Friends, not Foes – Synthesizing Existing Transport Strategies for Data Center Networks

Ali Munir
Michigan State University

Ghufran Baig, Syed M. Irteza, Ihsan A. Qazi, Alex X. Liu, Fahad R. Dogar
Data Center (DC) Applications

- **Distributed applications**
  Components interact via the network
e.g., a bing search query touches > 100 machines

- **Network impacts performance**
  “10% of search responses observe 1 to 14 ms of network queuing delay”
  [DCTCP, SIGCOMM 10]
DC Network Resource Allocation

- **Fair Sharing**
  Equal bandwidth sharing among jobs [TCP, DCTCP]
  - Increases completion time for everyone
  - Traditional “fairness” metrics less relevant

- **QoS Aware**
  Prioritize some jobs over other jobs (Priority Scheduling)
  - Minimize flow completion times [pFabric, L²DCT]
  - Meet flow deadlines [D³, D²TCP]
DC Transports

- DCTCP (SIGCOMM'10)
- D²TCP (SIGCOMM'12)
- L²DCT (INFOCOM'13)
- D³ (SIGCOMM'11)
- pFabric (SIGCOMM'13)
- PDQ (SIGCOMM'12)
DC Transports

- DCTCP
  - SIGCOMM
- D²TCP
  - SIGCOMM
- L²DCT
  - INFOCOM

Near Optimal but not Deployment Friendly
(Changes in data plane)
DC Transports

- DCTCP
  - Deployment Friendly but Suboptimal
  - SIGCOMM'12

- D²TCP

- Near Optimal but not Deployment Friendly (Changes in data plane)
  - pFabric
  - SIGCOMM'12
Step back and ask
How can we design a deployment friendly and near optimal data center transport while leveraging the insights offered by existing proposals?
DC Transports

- DCTCP
  - Deployment Friendly but Suboptimal
  - SIGCOMM’12
- D2TCP
- PDQ
- pFabric
  - Near Optimal but not Deployment Friendly
  - SIGCOMM’12
  (Changes in data plane)

Step back and ask:

How can we design a deployment friendly and near optimal data center transport while leveraging the insights offered by existing proposals?

PASE
Rest of the Talk ...

- DC Transport Strategies
- PASE Design
- Evaluation
Rest of the Talk ...

- DC Transport Strategies
- PASE Design
- Evaluation
DC Transport Strategies

- **Self-adjusting endpoints** e.g., TCP, DCTCP, L^2DCT
  - senders make independent decisions and adjust rate by themselves

- **Arbitration** e.g., D^3, PDQ
  - a common network entity (e.g., a switch) allocates rates to each flow

- **In-network prioritization** e.g., pFabric
  - switches schedule and drop packets based on the packet priority
DC Transport Strategies

- **Self-adjusting endpoints** e.g., TCP, DCTCP, L²DCT
  - senders make independent decisions and adjust rate by themselves

- **Arbitration** e.g., TCP, DCTCP
  - a common network entity (e.g., a switch) allocates rates to each flow

- **In-network prioritization** e.g., pFabric
  - switches schedule and drop packets based on the packet priority

Existing DC transport proposals use only one of these strategies
## Transport Strategies in Isolation

<table>
<thead>
<tr>
<th>Transport Strategy</th>
<th>Example</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Adjusting Endpoints</td>
<td>DCTCP, D²TCP, L²DCT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arbitration</td>
<td>PDQ, D³</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-network Prioritization</td>
<td>pFabric</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Transport Strategies in Isolation

<table>
<thead>
<tr>
<th>Transport Strategy</th>
<th>Example</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Adjusting Endpoints</td>
<td>DCTCP, D²TCP, L²DCT</td>
<td>Ease of deployment</td>
<td>No strict priority scheduling</td>
</tr>
<tr>
<td>Arbitration</td>
<td>PDQ, D³</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-network Prioritization</td>
<td>pFabric</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Transport Strategies in Isolation

