 1.3.2) beschreibt.

Abbildung 1 visualisiert das `cwnd` (vgl. 1.3.1) und die dazugehörigen Durchsätze (erwartet & tatsächlich) für ein Beispiel Vegas Durchlauf. Die in blau eingezeichnete Linie beschreibt den erwarteten Durchsatz, während die schwarz gefärbte Linie den tatsächlichen Durchsatz darstellt.

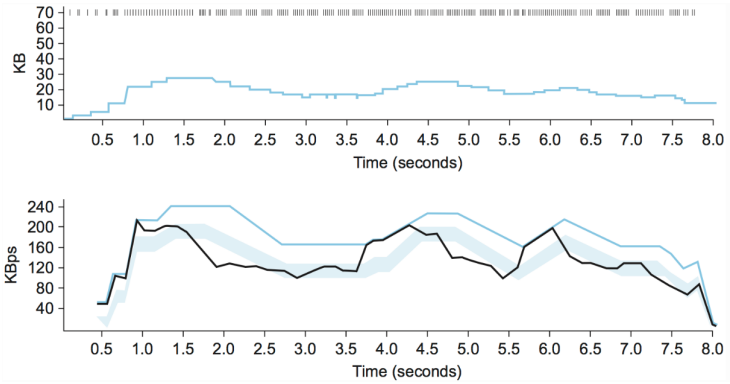


Abbildung 1: Beispiel von Vegas `cwnd` & Vegleich von erwarteten und tatsächlichen Durchsatz. [10]

1.2.5. Merkmale der CC Algorithmen auf einen Blick

Hier werden dem Leser unterschiedliche Merkmale von **CC** Algorithmen verständlich in einer Tabelle aufgelistet.

	Algorithmus	**Typ**	**Signal der Congestion avoidance**
BBR	a	b	c
NewReno	d	e	f
Cubic	g	h	i
Vegas	n	f	f

1.3. Definitionen

Hier werden dem Leser einige Definitionen über die in dem Bericht vorkommenden Begriffe näher gebracht.

1.3.1. Congestion Window

Das *Congestion Window*(`cwnd`) beschreibt die maximale Anzahl von Segmenten (Paketen) welche gleichzeitig über ein Netzwerk gesendet werden können, ohne dass Datenstau auftritt. Falls es zu Engpässen kommt, fällt der Verlauf von `cwnd`.

1.3.2. Round-Trip Time

Der Round-Trip Time (**RTT**)ist die Zeit von dem Absenden eines Segments bis zu dem Eintreffen der jeweiligen Bestätigungsnachricht ***ACK*** beim Sender.

2. Versuchsaufbau

Mit einem Test innerhalb des Command Line Simulators *NS-3.45* [1] lässt sich ein beliebieger CC-Algorithmus ausführen.

Netzwerktopologie

Zunächst definieren wir die Netzwerktopologie auf welcher die CC-Algorithmen laufen werden. [11]

```
            1000 Mbps           10 Mbps           1000 Mbps
Sender -------------- R1 -------------- R2 -------------- Receiver
            5 ms                10 ms              5 ms
```

So ist es möglich, einen Netzwerkstau zu simulieren, dieser bleibt für jeden Algorithmus gleich.

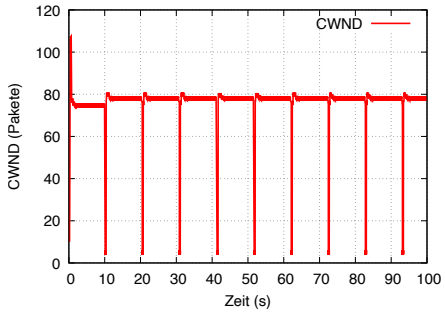
Durch das Verändern der folgenden *C++* Codezeile der Datei `tcp-bbr-example.cc` mit dem Namen des gewünschten Algorithmus lässt sich der Test unter gleichen Netzwerkbedingungen für die anderen Tesstfälle ausführen :

```
[...]
    std::string tcpTypeId = "ALGO_NAME";
[...]
```

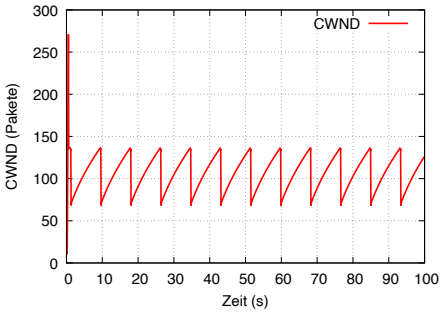
Mittels der Navigation in das Directory *ns-3.45* per Kommandozeile und dem Befehl

```
./ns3 run examples/tcp/tcp-bbr-example.cc
```

