The Network Stack (2)

Lecture 6, Part 2: TCP Implementation
Prof. Robert N. M. Watson
2022-2023
### Evolving BSD/FreeBSD TCP implementation

<table>
<thead>
<tr>
<th>Year</th>
<th>Version</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>4.2BSD</td>
<td>BSD sockets, TCP/IP implementation</td>
</tr>
<tr>
<td>1986</td>
<td>4.3BSD</td>
<td>VJ/Karels congestion control</td>
</tr>
<tr>
<td>1999</td>
<td>FreeBSD 3.1</td>
<td>sendfile(2)</td>
</tr>
<tr>
<td>2000</td>
<td>FreeBSD 4.2</td>
<td>TCP accept filters</td>
</tr>
<tr>
<td>2001</td>
<td>FreeBSD 4.4</td>
<td>TCP ISN randomisation</td>
</tr>
<tr>
<td>2002</td>
<td>FreeBSD 4.5</td>
<td>TCP SYN cache/cookies</td>
</tr>
<tr>
<td>2003</td>
<td>FreeBSD 5.0-5.1</td>
<td>IPv6, TCP TIMEWAIT state reduction</td>
</tr>
<tr>
<td>2004</td>
<td>FreeBSD 5.2-5.3</td>
<td>TCP host cache, SACK, fine-grained locking</td>
</tr>
<tr>
<td>2008</td>
<td>FreeBSD 6.3</td>
<td>TCP LRO, TSO</td>
</tr>
<tr>
<td>2008</td>
<td>FreeBSD 7.0</td>
<td>T/TCP removed, socket-buffer autosizing</td>
</tr>
<tr>
<td>2009</td>
<td>FreeBSD 7.1</td>
<td>Read-write locking, full TCP offload (TOE)</td>
</tr>
<tr>
<td>2009</td>
<td>FreeBSD 8.0</td>
<td>TCP ECN</td>
</tr>
<tr>
<td>2012</td>
<td>FreeBSD 9.0</td>
<td>Pluggable TCP congestion control, connection groups</td>
</tr>
</tbody>
</table>

- ... changes continue to this day ... BBR, RCU, pluggable TCP, KTLS, ...
- Which changes have protocol-visible effects vs. only code?
Reminder: Send/receive paths in the network stack
Denial of Service (DoS) – state minimisation

- Yahoo!, Amazon, CNN taken down by SYN floods in February 2000
- Attackers exploit automatic state allocation to overload servers
  - TCP state itself
  - Underlying routing state
  - Cost of walking data structures
- Attackers spoof SYN packets with random source addresses
  - IPv4 address use is sparse, so no RST
- D. Borman: TCP SYN cache – minimise state for new connections
- D. Bernstein: SYN cookies – eliminate state entirely – at a cost
- J. Lemon: TCP TIMEWAIT reduction – minimise state during long close sequences (e.g., 2MSL)
- J. Lemon: TCP TIMEWAIT recycle – release state early under load

Figure 3: Time needed to connect() to remote system.

Completely OK

Basically 100% dysfunctional

% of connections completed

Time needed to connect() to remote system
TCP connection lookup tables (original BSD)

- Global list of connections for monitoring (e.g., netstat)
- Connections are installed in a global hash table for lookup
  - NB: separate (similar) hash table for 2-tuple port-number allocations
- Tables protected by global read-write lock as reads dominate
  - New packets are more frequent than new connections
Reminder - Work dispatch: input path

- **Deferred dispatch**: ithread → netisr thread → user thread
- **Direct dispatch**: ithread → user thread
  - Pros: reduced latency, better cache locality, drop early on overload
  - Cons: reduced parallelism and work placement opportunities
• Network bandwidth growth > CPU frequency growth

• Locking overhead (space, contention) substantial
  • Getting ‘speedup’ is hard!

• Evaluate different strategies for TCP processing parallelisation
  • Message-based parallelism
  • Connection-based parallelism (threads)
  • Connection-based parallelism (locks)

• Coalescing locks over connections:
  • reduces overhead
  • increases parallelism
Connection groups, RSS (FreeBSD)

- From FreeBSD 9.x: **Connection groups** blend MsgP and ConnP-L models
  - PCBs assigned to group based on 4-tuple hash
  - Lookup requires group lock, not global lock
  - Global lock retained for 4–tuple reservation (e.g., setup, teardown)
- Problem: have to look at TCP headers (cache lines) to place work!
  - Microsoft: NIC **Receive-Side Steering (RSS)**
  - Multi-queue NICs deliver packets to queues using hash of 4-tuple
  - Align connection groups with RSS buckets / interrupt routing
- From FreeBSD 12.x: **Read-Copy-Update (RCU)** rather than RW locks protect lists