<table>
<thead>
<tr>
<th>Transport Strategy</th>
<th>Example</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Self-Adjusting Endpoints</strong></td>
<td>DCTCP, D²TCP, L²DCT</td>
<td>Ease of deployment</td>
<td>No strict priority scheduling</td>
</tr>
<tr>
<td><strong>Arbitration</strong></td>
<td>PDQ, D³</td>
<td>Strict priority scheduling</td>
<td>High flow switching overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hard to compute precise rates</td>
</tr>
<tr>
<td><strong>In-network Prioritization</strong></td>
<td>pFabric</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Transport Strategies in Isolation

<table>
<thead>
<tr>
<th>Transport Strategy</th>
<th>Example</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Adjusting Endpoints</td>
<td>DCTCP, D²TCP, L²DCT</td>
<td>Ease of deployment</td>
<td>No strict priority scheduling</td>
</tr>
<tr>
<td>Arbitration</td>
<td>PDQ, D³</td>
<td>Strict priority scheduling</td>
<td>High flow switching overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hard to compute precise rates</td>
</tr>
<tr>
<td>In-network Prioritization</td>
<td>pFabric</td>
<td>Low flow switching overhead</td>
<td>Switch-local decisions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Limited # of priority queues</td>
</tr>
</tbody>
</table>
## Transport Strategies in Unison

<table>
<thead>
<tr>
<th>Transport Strategy</th>
<th>Example</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Adjusting Endpoints</td>
<td>DCTCP, D²TCP, L²DCT</td>
<td>Ease of deployment</td>
<td>No strict priority scheduling</td>
</tr>
<tr>
<td>Arbitration</td>
<td>PDQ, D³</td>
<td>Strict priority scheduling</td>
<td>High flow switching overhead, Hard to compute precise rates</td>
</tr>
<tr>
<td>In-network Prioritization</td>
<td>pFabric</td>
<td>Low flow switching overhead</td>
<td>Switch-local decisions, Limited # of priority queues</td>
</tr>
</tbody>
</table>
Transport Strategies in Unison

In-network Prioritization Alone

Limited # of queues
More # of flows (priorities)

Flows
1
2
3
4

High Priority

Low Priority
Transport Strategies in Unison

In-network Prioritization Alone

**Limited # of queues**
More # of flows (priorities)

**Flow Multiplexing**
Limited performance gains!

Any static mapping mechanism degrades performance!
Transport Strategies in Unison

In-network Prioritization + Arbitration

**Arbitrator**
Dynamic mapping of flows to queues

**Idea**
As a flow’s turn comes, map it to the highest priority queue!
Transport Strategies in Unison

In-network Prioritization + Arbitration

**Arbitrator**
Dynamic mapping of flows to queues

**Idea**
As a flow’s turn comes, map it to the highest priority queue!
Transport Strategies in Unison

In-network Prioritization + Arbitration

Arbitrator
Dynamic mapping of flows to queues

Idea
As a flow’s turn comes, map it to the highest priority queue!

Time $t_1$

Time $t_2$
Transport Strategies in Unison

In-network Prioritization + Arbitration

Arbitrator
Dynamic mapping of flows to queues

Idea
As a flow’s turn comes, map it to the highest priority queue!

Similarly,
- Arbitration + Self-Adjusting Endpoints
- Arbitration + In-network Prioritization

PASE leverages these insights in its design!
Rest of the Talk ...

- DC Transport Strategies
- PASE Design
- Evaluation
PASE Design Principle

Each transport strategy should focus on what it is best at doing!