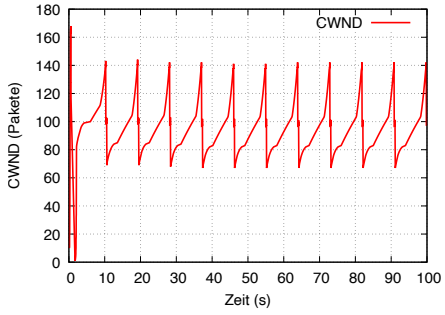
lässt sich der jeweilige Testlauf ausführen. Die aufgezeichneten Informationen können mit dem im Anhang hinterlegten Skript visualisiert werden.



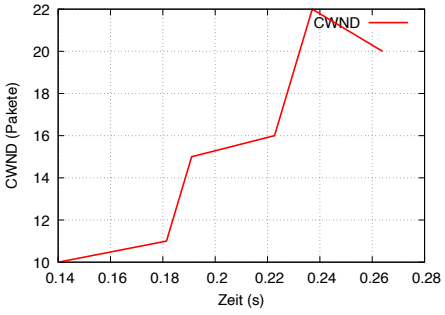
(a) BBR



(b) NewReno



(c) Cubic



(d) Vegas

Abbildung 2: Vergleich der aufgezeichneten Congestion Windows.

2.1. Analyse der Ergebnisse

Wie an den *Congestion Window* Graphen erkennbar, besitzt jeder der vorgestellten Algorithmen eine eigene Herangehensweise was das Regulieren von Datenstau angeht. Die RTT wird bei ungefähr 40 Milisekunden liegen, da die Verzögerung aufgrund der Netzwerktopologie bei `5ms + 10ms + 5ms` in jede Richtung liegt.

BBR (a)

Das im Graph in Abbildung 1(a) gezeigte `cwnd` weist auf klare, periodische Einbrüche des Verlaufs auf, diese lassen sich auf die Phase *ProbeRTT* des Algorithmus zurückführen. Dies unterscheidet BBR von den anderen CC-Algorithmen, da hier ein Einbruch im `cwnd` nicht direkt ein Hinweis auf Paketverlust ist, sondern jeweils eine gezielte Messung der RTT bedeutet. Mit jeder Messung wird eine neue minimale RTT bestimmt, und die Warteschlange von Paketen kann am Engpass geleert werden.

NewReno (b)

Der in Abbildung 1(b) aufgezeichnete Verlauf von NewReno zeigt im `cwnd` ein Sägezahnähnliches Muster. Hierbei lässt sich erst die lineare Steigung und dann der abrupte Einbruch der Anzahl der gesendeten Paketen leicht erkennen. Das `cwnd` wächst in der *congestion avoidance* Phase bis zum nächsten Einbruch. Nach dem Einbruch tritt der Algorithmus in eine *Recovery* Phase ein, wobei der Graph erneut linear zunimmt.

Cubic (c)

Mit dem in Abbildung 1(c) visualisierten Graphen kann der Leser das Arbeitsmuster von Cubic erkennen. Ähnlich wie NewReno in Abbildung 1(b) reagiert Cubic auf Paketverluste, hat allerdings eine strengere Anstiegskurve, da hier der Graph kubisch verläuft. Ebenfalls zu erkennen sind Plateaus, in denen für eine kurze Zeit das `cwnd` nicht ansteigt. Diese liegen an Orten, wo zuvor das W_{max} erreicht wurde.

Vegas (d)

In Abbildung 1 (d) wächst das `cwnd` ohne große Einbrüche. Dieser Verlauf passt zu Vegas, da der Algorithmus das `cwnd` nur langsam erhöht wenn ein Engpass erkannt wird. Somit wird die Warteschlange an Paketen klein gehalten. Wie an dem Graphen erkennbar, fällt das `cwnd` Verhalten linear.

3. Ergebnisse und Diskussion

Aufgrund der vorher durchgeführten Analyse werden hier die daraus gewonnenen `cwnd`-Ergebnissen präsentiert.

