BPF HOST NETWORK RESOURCE MANAGEMENT

Lawrence Brakmo
Alexei Starovoitov

Facebook

brakmo@fb.com  ast@fb.com
Introduction

• Linux supports allocating and managing many system resources such as CPU and memory.

• Network allocation and management is harder since it is both a local and a global resource.

• Require mechanisms to allocate and manage bandwidth both locally (i.e. per cgroup) and externally (i.e. per link or per switch).

• Ingress bandwidth management requires notifying senders to slow down.
Traffic control

• Current mechanism, traffic control (tc), allows shaping of outgoing traffic and policing of incoming traffic.
• It has been used for managing external bandwidths (e.g. Google’s BwE).
• However, tc has a history of performance issues when using many htb (Hierarchical Queuing Discipline) queues.
• tc bandwidth allocation usually results in standing queues* (other issues with codel).
• Lacks the flexibility usually provided by general programming constructs.
Goals

• a BPF based framework (NRM) for efficiently supporting shaping of both egress and ingress traffic based on both local and global network allocations.

• Initial assumption that majority of traffic is TCP (or it has similar congestion control).

• Eliminate/reduce standing queues.

• Flexibility (comes for free with BPF).
Overview

• Use existing egress and ingress cgroup skb hooks.
• For egress use ECN, calls to tcp_enter_cwr, or drops.
• For ingress use ECN (or similar) to notify sender to slow down.
• Use scopes to manage bandwidth
  • E.g. cgroup scope, particular link/switch scope, ...
  • Each socket belongs to a set of scopes
  • When sending a packet we update the bw utilization of the socket’s scopes
  • Congestion is determined by the most congested scope
Consider 2 flows from hosts 1 to host N belonging to the same service:

- Flow 1 could belong to scopes Red, Green & Blue.
- Flow 2 belongs to scopes Orange, Green & Blue.
BW management

• We use a virtual queue to track bw use (per scope)
  • Struct vqueue {
    int credit; /* in bytes */
    long long lasttime; /* in ns */
  }

• When sending a packet:
  • Credit += credit_per_ns(currtime – lasttime, rate); // need to bound
  • Credit -= wire_length_in_bytes(skb); // need to account for TSO

• Make decision based on credit and packet info
• Have to account for GRO and LRO packets
Current Congestion Algorithm

- If credit < Drop Threshold, then drop it (small packet buffer)
- If credit < Mark Threshold, then “mark it”
  - ECN: mark it
  - TCP – non-ECN: call tcp_cwr_enter() with a linear probability. The closer credit is to Drop Threshold, the more likely to call cwr
- Virtual buffer to absorb bursts
- Drop threshold: 600pkts (reserved space for small packets)
  - Mark threshold: 120pkts
CUBIC MARK FUNCTIONS

Current

Explore other response functions
Issues

• Packets dropped by cgroup skb BPF program do not trigger call to enter_cwr (cwnd reduction).
  • Solution: helper BPF function to call tcp_enter_cwr
  • Advantage: can make better decisions (probabilistic reductions)

• High tail latencies due to dropping packet and no more packets in transit (packets_out = 0).
  • For example, 1Gbps bound and 9 flows within rack => cwnd should be less than one.
  • When no more ACKs to trigger new packets, TCP depends on probe0 timer to resend. Default >= 200ms
  • Solution: reduce probe timer to 20ms in these cases
Issues (2)

• Update of Credit and Lasttime is a critical section
  • Needs to be protected
  • Currently we do not have spinlocks in BPF programs
  • Hack: spinlock the whole BPF program
  • Fix 1: work on bpf_spinlock support is happening in parallel
  • Fix 2: use data structure not requiring locks
experiments

- Only 1 scope (belonging to 1 cgroup)
- One hosts sends to another host in a rack
- One or more 1MB and 10KB RPCs
  - 1 - 1MB
  - 1 - 1MB and 1 - 10KB
  - 4 - 1MB and 1 - 10KB
  - 8 - 1MB and 1 - 10KB
- Limit rate by either by NRM or TC (htb)
- Introduce latency by netem on receiving host
### Experiment 1Gbps rate
Test smaller probe timer, 1MB RPC Latency

