High Performance Networking

Korrekturen des Netzwerkstacks um den Anforderungen in Hochgeschwindigkeitsnetzen Rechnung zu tragen

Hagen Paul Pfeifer
hagen@jauu.net

http://protocol-laboratories.net

27. November 2006

Agenda

1. IEEE 802.3 Anpassungen/Erweiterungen
   - Interrupt Coalescence
   - Dynamische Interrupt Moderation
   - Jumbo Frames
   - SG (Scatter/Gather), Multicast Filter, TCP Checksum Offload, 802.3x Flow Control, TSO (TCP segmentation offload), TOE (TCP Offload Engine), GSO (Generic Segmentation Offload)
2. Staukontrollalgorithmen
   - Tahoe (historische Einführung)
   - Reno, NewReno
   - SACK, FACK
   - BIC, CUBIC, Hybla, Westwood

- Linux Staukontrolle
- Paced TCP

3. Warteschlangentheorie und -praxis
   - Einführung Warteschlangentheorie
   - Warteschlagen bei Routern (Drop Tail, RED, WRED)

Kapitel 1
IEEE 802.3 Anpassungen/Erweiterungen

Interrupt Coalescence

- Bei 1538 Byte Frame und GigE haben wir ein Zeitdelta von $12,3\mu s$ pro Interrupt (oder 81000 Pakete pro Sekunde)
- Worst Case: CPU-Livelock oder packet drop
- Annahme: es werden in einem Zeitdelta mehr als ein Paket erwartet
- Ansatz: ein Interrupt für mehreren Pakete im Zeitdelta
- Reduziert den Interruptunterbrechungoverhead
- Vendor Default oft Priorisierung des Durchsatzes gegenüber geringer Latenz
- Werte tunen in Abhängigkeit des Verwendungszwecks (Server, Testbox, . . .)
- `drivers/net/e1000/e1000_main.c:1000_configure_rx()`
- Problematisch:
  1. Verzögerungen
     a) TCP ist „selbsttickendes“ Protokoll

b) Führt zu Häufung von ACKs und Datensegmenten

2. Änderung der Paket-Streuung (wichtig für Timingverhalten höherer Protokolle)

Interrupt Coalescence - Tuning

- Treiber Variablen (für e1000, `e1000_param.c`):
  - `InterruptThrottleRate` – maximale Anzahl von Interrupts pro Sekunde
  - `RxIntDelay` – minimale Verzögerung eines Paketes
  - `RxAbsIntDelay` – maximale Verzögerung eines Paketes
- Aufgepasst: wenn `RxIntDelay` inkrementiert - Deskriptoren für Ringpuffer auch inkrementieren
- Mehr zu Variablen: `drivers/net/e1000/e1000_param.c`
- Intel Untersuchungen zu günstiger InterruptThrottleRate:
  - Microsoft Windows: 4000 bis 12000
  - Linux: 1000 bis 8000

Dynamische Interrupt Moderation

- Dynamische Anpassung
- Grundlage: Größe und Anzahl Pakete zwischen zwei Interrupts
- Vorteile:
  - Moderation in Abhänigkeit des Paketstroms
  - Konvergiert schnell
- Implementierung:
  - Separation in `lowest_latency, low_latency, bulk_latency`
- Implementierung e1000:

  `drivers/net/e1000/e1000_main.c:e1000_update_itr()`

Jumbo Frames

- IEEE 802.3 Frames im Ethernet: max 1518 Byte
- Über 1518: Jumbo Frames
- Nicht standardisiert (9 kB, aber auch andere exotische größen!)
- Unterstützung durch Netzkomponenten
- LAN: gut (NFS, 8K Frames) — WAN: gut überlegt einsetzen
- Große Frames: weniger Interrupts, weniger Overhead
- Früher wurde versucht damit Mangel beim CA zu umgehen (man wusste es nur nicht . . .)
- Alteon Networks Paper:
  - CPU Belastung bis 50% minimieren
  - Datentransferrate bis 50% steigern
- Matt Mathis: Raising the Internet MTU – NO TINYGRAMS ;-)

- Ein paar Adapterdaten:
  - 8169: bis 7200 Bytes („Baby Jumbo Frames“)
  - bnx2 (v1.4.45): 9014 Bytes
  - tg3 (Broadcom Tigon3, v3.67): 9000 Bytes (MTU)
  - e1000: 16128 Bytes
  - Achtung: Adapterspezifische Unterschiede (Ethernet Controller Chip)!

