Hunting down an old Linux TCP bug

and what we learned from that

Apollon Oikonomopoulos
apollon@skroutz.gr — @apoikos

vGRNOG 11 — 2021-04-15

skroutz

Introduction

About me

- Mechanical engineer by training, sysadmin by trade
- CIO @ skroutz.gr
- FOSS enthusiast, Debian Developer

skroutz

About skroutz.gr

- #1 ~~Price comparison engine~~ Marketplace in Greece
- 9k+ merchants, 1.5M sessions/day
- >1Gbps web traffic, 3-4k app req/s
- On-premises, self-managed infrastructure
- Mostly Debian, mostly FOSS

skroutz

Tracking down a kernel bug

Heisensync

- Nightly cronjob, `rsync`'ing >1TB of fast-changing data from 2 source servers to 4 clients
- Sporadic hangs for no apparent reason, first observed ca. 2017
  - No data exchange - other than TCP keepalives
  - No other system- or network-level issues
  - Random incident frequency, impossible to reproduce reliably

skroutz

Operational realism

- `strace` showed signs of ”healthy” activity
- Low-effort debugging attempts led nowhere
  - We even ran the job in a loop for many days, without being able to reproduce the hang
- Not our most important problem overall
  - ”Just run the job by hand”
  - `rsync --bwlimit=40m` seemed to help

skroutz

Operational realism

- `strace` showed signs of ”healthy” activity
- Low-effort debugging attempts led nowhere
  - We even ran the job in a loop for many days, without being able to reproduce the hang
- Not our most important problem overall
  - ”Just run the job by hand”
  - `rsync --bwlimit=40m` seemed to help

Let’s blame some ”rsync event loop race condition”

skroutz

Things get worse

- From once/year to 4 times/week
  (for no apparent reason)
- Sometimes able to trigger the hang manually
- Tried upgrading rsync, using I/O and network timeouts

skroutz

Things get worse

- From once/year to 4 times/week
  (for no apparent reason)
- Sometimes able to trigger the hang manually
- Tried upgrading rsync, using I/O and network timeouts
- Annoyance threshold exceeded

skroutz

Things get worse

- From once/year to 4 times/week
  (for no apparent reason)
- Sometimes able to trigger the hang manually
- Tried upgrading rsync, using I/O and network timeouts
- Annoyance threshold exceeded

skroutz

One connection, two directions

```
[client]
$ ss -mito dst :873
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
ESTAB   0      392827  2001:db8:2a::3:38022  2001:db8:2a::18:rsync  timer:(persist,1min56sec,0)

[server]
$ ss -mito src :873
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
ESTAB   0      0       2001:db8:2a::18:rsync 2001:db8:2a::3:38022   timer:(keepalive,3min7sec,0)
```

skroutz

One connection, two directions

```
[client]
$ ss -mito dst :873
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
ESTAB   0      392827  2001:db8:2a::3:38022  2001:db8:2a::18:rsync  timer:(persist,1min56sec,0)

[server]
$ ss -mito src :873
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
ESTAB   0      0       2001:db8:2a::18:rsync 2001:db8:2a::3:38022   timer:(keepalive,3min7sec,0)
```

Non-empty Send-Q + empty Recv-Q = Zero Window (aka `persist`)

skroutz

Interlude: TCP sequence numbers

- TCP is *stateful* and *order-preserving*
- Both ends keep track of what they've seen on the wire
- … using *sequence numbers*: monotonically increasing, unsigned, 32-bit integers
- Essentially *byte offsets* into the data stream, counting from an *arbitrary* initial value (*ISN*)
- Two fields in the TCP header: *SEQ* and *ACK*

  SEQ is what I'm sending

  ACK is what I expect from you to send

skroutz

Interlude: Sequence number arithmetic

```c
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```

skroutz

Interlude: Sequence number arithmetic

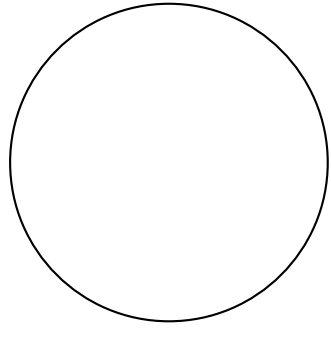
```
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```

- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

skroutz

Interlude: Sequence number arithmetic

```
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```

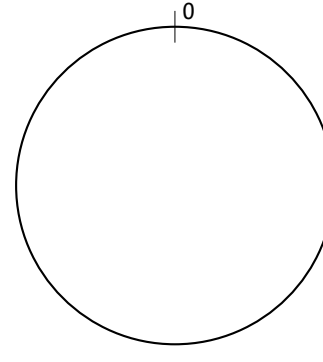


- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

skroutz

Interlude: Sequence number arithmetic

```c
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```

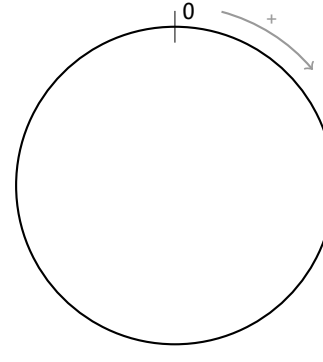


- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

skroutz

Interlude: Sequence number arithmetic

```c
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```

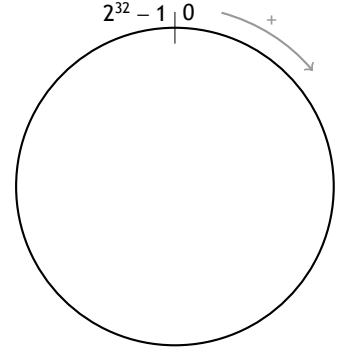


- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

skroutz

Interlude: Sequence number arithmetic

```c
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```

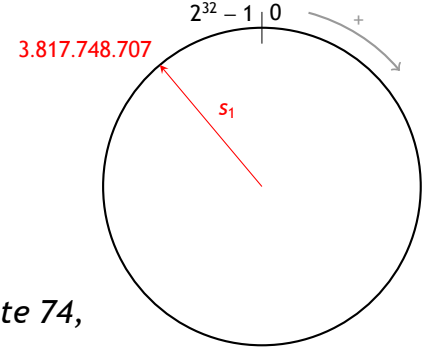


- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

skroutz

Interlude: Sequence number arithmetic

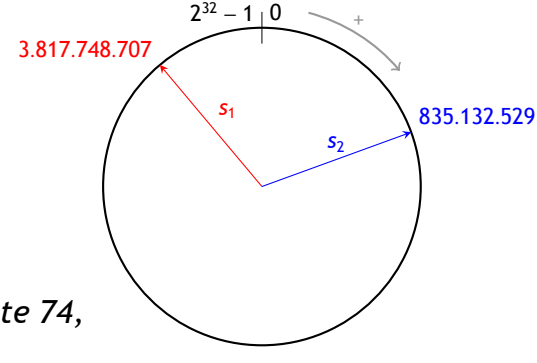
```
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```



- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

Interlude: Sequence number arithmetic

```c
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```

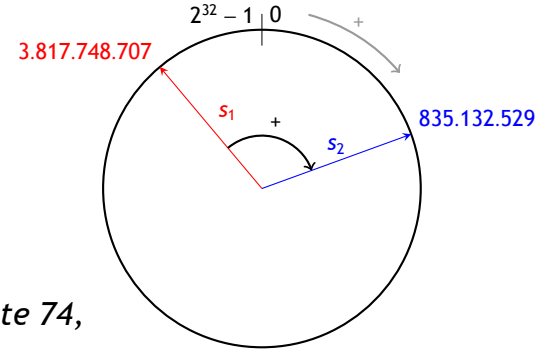


- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

skroutz

Interlude: Sequence number arithmetic

```c
/*
 * The next routines deal with comparing 32 bit unsigned ints
 * and worry about wraparound (automatic with unsigned arithmetic).
 */
static inline bool before(__u32 seq1, __u32 seq2)
{
        return (__s32)(seq1-seq2) < 0;
}
#define after(seq2, seq1)       before(seq1, seq2)
```