<table>
<thead>
<tr>
<th># Flows</th>
<th>Cubic Aggr BW</th>
<th>Cubic 99.9% Lat</th>
<th>DC-TCP Aggr BW</th>
<th>DC-TCP 99.9% Lat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt; timer</td>
<td>&lt; timer</td>
<td>&lt; timer</td>
<td>&lt; timer</td>
</tr>
<tr>
<td>1</td>
<td>496M</td>
<td>794M</td>
<td>250ms</td>
<td>84ms</td>
</tr>
<tr>
<td>2</td>
<td>856M</td>
<td>922M</td>
<td>260ms</td>
<td>44ms</td>
</tr>
<tr>
<td>5</td>
<td>935M</td>
<td>989M</td>
<td>465ms</td>
<td>92ms</td>
</tr>
<tr>
<td>9</td>
<td>999M</td>
<td>1006M</td>
<td>600ms</td>
<td>345ms</td>
</tr>
</tbody>
</table>

- Reducing probe timer to 20ms reduces tail latency significantly!
- From now on assume reduced probe timer
1Gbps Limit: Aggregate Rate and RTT

- **Experiments:**
  - cubic ecn=0 1G TC
  - dctcp 1G
  - cubic ecn=0 1G
  - cubic ecn=1 1G

- **Graph Parameters:**
  - **Y-axis:** Aggregated Rate (Mbps)
  - **X-axis:** Number of Flows
  - **RTT (us)**

- **Observations:**
  - The graph shows the aggregate rate and RTT for different numbers of flows and varying congestion control methods.
  - Different congestion control methods impact the aggregate rate and RTT differently.
  - The red line indicates a specific trend or result that is highlighted for comparison.
1Gbps Limit: 1MB and 10KB Rates

The diagram illustrates the average rate of 1MB RPCs (Mbps) and the rate of 10KB RPCs (Mbps) as the number of flows increases. The graph shows different experiment conditions and their corresponding performance metrics.
1Gbps Limit: 1MB RPC Latencies

Experiment
- cubic ecn=0 1G TC
- dctcp 1G
- cubic ecn=0 1G
- cubic ecn=1 1G

99.9% Latency (ms)

Number of Flows
- 1c1s-1f
- 1c1s-2f
- 1c1s-5f
- 1c1s-9f

50% Latency (ms)
1 Gbps Limit Conclusions

• Similar aggregate rate (except for 1 flow Cubic)
• High RTTs when using TC (standing queue)
  • Default output_limit_bytes is 260KB, at 1Gbps => 2ms
• TC is size unfair, 10KB RPCs get up to 20x less rate
• Cubic and Cubic-ecn have higher 1MB RPC tail latencies
• DC-TCP has much better 10KB tail latency (10x to 80x lower)
## 1 Gbps Limit & 10ms

<table>
<thead>
<tr>
<th>Cong Control</th>
<th>qdisc</th>
<th>Rate Control</th>
<th>1-flow Rate (Mbps)</th>
<th>9-flow Aggr Rate (Mbps)</th>
<th>1-flow 1-MB 99.9% Lat (ms)</th>
<th>9-flow 1-MB 99.9% Lat (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cubic</td>
<td>HTB – fq</td>
<td>HTB</td>
<td>441</td>
<td>858</td>
<td>58</td>
<td>85</td>
</tr>
<tr>
<td>Cubic</td>
<td>HTB</td>
<td>HTB</td>
<td>437</td>
<td>945</td>
<td>20</td>
<td>120</td>
</tr>
<tr>
<td>Cubic</td>
<td>mq – fq</td>
<td>NRM-BPF</td>
<td>410</td>
<td>915</td>
<td>46</td>
<td>141</td>
</tr>
<tr>
<td>Cubic</td>
<td>mq – fqc</td>
<td>NRM-BPF</td>
<td>754</td>
<td>944</td>
<td>12</td>
<td>218</td>
</tr>
<tr>
<td>DC-TCP</td>
<td>mq – fq</td>
<td>NRM-BPF</td>
<td>666</td>
<td>947</td>
<td>13</td>
<td>143</td>
</tr>
</tbody>
</table>
1 Gbps Limit & 10ms RTT Conclusions

- Best rate is achieved with Cubic, mq-fq_codel and NRM-BPF for rate control
  - However, 9-flow tail latency is higher at 218ms
- Using Cubic with HTB for rate control reduces tail latency (85 or 120ms)
  - However, 1-flow rate decreases (to 441 from 760Mbps) and also increases 1-flow tail latency (to 20 or 58ms from 12ms).
- Using DC-TCP with NRM-BPF for rate control produces results between the previous 2: 666Mbps and 13/120ms tail latencies
  - Note: There seems to be an issue with DC-TCP that may increase latencies.
## 5Gbps Limit