BBR weist periodisch auftretende Einbrüche auf, welche sich allerdings auf *ProbeRTT* zurückführen lassen. Grundsätzlich ist das Sendeverhalten von BBR stabil, und die auftretenden Warteschlangen werden regelmäßig abgebaut. Bei mehreren, nebenläufigen Übertragungen hat BBR Schwierigkeiten mit der Fairness, allerdings wurden diese in den späteren

Versionen adressiert. Da wie beschreiben der Verlust von Paketen nicht das primäre Signal ist, eignet sich BBR für Netzwerke in denen oft ein zufälliger Verlust auftritt, wie etwa bei WLAN.
NewReno welcher auch im Linux Kernel vertreten ist, hat eine lineare Steigung und abrupte Einbrüche. Da dieser Algorithmus stark auf den Verlust von Paketen reagiert, kann es bei Netzwerken mit zufälligen Signalschwankungen zu Unterauslastung kommen, hierbei wird aufgrund von ständig aufeinanderfolgenden *congestion avoidance* Phasen nicht die volle Bandbreite des Netzwerks ausgenutzt.
Cubic kann durch die kubische Wachstumskurve effizient nach Einbrüchen des `cwnd` wieder die gewünschte Kapazität an Paketen wiederfinden. Heutzutage wurd Cubic in Linux, aber auch in Netzwerken mit hohen Bandbreiten verwendet. [12]
Vegas besitzt duch die langsame Steigung des `cwnd` eine niedrige Latenz. Hierbei wird nicht direkt auf Verlust geachtet, sondern wie beschrieben die RTT gemessen, verglichen, und anschließend entschieden.

4. Fazit und Schlussfolgerungen

Ziel dieses Berichts war es die Staukontrollverfahren unter identischen Bedingungen zu vergleichen. Dies ist mit **ns-3** gelungen, und die Algorithmen konnten in Form vom `cwnd` Graphen aufgezeichnet werden. Auch wenn diese Verfahren das gleiche Ziel befolgen, nämlich den Datenfluss von Paketen über das Netzwerk zu verwalten, tun Sie dies mit verschiedenen Vorgehensweisen. Es soll eine möglichst hohe Bandbreite gewährleistet werden, und gleichzeitig kein Verlust von Paketen an Netzwerkschnittstellen erfolgen. Es ist klar, dass CC-Algorithmen ein nötiges Bestandteil der heutigen Rechnernetze ist, und bleibt.

5. Ausblick

Um den Rahmen dieses Berichts einfach zu halten, habe Ich mich entschieden, nur das `cwnd` in den Vergleich einzubringen. Interessant wäre zusätzlich die Messung der Bandbreite, oder etwa im Fall von **BBR** mehrere Datenflüsse(*Flows*) eizuführen, und diese mit **ns-3** und dem **JFI** auszuwerten.

Literaturverzeichnis

[1] "a discrete-event network simulator for internet systems," version 3.45. [Online]. Available: https://www.nsnam.org/

[2] DE-CIX, "Traffic statistics frankfurt," Frankfurt am Main, Deutschland, 2025, letzter Zugriff: 29. August 2025. [Online]. Available: https://www.de-cix.net/en/locations/frankfurt/statistics

[3] J. Postel, "transmission control protocol," Marina Del Rey, California, 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc793

[4] K. W. James F.Kurose, *Computer Networking: A Top-Down Approach*, 8th ed. London: Pearson, 2022.

[5] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, and V. Jacobson, "BBR Congestion Control," Internet Engineering Task Force, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02, Mar. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/02/

[6] L. Tang, "Bbr fairness evaluation using ns-3," arXiv preprint arXiv:2410.22560 [cs.NI], 2024. [Online]. Available: https://arxiv.org/abs/2410.22560v1

[7] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *CoRR*, vol. cs.NI/9809099, 1998. [Online]. Available: https://arxiv.org/abs/cs/9809099

[8] T. Henderson and S. Floyd, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 2582, Apr. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2582

[9] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, p. 64–74, Jul. 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400105

[10] L. Peterson, L. Brakmo, and B. Davie, *TCP Congestion Control: A Systems Approach*. Systems Approach LLC, 2022. [Online]. Available: https://tcpcc.systemsapproach.org/index.html

[11] M. P. T. Aarti Nandagiri, Vivek Jain, "Tcp bbr example," 2020, copyright (c) 2018-20 NITK Surathkal. [Online]. Available: https://github.com/nsnam/ns-3-dev-git/blob/master/examples/tcp/tcp-bbr-example.cc

[12] P. F. team, "Why does cubic take us back to tcp congestion control?" Online, 2021. [Online]. Available: https://pandorafms.com/blog/tcp-congestion-control/