- Described in *Internet Experiment Note 74, 21 September 1978*
- Similar approach for DNS serial numbers in RFC 1982 (1996)

skroutz

Interlude: TCP receive window

- Each end signals how much data it is *currently* willing to receive
- Other end *may* send *that* much data before waiting for ACK (buffering/RTT suppression)
- If running out of *receive space*, advertise a *zero-sized* window (flow control)
- A zero window *opens* with a subsequent non-zero advertisement
- To avoid missing re-opening due to packet loss, send *Zero Window Probes* (`persist` timer)

skroutz

tcpdump to the rescue

```
[client]
$ ss -mito dst :873
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
ESTAB   0      392827  2001:db8:2a::3:38022  2001:db8:2a::18:rsync  timer:(persist,1min56sec,0)
```

skroutz

tcpdump to the rescue

```
[client]
$ ss -mito dst :873
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
ESTAB   0      392827  2001:db8:2a::3:38022  2001:db8:2a::18:rsync  timer:(persist,1min56sec,0)

09:34:34.165148 0c:c4:7a:f9:68:e4 > 0c:c4:7a:f9:69:78, ethertype IPv6 (0x86dd), length 86:
    (flowlabel 0xcbf6f, hlim 64, next-header TCP (6) payload length: 32)
    2001:db8:2a::3.38022 > 2001:db8:2a::18.873:
      Flags [.], cksum 0x711b (incorrect -> 0x4d39), seq 4212361595, ack 1253278418,
      win 16384, options [nop,nop,TS val 2864739840 ecr 2885730760], length 0
09:34:34.165354 0c:c4:7a:f9:69:78 > 0c:c4:7a:f9:68:e4, ethertype IPv6 (0x86dd), length 86:
    (flowlabel 0x25712, hlim 64, next-header TCP (6) payload length: 32)
    2001:db8:2a::18.873 > 2001:db8:2a::3.38022:
      Flags [.], cksum 0x1914 (correct), seq 1253278418, ack 4212361596,
      win 13831, options [nop,nop,TS val 2885760967 ecr 2863021624], length 0
[… repeats every 2 mins]
```

skroutz

tcpdump to the rescue

```
[client]
$ ss -mito dst :873
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
ESTAB   0      392827  2001:db8:2a::3:38022  2001:db8:2a::18:rsync  timer:(persist,1min56sec,0)

09:34:34.165148 0c:c4:7a:f9:68:e4 > 0c:c4:7a:f9:69:78, ethertype IPv6 (0x86dd), length 86:
    (flowlabel 0xcbf6f, hlim 64, next-header TCP (6) payload length: 32)
    2001:db8:2a::3.38022 > 2001:db8:2a::18.873:
      Flags [.], cksum 0x711b (incorrect -> 0x4d39), seq 4212361595, ack 1253278418,
      win 16384, options [nop,nop,TS val 2864739840 ecr 2885730760], length 0
09:34:34.165354 0c:c4:7a:f9:69:78 > 0c:c4:7a:f9:68:e4, ethertype IPv6 (0x86dd), length 86:
    (flowlabel 0x25712, hlim 64, next-header TCP (6) payload length: 32)
    2001:db8:2a::18.873 > 2001:db8:2a::3.38022:
      Flags [.], cksum 0x1914 (correct), seq 1253278418, ack 4212361596,
      win 13831, options [nop,nop,TS val 2885760967 ecr 2863021624], length 0
[… repeats every 2 mins]
```

For some reason, the client *rejects* the window update

skroutz

♥ Free Software

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
$ git checkout v4.9.144
$ git grep …
$ vi net/ipv4/tcp_input.c
$ git blame …
$ git log --cherry v4.9.144...linus/master -- net/ipv4/tcp_input.c
$ git show …
$ git grep …
$ vi include/net/tcp.h
  …
  coffee
  more git
  sleep & dream of more git
```

skroutz

♥ Free Software

net/ipv4/tcp_input.c

```c
/* Check that window update is acceptable.
 * The function assumes that snd_una<=ack<=snd_next.
 */
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

skroutz

♥ Free Software

net/ipv4/tcp_input.c

```c
/* Check that window update is acceptable.
 * The function assumes that snd_una<=ack<=snd_next.
 */
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

skroutz

♥ Free Software

net/ipv4/tcp_input.c

```
/* Check that window update is acceptable.
 * The function assumes that snd_una<=ack<=snd_next.
 */
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