- **Arbitrators**
  - Do inter-flow prioritization at coarse time-scales

- **Endpoints**
  - Probe for any spare link capacity

- **In-network prioritization**
  - Do per-packet prioritization at sub-RTT timescales
PASE Overview

Sender

Arbitrator

Receiver
PASE Overview

- Arbitration: Control plane
  Calculate “reference rate” and “priority queue”
PASE Overview

- **Arbitration**: Control plane
  - Calculate “reference rate” and “priority queue”
- **Self-Adjusting Endpoints**: Guided rate control
  - Use arbitrator feedback as a pivot
**PASE Overview**

- **Arbitration:** Control plane
  Calculate “reference rate” and “priority queue”
- **Self-Adjusting Endpoints:** Guided rate control
  Use arbitrator feedback as a pivot
- **In-network Prioritization:** Existing priority queues
PASE Overview

Arbitration: Control plane
Calculate “reference rate” and “priority queue”
Self-Adjusting Endpoints: Guided rate control
Use arbitrator feedback as a pivot
In-network Prioritization: Existing priority queues

Key Components
PASE Arbitration

Sender

Arbitrator

Receiver
**Distributed Arbitration**

- per link arbitration done in **control plane**
- existing protocols implement in **data plane**
PASE Arbitration

Distributed Arbitration
- per link arbitration done in control plane
- existing protocols implement in data plane

Arbitrator Location
- at the end hosts (e.g., for their own links to the switch) OR
- on dedicated hosts inside the DC
PASE Arbitration

Distributed Arbitration

- per link arbitration done in control plane
- existing protocols implement in data plane

Arbitrator Location

- at the end hosts (e.g., for their own links to the switch) OR
- on dedicated hosts inside the DC
PASE Arbitration

Distributed Arbitration
- per link arbitration done in control plane
- existing protocols implement in data plane

Arbitrator Location
- at the end hosts (e.g., for their own links to the switch) OR
- on dedicated hosts inside the DC
PASE Arbitration – Challenges

- **Challenges**
  - Arbitration latency
  - Processing overhead
  - Network overhead
PASE Arbitration – Challenges

- **Challenges**
  - Arbitration latency
  - Processing overhead
  - Network overhead

Solution: Leverage the tree-like structure of typical DC topologies
Bottom Up Arbitration

- Leverage Tree Structure from leaves up to the root
Bottom Up Arbitration

- **Leverage Tree Structure** from leaves up to the root

![Diagram showing a tree structure with nodes labeled ToR, Aggregation, Core, Inter-Rack, Sender, and Receiver. The diagram illustrates the concept of bottom up arbitration through a tree structure.]
Bottom Up Arbitration

- **Leverage Tree Structure** from leaves up to the root

Diagram:
- Sender
- Inter-Rack
- Receiver
- ToR
- Core Aggregation
- Aggregation
- Arrow: Arbitration Message
Bottom Up Arbitration

- **Leverage Tree Structure** from leaves up to the root

![Diagram showing Bottom Up Arbitration]

- Sender 
- Receiver 
- Inter-Rack 
- Arbitration Message
Bottom Up Arbitration

- **Leverage Tree Structure** from leaves up to the root

![Diagram showing Bottom Up Arbitration with nodes and arrows labeled as follows: Sender, Aggregation, ToR, Core, Inter-Rack, and Receiver. The diagram illustrates the flow of messages and responses with red arrows for Arbitration Message and blue arrows for Receiver Response.]
Bottom Up Arbitration

- **Leverage Tree Structure from leaves up to the root**

**Intra-Rack**
- Sender
- Receiver

**Inter-Rack**
- Sender
- Receiver

**Core**
- Aggregation

No external arbitrators required!
Bottom Up Arbitration

- **Leverage Tree Structure** from leaves up to the root

No external arbitrators required!

Facilitates inter-rack optimizations (early pruning & delegation) to reduce arbitration overhead.
Early Pruning

Arbitration involves sorting flows and picking top $k$ for immediate scheduling.

Flows that won’t make it to top $k$ queues should be pruned at lower levels.
Early Pruning

Arbitration involves sorting flows and picking \textit{top k} for immediate scheduling.

Reduces Network and Processing overhead
Fewer flows contact the higher level arbitrators!

