Technische Universität Ilmenau 15. April 2024

Vorlesung

Advanced Networking Technologies

Dr. Michael Roßberg

NEXTEX

Vorlesungsmitschrift von
Adrian Schollmeyer

Inhaltsverzeichnis

1. Routers and Switches

Router sind zuständig dafür, Pakete anhand der Informationen im IP-Header (i. d. R.) weiterzuleiten. Das passiert anhand der Forwarding-Tabelle (berechnet aus der Routing-Tabelle), die eindeutig einen Next Hop für eine Zieladresse festlegt. Router werden üblicherweise in *Points of Presence* (POPs) (z. B. DE-CIX) zusammengeschlossen. Das Internet ist also kein gut vermaschtes Netz, sondern die Vermaschung konzentriert sich eher auf wenige POPs.

1.1. Why we Need Faster Routers

Router werden i. d .R. für größe Bandbreiten ausgelegt, da diese sonst leich zum Bottleneck werden können. Schnelle Router sind wichtig, um Kapazitäten, Kosten, Größe und Stromverbrauch am POP zu senken. Entscheidend sind die Port-Kosten. Je weniger Ports benutzt werden sollen, umso schneller muss der Router werden. In POPs mit großen Routern können Pakete innerhalb des POPs mit deutlich weniger Interconnections weitergeleitet werden, während bei kleineren Routern deutlich mehr Verbindungen nötig sind (um alles miteinander zu verbinden).

Zur Steigerung von Übertragungsgeschwindigkeiten ist es bspw. auch möglich, die Wandlung der Daten in elektrische Signale an Routern für einzelne Farben in einem WDM auszulassen und stattdessen über ein Prisma die Farbe direkt an die passende Zielfaser weiterzuleiten. Dies spart teure Umwandlungen und erhöht den Durchsatz.

1.2. Why Making Fast Routers is Difficult

Moore's Law ist für CPUs nicht mehr gültig und wurde für Speicher nie erreicht. Weder Kapazität, Bandbreite noch Zugriffszeiten bei Speicher folgen Moore's Law, sondern steigen deutlich langsamer.

1.3. First Generationn Routers

Zu BNC-Zeiten gab es für jeden Anschluss ein Line Interface mit BNC-Anschlüssen, die an einer gemeinsamen Backplane angeschlossen waren. Durch die Bus-Architektur muss jedes Pakete zweimal über den Datenbus gesendet werden.

1.4. Second Generation Routers

Zusätzlich werden auf den Line Cards Forwarding Caches eingebaut, die ausgehende Interfaces zwischenspeichern, wodurch die meisten Pakete nur einmal über den Datenbus geschickt werden müssen. Durch das Caching können jedoch Reordering-Probleme auftreten, wenn ein zweites Paket dank Cache schon verschickt werden kann, während das erste Paket noch im Paketpuffer ist.

1.5. Third Generation Routers

In der Backplane wird eine Switchingmatrix gepflegt, um Pakete hin- und herzusenden. In den Line Cards wird jeweils eine Forwarding-Tabelle von der CPU gepflegt, sodass keine Zugriffe mehr auf die Backplane nötig sind, um das Zielinterface zu bestimmen.

1.6. Fourth Generation Routers

Router sind teilweise als Multi-Chassis-Systeme mit optischen Links zwischen den Chassis aufgebaut. Damit hat man im Prinzip schon ein Netzwerk innerhalb des Routers.

1.7. Generic Router Architecture

Bei jeden eingehenden Paket wird anhand der IP-Adresse der Zielport aus der Forwarding-Table ausgelesen. Header (TTLs) werden aktualisiert und über ein Switching Fabric an den Output Buffer des ausgehenden Interfaces geschickt. Anschließend wird das Paket am ausgehenden Link verschickt.