1. We can only observe `ack`, `ack_seq` and `nwin`

skroutz

♥ Free Software

net/ipv4/tcp_input.c

```
/* Check that window update is acceptable.
 * The function assumes that snd_una<=ack<=snd_next.
 */
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

1. We can only observe `ack`, `ack_seq` and `nwin`
2. What are `snd_una`, `snd_wl1`, `snd_next`?

skroutz

♥ Free Software

net/ipv4/tcp_input.c

```c
/* Check that window update is acceptable.
 * The function assumes that snd_una<=ack<=snd_next.
 */
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

1. We can only observe `ack`, `ack_seq` and `nwin`
2. What are `snd_una`, `snd_wl1`, `snd_next`? → RFC 793 (TCP, 1981)

skroutz

♥ Free Software

net/ipv4/tcp_input.c

```
/* Check that window update is acceptable.
 * The function assumes that snd_una<=ack<=snd_next.
 */
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

1. We can only observe `ack`, `ack_seq` and `nwin`
2. What are `snd_una`, `snd_wl1`, `snd_next`? → RFC 793 (TCP, 1981)
   - `snd_wl1`: **w**indow **l**imit used to guard window updates
   - `snd_una`: **una**cknowledged sent data seq. no

skroutz

Probing the kernel in vivo

SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel

skroutz

Probing the kernel in vivo

SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel

```
probe kernel.statement("tcp_ack@./net/ipv4/tcp_input.c:3751")
{
  if ($sk->sk_send_head != NULL) {
    ack_seq = @cast(&$skb->cb[0], "tcp_skb_cb", "kernel<net/tcp.h>")->seq
    printf("ack: %d, ack_seq: %d, prior_snd_una: %d\n", $ack, ack_seq, $prior_snd_una)
    seq = @cast(&$sk->sk_send_head->cb[0], "tcp_skb_cb", "kernel<net/tcp.h>")->seq
    end_seq = @cast(&$sk->sk_send_head->cb[0], "tcp_skb_cb", "kernel<net/tcp.h>")->end_seq
    printf("sk_send_head seq:%d, end_seq: %d\n", seq, end_seq)

    snd_wnd = @cast($sk, "tcp_sock", "kernel<linux/tcp.h>")->snd_wnd
    snd_wl1 = @cast($sk, "tcp_sock", "kernel<linux/tcp.h>")->snd_wl1
    ts_recent = @cast($sk, "tcp_sock", "kernel<linux/tcp.h>")->rx_opt->ts_recent
    rcv_tsval = @cast($sk, "tcp_sock", "kernel<linux/tcp.h>")->rx_opt->rcv_tsval
    printf("snd_wnd: %d, tcp_wnd_end: %d, snd_wl1: %d\n", snd_wnd,
           $prior_snd_una + snd_wnd, snd_wl1)
    printf("flag: %x, may update window: %d\n\n", $flag, $flag & 0x02)
   }
}
```

skroutz

Probing the kernel in vivo

```
$ sudo apt install systemtap linux-headers-$(uname -r) linux-image-$(uname -r)-dbg
```

skroutz

Probing the kernel in vivo

```
$ sudo apt install systemtap linux-headers-$(uname -r) linux-image-$(uname -r)-dbg

$ stap -v zwp.stp
```

skroutz

Probing the kernel in vivo

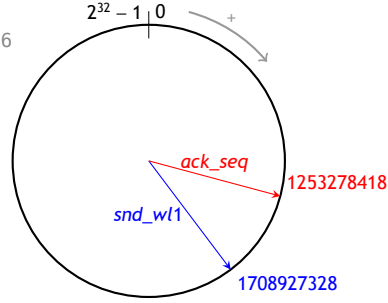
```
$ sudo apt install systemtap linux-headers-$(uname -r) linux-image-$(uname -r)-dbg

$ stap -v zwp.stp
ack: 4212361596, ack_seq: 1253278418, prior_snd_una: 4212361596
sk_send_head seq:4212361596, end_seq: 4212425472
snd_wnd: 0, tcp_wnd_end: 4212361596, snd_wl1: 1708927328
flag: 4100, may update window: 0
```

skroutz

Probing the kernel in vivo

