Reflections on TCP

Florian Westphal

22. Juni 2008

1024D/F260502D <fw@strlen.de>
1C81 1AD5 EA8F 3047 7555
E8EE 5E2F DA6C F260 502D

Abstract

In the 25+ years the TCP/IP protocol family has been deployed on the internet, it has often faced new problems and challenges. This talk reflects on some of these issues and how TCP was adapted to meet new requirements.

Outline

1. Problems during the early deployment (1980s)
   - Silly Window Syndrome and Tinygrams
   - Initial Sequence Numbers
   - Congestion Control
2. Addition of Explicit Congestion Notification to IP
3. Syn Cookies
4. Selective Acknowledgments
5. future directions

Tinygrams/Silly Window Syndrome

- one-byte writes resulted in 41 bytes packets on the network (tinygrams)
- RFC 896 / '84: "Nagle Algorithm" – delays sending of data, if
  - MSS-sized packet cannot be sent (not enough data)
  - and ACK is outstanding

(Receiver-side) silly window syndrome:

- if receiver is unable to process all incoming data, the window becomes smaller and smaller
- fix: close window until full MSS-sized packet can be received again (or rx buffer is at half empty)

TCP & Sequence Numbers

- TCP – Bytestream
- TCP is designed to work in (possibly) unreliable networks
  - packet loss
  - packets may be delayed and arrive out-of-order
- TCP sequence numbers refer to bytes
- sequence number does **not** start at 0
  - Need to be able to detect packets belonging to "old" connections
  - peer sequence numbers are synchronized during TCP handshake
- RFC 793: sequence numbers are clock based

Initial Sequence Number

- Historic 4.4 BSD implementations:
  - Initialize ISN with 1
  - increment by a constant every 0.5s and for every connection
- Allows others to guess valid sequence numbers (Morris '85):
  - establish valid connections without receiving replies
  - when addresses/port numbers known: Allows to inject data, reset connections, etc.

Initial Sequence Number (2)

- 1996 RFC 1948: 'Defending Against Sequence Number Attacks'
- ISN is created by combining counters and random numbers, eg. Linux:
  - low 24 Bits (partial) MD4 digest using src/dest IP address and a secret
  - High bits: Adds a counter
  - adds current time (resolution 64ns)

Initial Sequence Number problem today

- no need to guess the exact number, only need one within current window
- larger window increases chances
- Window Scaling increases this problem (also see PAWS)
- esp. problematic with long-living connections using large windows
- 1998 RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option
- SCTP: Verification tag; initial TSN is random
- DCCP: Sequence number is per-datagram (optionally 48 Bit)

Congestion and Congestion Control

- "Because segments may be lost due to [..] network congestion, TCP uses retransmission (after a timeout) to ensure delivery.." (RFC 793)

- Massive problems in 1986: routers can't signal congestion – senders increase data rate to retransmit missing packets $\rightarrow$ Congestion Control

- Van Jacobson/Karels – Paxon/Allman/Stevens (RFC 2581 – TCP congestion control)
  - Slow Start: introduces `cwnd` – sender imposed flow control
  - Exponential Retransmit Timer, Fast Retransmit, . . .

- ICMP Source Quench $\rightarrow$ has to send additional packets if network is congested

Congestion Control

- Packet loss does not neccessarily due to congestion (eg wireless networks)
- `LFN` (links with high BDP)
- Global Synchronization $\rightarrow$ RED (Random Early Detection)
- Network layer problem – not transport layer

Explicit Congestion Notification (ECN)

- RFC 3168 / 2001 (Ramakrishnan/Floyd/Black)
- extension to the Internet Protocol that allows routers to signal congestion before packets have to be dropped
- uses two bits in the IPv4 header TOS octet
- ECN-unaware (00), congestion experienced (11)
- two ECN-Nonces, 01 and 10 – nonce tells router that endpoint is ECN-aware
  (they are aequivalent – allows detection of middleboxes that delete CE bit)

ECN & TCP

- Two new TCP header flags: ECN-echo & Congestion Window Reduced.
- fortuanetely, TCP had 6 bits reserved for future use
- An ECN aware TCP sets both CWR and ECN-Echo in SYN packet
- If the peer TCP is ECN aware too, it sets ECN-Echo and unsets CWR in SYN-ACK
  (avoids hassle with TCP endpoints that 'copy' the reserved flags)
- ECN-Echo: Used by receiver to inform sender that IP header was CE-tagged
- CWR: Used by sender to inform Receiver that congestion window was reduced

SYN Cookies

- TCP uses a three-way handshake. When packet with SYN flag set received,
  - need to send SYN-ACK to peer
  - also need to create SYN-Queue entry with peer information (address, TCP options, ...)
- Problem: When queue is full/exhausted $\longrightarrow$ can't accept new connections
- Problem known for long time
- 1996: PANIX DoS Syn-Flood