802.3x Flow Control

- Sender ist schneller als Empfänger
- Eine Endstation sendet Pause Frame
- Verbot für eine gewisse Zeit Daten zu senden (ausser Kontroll-Frames)
- Bidirektional und Full-Duplex Operation
- Einer speziellen MAC-Adresse oder Multicast Adresse (01-80-C2-00-00-01)
- Problematisch: Switch Flow-Control, Multicast und ein langsamer Empfänger
- Zusammenspiel Linux NAPI und Flow Control
- Optimal: nur bei single Links
- Optional – kann man(n) aus- oder anschalten
- Implementierung: siehe Linkliste im Anhang

TSO

- TCP Segmentierung durch NIC
- `64k -> 1500`
- NIC bekommt Zeiger auf großen Datenbereich sowie IP/TCP Header Template
- Nicht mittelbar um Durchsatz zu erhöhen (Limit erreicht NIC auch)
- CPU wird spürbar entlastet:
  - Weniger Daten via DMA werden übertragen (über PCI Bus)
  - Weniger CPU-Zyklen für die Bearbeitung (für Segmentierung)
- `NETIF_F_TSO`: TCP Checksum Offloading und Scatter Gather Support
- Wenn von NIC unterstützt: `ethtool -K eth0 tso {on|off}`
- Nette Regel: CPU von 1KHz benötigt um 1Kbit/s TCP-Overhead zu verarbeiten
  (2GHz-CPU für Gigabit Ethernet) (Chinh Le)

Checksum Offload, TOE, . . .

- Checksum Offload
- TCP Offload Engine (TOE)
- Generic Segmentation Offload (GSO)
- UDP Fragmentation Offload (UFO)

Kapitel 2
Staukontrolle

Historisch

- Mit ansteigender Benutzung traten immer häufiger Überlastsituationen ein
- Oktober 85 kam es zu ersten Zusammenbrüchen aufgrund von Überlast
- TCP konnte darauf nicht reagieren – es erkannte dies nicht (wie UDP)
- Oberstes Ziel: Netzwerkstruktur intakt halten
- Mann der ersten Stunde: Van Jacobson (Header-Komprimierung, Slow Start, Congestion Avoidance, . . .)

Hintergrund – Implementierung

- Annahme:
  - Paketverlust aufgrund von Stausituationen
  - Wahrscheinlichkeit von zerstörten Paketen geht gegen 0 (naja das ist nicht richtig)
  - Router wissen darüber und behandeln dies intelligent (mehr oder weniger)
- TCP – ACK-Triggert:
  - Self-Clocking: der Empfänger kann ACK's nicht schneller schicken als Senderfluss
  - Es reagiert somit auf Verzögerungs- sowie Bandbreitenänderung
  - Alternativer Ansatz: Timesloted Verfahren
- „Jacobson Patch“ (in Tahoe):
  - Slow-Start:
    - Kernelvariable für jede Verbindung: `cwnd` (initialisiert mit eins oder zwei)

- Wenn Paketverlußt: cwnd auf ein Paket setzten
- Jedes empfangene ACK inkremmentiert `cwnd` um eins
- Beim senden: `min(cwnd, gewaehrtes window)`

- Congestion Avoidance:
  - Inkrementierung um ein Segment pro RTT
  - Slow Start Threshold wird benutzt um Unterscheidung zu treffen (Slow-Start oder Congestion Avoidance)
- Zusammenspiel:
  - Wenn drei Dup-ACKs eintreffen: `ssthresh` auf hälfe von `cwnd`, `cwnd` auf `ssthresh` plus drei (`TCP_FASTRETRANS_THRESH`)
  - Bei Timeout: `ssthresh` auf Hälfte von `cwnd` und `cwnd` auf eins
- Fast Retransmit
  - Fehlendes Paket übertragen: Fast Retransmit