```
$ sudo apt install systemtap linux-headers-$(uname -r) linux-image-$(uname -r)-dbg
```

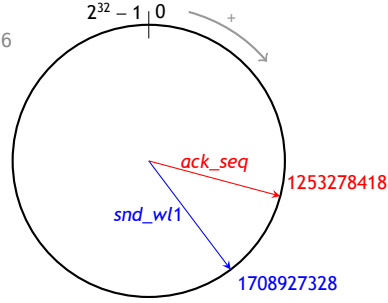
```
$ stap -v zwp.stp
ack: 4212361596, ack_seq: 1253278418, prior_snd_una: 4212361596
sk_send_head seq:4212361596, end_seq: 4212425472
snd_wnd: 0, tcp_wnd_end: 4212361596, snd_wl1: 1708927328
flag: 4100, may update window: 0
```



Probing the kernel in vivo

```
$ sudo apt install systemtap linux-headers-$(uname -r) linux-image-$(uname -r)-dbg
```

```
$ stap -v zwp.stp
ack: 4212361596, ack_seq: 1253278418, prior_snd_una: 4212361596
sk_send_head seq:4212361596, end_seq: 4212425472
snd_wnd: 0, tcp_wnd_end: 4212361596, snd_wl1: 1708927328
flag: 4100, may update window: 0
```



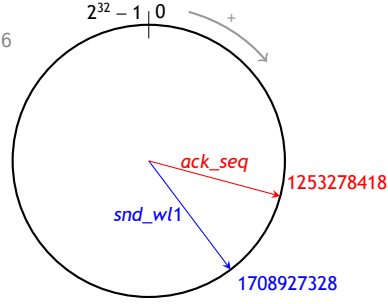

```
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

skroutz

Probing the kernel in vivo

```
$ sudo apt install systemtap linux-headers-$(uname -r) linux-image-$(uname -r)-dbg
```

```
$ stap -v zwp.stp
ack: 4212361596, ack_seq: 1253278418, prior_snd_una: 4212361596
sk_send_head seq:4212361596, end_seq: 4212425472
snd_wnd: 0, tcp_wnd_end: 4212361596, snd_wl1: 1708927328
flag: 4100, may update window: 0
```




```
static inline bool tcp_may_update_window(const struct tcp_sock *tp,
                                         const u32 ack, const u32 ack_seq,
                                         const u32 nwin)
{
        return  after(ack, tp->snd_una) ||
                after(ack_seq, tp->snd_wl1) ||
                (ack_seq == tp->snd_wl1 && nwin > tp->snd_wnd);
}
```

skroutz

Taking things upstream

- Hypothesis: some code path fails to update `snd_wl1`, and `ack_seq` eventually wraps around with respect to it, making it impossible to re-open a zero window
- Mailed `netdev@vger.kernel.org`
- Confirmation came a couple of *hours* later, with a one-line patch
- Apparently the bug was there since Linux 2.1.8 (1996)
- Applied the patch, rebuilt and tested, reported back a couple of days later

skroutz

Random thoughts

What's so special about rsync?

- rsync operates as an aggressive pipeline

The checksums and file data are sent over the *same* TCP connection

- Each process sends data as fast as it can over the network
- There is no protocol-level signaling
- The pipeline is throttled using TCP flow control (zero windows)
- It lingers forever with no timeout

On Linux release management

- Patch commited to `netdev/net.git`
- Merged in mainline for 5.10-rc1
- Cherry-picked and applied to a series of stable kernels in linux-stable.git *before* 5.10 was released
- Trickled down to Debian Buster and Stretch a couple of months later

skroutz

On (kernel) observability

- Debugging this issue would have been a herculean task 10 years ago
- Tools like systemtap and LTTng made instrumenting a live kernel feasible
- Changes in distribution toolchains have improved DWARF symbol availability
- The past 10 years have seen a lot of work going into kernel observability, leading to standardized interfaces

skroutz

Observability today and tomorrow

- eBPF is eating the world. eBPF + perf_events + kprobes will replace most other tools (however learning curve is still steep and eBPF tooling has some way to go)
- `debuginfod` makes debug symbols available on demand, transparently, over the network
  - `debuginfod` *may* also make debugging containers easier

skroutz

Thank you!

https://engineering.skroutz.gr/

skroutz