TCP-NV
Congestion Avoidance for Data Centers

Lawrence Brakmo

Google
TCP Congestion Control

- Algorithm for utilizing available bandwidth without too many losses
  - No attempt at eliminating losses
  - Works well at eliminating congestion collapse
- TCP’s congestion control will continually increase its congestion window (# packets in transit) until losses occur
  - rate = cwnd / RTT
- I.e. congestion control repeatedly creates congestion and packet losses
- Unfairness
  - Between flows with different RTTs (up to 20-40x)
- Works well for large transfers and it has proven to work well in many different environments
Congestion Window Graph
Data Center Perspective

- RPCs make large percent of traffic
  - Important to test TCP behavior with RPCs
- Latency sensitive
- Congestion control results in queue build-up at host, switches and routers
  - Increasing average latency of smaller size RPCs
- Congestion control results in periodic packet losses
  - Increasing high percentile latency of smaller size RPCs
- Need Congestion avoidance
  - Reduce queue buildup and losses
Congestion Avoidance

- **Goal:** Prevent queue build-up and packet losses while making full utilization of available bandwidth
  - Only grow congestion window when there is bandwidth available
  - Decrease congestion window when congestion (queue build-up) is detected
    - Unlike congestion control which only decreases with losses
- **Typically delay (RTT) based**
  - Increase in RTT implies queue buildup
  - Problem: RTT is VERY noisy
TCP-NV Congestion Avoidance

- We can only detect the level of congestion, not the available bandwidth

![Graph showing TCP-NV congestion avoidance](image)

- Max Slope
- Grow cwnd
- cwnd = 19
Basic Concept of TCP-NV Congestion Avoidance

- In a perfect world, and while there is no congestion, if a cwnd of $X$ achieves rate $Y$, then a cwnd of $2X$ should achieve rate $2Y$
TCP-NV Implementation

- In Linux 2.6.34, soon in 2.6.36
- On top of BIC
  - BIC behavior when there is no congestion
- Most of the changes are self contained in one CA function: `nvtcp_acked()`
  - Collects appropriate info
  - Every so often makes a CA decision
    - Allow BIC to grow cwnd
    - Don't allow BIC to grow cwnd
    - Decrease cwnd and don't allow BIC to grow cwnd
- Need to store in-flight every time a packet is sent
  - Needed by `ntcp_acked()`, stored in skb
- Can be disabled through `sysctl` (in which case it behaves like BIC)
More Implementation Details

- Every time a packet is sent
  - Store time in us, bytes in-flight

- Every time an ACK is received
  - Use sent time and in-flight for newest packet ACKed
  - Calculate slope = rate/in-flight
  - Keep largest slope seen so far (recalc every so often)
  - Keep largest rate seen since last CA decision

- Every so often
  - \( \text{pred}_\text{cwnd} = \frac{\text{max}_\text{rate}}{\text{max}_\text{slope}} \)
  - if \( \text{pred}_\text{cwnd} + \text{pad} > \text{cwnd} \)
    - Reduce cwnd by percent of congestion
    - Stop BIC from growing cwnd
  - else
    - Allow BIC to grow cwnd
Congestion Window Graph
Issues

- **TSO, LRO and interrupt coalescence**
  - Reduces number of data points
    - May only get one ACK per RTT
  - Need to inflate window for one flow to use full bandwidth

- **Coexistence with congestion control**
  - As CC increases cwnd, CA reduces its own
  - Usually cannot do both
  - Unless RTT of CA is << RTT of CC
    - Traffic within DC CA, traffic outside DC CC

- **TCP-NV needs pacing when** RTT > 1 to 5ms
  - due to big bursts when starting a new RPC
  - not an issue for traffic within a data center or cluster

- **Cannot do CA with small RPCs**
  - Needs a couple of RTTs of data
## CC vs CA Fairness

<table>
<thead>
<tr>
<th>NV</th>
<th>small RTT</th>
<th>large RTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>small RTT</td>
<td><strong>BIC 5X &gt; NV</strong></td>
<td><strong>BIC 10-100X &gt; NV</strong> (avoid)</td>
</tr>
<tr>
<td>large RTT</td>
<td><strong>NV 2-5X &gt; BIC</strong> (= BIC vs. BIC)</td>
<td><strong>NV losses ~0</strong></td>
</tr>
</tbody>
</table>
Issues ...

- Reverse congestion
  - Congestion in the ACK direction will increase RTT
  - CA mechanism will react to this

- Solutions
  - Prioritize pure ACKs in host and switches/routers
    - so they are not affected by congestion
  - Measure reverse delay and adjust appropriately
    - New TCP option
    - Use TCP us Timestamp option
  - Limit how much NV can decrease the cwnd
Issues ...

- Too many flows decrease effectiveness of Congestion Avoidance
  - # of flows w/o losses is a function of router buffer size
  - With 1.2MB buffer size, can handle up to 128 bulk flows with no losses
    ■ This has improved and we are working on further improvements
Results

- **Simulations using actual Linux network stack**
  - Using Sam Jansen's Network Simulation Cradle
  - Up to 1000 hosts, 50us to 100ms RTTs
  - Can examine detail behavior of protocol, routers, etc.
  - Can easily move the code to Linux

- **Rack tests with 1G and 10G NICs**
  - Actual hardware
    - Effects of TSO, LRO, interrupt coalescence

- **Starting tests with production workloads**
  - More machines (currently 38)
  - Actual workloads
TCP Burstiness with Small RTT Large RPCs

- Experiment: 128 flows, 100us RTT
- Graph: In-flight (dark blue) and rate (aqua) of 1 flow
- Each large peak is the result of starting a new RPC.
TCP Burstiness with TCP-NV