Tahoe und Reno

- Tahoe:
  - `cwnd` bei Drop auf 1 (egal ab Dup-ACK oder RTO) – kein Fast Recovery
  - Verringerung der Datentranferrate von 50% bis 75% bei 1% Packetverlust
- Zwei Fälle werden unterschieden:
  1. Retransmit Timeout
     - Stausituation akkut
  2. Duplicate ACK
     - Stausituation verkraftbar
- Probleme:
  - Paketverlust von mehr als ein Paket im aktuellen Fenster
  - Dies triggert oft ein RTO
- Fast Recovery

- Verhindert das der Link entleert
- Bei drei Dup-ACKs reduzierung `ssthresh` auf $\frac{cwnd}{2}$
- Deutliche Verbesserung wenn ein Packet verloren gegangen ist

NewReno

- Reno plus verbeserter Fast Recovery
- Verbesserung bei Verlust von meheren Paketen im Fenster
- Beendet Fast Recovery erst wenn komplettes Window geACKed
- Funktioniert auch wenn Empfänger kein SACK kennt
- RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm

TCP Extensions for High Performance

- Performance Probleme:
  - Window Size: 2^{16} (Bei 100ms RTT: $\frac{2^{16}}{0.1} \Rightarrow$ 0.7MBps)
  - Altes Paket auf Link (Gigabit: $\frac{2^{31}}{125000000} \Rightarrow$ 17.2s)
  - Ungenaues Retransmission Timeout
- Neuerungen:
  - TCP Window Scale Option
  - RTTM – Round-Trip Time Measurement
  - PAWS – Protect Against Wrapped Sequence Numbers
- RFC 1323

SACK und FACK

- SACK (Selective Acknowledegment)
  - Standard TCP reagiert schlecht wenn mehr als ein Paket in einem Fenster verloren sind
  - Cumulative ACKs ermöglichen nur eine Information per RTT
  - Alternativ und Naiv: bei DUP-ACK Fenster komplett neu zu übertragen
  - SACK Empfänger informiert Sender welche Packete empfangen wurden
  - Muss beim Verbindungsaufbau ausgehandelt werden (SYN,SYN/ACK)
  - Implementierung auf Sender- sowie Empfangsseite
  - Default bei Linux und Microsoft (seit Windows 98)
- FACK (Forward Acknowledgment)
  - Baut auf SACK auf
  - Betrachtet nicht bestätigte Blöcke als verworfen

Probleme

- Van Jacobson hat für zwei Jahrzehnte Problem beseitigt (Satellitenlinks etc. mal abgesehen)
- VDSL2, Gigabit, 10 Gigabit erobern den breiten Markt
- Verbindung mit hohem BDP (Bandbreitenverzögerungsprodukt)
- BDP (Bandwith Delay Produkt)
  - Produkt aus Bandbreite und RTT (latür ;-)
  - RTT mit Ping (Achtung: ICMP erfährt oft Begrenzungen)
  - Bandbreitenbestimmung komplizierter (kommt später)
  - Signalausbreitung ca. 60% der Lichtgeschwindigkeit
    - 3,000km: 15ms
    - 6,400km: 33 ms
  - Link Serialization fügt weitere Latenz hinzu:

- Paket kann erst versendet werden wenn letztes Bit angekommen ist
- 56 Kbps: 214 ms
- 1.5 Mbps: 8 ms
- 100 Mbps: 120 μs

- Sally: 1500 Byte und 100ms um 10Gb „steady-state“ zu erreichen:
  - 83,3 Segmente
  - 5 Millionen Pakete ohne Drop `oder`
  - $1\frac{2}{3}$ Stunden ohne packet drop
- Kurzlebige Verbindungen erreichen MAX nie (z.B. HTTP)
- Annahme: Paketverlust == Stausituation
  - Die ist aber bei Radio Link nicht der Fall (802.11)
  - Westwood
- $\Rightarrow$ es müssen (wieder) Anpassungen im TCP-Protokoll getroffen werden!