Für diese Vorgänge ist generell sehr wenig Zeit möglich; bspw. sind für 40 Gbps-Switching 8 ns Zeit für IP-Adress-Lookup verfügbar. Solche Lookups können jedoch nicht mit einfachen Hash-Tabellen gelöst werden, da diese Worst Case mehr Speicher nehmen als verfügbar ist. Durch Bäume lassen sich zwar die hierarchischen Strukturen, die für Lookups in CIDR nötig sind, speichern, jedoch sind bei Baumsuchen Speicherzugriffe nicht sehr cacheeffizient, was Lookups durch Cache Misses sehr teuer machen kann.

Für die Speicherung im Routern werden statt normalem RAM TCAMs benutzt. Dabei werden die Adressen für die Routen hinterlegt. Alles hinter der Subnetzmaske wird dabei auf „don't care“ gesetzt. Für eingehende Pakete werden dann einfach alle Lines im TCAM parallel durchsucht und der Eintrag mit der höchsten Priorität benutzt. So kann deterministisch und schnell eine Pattern-Suche gemacht werden. Nachteil dieser Technik sind hoher Energieverbrauch (es wird immer der ganze Speicher angesprochen) und hohe Kosten.

Für die Logik werden gerne FPGAs/ASICs eingesetzt.

1.8. Buffer Placement

Pakete müssen gelegentlich auch mal gespeichert werden. Dies kann entweder am Input Port oder am Output Port gemacht werden. Je nachdem, wo man dies tut, hat es verschiedene signifkant Eigenschaften.

1.8.1. Output queueing

Output queueing ist hinsichtlich der Latenz optimal. Worst Case kommt an jedem Input Port etwas für denselben Output Port an, die jedoch dort einfach gespeichert werden können, ohne andere Prozesse zu stören. Allerdings muss das Switching Fabric das n-fache (n ist die Anzahl Ports) der Line Rate schaffen.

1.8.2. Input queueing

Queues werden kurz vor dem Switching Fabric eingesetzt. Wenn jetzt also mehrere Ports einen Output Port ansprechen wollen, müssen die meisten warten. Für die Entscheidung, welches Paket wann geschickt wird, ist jetzt ein Scheduler nötig. Dafür muss die Switching Fabric aber nur so viel Durchsatz wie Line Rate haben, da an jeden Output Port höchstens mit Line Rate gesendet wird.

Input queueing ist anfällig für Head of Line Blocking. Wenn ein Paket in der Queue nicht verschickt werden kann, können auch nachfolgende Pakete dieses Input Ports (also auch solche, die an andere, freie Ports gerichtet sind), nicht verschickt werden. Also erhöht sich die Latenz und der maximale Durchsatz verringert sich.

1.8.3. Virtual Output Queueing

An die Input Queues werden jetzt mehrere Queues gesetzt (eine für jeden Output Port). Jetzt kann ein Scheduler entscheiden, welcher Port jetzt für welchen Output Port drankommt. Dies erlaubt es, dass Pakete andere überholen können, wenn sie an einen anderen Output Port gehen. Damit wird es wieder latenzoptimal.

1.9. The Evolution of Switching

Die gezeigten Verfahren sind zwar in der Theorie sehr gut, in der Praxis aber nur schwer umzusetzen. In der Praxis werden stattdessen schnellere Interfaces benutzt, um mehr Durchsatz zu erzielen.

2. Input-buffered Switches

2.1. Input-Queued Switch

Bei reinem Input-Queueing ist der switch nicht **work-conserving**. Wäre er das, würde ein Output Port, wann immer ein Paket für diesen Output Port irgendwo verfügbar ist, der Output Port nicht idle sein. Dies kann aber auftreten, wenn so ein Paket weiter hinten in einer Queue ist.

2.2. Simple Analysis

- Assumptions:
  - Time is slotted, trifft bspw. für ATM zu.

Die Frage ist jetzt, wie viele Zustände der Switch haben kann. Hierzu betrachtet man, wann welche Pakete in welche Input Queue liegen und an welchen Output Port diese geschickt werden sollen. Das ergibt für den 2x2-Switch 4 Zustände.

