Protokoll Turbulenzen

Wirr- und Irrungen bei der Implementierung und Standardisierung von Netzwerkprotokollen

Hagen Paul Pfeifer

‹hagen@jauu.net›

Florian Westphal

‹fw@strlen.de›

22. März 2008

Einführung

"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

- Differenzierung zwischen Standard und technischer Implementierung
  - Ohne uns etwas vorzumachen: viele Forscher sind am Ph.D interessiert
  - Forscher und Netzwerkhacker sind die wenigsten
  - Viele Augen sind teilweise nicht genug (→ komplexe Thematik)
- Oft mangelndes Wissen über Betriebsysteminternes (Timerhandling, TCP SACK, . . .)
- Neu entstehende Infrastrukturen bieten die Möglichkeit zum "Technologierefresh"
- Proprietäre oder statische Strukturen behindern Evolution ("Evolution durch Mutation")
- "Flag Days" wird es nicht mehr geben (*NCP* → *IP/TCP*)

TCP und Sequenznummern

- TCP: Bytestrom
- TCP ist auf unzuverlässige Umgebung ausgelegt
  - Pakete gehen verloren
  - Pakete werden verzögert
- Sequenznummern nummerieren Bytes
- Start-Sequenznummern sind nicht 0
  - Es muss gewährleistet sein, das ein zu einer 'alten' Verbindung gehörendes Paket auch als solches erkannt werden kann
  - Synchronisation der Peer-Sequenznummern bei TCP-Handshake
- RFC 793: Erzeugung der ISN mittels Uhr

ISN Erzeugung

- Morris 1985: ’Uhm.... Moment!’
  - Monoton wachsende ISN erlaubt dritten einfaches ’raten’ gültiger Sequenznummern
  - Aufbau gültiger Verbindungen (ohne Pakete empfangen zu müssen)
  - Wenn Adressen und Ports bekannt: Einschleusen von Daten, Resets
- 1996 RFC 1948: ’Defending Against Sequence Number Attacks’
- Wahl der ISN durch Kombination von Zähler und Zufallswert, z.B. Linux:
  - Untere 24 Bit: (partieller) MD4 über Quell/Ziel-IP und Secret
  - Highbits: Addieren eines Zähler
  - Addieren der aktuellen Zeit (Auflösung: 64ns)

Seqenznummer-Problematik heute

- Um Pakete einzuschleusen/Verbindungen zu resetten muss nächste Sequenznummer nicht exakt erraten werden
- Es reicht eine Seqenznummer, die innerhalb des Sendefensters liegt
- Je größer das Fenster, desto einfacher
- Window Scaling vergrößert das Problem (nicht nur hier: vgl. PAWS)
- 1998 RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option
- *SCTP*: Verification tag; initiale TSN beliebig wählbar
- *DCCP*: Sequenznummer ist per-Datagramm-Zähler, optional 48 bit

SYN Cookies

- TCP verwendet Three-Way-Handshake:
- Wenn TCP Diagramm mit gesetzem SYN-Bit für einen Dienst empfangen wird:
  - Syn-Queue eintrag erstellen: Adressen, MSS, Peer TCP-Optionen, etc.
  - SYN-ACK an Empfänger zurückschicken
- Problem: Was, wenn die Queue voll ist (oder Ressourcen alle)?
  - Es kann keine neue Verbindung mehr angenommen werden
  - Auch dann, wenn Leitung nicht überlastet ist
- Problem lange bekannt
- 1996 ist es soweit: PANIX DoS

SYN Cookies (2)

- Keine Queue mehr, es wird nur *SYN/ACK* versendet
- Muss erkennen können, ob empfangenes *ACK* Antwort auf ein vorher gesendetes *SYN/ACK* ist – ohne Zustände zu speichern
- Ohne Modifikationen am *TCP* Protokoll, d.h. für Client transparent

SYN Cookies (3)

- Lösung durch Bernstein/Schenk 1996
- Verwendet eine gezielt gewählte TCP Sequenznummer im *SYN/ACK*, um die Allernötigsten Informationen zu kodieren
- Bei ACK-Empfang kann Sequenznummer aus dem Paket entnommen, dekodiert und der Handshake beendet werden
- Wie kann man verhindern, das dritte nun einfach ’falsche’ ACK-Datagramme versenden um TCP-Verbindungen zu erzeugen?

