Adaptive Weighted Traffic Splitting in Programmable Data planes

Kuo-Feng Hsu*, Praveen Tammana†, Ryan Beckett‡, Ang Chen*, Jennifer Rexford†, David Walker†

Rice University*, Princeton University†, Microsoft Research‡
Multiple paths between source-destination pairs

Datacenter networks
Capacity: 20.4Tbps
Switch chip: 32X40G
16 rack switches*

Efficient load balancing is crucial to achieve good performance

Private WANs
E.g: B4, SWAN
33 datacenter sites* and growing..

*src: B4andAfter [SIGCOMM’18].
Motivation: Load-aware traffic splitting

Fast adaptation to real-time traffic conditions at RTT timescales

Equal split is desired

Unequal split is desired

Traffic flows

load = 0

load > 0
Existing approaches

<table>
<thead>
<tr>
<th>Operates entirely in data plane</th>
<th>Control plane + Data plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.g.: HULA [SOSR’16], CONGA [SIGCOMM’14]</td>
<td>E.g.: HALO [ToN’15], TeXCP [SIGCOMM’05]</td>
</tr>
</tbody>
</table>

- **Operates entirely in data plane**
  - E.g.: HULA [SOSR’16], CONGA [SIGCOMM’14]
  - Overloads the best path when RTTs are large

- **Control plane + Data plane**
  - E.g.: HALO [ToN’15], TeXCP [SIGCOMM’05]
  - Unclear how these solutions can be realized using commodity data planes
In this work...

Question

How to balance load dynamically across multiple paths in the data plane?

Contributions

1. We design new data structures for load-aware traffic splitting
2. We characterize and study tradeoffs of these data structures
3. We propose a data structure called DASH
Key problems

(1) Spreading flows using a **Path-to-Weight** data structure

(2) Updating that data structure as probes arrive
PISA: Protocol Independent Switch Architecture
But, a constrained computational model

1. Limited #per-packet register accesses

2. No access to registers mapped to a different stage

3. Small #stages and limited computation in each stage
Existing technique: Weighted Cost Multipathing (WCMP)

Replicates table entries with same pathID in proportion to its weight

Paths: A, B, and C

Weight vector
1 : 2 : 3

How can we update the table as probes arrive?

WCMP table is stored in a stage registers
Updating WCMP table

Iterate over the table

Current vector
1 : 2 : 3

Desired vector
3 : 2 : 1

Requires many per-packet accesses to a stage registers
Updating WCMP table

Assign entries of non-deficit paths to deficit paths

<table>
<thead>
<tr>
<th>Path</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
</tbody>
</table>

Desired vector

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Before packet 1

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
</tbody>
</table>

After packet 1

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

After packet 2

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

Final result

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
</tbody>
</table>

❌ Requires read and write to same register from different stages
DASH: Data-plane Adaptive Splitting with Hash threshold

Replicate pathID in WCMP table

Idea: Partition hash space into unique regions of size proportion to path weights

Weight vector: 1 : 2 : 3

Packet header → Hash → H = Hash value

0 1 3 6
A B C

Stage 1: H < 1
Stage 2: H < 3
Stage 3: H < 6

Boundaries are stored in per-stage registers

Check packet’s hash value against boundaries
Updating DASH boundaries is simple

New path boundary = Previous path’s boundary + Path region size

Desired vector = 3 : 2 : 1

Stage 1

Probe

Stage 2

Stage 3

Fast and efficient

One per-packet access to stage registers
Read-write in the same stage

Used per-stage SALUs to execute addition operation
Evaluation

- Prototype: Bmv2
- Environment: Ns3
- Workload:
  - Web search [Alizadeh-SIGCOMM'15]
- Schemes
  - CP (Control plane): slow adaptation
  - Hula: fast but single best path
  - ECMP: traffic splitting but no adaptation
  - DASH: traffic splitting and fast adaptation
Summary and future work

• Adaptive weighted traffic splitting in the data plane

• **DASH** is fast and efficient

• Future work:
  • Implementation in commodity switches (e.g., P4-Tofino)
  • Building a distributed online traffic engineering system
Thank you