Host 4 (2.6.18 nv)
Number of RTOs: 0

Packets

Time in seconds

Rate (Mbps)
Router Queues with 128 flows
1-8 MB RPCs, 100us RTT, 1.2MB switch buffer

TCP-BIC

13,000 packet losses/s

TCP-NV

0 packet drops
Router Queues with 128 flows
1-8 MB RPCs, 100us RTT, 175KB switch buffer

TCP-BIC
17,800 packet drops/s

TCP-NV
75 packet drops/s
## Simulator, 10Gbps, ~1MB RPCs

<table>
<thead>
<tr>
<th></th>
<th>TCP-BIC avg of 2,8,32 flows</th>
<th>TCP-NV avg of 2,8,32 flows</th>
<th>TCP-BIC 128 flows</th>
<th>TCP-NV 128 flows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goodput (Gbps)</strong></td>
<td>8.1</td>
<td>8.8</td>
<td>9.5</td>
<td>9.6</td>
</tr>
<tr>
<td><strong>Fairness</strong></td>
<td>0.96</td>
<td>0.94</td>
<td>0.82</td>
<td>0.93</td>
</tr>
<tr>
<td><strong>Average latency</strong></td>
<td>642 us</td>
<td>251 us</td>
<td>1.7 ms</td>
<td>527 us</td>
</tr>
<tr>
<td><strong>90% latency</strong></td>
<td>448 us</td>
<td>274 us</td>
<td>589 us</td>
<td>559 us</td>
</tr>
<tr>
<td><strong>95% latency</strong></td>
<td>494 us</td>
<td>285 us</td>
<td>14.5 ms</td>
<td>584 us</td>
</tr>
<tr>
<td><strong>99% latency</strong></td>
<td>11.9 ms</td>
<td>305 us</td>
<td>21.0 ms</td>
<td>616 us</td>
</tr>
<tr>
<td><strong>Average queue size</strong></td>
<td>94.6</td>
<td>39.3</td>
<td>152</td>
<td>194</td>
</tr>
<tr>
<td><strong>Packets dropped/sec</strong></td>
<td>19000</td>
<td>0</td>
<td>63400</td>
<td>5300</td>
</tr>
</tbody>
</table>
## 2 1G hosts sending to 1G host (Rack test)

<table>
<thead>
<tr>
<th>Flows per host</th>
<th>Busy %</th>
<th>avg cwnd</th>
<th>avg RTT (ms)</th>
<th>Aggregate Rate (Mbps)</th>
<th>Losses %</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIC</strong></td>
<td>1</td>
<td>9.3</td>
<td>119</td>
<td>2.2</td>
<td>918</td>
</tr>
<tr>
<td><strong>NV</strong></td>
<td>1</td>
<td>3.3</td>
<td>14</td>
<td>1.0</td>
<td>925</td>
</tr>
<tr>
<td><strong>BIC</strong></td>
<td>4</td>
<td>11.0</td>
<td>25</td>
<td>1.9</td>
<td>952</td>
</tr>
<tr>
<td><strong>NV</strong></td>
<td>4</td>
<td>6.0</td>
<td>6.4</td>
<td>1.0</td>
<td>944</td>
</tr>
<tr>
<td><strong>BIC</strong></td>
<td>8</td>
<td>9.6</td>
<td>22</td>
<td>1.8</td>
<td>981</td>
</tr>
<tr>
<td><strong>NV</strong></td>
<td>8</td>
<td>6.5</td>
<td>6.5</td>
<td>1.0</td>
<td>974</td>
</tr>
</tbody>
</table>
### 2 1G hosts sending to 1G host, Flow Control Enabled

<table>
<thead>
<tr>
<th></th>
<th>Flows per host</th>
<th>Busy %</th>
<th>avg cwnd</th>
<th>avg RTT (ms)</th>
<th>Aggregate Rate (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIC</td>
<td>1</td>
<td>7.5</td>
<td>737</td>
<td>15</td>
<td>943</td>
</tr>
<tr>
<td>NV</td>
<td>1</td>
<td>2.9</td>
<td>14</td>
<td>1</td>
<td>938</td>
</tr>
<tr>
<td>BIC</td>
<td>4</td>
<td>8.7</td>
<td>736</td>
<td>66</td>
<td>944</td>
</tr>
<tr>
<td>NV</td>
<td>4</td>
<td>5.9</td>
<td>7</td>
<td>1</td>
<td>944</td>
</tr>
<tr>
<td>BIC</td>
<td>8</td>
<td>8.9</td>
<td>737</td>
<td>134</td>
<td>944</td>
</tr>
<tr>
<td>NV</td>
<td>8</td>
<td>7.2</td>
<td>8</td>
<td>1.5</td>
<td>944</td>
</tr>
</tbody>
</table>
### Distributed Sorting (38 1Gbps machines, real HW)

<table>
<thead>
<tr>
<th>TCP version</th>
<th>Execution Time</th>
<th>Retransmits</th>
<th>RTOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cubic</td>
<td>baseline</td>
<td>baseline</td>
<td>baseline</td>
</tr>
<tr>
<td>BIC</td>
<td>-3%</td>
<td>+36%</td>
<td>-25</td>
</tr>
<tr>
<td>NV</td>
<td>-5%</td>
<td>-45%</td>
<td>-17%</td>
</tr>
</tbody>
</table>
Conclusions

- Congestion avoidance is possible in the data center
  - Can insure everyone is running CA
- Can reduce losses and queue buildup
  - But these are workload dependent
- Can reduce cpu utilization
  - Due to fewer losses and smaller host queues
- Small changes to the network stack
  - Most of the code in a new ca module.
- Started running larger tests
  - multi-rack to cluster size
- Code will be released in December
The End