2.2.1. Balls-and-Bins Model

Der Backlog wird hier nicht betrachtet, sondern nur die Pakete an der HoL. In jedem Schritt wird von jeder Farbe eine Kugel entnommen (sprich: ein Pakete aus der Queue verschickt). Damit lässt sich für das 2x2-Modell die Anzahl Zustände auf 3 reduzieren, da es jetzt egal ist, an welchen Port welche Farbe in der HoL liegt. Dies lässt sich jetzt zu einer Markov-Kette umbauen.

2.2.2. Markov Chain

Bei Markov-Ketten werden alle Zustände und die Wahrscheinlichkeiten für Zustandswechsel aufgetragen. Zusätzlich wird zu jedem Zustand der Durchsatz aufgetragen. Die Wahrscheinlichkeiten für Zustandswechsel ergeben sich durch Überlegung, welche Pakete in einem Zustand verschickt werden können und wie viele Pakete welchen Typs mit welcher Wahrscheinlichkeit „nachrücken“.

In einem eingeschwungenen Prozess sollte die Wahrscheinlichkeit, sich in einem Zustand zu befinden mal dem Zustandswechsel in den Nachbarzustand genau so groß sein wie die Wahrscheinlichkeit, im Nachbarzustand zu sein mal die Wahrscheinlichkeit, zurückzuwechseln. Dies lässt sich mit der zweiten Hälfte der Kette fortführen. Wenn man nun noch fordert, dass alle Wahrscheinlichkeiten zusammen 1 ergeben müssen (was ja

gelten muss), lässt sich das Gleichungssystem lösen. Daraus ergibt sich am Ende ein Gesamtdurchsatz von 75 %.

Die Markov-Kette lässt sich sogar noch weiter vereinfachen, wenn als Zustände nur betrachtet, wie viele Pakete gleichzeitig verschickt werden können (die Zustände werden „kollabiert“). Dies macht auch die Analyse des 3x3-Szenarios einfacher. Bei der Berechnung der Wahrscheinlichkeiten muss trotzdem betrachtet werden, welche Zustände eigentlich hinter den kollabierten Zuständen liegen. nach Analyse ergibt sich hier ein Durchsatz von 68 %. Der Durchsatz fällt als weiter. Konkret: je mehr Ports, umso weniger Durchsatz gibt es, da es mehr Congestion gibt.

2.3. Closed Form Equations for Balls in Bins

Ein M/D/1-System ist ein Warteschlangesystem mit Markov-verteiltem Input, degeneriertem Output und einer Bedieneinheit (ein Port).

Nomenklatur:

- $E\{k\}$... Erwartungswert
- ρ... Durchsatz
- μ... Verarbeitungszeit
- σ... Standardabweichung von der Verarbeitungszeit

Da die Verarbeitungszeit im M/D/1-System immer gleich ist, ist $\sigma = 0$. Man kann jetzt $E\{k\} = 1$ setzen um zu betrachten, wie groß ρ am Port sein muss, damit das System instabil wird. Das Limit von 58 % zeigt auch, dass ab etwas mehr als 50 % Last der Delay steigt.

Zu beachten ist hierbei, dass es natürlich nur eine theoretische Annahme ist, dass das System ein M/D/1-System ist. Ethernet ist bspw. kein M/D/1-System und insbesondere gibt es durch unterschiedliche Paketgrößen auch unterschiedliche Zeiten, die ein Paket braucht, um verschickt zu werden.

2.4. Virtual Output Queues

Idee ist jetzt, an jeden Input Port für jeden Output eine Queue aufzubauen. Ein Scheduler entscheidet dann, welches Paket an welchen Output verschickt wird.

2.4.1. Basic Switch Model

An jedem Port i gibt es einen **Ankunftsprozess** $A_i(n)$, der Pakete für Port j empfängt $(A_{ij}(n))$, Warteschlangen $Q_{ij}(n)$, eine Switching-Matrix $S(n) \in \{0,1\}^{n \times n}$.