SYN Cookie: Vorgehen

- In SYN-ACK versendete TCP Sequenznummer basiert auf Hash (Linux: SHA-1)
  - Ziel/Quelladressen, Ziel/Quellports, Peer-Sequenznummer
  - Secret
  - Zähler (wird jede Minute inkrementiert)
- MSS wird im Ergebnis codiert (nur d. häufigsten)
- Beim Empfang eines ACK Paketes wird versucht, den Cookie für die letzten vier Zähler zu rekonstruieren
- Wenn erfolgreich → Established

SYN Cookie: Auswirkungen

- Keine TCP-Optionen
- ”hängende” TCP Verbindung, wenn SYN-ACK des Servers verloren geht (und Client auf Daten wartet)
- Aber:
  - Cookies werden nur dann verwendet, wenn die Queue voll ist
  - $\rightarrow$ ”besser eine schlechte als gar keine Verbindung”
  - FreeBSD: Optionen werden via Timestamp-Option codiert
- Notwendigkeit heute?
  - Heute mehr Speicher, viel größere Queues
  - Linux Minisocks
  - Egress Filtering durch Provider vs. Botnets

Stausituationen

- Massive Probleme im Oktober 1986 (*ARPANET*/*MILNET*) – Leistungverringerung um Faktor 1000
- Problem: Netzelemente kannten keine Indikatoren um Überlast zu signalisieren
- Im Gegenteil, sendende Host reagierten mit einer vergrößerten Datenrate
- Apropos: Stand der Dinge war *4.3BSD*

Staukontrolle

- Stausituationen → Staukontrolle
- Van Jacobson, Michael J. Karels:
  1. Slow Start
  2. Fast Retransmit
  3. Exponentieller Retransmit Timer
  4. RTT Varianz Abschätzungen
  5. ...
- ICMP Source Quench → zusätzliche Last bei akuter Überlast!

Staukontrolle (2)

- Problem gelöst? Mitnichten!
- Vermittlungsschichtproblem nicht Transportschichtproblem
  - TCP, DCCP kontra UDP, UDPLite
  - XCP (eXplicit Congestion control Protocol)
- Routing Anomalien - Retransmittimer nicht 100%
- `LFN` - Links mit einem hohen BDP
  - Congestion Avoidance Phase:
    - Vergrößerung von `cwnd` mit 1 per RTT -ein Paket pro RTT!
    - 1000, 2000, 5000, . . . RTT's ohne Paketdrop um Linkkapazität zu erreichen
  - HighSpeed TCP, BIC, Cubic, Hybla, Compound TCP, . . .
- Paketverlust kann nicht immer als Stau-Indikator dienen, 802.11!
- Nicht alle TCP Stacks Implementieren den gleichen CC-Algorithmus - Fairshare!

- Netzkoppelelemente müssen involivert sein - Drop Tail
- Globale Synchronisierung → *RED* (Random Early Detection) - vorzeitiges verwerfen
- Packetdrop als Indikator - extrem unsauber (Engineering Sicht)

Staukontrolle (3) - ECN

- *ECN* - Explicit Congestion Notification
- Explicietes Anzeigen das Stausituation auftritt ohne Paketverlust
- Middlebox Problematik

SACK - Selective Acknowledgment

- falls in einem Sendefenster mehrere Pakete verloren gehen, bekommt TCP ein performance-Problem
  - ACKs sind Kummulativ. Aber:
    - Sender schickt Segmente s_1, s_2, s_3, $\ldots s_n$
    - Sender erhält ACK für s_2
    - Muss s_3 neu übertragen werden? s_4? s_n?
  - Sender muss entweder:
    - pro verlorenem Paket eine ganze RTT warten, oder
    - (alle) Pakete ab s_3 'auf Verdacht' neu übertragen
- RFC 3517: A *Conservative* Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP
- "Nachrüstung" via TCP-Options: SACK-Permitted, SACK
- FastRetransmit: geSACKte Blöcke nicht erneut senden