BIC, CUBIC, Hybla, Westwood

- HighSpeed TCP (Sally)
  - `cwnd = cwnd + a(cwnd) / cwnd` (bei ACK-Empfang)
  - `cwnd = (1-b(cwnd)) * cwnd` (bei packet drop)
  - `a` und `b` abhängig von `cwnd`
  - 10Gb bei maximal einmal in 12 Sekunden
- Scalable TCP
  - `cwnd = cwnd + 0.01`
  - `cwnd = cwnd - 0.125 * cwnd`
  - Wie bei HighSpeed TCP sind Schwellwerte vorhanden, erst wenn diese erreicht sind werden die modifizierten Congestion Avoidance Algorithmen verwendet
- BIC und CUBIC

- Congestion Control betrachtet wie ein Binary-Search Problem
- Konvergiert schnell wenn großer Diff und langsamer wenn nah am Ziel
- BIC problematisch bei kurzen RTT: CUBIC (nebenbei auch vereinfachter Algorithmus)

- Westwood+
  - Packetverlust != Stausituation
  - Berechnet End-To-End Bandbreite der Verbindung (ACK Enpfangsrate; kommt später genau)
- Ansatz der sich stärker unterscheidet:
  - Packet drop beachten
  - Aber auch: Queueing Delay in Berechnung einbeziehen!
  - Beispiele: Fast TCP oder Vegas TCP
- Die Algorithmen sind stark verkürzt, für eine ausführliche Beschreibung wird auf die Entwicklerseiten verwiesen!

Linux Staukontrolle

- Linux unterstützt folgenden Algorithmen (v2.6.19-rc6):
  1. Binary Increase Congestion (BIC) control
  2. CUBIC TCP
  3. TCP Westwood+
  4. H-TCP
  5. High Speed TCP
  6. TCP Hybla
  7. TCP Vegas
  8. Scalable TCP
  9. TCP Low Priority
  10. TCP Veno
  11. (TCP Compound)

Paced TCP

- Gewöhnlich für LFNs
- Router Warteschlangengröße - diese sind oft kleiner als BDP!
- TCP ist ACK-triggered – und aus diesem Grund oft „bürsti“
- Normalverteilung fehlt an dieser Stelle
  - Delayed ACKs und ACK-Compression
  - NIC Problematik (siehe NIC Folie)
- Idee: statt am Anfang Burst zu versenden - Daten gleichverteilt auf RTT versenden
- Aber oft auch:
  - geringerer Durchsatz
  - größere Latenz
- Abschließend: folgende Forschungsergebnisse im Blick behalten!

Kapitel 3
Warteschlangentheorie und -praxis

Warteschlangentheorie

- Gegebenen Zeitintervall der Ankunftstrom auf Bedienstelle größer ist als Abgangsstrom
- Beispiel:
  - Fahrzeugstau (Transportwesen)
  - Kasse im Einkaufsmarkt
  - Sylvester Telefonat unter den Linden (Telekommunikation)
  - Und: Netzwerktheorie, Scheduler, . . . ;-)
- Einreihung in Warteschlange wenn Bedienstelle besetzt
- Zufällige Einflüße (Zeitintervall beim Eintreffen)
- Theorie versucht dies nun abzubilden

Warteschlangentheorie – Bediensystem

- Besteht aus
  1. Ankunftsstrom
  2. Warteraum
  3. Bedienstelle
  4. Abgangsstrom
- Ankunftsstrom:
  - Ankunftsrate: mittlere Anzahl von Ankünften pro Zeiteinheit
- Warteraum
  - Bis Bedienung warten
  - Größe des Raumes begrenzt oder unbegrenzt (Router)
  - englische Schlange – italienische Schlange
- Bedienstelle:

- Anzahl Bedienstellen: 1 bis ∞
- Bei $n \geq 2$ Parallel (Kassen) oder Seriell (Ampeln)
- Bedienrate: wie viele Elemente pro Zeiteinheit bedient
- Belegungsgrad: Verhältnis aus Ankunftrate und Bedienrate
- Vergrößerung wenn: Belegungsgrad $\geq$ Bedienstellen (gleich? ja, wenn Ankünfte und Abgänge nicht synchronisiert!)