Der Ankunftsprozess wird wieder mit einer Matrix λ beschrieben, die die Ankunftswahrscheinlichkeit angibt, dass von Port i an Port j gesendet wird. Dabei muss sichergestellt werden, dass keine der Spalten- oder Zeilensummen > 1 ist, da sonst der Input überlastet wird.

2.4.2. Scheduling Algorithm

Ziel ist jetzt, $S(n)$ immer so zu wählen, dass der Switch möglichst 100 % Durchsatz hat. Das ist per Definition der Fall, wenn der Switch work conserving ist.

Für fixe Paketgrößen wird bei Work-Conservation auch der Delay minimiert. Für variable Paketgrößen hingegen kann es sinnvoller sein, kleinere Pakete schneller zu switchen als größere Pakete.

2.4.3. Common Definitions for 100 % Throughput

Die dritte Definition sagt aus, dass auch größere Bursts von Pakete mal über einem gewissen C sein dürfen, solange der Erwartungswert kleiner ist. Es ist also möglich, dass Bursts auch mal überschießen. In der Praxis reicht dies aus, damit nur wenig Paketverlust auftritt.

2.4.4. Uniform Traffic

Hier wird zunächt sehr unrealistisch angenommen, dass jeder an jeden mit der gleichen Paketrate sendet. Dann muss lediglich $\lambda < \frac{1}{N}$ sichergestellt werden, damit es nicht explodiert. Dieser Verkehr ist so einfach, dass quasi jeder Scheduling-Algorithmus 100 % Durchsatz schafft.

Uniform Cyclic Scheduling

In jedem Zyklus wird einfach die Switching-Matrix durchgeschaltet, sodass im Kreis jeder Input Port der Reihe nach auf jeden Output Port geschaltet wird. Dieser Scheduler ist fair und deterministisch, jedoch nicht latenzoptimal.

Wait Until Full

Es wird gewartet, bis auf jeder VOQ ein Paket anliegt und erst dann wird eine beliebige Permutation geschaltet. Dies ist zwar optimal für den Durchsatz, aber in Hinblick auf Delay sehr schlecht. Insbesondere führt das dazu, dass sich der Delay erst verbessert, wenn die Last *steigt*.

Uniform Random Scheduling

Abwandlung von UCS, bei der zu jedem Zeitpunkt eine zufällige Permutation gewählt wird. Dies erlaubt wieder die Analyse mit Markov-Ketten (mit unendlich vielen Zuständen), da es sich um ein M/M/1-System handelt. Durch Analyse der lokalen Gleichgewichtsbedingung für jeden Zustand lässt sich ein Erwartungswert für jeden Zustand berechnen. Damit wiederum lässt sich berechnen, wie lang ein Paket im System verbleibt (sprich: der Delay).

2.4.5. Non-Uniform Traffic with Known Traffic Matrix

Der Traffic ist zwar nicht uniform verteilt, aber die Traffic-Matrix λ ist bekannt (fix).

Bei einem uniformen Schedule wird das System sofort instabil. Es besteht jedoch die Möglichkeit, verschiedene Permutationen mit verschiedenen Wahrscheinlichkeiten zu schalten. Dies kann man bspw machen, indem eine feste Abfolge von Schedules in Reihe oder zufällig durchgeschaltet wird. Letzteres macht wieder eine Analyse mit Markov-Ketten möglich.

Die Kernfrage ist nun, ob eine solche Zerlegung sich verallgemeinern lässt. Tatsächlich lässt sich dies algorithmisch lösen.

2.4.6. Double Stochastic Matrices

Bei einem doppelt stochastischen Prozess können sich die Wahrscheinlichkeiten ändern. Dies tritt beispielsweise beim Besuch von Websites auf, wo die Wahrscheinlichkeit, dass Pakete verschickt werden, von der Wahrscheinlichkeit abhängig, wie häufig der Nutzer auf Links klickt. Es gibt also einen übergeordneten Prozess, der bestimmt, wie hoch die Wahrscheinlichkeiten sind, dass ein untergeordneter Prozess (hier: der Versand von Paketen) bestimmte Wahrscheinlichkeiten annimmt.

