Byte Queue Limits

Tomáš Hrubý

August 24, 2012

Google™
BQL - Motivation

- Packets spend enough time enqueued within the stack
- When a packet gets to a NIC it is enqueued again
- HW queue length in TX descriptors are decided ad-hoc
- The queue length is not directly related to bytes or time
What BQL does

• Reduces unnecessary queuing in the NIC by setting the ”right” limit
• Limits queuing to what the NIC is able to TX
• Pushes queuing from NIC to the network stack
• Reduces latency
The limit

- Based on bytes the NIC dequeued recently
- Changes dynamically
- Soft - allows to be exceeded once (overlimit)
- Grows fast and shrinks slowly
The algorithm

Generic implementation in dynamic queue limits (DQL) library

- Packets are queued up to a dynamic limit
- Packets should be *periodically* retired by a completion event
- Limit is increased whenever the NIC *starves* (dequeues too fast)
- Limit is decreased when the NIC cannot keep up for a while

Starvation

- When queuing is blocked (limit has been reached)
- All data were consumed
- Completion has not happened yet
Increasing the limit

Interval - the period between to invocations of `dql_completed()`

`ovlimit && !inprogress`

- Queue was certainly over limit in this interval

`dql->prev_ovlimit && all_prev_completed`

- Loose condition, queue might have been starving during the interval

Increment

* When queue is starved increase the limit by the amount
  * of bytes both sent and completed in the last interval,
  * plus any previous over-limit.
  */

```
limit += POSDIFF(completed, dql->prev_num_queued) +
        dql->prev_ovlimit;
```

- Limit grows aggressively, can more than double
- Limit stops increasing when it is sufficient
Decreasing the limit

\texttt{inprogress} \&\& \texttt{prev\_inprogress} \&\& \texttt{!all\_prev\_completed}

- Slack is only detected when the queue is busy for the entire interval.
- Decrease happens only when slacking for a period of time (1s).
- Conservative - decrease by the minimal slack over the period.
- Any increase in the limit resets the period.

2 flavors of slack

- \texttt{slack} = \texttt{POSDIFF(limit + dql\rightarrow prev\_ovlimit,}
  \texttt{2 * (completed - dql\rightarrow num\_completed));}

- \texttt{slack\_last\_objs} = \texttt{dql\rightarrow prev\_ovlimit ?}
  \texttt{POSDIFF(dql\rightarrow prev\_last\_obj\_cnt, dql\rightarrow prev\_ovlimit) : 0;}

BQL benefits

Reduced latency

![Graph showing reduced latency with BQL benefits]
BQL benefits

The "right" TX right size
# BQL benefits

## The ”right” TX Right size

<table>
<thead>
<tr>
<th>Packet size</th>
<th>Bit rate Mbps</th>
<th>Pkt rate kpps</th>
<th>Q size bytes</th>
<th>Q size pkts</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>~1025</td>
<td>~1490</td>
<td>~9577</td>
<td>~145</td>
</tr>
<tr>
<td>1514</td>
<td>~9995</td>
<td>~810</td>
<td>~122974</td>
<td>~81</td>
</tr>
</tbody>
</table>

single TX process, single queue only

- Default bnx2x TX descriptor queue size is 4078
- Default ixgbe TX descriptor queue size is 4096
- Using BQL, the actually used size is significantly smaller
BQL in action

Minimal packets ~1Gbps
BQL in action

Minimal packets ~1Gbps - initial ramp up
BQL in action

Maximal packets \(\sim 10\text{Gbps}\)
BQL in action

1 TCP stream ~10Gbps (TSO on)
BQL in action

Idle noise

![Chart showing bytes limit and inflight values]
BQL problems

The good news
No case where BQL on would work worse than BQL off (except non-cooperative hw)

The not so bad news
In some situations, BQL can be tricked to virtually turning itself off by setting the limit to high.

What is the trick?
Kernel does not guarantee that completion is called periodically
Completion not periodical

Periodicity required

- Impossible to say whether the interval was smaller or longer
- Shorter interval looks like congestion → slack
- Longer interval looks like the NIC can dequeue more → limit too low

Completion called in NAPI loop

- Some driver (e.g., bnx2x, gianfar) retire as much as they can
- Last iteration usually retires much less
- Retiring when exiting NAPI may lead to excessively long interval
- Different interval between iterations and new polling start
Excessively Large limit

- Occasional large interval leads to large limits
- Large limit leads to large increments, which leads to larger limit ...
- Limit decreases only after fairly long period
- Zero or negative slack cancels decreasing (minimum!)
- Large limit tends to stay
Excessive decrease

- Due to occasionally shorter intervals on a loaded link
- Sudden bursts on a lightly loaded link
- Much less of a problem than large limit
- Instantly fixed by immediate limit increase
  - Although the limit is low, the NIC dequeues a lot since the queue is long due to the previously large limit
  - The large dequeue is usually complete
Observation

- Limit tends to change when timing changes for the same load
Possible cure

Execute completion only when fixed amount of time elapsed

- What should be the period?
- Needs a timer to deal with corner cases
- Allows calling a completion wrapper for each packet
128 Streams vs 1 RR

<table>
<thead>
<tr>
<th>Feedback</th>
<th>RR rate rps</th>
<th>Qlen pkt</th>
</tr>
</thead>
<tbody>
<tr>
<td>no BQL (DROP/ECN)</td>
<td>~94</td>
<td>~28</td>
</tr>
<tr>
<td>DROP</td>
<td>~140</td>
<td>~113</td>
</tr>
<tr>
<td>ECN</td>
<td>~749</td>
<td>~82</td>
</tr>
</tbody>
</table>

- Codel has little effect for RR without BQL
- DROP results in a huge tail latency due to retransmissions
Conclusions

- BQL works fine in vast majority of cases
- BQL may set excessive limit
- Never worse than not having BQL
- Essential for CoDel