Roadmap

• Other Data Center network topologies
• What’s TCP incast
• Solutions
Other Datacenter network topologies
http://www.infotechlead.com/2013/03/28/gartner-data-center-spending-to-grow-3-7-to-146-billion
What to build?

This question has spawned a cottage industry in the computer networking research community.

- “Fat-tree” [SIGCOMM 2008]
- VL2 [SIGCOMM 2009, CoNEXT 2013]
- DCell [SIGCOMM 2008]
- BCube [SIGCOMM 2009]
- Jellyfish [NSDI 2012]
“Fat-tree” SIGCOMM 2008

Isomorphic to butterfly network except at top level

Bisection width n/2, oversubscription ratio 1
VL2 (SIGCOMM 2009, CoNEXT 2013)

called a Clos network
oversubscription ratio 1
(but 1Gbps links at leaves, 10Gbps elsewhere)
Dcell 2008

A “clique of cliques”

Servers forward packets

n servers in DCell_0
(n+1)n servers in DCell_1
(((n+1)n)+1)(n+1)n in DCell_2

oversubscription ratio 1

DCell_1  n=4
Mesh of stars (analogous to a mesh of trees)
Bisection width between .25n and .35n
Jellyfish (NSDI 2012)

Random connections
Bisection width $\Theta(n)$

"fat-tree" butterfly
Datacenter Transport Protocols
Data Center Packet Transport

• Large purpose-built DCs
  – Huge investment: R&D, business

• Transport inside the DC
  – TCP rules (99.9% of traffic)

• How’s TCP doing?
TCP does not meet demands of DC applications

• Goal:
  – Low latency
  – High throughput

• TCP does not meet these demands.
  – Incast
    • Suffers from bursty packet drops

  – Large queues:
    ➢ Adds significant latency.
    ➢ Wastes precious buffers, esp. bad with shallow-buffered switches.
Case Study: Microsoft Bing

• Measurements from 6000 server production cluster

• Instrumentation passively collects logs
  – Application-level
  – Socket-level
  – Selected packet-level

• More than **150TB** of compressed data over a month
Partition/Aggregate Application Structure

- Picasso
  - Time is money
    - Strict deadlines (SLAs)
    - Missed deadline
      - Lower quality result

1. Art is a lie…
2. The chief...
3. ...

Deadline = 250ms
Deadline = 50ms
Deadline = 10ms

“Everything you can imagine is real.”
“Bad artists copy. Good artists steal.”
“It is your work in life that is the ultimate seduction.”
“The chief enemy of creativity is good sense.”
“Inspiration does exist, but it must find you working.”
“I’d like to live as a poor man with lots of money.”
“Art is a lie that makes us realize the truth.”
“Computers are useless. They can only give you answers.”

Worker Nodes
Generality of Partition/Aggregate

• The foundation for many large-scale web applications.
  – Web search, Social network composition, Ad selection, etc.

• Example: Facebook

Partition/Aggregate ~ Multiget
  – Aggregators: Web Servers
  – Workers: Memcached Servers
Workloads

- Partition/Aggregate (Query)
- Short messages [50KB-1MB] (Coordination, Control state)
- Large flows [1MB-50MB] (Data update)
Work load characterization

Figure 3: Time between arrival of new work for the Aggregator (queries) and between background flows between servers (update and short message).
Workload characterization

Figure 4: PDF of flow size distribution for background traffic. PDF of Total Bytes shows probability a randomly selected byte would come from a flow of given size.
Figure 5: Distribution of number of concurrent connections.
Impairments

- Incast
- Queue Buildup
- Buffer Pressure
Impairments

• Incast

• Queue Buildup

• Buffer Pressure
Incast

- Synchronized mice collide.
  - Caused by Partition/Aggregate.

Worker 1
Worker 2
Worker 3
Worker 4

Aggregator

RTO_{min} = 300 ms

TCP timeout
Incast Really Happens

MLA Query Completion Time (ms)

- Requests are jittered over 10ms window.
- Jittering switched off around 8:30 am.
- Jittering trades off median against high percentiles.

99.9th percentile is being tracked.
Queue Buildup

- Big flows buildup queues.
  - Increased latency for short flows.

- Measurements in Bing cluster
  - For 90% packets: $\text{RTT} < 1\text{ms}$
  - For 10% packets: $1\text{ms} < \text{RTT} < 15\text{ms}$
Many solutions to this problem

• Fine-grained TCP retransmissions
• DCTCP
• DCQCN
• NDP – a new DC transport architecture
• High Precision Congestion Control (HPCC)
Data Center TCP
(DCTCP)

Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye
Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, Murari Sridharan

Microsoft Research                Stanford University
Data Center Transport Requirements

1. **High Burst Tolerance**
   - Incast due to Partition/Aggregate is common.

2. **Low Latency**
   - Short flows, queries

3. **High Throughput**
   - Continuous data updates, large file transfers

The challenge is to achieve these three together.
Tension Between Requirements

High Throughput
High Burst Tolerance

Low Latency

Deep Buffers:
- Queuing Delays Increase Latency

Shallow Buffers:
- Bad for Bursts & Throughput

Objective:
Low Queue Occupancy & High Throughput

Reduced $\text{RTO}_{\text{min}}$ (SIGCOMM ‘09)
- Doesn’t Help Latency

AQM – RED:
- Avg Queue Not Fast Enough for Incast
The DCTCP Algorithm
Review: The TCP/ECN Control Loop

ECN = Explicit Congestion Notification

ECN Mark (1 bit)
Bandwidth-delay product rule of thumb:
- A single flow needs $C \times RTT$ buffers for 100% Throughput.
Small Queues & TCP Throughput: The Buffer Sizing Story

• Bandwidth-delay product rule of thumb:
  – A single flow needs $C \times RTT$ buffers for \textbf{100\% Throughput}.