Eine zulässige Verkehrsmatrix ist doppelt substochastisch. Durch geschicktes Aufrunden lässt sich diese in eine doppelt stochastische Matrix überführen. Diese Matrizen wiederum können als Linearkombination endlich vieler Permutationsmatrizen dargestellt werden. Genau genommen werden „nur“ höchstens quadratisch viele benötigt.

A. Buzzword Of The Day

In den Vorlesungen wird ein „Buzzword“ bzw. Begriff aus der Welt der Netzwerktechnik genannt. Dieser Begriff ist von den Hörenden zu recherchieren und wird in der anschließenden Vorlesung diskutiert. Hier sind einige Begriffe samt Diskussionen und zusammengetragen.

A.1. Cut-Through Switching [1, 3]

- Was bedeutet das?
- Ist das eine sinnvolle Technologie?

Cut-Through Switching vermeidet Latenz, indem ein Paket bereits an den ausgenden Port verschickt wird, sobald die Zieladresse und der ausgehende Port bestimmt werden konnten. Das Paket kann also weitergeleitet werden, noch bevor es vollständig empfangen wurde.

Cut-Through Switching erfordert, dass die Linkgeschwindigkeit des ausgehenden Ports mindestens so groß ist wie die des eingehenden Ports, da sonst das Paket wieder gepuffert werden muss. Ein weiteres Problem ergibt sich dadurch, dass die Checksumme des Ethernet Frames nicht geprüft werden kann, bevor das Paket verschickt wird (einerseits, da die Payload zu diesem Zeitpunkt noch nicht ganz bekannt ist, andererseits, da die Checksumme erst am Ende des Ethernet Frames steht). Daher leitet ein Cut-Through Switch auch beschädigte Frames weiter, die ein Store-and-Forward Switch verwerfen würde.

Cut-Through Switching kann auch in einer adaptiven Variante betrieben werden. In diesem Fall wird es nur benutzt, solange Fehler am Link nur selten passieren. Steigt hingegen die Fehlerrate am Port, wird auf Store and Forward umgeschaltet, um unnötiges Weiterleiten von Paketen zu vermeiden.

Eine andere Erweiterung ist Fragment-Free-Cut-Through, bei der nur die ersten 64 Bytes eines Ethernet Frames geprüft werden. Dies beruht auf der Erfahrung, dass Framefehler meistens nur innerhalb der ersten 64 Byte auftreten.

In einem RZ-Setup ist üblicherweise für jedes Rack ein Top-Of-Rack-Switch verbaut, an dem die Server des Racks angeschlossen sind und der das Rack mit dem Backbone verbindet. Das Problem dabei ist, dass Pakete fast immer im Top-Of-Rack-Switch gepuffert werden müssen, weil sie von den Servern zeitgleich Pakete empfangen. In so einem Fall bringt Cut-Through also nichts. Änlich bringen die Latengewinne im Nanosekundenbereich selbst für latenzkritische Anwendungen (wie bspw. PTP) nicht viel, da die

Latenzen an den Endsystemen meist deutlich größer sind und somit die Gewinne durch Cut-Through-Switching zunichte machen.

A.2. Hairpin Turn

Stichwortverzeichnis

Literatur

[1] *Cut-through switching*. In: *Wikipedia*. Page Version ID: 1205281012. 9. Feb. 2024. URL: https://en.wikipedia.org/w/index.php?title=Cut-through_switching&oldid=1205281012 (besucht am 08. 04. 2024).

[2] Michael Roßberg. "Advanced Networking Technologies". Vorlesung. Vorlesung. 2024.

[3] *Switching*. Elektronik-Kompendium. URL: https://www.elektronik-kompendium.de/sites/net/0907141.htm (besucht am 08. 04. 2024).