<table>
<thead>
<tr>
<th>Cong Control</th>
<th>qdisc</th>
<th>Rate Control</th>
<th>Aggr Rate (Mbps)</th>
<th>1-MB 99.9% Lat (ms)</th>
<th>10-KB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-flow</td>
<td>9-flow</td>
<td>1-flow</td>
<td>9-flow</td>
<td>Rate (Mbps)</td>
</tr>
<tr>
<td>Cubic</td>
<td>HTB</td>
<td>HTB</td>
<td>4.5</td>
<td>4.8</td>
<td>2.0</td>
</tr>
<tr>
<td>Cubic</td>
<td>mq-fqc</td>
<td>NRM-BPF</td>
<td>4.2</td>
<td>4.7</td>
<td>6.5</td>
</tr>
<tr>
<td>Cubic-ECN</td>
<td>mq-fqc</td>
<td>NRM-BPF</td>
<td>4.4</td>
<td>4.2</td>
<td>4.7</td>
</tr>
<tr>
<td>DC-TCP</td>
<td>mq-fqc</td>
<td>NRM-BPF</td>
<td>4.6</td>
<td>4.9</td>
<td>4</td>
</tr>
</tbody>
</table>
5Gbps Limit Conclusions

- Cubic with HTB for rate control produce the best 1MB results at the cost of 10KB results
  - 9-flow 1MB: 4.8Gbps and 18ms tail latency
  - 9-flow 10KB: 35Mbps and 3.7ms tail latency
- NRM-BPF produced much better 10KB results, but worst 1MB results
  - Cubic 9-flow 1MB: 4.2Gbps and 113ms tail latency
  - Cubic 9-flow 10KB: 243Mbps, 0.8ms tail latency
  - DC-TCP 9-flow 1MB: 4.6Gbps, 27ms tail latency
  - DC-TCP 9-flow 10KB: 295Mbps, 0.8ms tail latency
NRM Ingress

• Similar idea: use a virtual queue to track usage
• Want a mechanism to notify sender
  • Otherwise need to depend on packet drops (bad)
• Options
  • DC-TCP – uses ECN to notify sender
  • Cubic – use a side channel to notify sender
    • Use ECN markings (maybe ECT1 if ECT0 is default as in Linux)
    • BPF program on other side does cwr
• Incast prevention does not drop, only mark so it can use switch buffers
# DCTCP Ingress

<table>
<thead>
<tr>
<th>Limit (Mbps)</th>
<th># Flows</th>
<th>Aggregate Rate (Mbps)</th>
<th>Retrans</th>
<th>99.9% Latency (ms)</th>
<th>50% Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>925</td>
<td>0</td>
<td>9.5</td>
<td>9.0</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>922</td>
<td>0</td>
<td>19.0</td>
<td>13.0</td>
</tr>
<tr>
<td>1000</td>
<td>5</td>
<td>931</td>
<td>0</td>
<td>47.9</td>
<td>43.0</td>
</tr>
<tr>
<td>1000</td>
<td>9</td>
<td>945</td>
<td>1493</td>
<td>336.0</td>
<td>54.0</td>
</tr>
<tr>
<td>5000</td>
<td>1</td>
<td>4600</td>
<td>0</td>
<td>4.1</td>
<td>1.7</td>
</tr>
<tr>
<td>5000</td>
<td>2</td>
<td>4600</td>
<td>0</td>
<td>4.7</td>
<td>1.9</td>
</tr>
<tr>
<td>5000</td>
<td>5</td>
<td>4670</td>
<td>0</td>
<td>12.4</td>
<td>7.7</td>
</tr>
<tr>
<td>5000</td>
<td>9</td>
<td>4630</td>
<td>0</td>
<td>18.5</td>
<td>15.5</td>
</tr>
</tbody>
</table>
Conclusions

- Egress NRM prevents standing queues (i.e. smaller RTT) as long as host average BW utilization is smaller than NIC rate.
- As a result smaller RPCs have smaller latencies
- Using BPF provides great flexibility and is a great platform for experimentation.
Future work

• Explore different marking algorithms (response functions)
• Explore using connection RTT in marking algorithm
• Test multiple scopes
  • Multiple cgroups (each flow only has one scope)
  • Multiple scopes per flow
• Test concurrent flows with different RTTs
• Test concurrent flows with different TCP variants
• Ingress NRM with sender notifications