Unix Netzwerk-Stack Implementierungen

Entwicklungstrends der TCP/IP-Suite

„Fragmentation is like classful addressing – an interesting early architectural error that shows how much experimentation was going on while IP was being designed.“ — Paul Vixie

Hagen Paul Pfeifer <hagen@jauu.net>

27. April 2006

TCP Hbt

„zum langesamen Einschwingen . . . “

- Endknoten versus Routeroptimierungen
- „Das Ebenenmodel der Optimierungen“
  - Hardwareebendee
  - Betriebssystemebene
  - Anwendungsebene
- Tendenzen der Hardwareentwicklung (Multi-Core)
- Prozessor, Hard-Disk und Ram Grenzen werden erreicht und überschritten
- Aufgabe NIC:
  - Möglichst hoher Durchsatz
  - CPU und Bus-Auslastung so gering wie möglich zu halten

Vortrags-Fahrplan

1. Allgemeine Herausforderungen
2. Ältere Kamelen der Protokoll-Reformation
3. Hot Off The Network Source Code Cuisine
4. Betriebssytemdifferenzen
   - Net,Free,Dragonfly und OpenBSD
   - Solaris
   - Linux (latür)
   - Windows

Allgemeine Herausforderungen

- RFC sind teilweise zu bestehenden RFC's inkompatibel
  - es wird viel Aufmerksamkeit verwendet um dies zu verhindern, aufgrund der Komplexität ist dies nicht immer möglich
  - Evolution bedarf Mutation
- RFC leben von einer korrekte Implementierung und an die Einhaltung derer (einig Herstellen verlassen an einigen Stellen den Pfad; dies endet oft in riesigen Problemen (z.b. ECN))
- Höhere Bandbreite verlangt an einigen Stellen Modifikationen im Stack (um „unvorhersehbare"Situationen in den RFC's zu fixen)

  (z.B PAWS. PAWS garantiert keine wrapped ACK's in Netzen mit hoher Bandbreite – naja PAWS ist eh merkwürdig und ein schlechter Workaround)
- X-WiN, GEANT2, Internet2

Zero Copy Mechanismen

- Sinnfreie Kopieraktionen zwischen Userspace - Kernelspace können werden vermieden
- Erreicht durch Schlüßelfeature wie DMA und MMU
- Einläuchtendes Beispiel: `sendfile(2)`
- Älteres Konzept (1995)
- Passt an dieser Stelle am besten: `TCP_CORK`

NAPI(Linux) und Interrupt Mitigation

- Ein IRQ pro Packet performt schlecht
- Automatisches Umstellen auf Poll-Modus (Interruptline abgeschaltet, Softirq)
- Veränder das Interrupt/Packet Verhältniss
- Achtung: Latenz
- Tipp: e1000 Implementierung

DMA/RDMA

- DMA
  - I/O-Daten direkt in Hauptspeicher
  - NIC: asynchronen Ansatz
  - ...
- RDMA - Remote Direct Memory Access
  - Kopieroperationen werden vermieden: „ein Computer kopiert direkt in den anderen“
  - Ansatz um Netzwerkbandbreite und CPU-Leistung/Speicherbandbreite auszugleichen
  - RDMA over TCP/IP
  - Einsatz: z.B .InfiniBand

Jumbo Frames

- Ethernet: MTU von 1500 byte
- Jumbo Frames: MTU bis 9000 byte (>12000 bytes verringerte Leistunsfähigkeit des CRC)
- bis 50% Steigerung von Durchsatz und bis 50% Verringerung der CPU Auslastung
- Pro Paket Bearbeitung verringert sich (Interrupts, Down- und Upstream Paths)
- Achtung: Vorteile auch im WAN: MSS entspricht MTU (minus TCP/IP Header)
- (Nur am Rande: wenn die Firewall alles droped was nach ICMP auschaut wird es Probleme geben - aber Hauptsache das Netz ist sicher ;-)
- Nachteil: jeder Herstellen kocht sein eigenes Süpchen
- Einsatzgebiet: Lokal, Cluster und Forschungsnetze (z.B. NFS: 8,192-byte Blöcke)

Dynamic Right Sizing

- Empfangspuffer konservative Größe (Embedded Geräte)
- Empfänger bestimmt cwnd des Senders
  - granularität Timer
  - Bulk Transfer
- Empfänger passt eigene Windowsize an
- Sender ist Staukontrollfenster kontrolliert – nicht Flow-Kontroll-Window
- Ergebnis:
  - Durchsatz wird begrenzt durch Bandbreite und nicht durch magische Konstanten
- BTM: Linux als General Purpose OS hat einen riesigen Netzwerkstack – Embedded Hersteller macht das nicht gerade glücklich

Dynamic Right Sizing - von Hand ;-)