SACK-Option

- Wenn nicht-zusammenhängende Daten empfangen wurden:
  - 'normales' ACK auf letzte vollständige Sequenznummer
  - SACK-Option beinhaltet (Teil-) Liste empfangener Blöcke
    - erste zum Block gehörende Sequenznummer
    - erste nicht mehr zum Block gehörende Sequenznummer (= noch nicht empfangen)
  - Da Liste ins Optionen Feld passen muss: max. 4 Blöcke möglich
- Weit verbreitete TCP Option
- auch SCTP integriert SACK

SACK Implementierung

```
struct tcp_sack_block { u32 start_seq; u32 end_seq; };

struct tcp_sock {

    [..]

    struct tcp_sack_block duplicate_sack[1]; /* D-SACK block */

    struct tcp_sack_block selective_acks[4]; /* The SACKS themselves*/
```

- Linux Kernel hat receive queue & out_of_order queue
  - Eintreffendes Segment (=skbuff) nicht in Reihenfolge?
    → out-of-order queue (nach seq sortiert)
  - neuer SACK-Eintrag, oder exisitierenden SACK block anpassen
    (fast wie bei Tetris ;-)
- Jetzt gibt es da aber auch noch den D-SACK-Block...

Duplicate SACK

- RFC 2883 – keine neuen TCP Optionen, nutzt exisitierende SACK Infrastruktur
- Pakete können nicht nur in beliebiger Reihenfolge eintreffen ...
- ... sondern sich auch überlappen
- Bei Nutzung von D-SACK wird der Block dem Peer auf jeden Fall im nächsten SACK mitgeteilt
- SACK-Empfänger:
  - erster SACK Block $<$ ACK? Oder...
  - zweiter SACK Block vorhanden und start_seq $<=$ erster SACK-Block?
- Linux Kernel: Bei D-SACK `cwnd` senken
- SCTP: ”Gap ACK Block” und ”Duplicate TSN” Listen

FACK

- Forward Acknowledgment (Mathis/Mahdavi ’96)
- Keine TCP-Erweiterung, nutzt SACK-Informationen für bessere Staukontrolle
- bessere abschätzung, wieviele Byte noch ”unterwegs” sind
  - Man merke sich vordersten SACK Block (”most forward SACK”)
  - $awnd = snd\_next - fack + retrans$
    - Funktioniert nur dann, wenn Pakete nicht ausserhalb der Reihenfolge eintreffen
    - Linux: FACK abschalten, wenn Reordering erkannt wird
    - Wenn Paketverlust erkannt, *alle* nicht SACK-Blöcke bis zum ’forward SACK’ als verloren Betrachten

SACK Probleme

- Entwurfsansatz: SACK-Blöcke sind verlustbehaftete Information
  - Der Empfänger von out-of-order Paketen darf diese verwerfen
  - Annociert diese Pakete nicht mehr in SACK blöcken
- SACK-Blöcke sind somit immer nur *Hinweis* und ersetzen ACKs nicht
- Sender muss unbestätigten Pakete auch dann in Queue halten, wenn diese in einem SACK-Block annociert wurden
- → Sender muss mehr Resourcen bereitstellen als Empfänger
- → `kfree_skb`-Lawine

Fazit

- Innovation stagniert – das innovative, wissenschaftlich geprägte Medium ist das Internet schon lange nicht mehr (siehe den durchschlagenden Erfolg von IPv6)
- Änderungen an der Core Funktionalität nur dann wenn:
  1. Zentrale Funktionaliät in Gefahr ist
  2. Wenn ISPs/Global Player die Gewinne maximieren können
- Legacy Systeme werden immer mehr zum Problem
- Standardiserungsprozeß ist ein fehleranfälliger da komplexer Prozeß
- Langlebige standardisierte Funktionalität ist nicht gefeit von “Wetteränderungen”
- Komplexer werdender Standardisierungsprozeß
  - → das Gehirn skaliert nicht ausreichend, mit der Menge an benötigten interdisziplinären Wissen ;-)