Linked Congestion Control

Costin Raiciu, UCL
Multipath TCP at work

• Source can use both paths to send traffic
• How should it allocate traffic to the two paths?
  – Using a window based protocol
  – Playing fair with TCP
Multipath TCP at work

• Source can use both paths to send traffic
• How should it allocate traffic to the two paths?
  – Using a window based protocol
  – Playing fair with TCP
Aims

• **Goal 1** (improve throughput): when compared to using the best single path
Aims

• **Goal 1** (improve throughput): when compared to using the best single path
• **Goal 2** (do no harm): on any available path, take at most the same throughput a single TCP would
Aims

• **Goal 1** (improve throughput): when compared to using the best single path

• **Goal 2** (do no harm): on any available path, take at most the same throughput a single TCP would

• **Goal 3** (balance congestion) move traffic onto least congested links as long as goals 1 and 2 are met
Goals 1&2 Imply Bottleneck Fairness

Diagram:
- Source (Src)
- Source TCP (Src_{TCP})
- Destination TCP (Dst_{TCP})
- Destination (Dst)

Connections:
- Source to Source TCP
- Source TCP to Destination TCP
- Destination TCP to Destination
Goal 3 Implies Resource Pooling
This Talk

• Show that goals can be met
• Present a simple, safe, deployable protocol
  – Achieves reasonable resource pooling
• There are probably other solutions that
  – Get better resource pooling
  – Are possibly safe to deploy
  – We just don’t know them yet
Default

• Use independent TCP CC on each path
Default

- Use independent TCP CC on each path
- Problem: bottleneck fairness
Default

- Use independent TCP CC on each path
- Problem: bottleneck fairness
- Problem: resource pooling
Solution:
Couple Congestion Controllers

- $w_r$ - congestion window on subflow $r$
- $w = \text{sum}(w_r)$
- **Fully Coupled** algorithm
  - Increase $w_r$ by $1 / w$ per ack on subflow $r$
  - Decrease $w_r$ by $w / 2$ per drop on subflow $r$
- Behaves like a single TCP
Fully Coupled is Flappy
Better Solution

• **Linked Increases** Algorithm
  – Increase $w_r$ with $a / w$ for each ack on subflow $r$
  – Decrease $w_r$ by $w_r / 2$ for each drop on subflow $r$

• $a$ is a parameter that controls aggressiveness
Linked Increases

• Not Flappy
Resource Pooling of Linked Increases Algorithm
Effect of RTT

• Assume equal drop rates: $p_1 = p_2$

$$v_1 = 10$$

$$v_2 = 10$$
Equal RTTs

• Assume equal drop rates: $p_1=p_2$

\[ v_1 = 10 \quad \text{RTT}_1 = 10\text{ms} \quad \Rightarrow \quad 1000\text{pkts/s} \]

\[ v_2 = 10 \quad \text{RTT}_2 = 10\text{ms} \quad \Rightarrow \quad 1000\text{pkts/s} \]

Rate = 2000 pkts/s
Dissimilar RTTs

• Assume equal drop rates: $p_1 = p_2$

Rate = 1100 pkts/s
Dissimilar RTTs

- Assume equal drop rates: $p_1 = p_2$

$v_1 = 10 \quad \text{RTT}_1 = 10\text{ms}$
$v_2 = 10 \quad \text{RTT}_2 = 100\text{ms}$

Rate = 1100 pkts/s

A TCP on path 1 would get 2000 pkts/s
Multipath is doing worse!
Dissimilar RTTs

Assume $p_1 > p_2$, so $w_1 < w_2$
Dissimilar RTTs

Uncoupled TCP on path 1
Dissimilar RTTs

Uncoupled TCP on path 1
Uncoupled TCP on path 2
Dissimilar RTTs

![Graph showing Dissimilar RTTs with Uncoupled Multipath TCP]
Dissimilar RTTs

Assume $\text{RTT}_1 < \text{RTT}_2$, and $\text{TCP}_1 > \text{TCP}_2$
Dissimilar RTTs

Lines of Equal Throughput

Uncoupled Multipath TCP
Dissimilar RTTs

Uncoupled TCP

Minimum Throughput for Multipath
Dissimilar RTTs

Acceptable Throughput for Multipath

Uncoupled TCP
Dissimilar RTTs

Uncoupled TCP

Resource Pooling
Dissimilar RTTs

Linked Increases

Uncoupled TCP

Resource Pooling
Dissimilar RTTs

Linked Increases

Uncoupled TCP

Resource Pooling
It Works

• Experiment

• Results (Mb/s)

<table>
<thead>
<tr>
<th></th>
<th>Coupled</th>
<th>Linked Inc.</th>
<th>Uncoupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src_A</td>
<td>7.1</td>
<td>5.3</td>
<td>4.8</td>
</tr>
<tr>
<td>Src_B</td>
<td>3.3</td>
<td>5.4</td>
<td>5.8</td>
</tr>
<tr>
<td>Src_C</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Conclusions

• We must couple congestion control loops to get resource pooling and bottleneck fairness
• It is not hard to do so
  – Must remove flappiness
  – Must take into account RTT fairness
• Our proposal
  – Simple and works
  – We have a working implementation
• Other solutions possible
It Works

- Simulation Run: $p_1=p_2=1/1000$, 5 RTT$_1$=RTT$_2$