Anhang

A. Zusätzliche Abbildungen

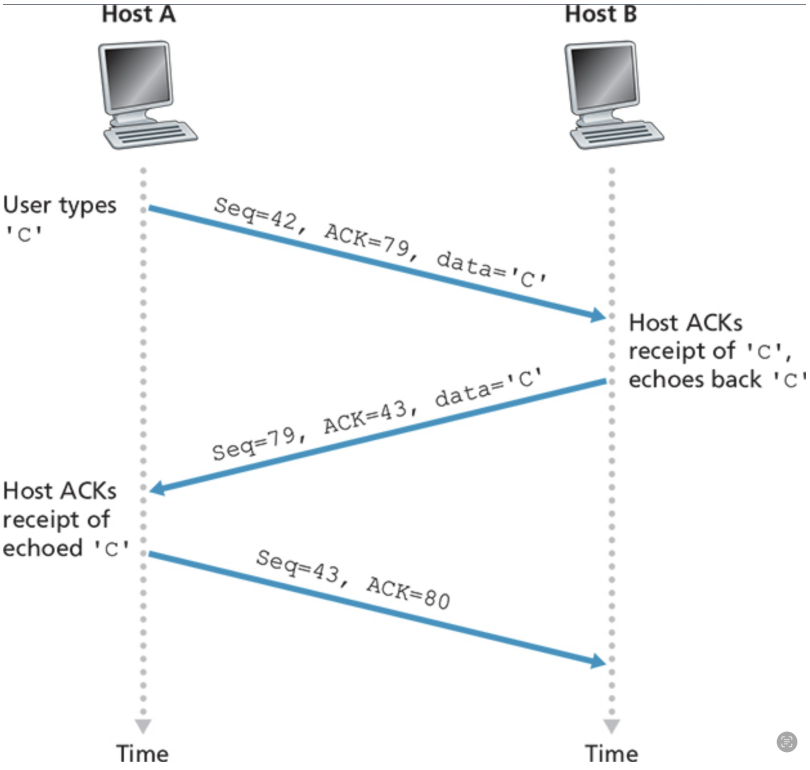


Abbildung 3: Detaillierte Darstellung des Three-Way Handshake.

B. GnuPlot Skript zur Erstellung der CWND Graphen

Hier folgt das Skript mit dem die in Abbildung 2 vorgestellten `cwnd` Graphen erstellt worden sind.

```
set terminal pdfcairo size 10cm,7cm enhanced font "Helvetica,14"
set output "cwnd.pdf"

set xlabel "Zeit (s)"
set ylabel "CWND (Pakete)"
set grid
set key top right

plot "cwnd.dat" using 1:2 with lines lw 2 lc rgb "red" title "CWND"
```

C. Nicht erfüllte Vorgaben

Aufgrund des Mangels von unter anderem der Anzahl an komplexen Formeln und Tabellen zu dem Thema des Berichts folgen hier die restlichen Vorgaben.

C.1. Formel von Bayes

Die Bayessche Formel ist eine grundlegende Formel in der Statistik. Sie beschreibt die bedingte Wahrscheinlichkeit für das Aufreten eines Ereignisses A_k gegeben Ereignis B. Ereignis A_k wird oft als die ***a priori*** Wahrscheinlichkeit bezeichnet.

$$Sei\ P(B) > 0.\ A_1 \dot{\cup} \ldots \dot{\cup} A_n\ = \Omega \text{ Dann gilt :}$$

$$P(A_k|B) = \frac{P(A_k) \cdot P(B|A_k)}{\sum_{i=1}^{n} P(A_i \cdot P(B|A_i))}$$

C.2. Beweis der Formel von Bayes

Der Satz liefert :

$$P(A_k|B) = \frac{P(A_k \cap B)}{P(B)} = \frac{P(A_k) \cdot \frac{P(B \cap A_k)}{P(A_k)}}{P(B)} = \frac{P(A_k) \cdot P(B|A_k)}{\sum_{i=1}^{n} P(A_i) \cdot P(B|A_i)}$$

□

C.3. Komplexe Tabelle

Programmiersprache	Bemerkung	Funktional	Prozedural	Objektorientiert
Java	JVM	✗	✗	✓
OCamkl	Nachfolger von StandardML	✓	✗	✗
Lisp	Common Lisp	✓	✗	✗
Pascal	ISO Pascal	✗	✓	✗
C	Low-Level	✗	✓	✗

Tabelle 1: Einige Programmiersprachen und ihre Einordnung ob Sie Funktional, Prozedural, oder Objektorientiert sind.