(für die Netzwerkhacker mit GE ;-)

```
ifconfig eth0 mtu 9000
sysctl  -w net.ipv4.tcp_sack=0
sysctl  -w net.ipv4.tcp_timestamps=0
sysctl  -w net.core.rmem_max=524287
sysctl  -w net.core.wmem_max=524287
sysctl  -w net.core.optmem_max=524287
sysctl  -w net.core.netdev_max_backlog=300000
sysctl  -w net.ipv4.tcp_rmem="10000000 10000000 10000000"
sysctl  -w net.ipv4.tcp_wmem="10000000 10000000 10000000"
sysctl  -w net.ipv4.tcp_mem="10000000 10000000 10000000"
sysctl  -w net.ipv4.tcp_tw_recycle=1
sysctl  -w net.ipv4.tcp_tw_reuse=1
```

TOE - TCP Offload Engine

- Viel CPU-Zyklen im RX/TX Pfad
- Verlagerung auf NIC Hardware Ebene (Protokoll Unterstützung auf dem Karten)(DAG-Cards)
- Beispiel: Alacritec Gigabit Accelerator TNIC
  - TCP/IP Checksum
  - TCP Segmentation und Reassembly
  - On-Chip ACK processing
  - On-Chip slow-start processing
  - 66 MHz/64-bit
  - Grundsätzlich geht aber auch mehr: z.B. SBE toePCI-2Gx

TOE - TCP Offload Engine

- Aber:
  - Linux Feature werden umgangen: Netfilter, Trafficshaping
  - Aber: Separation möglich: Verbindungsaufbau, Offload unterbinden
  - Rewrite des Stacks notwendig
  - Debugging wird komplizierter (`netstat -ni`)
  - NDIS-5 Unterstützung
  - Patentbelastet (SPTO 20040042487)!

TCP Segmentation Offloading (TSO)

- Grosse Datenpakete müssen segmentiert (normalerweise in der TCP Implementierung `tcp_output.c`)
- Wenn Arbeit von der NIC übernohmen wird: TSO
- e1000 Test von Scott Feldman (ein Pseudo TCP-Header und 64K Daten)
  - `Tx TCP file send long (Tx only)`
  - `w/o TSO: 940Mbps, 40% CPU`
  - `w/  TSO: 940Mbps, 19% CPU`
- NIC muss Scatter/Gather unterstützen
- Apropos: IO Scatter/Gather der glibc: readv()/writev()
- Linux: tg3, e1000, ixgb, s2io, bnx2, qeth, tg3, 8139cp

Net Channels - Van Jacobson

- Van Jacobson Gedanken für ein performanteren Netzwerkstack
- Problem:
  - aktuelles Hardware Problem: Memory Latez (Ram (DDR333), Cache)
  - Cache Performance wichtiger Faktor (`-Os` teilweise schneller Code als `-O3` da Cacheline freundlich)
  - Viele Cache Penalties durch die Bearbeitung in verschiedenen Schichten
  - Viele Locks im RX/TX Pfad
  - Viele doppelt verkettete Listen welche weniger gut mit dem Cache interagieren
- Multi-Core und SMP Systeme verstärken diese Problem

Net Channels - Van Jacobson

- Ansatz:
  - Locks verringern
  - Shared Daten verringen
  - Ausführung auf derselben CPU
- Wie:
  - Verlagerung der Arbeit in den Userspace
  - Producer/Consumer
- Probleme mit diesem Ansatz:
  - erstmal viele! ;-)
  - tcp/ip stack im userspace

Solaris

- Ursprung der Netzwerkstackes: BSD (BSD-Reno, latür)
- STREAMS
  - Message Passing Interface
  - ungünstig für SMP da keine CPU Affinität
  - unglücklich da STREAMS Initialisierung teure Operation ist (in Hinblick auf kurz-lebige Verbindungen)
- FireEngine[TM]
  - Funktionsaufruf basiertes Interface
  - Multi-Threaded Netzwerkmodule
  - CPU lokale Synchronisation (vertical perimeter (squeue))
  - Verbindung wird CPU gebunden (cache lokal, IP-Classifier)
- Yosemite

BSD

- 30 jahre alte codebasis: schwaerfalliger, teilweise schlecht wartbarer code
- DragonFly
  - CPU lokale Threads (nett für unsere Cachelines)
  - Multi-threaded Network Stack
- NetBSD
  - Hardware Checksumming
  - NewReno, SACK, und TCP Westwood+
  - Skaliert sehr gut
- FreeBSD skaliert auch gut!
- OpenBSD verfolgt andere Ziele . . . ;-)

Windows

- Scalable Networking Initiative
  - Microsoft Chimney Offload Architecture (TOE Interface)
  - Receive Side Scaling (CPU Balancer)
- 1323, SACK, LSO
- mind. Windows Server 2003 (Checksuming und Large-Send Offload(NDIS 5))
- Automatische Window Size Anpassungen (Media-Size)
- Ipv6
- Grundsätzlich debeckter Informationsfluß (im Zweifel: `softice tcpip.sys` :-)