- Abgangsstrom:
  - Wird in der Regel als unendlich groß betrachtet
- Wartediziplinen
  - FIFO, LIFO, SIRO, PRI, RR, ...

Queue Managment

- Sicht der Netzkoppelelementen (hier Router)
- Paketverwurf als Standardmittel zum Zweck
- Drop Tail
- RED (Random Early Detection)
- WRED (Weighted Random Early Detection)

Drop Tail

- Wenn Warteschlange voll: packet drop
- Default bei vielen Routern – auch wenn sie mehr können
- Netzverkehr wird bei packet drop gleich behandelt
- Problem: Globale Synchronisierung bei TCP
  - Viele TCP Sessions gehen simultan in Slow Start
  - Zeitspannen mit Bursts
  - Zeitspannen mit wenig Link Auslastung

Random Early Detection

- Pakete werden vor eigentlicher Stausituation verworfen
- RED-Router versuchen Puffer zu kontrollieren
- Reaktive Maßnahme (im gegensatz zu Drop Tail)
- Annahme:
  1. großer Prozentsatz der Transportprotokolle mit Staukontrollfunktionalität (TCP, DCCP kontra UDP, NetWare oder AppleTalk)
  2. Packet drop als Stausignalisierung
- Sally Floyd und Van Jacobson
- Implementierung (Cisco Variablenbezeichnung):
  - `minimum threshold`
  - `maximum threshold`
  - `mark probability denominator`

Verschiedenes

- ICMP Source Quench
- Explicit Congestion Notification (ECN)
  - Router haben oft besseren Überblick auf Netzsituation als Endknoten

Kapitel 4
Prolog

Ende

- Danke für eure Aufmerksamkeit!
- Fragen – Anregungen – Bemerkungen?

Weiterführende Informationen

- Weiterführende Informationen:
  - What is inside a router?
  - Cisco: Weighted Random Early Detection
  - Sally: RED Queue Management
  - Matt Mathis: Raising the Internet MTU
  - Intel Gigabit Ethernet Controllers Application Note
  - Transmission versus Propagation Delay Java Applet
  - Probleme mit Ethernet Flow Control
  - Vendors on flow control
  - TCP Timestamping and Remotely gathering uptime information

Kontakt

- Hagen Paul Pfeifer
- E-Mail: hagen@jauu.net
  - Key-ID: `0x98350C22`
  - Fingerprint: `490F 557B 6C48 6D7E 5706 2EA2 4A22 8D45 9835 0C22`

Kapitel 5
Vortrags-Sicherungs-Folien

ACK Behandlungen

- Linux verwendet bei Verbindungsaufbau Quick-ACKs
- Normal: Delayed-ACKs wegen Piggybacking und der Bestätigung von mehr als ein Paket
- Maximale Zeit für Delayed-ACK: 0.2s
- Minimale Zeit für Delayed-ACK: 0.04s
- `net/ipv4/tcp_output.c:tcp_send_delayed_ack()`

WRED

- Einsatz in Core-Routern
- Priorisierung des Verkehrs
- Für jede Verkehrsklasse können eigene Schwellenwerte definiert werden
- Nicht IP-Verkehr bekommt die niedrigste Priorität

Fairshare 1 kontra n Streams

- Additive Increase
  - $a = 1 * MSS$
  - $a = n * MSS$
- n ist natürlich aggressiver
- Höhere Fairness gegenüber dem Bottleneck-Link
- Wenn limitiert durch Socketpufferspeicher - n kann dann BDP füllen
- Last but not least: CPU limited - bei SMP-Systemen bessere Leistung

Bottleneck Bandwidth

- Kapazität einer Verbindung
  - Maximaler Durchsatz von Sender zu Empfänger
  - Unabhängig von der aktuellen Auslastung
  - Verbindung mit kleinsten Transferrate ist verantwortlich
  - Maximale Durchsatz wenn kein „Cross-Traffic“
- Werkzeuge:
  - End-To-End Network Kapazität
    - Pathrate