TCP stack scalability

multiqueue NICS can hurt, when stack still use a 15 years old design

We use per socket spinlock (sk_lock.slock)

It mostly works in the fast path, when there is no contention.

We use RCU lookups (2.6.29+) to find ESTABLISHED/TIMEWAIT sockets. Tricky part is the SLAB_DESTROY_RCU to avoid having too much memory queued for freeing.

It mostly works for well behaving patterns, as packets can be processed in // on different cpus.

TCP listener syndrom

All non matching packets (SYN packets, ACK packets, ...) must lock the listener socket.

Many cpu can spin on this listener spinlock, and the hold duration of this spinlock can be very long in some pathological cases.

The SYNACK retransmit timer syndrom

For historical reasons, tcp_request_sock are very small (112 bytes), contain no individual timer for example.

So the SYNACK retransmit timer uses a single timer (per listener), and has to lock the listener when doing the retransmit logic.

If the listen(backlog) is increased too much, inet_csk_reqsk_queue_prune() takes age, while holding the listener lock. All other cpus are spining...

The recent (linux-3.1) change of initial SYNACK timer from 3 sec to 1 sec increased the problem by a 200% factor.

```
#define TCP_TIMEOUT_INIT ((unsigned)(1*HZ))       /* RFC6298 2.1 initial RTO value       */
```

Add a per {tcp}_request_sock timer

Permits better scalability, at the cost of an increase of memory requirements. It makes TCP_DEFER_ACCEPT handling way more practical. (check netdev archives for attempts to avoid spurious SYNACK retransmits)

Typical listener queue is limited by 65536 elements. Adding more SYN_RECFV is a problem because of the SYNACK retransmits (We still need SYNCOOKIES !)

65536*80 = 5 MBytes

In practice, many listeners use a much smaller backlog (somaxconn default value is 128, tcp_max_syn_backlog is 1024)

The SO_REUSEPORT problem

A SYN message has to select one listener in a list of available listeners, using a custom hash function.

Problems :

- This hash is different from the NIC mq or RPS hash.
- If a listener disapears or a new one is created,
  the third packet might not find the corresponding request_sock
- The thing doesnt really scale (say if you have 128 cpus)
- SYN packet (and 3rd packet) can be handled by a different cpu than the cpu doing the accept()
- many cpus can still spinlock on a listener, while SO_REUSEPORT was meant to address this problem.

Google

New common shared_listener structure

Each listener 'socket' has a link to a (possibly shared if SO_REUSEPORT is used) structure

- Structure protected by refcount and RCU

- SYN packets (or more generally all tcp packets) should only perform RCU lookups.

- Not sure we need to link all request_sock in this structure : We could rely on SYNACK retransmit timer to perform the garbage collection if the listener was dismantled. If we want to keep track of them, we probably need a hashed array to keep the cost of inserting/removing a request_sock small enough.

New common shared_listener structure (cont)

- Use a per_cpu_counter to keep track of count of request_sock
  (maybe need to check what are precise @backlog requirements : an atomic_t might be needed)

- Keep track of all real sockets bound to the same entity (SO_REUSEPORT)

- After 3WHS is completed, the selection of the socket can now use better strategy to favor cpu affinity, instead of a custom hash, having to parse all sockets. (no false sharing)

request_sock (SYN_RECV) inserted in regular TCP ehash table

- Current ehash has one chain for ESTABLISHED sockets, one chain for TIME_WAIT sockets.

524.288 slots on typical machines today.

- Extend TIME_WAIT chain to contain SYN_RECV sockets as well.

-> no need for a separate hash table, better cache locality.
-> no need for separate locking. Reuse the existing ehash_locks
-> The third packet of 3WHS does a regular lookup and finds the SYN_RECV immediately.

The listener socket becomes a small entity

Its lock protects:

- the fields read/updated by get/setsockopt()
- The accept() queue of sockets
- cloning the socket when 3WHS is completed

MISC & User visible changes

tcp_diag might be easier, we can remove some complex code dealing with the SYN_RECV sockets.

netstat (/proc/sys/net/tcp) / ss (netlink) output might interleave SYN_RECV 'sockets' in the other sockets, instead of giving separate blocks.

SYN attacks, and other kind of attacks dont hit anymore a highly contended spinlock. No more multiqueue multiplicative slowdown. No more hacks to try to filter the packets.