SYN Cookies (2)

Problem is solved by removing the Syn queue

- Only send out the SYN/ACK
- need to detect if incoming ACK is a response to a SYN/ACK (without remembering any states)
- without modifying the TCP protocol (i.e. transparent for TCP clients)

SYN Cookies (3)

- Solution: Bernstein/Schenk 1996
- uses a specially crafted TCP ISN in SYN/ACK packet to encode needed information (the syn cookie).
  - e.g. TCP MSS
  - this information would normally be stored in a syn queue entry
- Can read this cookie again when ACK is received
- cookie can then be decoded and the connection is established
- Problem: can now create TCP connections by sending ACK $\rightarrow$ need to detect if cookie is valid

Basic algorithm

- create the cookie ISN using a hash function (Linux: SHA-1)
  - src/dest addresses and port number, peer ISN
  - secret
  - counter (incremented once very minute)
- MSS is encoded in the results (only most common)
- when ACK is received try to re-compute cookie for the (most recent) counter values
- on success $\longrightarrow$ Established

SYN Cookie: drawbacks

- TCP connections hang if ACK (for the SYN/ACK) is lost and client waits for data (can also happen without cookies)

- no TCP options due to lack of space in cookie value, BUT:
  - Cookies are only used if queue is exhausted → no cookies, no connection
  - FreeBSD encodes options in the TCP Timestamp option, if client advertised TIMESTAMP support
  - Linux starting with 2.6.26
  - Minor difference in implementation: FreeBSD encodes value in High-bits, Linux in low bits
  - FreeBSD uses per connection TS offset to adjust TS accordingly

- Today: larger queues, Linux Minisocks, Egress Filtering vs. Botnets, ...

Impact on other protocols

```
Note: After sending out INIT ACK with the State Cookie parameter,
"Z" MUST NOT allocate any resources, nor keep any states [..]
Otherwise, "Z" will be vulnerable to resource attacks.
```

(RFC 4960 – Stream Control Transmission Protocol)

- DCCP: Optional Init-Cookies

SACK - Selective Acknowledgment

TCP has problems if several packets in the same window are lost

- ACKs are cummulative, but:
  - Sender sends segments s_1, s_2, s_3, . . . s_n
  - Sender receives ACK for s_2
  - Should s_3 be retransmitted? What about s_4? s_n?
- Sender needs to:
  - either retransmit s_2 and wait one RTT to see if it was enough
  - or assume several packets got lost and re-send them all

RFC 3517: A **Conservative** Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP

SACK-Option

When packets are received out of order:

- ACK the last in-sequence number
- TCP-Header stores SACK-Block(s) with information about received block
- recall previous example: If s_3 got lost, could ack s_2 and announce receipt of s_4, s_5 in SACK block to avoid unneeded retransmit
  - first sequence number of the block
  - last sequence number not within block (= i.e. not (yet) received)
- SACK Blocks must fit within TCP header option field: Max. 4 SACK Blocks

Widely supported TCP option – SCTP also has a SACK mechanism

SACK Implementation (Linux)

```
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 themselve
```

- Linux Kernel hat receive queue & out_of_order queue
  - Incoming segment not in-order? $\rightarrow$ out-of-order queue (sorted by sequence number)
  - new SACK-Entry or change existing SACK block if adjacent segment
- Linux supports DSACK, too.

Duplicate-SACK

- RFC 2883 – no new TCP options, uses SACK
- data may not only arrive out of order, but also more than once
- D-SACK is a method to inform the sender about such duplicates
- SACK-Reciepient:
  - first SACK Block $<$ ACK? or...
  - 2nd SACK Block present and start_seq $<=$ first SACK block?
- Linux Kernel: Decrease Window (`cwnd`)
- SCTP: "Gap ACK Block" and "Duplicate TSN" Lists

F-ACK

Forward Acknowledgment (Mathis/Mahdavi '96)

- not a TCP-extension, uses SACK information for congestion control
- fack: "most forward SACK" – highest sequence number know to have reached receiver
- Estimates number of bytes in flight:
- $awnd = snd_{next} - snd_{fack} + retrans$
- `while (awnd < cwnd) send_more_data()`

– Does not work when packets are reordered

SACK Problems

- SACK blocks are a lossy indicator
  - The Recipient of out-of-order packets may discard them (and then stop advertising them in SACK blocks)
- so, SACK-Blocks are only a $hint$ and are not an ACK replacement
- sender has to keep unacknowledged packets, even if they've been announced in SACK blocks
- when ACK arrives, all bufs are freed in one go

Conclusions

The internet protocols are still evolving:

- Wifi – Congestion detection
- TCP LO / SLO (`draft-eddy-tcp-loo-03`)
- ECN+
- 10G Ethernet – TSO/GSO
- Rethinking the implementation: Netchannels (Van Jacobson)
- New protocols: SCTP, DCCP, UDPLITE, . . .