Flows that won't make it to \textit{top k} queues should be pruned at lower levels.
Delegation

Key Idea: Divide a link into virtual links and delegate responsibility to child arbitrators
Delegation

Key Idea: Divide a link into virtual links and delegate responsibility to child arbitrators

Algorithm
Link capacity C is split in N virtual links
Delegation

Key Idea: Divide a link into virtual links and delegate responsibility to child arbitrators

Algorithm
Link capacity C is split in N virtual links
Parent arbitrator delegates virtual link to child arbitrator
Delegation

Key Idea: Divide a link into virtual links and delegate responsibility to child arbitrators

Algorithm
Link capacity C is split in N virtual links

Parent arbitrator delegates virtual link to child arbitrator

Child arbitrator does arbitration for virtual link
Delegation

Key Idea: **Divide a link into virtual links and delegate responsibility to child arbitrators**

- **Algorithm**
  - Link capacity $C$ is split in $N$ virtual links.
  - Parent arbitrator delegates virtual link to child arbitrator.
  - Child arbitrator does arbitration for virtual link.
  - Virtual link capacity is periodically updated based on the top $k$ flows of all child arbitrators.

![Diagram showing aggregation, delegation, and virtual link capacity management.](Image)
Delegation

Key Idea: Divide a link into virtual links and delegate responsibility to child arbitrators

Algorithm
Link capacity $C$ is split in $N$ virtual links

Reduces Arbitration Latency
Make arbitration decision close to the sources

Child arbitrator does arbitration for virtual link

Virtual link capacity is periodically updated based on the top $k$ flows of all child arbitrators
PASE Overview

- **Arbitration**: Control plane
  Calculate “reference rate” and “priority queue”
- **Self-Adjusting Endpoints**: Guided rate control
  Use arbitrator feedback as a pivot
- **In-network Prioritization**: Existing priority queues
PASE Endhost Transport

- Rate Control

- Loss Recovery Mechanism
PASE Endhost Transport

- **Rate Control**
  - Use reference rate and priority feedback from arbitrators
  - Use reference-rate as pivot, and
  - Follow DCTCP control laws

- **Loss Recovery Mechanism**
PASE Endhost Transport

- **Rate Control**
  - Use reference rate and priority feedback from arbitrators
    - Use reference-rate as pivot, and
    - Follow DCTCP control laws

- **Loss Recovery Mechanism**
  - Packets in lower priority queues can be delayed for several RTTs
    - large RTO OR small probe to avoid spurious retransmissions
PASE -- Putting it Together

- Efficient arbitration control plane
- Simple TCP-like transport
- Existing priority queues inside switches
Rest of the Talk ...

- DC Transport Strategies
- PASE Design
- Evaluation
Evaluation

- **Platforms**
  - Small scale testbed
  - NS2

- **Workloads**
  - Web search (DCTCP), Data mining (VL2)

- **Comparison with** deployment friendly
  - DCTCP, $D^2TCP$, $L^2DCT$

- **Comparison with** state of the art
  - pFabric
Simulation Setup

<table>
<thead>
<tr>
<th>Queue Size</th>
<th>250KB (per queue)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT</td>
<td>300usec</td>
</tr>
<tr>
<td>RTO</td>
<td>1 msec</td>
</tr>
<tr>
<td>L</td>
<td>40</td>
</tr>
</tbody>
</table>
Comparison with Deployment Friendly

Settings similar to D²TCP
- Flow Sizes: 100-500KB
- Deadlines: 5-25msec
Comparison with Deployment Friendly

Settings similar to $D^2$TCP
- Flow Sizes: 100-500KB
- Deadlines: 5-25msec

PASE is deployment friendly yet performs BETTER than existing protocols!
Comparison with State of the Art

Settings
- Flow Sizes: 2-98KB
- Left-to-right traffic
Comparison with State of the Art

Settings
- Flow Sizes: 2-98KB
- Left-to-right traffic

PASE performs comparable and does not require changes to data plane
Summary

- **Key Strategies** for Existing DC Transport
  - Arbitration, in-network Prioritization, Self-Adjusting End-points
  - Complimentary rather than substitutes

- **PASE**
  - Combines the three strategies
  - Efficient arbitration control plane; simple TCP-like transport; leverages existing priority queues inside switches

- **Performance**
  - Comparable to or better than earlier proposals that even require changes to the network fabric
Thank you!