GNU/Linux TCP

- Grundzüge der Linux TCP/IP Implementierung
  - schnelle (schnellste?) Stack Implementierung
  - stabilität, fairnis und
  - general purpose
- Problembereiche:
  - Fat Pipes
  - Leaky Pipes
- Timer Granularität (und Anzahl der Timer (Molnar/Gleixner Patch))

GNU/Linux TCP

- splice() und tee() - Zero-Copy die zweite
  - kommende neue Syscalls (2.6.17)
  - KS-Puffer welcher vom User kontrolliert wird
  - Wenn Daten VON irgenwo NACH irgendwo kopiert werden
  - `splice()` von/nach ks-buffer von/nach fd
  - `tee()` „kopiert buffer“

GNU/Linux

- Linux wird häufig als BS der Wahl eingesetz wenn es um die Implementierung und Evaluation neuer Netzwerkfeatures geht
  1. Kernel unterliegt GPL (dürfte sich herumgesprochen haben ;-)
  2. Codebasis (aber auch Qualität) ist gut bis sehr gut
  3. Gut Dokumentiert
  4. GNU/Linux liefert eine ausgesprochen potente Entwicklungsumgebung
  5. All das predesteniert Linux als BS der Wahl für die Implementierung
- Nichtzuletzt profitiert Linux von dieser regen Forschungstätigkeit!
- (das war die BWL-Folie in diesem Vortrag ;-)

GNU/Linux Sahnestücke

1. TCP Metrics
   - `net/ipv4/tcp_input.c:tcp_update_metrics()`
   - gespeichert wenn Session den Zustand `TIME-WAIT` oder `CLOSE` erreicht - sich also erfolgreich beendete.
   - Unter anderem:
     - `RTAX_MTU`
     - `RTAX_WINDOW`
     - `RTAX_RTT`
     - `RTAX_SSTHRESH`
     - `RTAX_ADVMSS`
     - `RTAX_HOPLIMIT`

GNU/Linux Sahnestücke

1. Plugable TCP Congestion Control
   - Dynamisch austauschbarer Staukontroll-Algorithmus
   - Default: NewReno (Jacobson's slow start and congestion avoidance)
   - Via sysctl(8) oder per setsockopt(2) (socket spezifisch!)
2. `TCP_CORK` (existiert auch in anderen Unix-Derivaten)
   - ideal für HTTP:

   (a) `setsockopt(TCP_CORK, 1)`
   (b) `write(2)` für HTTP-Header
   (c) `sendfile(2)` für HTML-Datei
   (d) `setsockopt(TCP_CORK, 0)`

GNU/Linux Sahnestücke

1. `tcp_frto`
   - packet loss != intermediate router congestion
   - bei RTO: 2 Segmente erneut übertragen und auf ACK’s schauen

GNU/Linux Sahnestücke

1. InfiniBand Support
   - bidirektionalen seriellen Bus
   - PCIe war/ist Alternative
   - Protokollstack in Hardware Implementiert
   - RDMA
   - latenzarm im Bezug auf Ethernet
   - Bandbreite: 2,5/5/10/20 GBit/s (Bündelung und DDR)
   - Forschunszentrum Karlsruhe Messreihe:
     (a) Latenz: 7 Mikrosekunden (60 bei GE (30 fuer Switch))
     (b) Bandbreite: 780MB/s (60MB/s bei GE)
   - Einsatz: Cluster Technologie (z.Z.)
   - `http://infiniband.sourceforge.net/`

. . . zum ausschwingen

- NIC Hardware Encryption
- Alignment (Ethernet Header 14 Byte)
- Multicast Support
- `invXpg` unendlich viele Zyklen, besonders wenn global bei SMP-Box

FIN

- Vielen Dank!
- Fragen/Anregungen/Pizza: bitte an Hagen Paul Pfeifer <hagen@jauu.net>
- Mit der Bitte um `!(world_readable) Key-ID: 0x98350C22` ;-)

Quellen

1. http://www.microsoft.com/technet/itsolutions/network/deploy/depovg/tcpip2k.mspx
2. Jumbo Frames
   - http://www.psc.edu/~mathis/MTU/
   - http://darkwing.uoregon.edu/~joe/jumbo-clean-gear.html
3. TOE
   - http://www.alacritech.com/assets/applets/Alacritech_Accelerator.pdf
   - http://www.microsoft.com/windowsserver2003/evaluation/news/bulletins/ws03net.mspx
4. RDMA
   - http://www.rdmaconsortium.org
   - http://datatag.web.cern.ch/datatag/pfldnet2003/papers/romanow.pdf