• Appenzeller rule of thumb (SIGCOMM ‘04):
  – Large # of flows: $C \times RTT / \sqrt{N}$ is enough.
Small Queues & TCP Throughput: The Buffer Sizing Story

• Bandwidth-delay product rule of thumb:
  – A single flow needs $C \times RTT$ buffers for 100% Throughput.

• Appenzeller rule of thumb (SIGCOMM ‘04):
  – Large # of flows: $C \times RTT / \sqrt{N}$ is enough.

• Can’t rely on stat-mux benefit in the DC.
  – Measurements show typically 1-2 big flows at each server, at most 4.
Small Queues & TCP Throughput: The Buffer Sizing Story

- **Bandwidth-delay product rule of thumb:**
  - A single flow needs $C \times RTT$ buffers for **100% Throughput**.

- **Appenzeller rule of thumb (SIGCOMM ‘04):**
  - Large # of flows: $C \times RTT / \sqrt{N}$ is enough.

- **Can’t rely on stat-mux benefit in the DC:**
  - Measurements show typically **1-2 big flows** at each server, **at most 4**.

---

**Real Rule of Thumb:**
Low Variance in Sending Rate $\rightarrow$ Small Buffers Suffice
Two Key Ideas

1. React in proportion to the *extent* of congestion, not its *presence*.
   - Reduces *variance* in sending rates, lowering queuing requirements.

<table>
<thead>
<tr>
<th>ECN Marks</th>
<th>TCP</th>
<th>DCTCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 1 1 0 1 1 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 40%</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 5%</td>
</tr>
</tbody>
</table>

2. Mark based on *instantaneous* queue length.
   - Fast feedback to better deal with bursts.
Data Center TCP Algorithm

Switch side:
- Mark packets when Queue Length > K.

Sender side:
- Maintain running average of fraction of packets marked ($\alpha$).

In each RTT:

$$ F = \frac{\#\ of\ marked\ ACKs}{Total\ #\ of\ ACKs} \quad \alpha \leftarrow (1 - g)\alpha + gF $$

- Adaptive window decreases: $$ Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd $$
  - Note: decrease factor between 1 and 2.
DCTCP in Action

Setup: Win 7, Broadcom 1Gbps Switch
Scenario: 2 long-lived flows, K = 30KB
Why it Works

1. High Burst Tolerance
   - Large buffer headroom → bursts fit.
   - Aggressive marking → sources react before packets are dropped.

2. Low Latency
   - Small buffer occupancies → low queuing delay.

3. High Throughput
   - ECN averaging → smooth rate adjustments, low variance.
Evaluation

• Implemented in Windows stack.
• Real hardware, 1Gbps and 10Gbps experiments
  – 90 server testbed
  – Broadcom Triumph 48 1G ports – 4MB shared memory
  – Cisco Cat4948 48 1G ports – 16MB shared memory
  – Broadcom Scorpion 24 10G ports – 4MB shared memory
• Numerous micro-benchmarks
  – Throughput and Queue Length
  – Multi-hop
  – Queue Buildup
  – Buffer Pressure
  – Fairness and Convergence
  – Incast
  – Static vs Dynamic Buffer Mgmt
• Cluster traffic benchmark
Cluster Traffic Benchmark

• Emulate traffic within 1 Rack of Bing cluster
  – 45 1G servers, 10G server for external traffic

• Generate query, and background traffic
  – Flow sizes and arrival times follow distributions seen in Bing

• Metric: We use $RTO_{min} = 10$ms for both TCP & DCTCP.
  – Flow completion time for queries and background flows.
Low latency for short flows.
Low latency for short flows.
High throughput for long flows.
 Baseline

- Low latency for short flows.
- High throughput for long flows.
- High burst tolerance for query flows.
Scaled Background & Query

10x Background, 10x Query

Completion Time (ms)

- DCTCP/ShallowBuf
- TCP/ShallowBuf
- TCP-RED/ShallowBuf
- TCP/DeepBuf

Short messages & Query
Conclusions

• DCTCP satisfies all our requirements for Data Center packet transport.
  ✓ Handles bursts well
  ✓ Keeps queuing delays low
  ✓ Achieves high throughput

• Features:
  ✓ Very simple change to TCP and a single switch parameter.
  ✓ Based on mechanisms already available in Silicon.
HPCC:
High Precision Congestion Control

Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, Minlan Yu
Problems of state-of-the-art

- Slow convergence
  ➢ One fundamental issue: coarse-grained feedback
  ➢ No precise feedback indicating how much to increase/decrease

- Standing queue
  ➢ Feedback relies on queue
- Complex parameter tuning
  ➢ No precise feedback → need heuristics: lots of parameters

What if we have precise feedback?
HPCC: use INT as precise feedback

- In-band network telemetry (INT) provides many details per packet
- Broadcom&Barefoot have INT in recent products.
- Widely used for diagnosis and monitoring in production
How good can CC do with INT as the feedback?
We design HPCC to answer this question
Challenge 1: Feedback delay:
- Pkt/ACK may get delayed

Challenge 2: Overreaction:
- Diff ACKs bring overlapping feedback

Adjust rate per ACK

Sender ─ Link-1 ─ Receiver

Sender ─ Link-2 ─ Receiver

99
Challenge 1: tolerate feedback delay

Rate (DCQCN/TIMELY)

No congestion

<table>
<thead>
<tr>
<th>Time (T)</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>2T</td>
<td></td>
</tr>
</tbody>
</table>

(T is the base RTT)

Congestion

<table>
<thead>
<tr>
<th>Time (T)</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>2T</td>
<td></td>
</tr>
</tbody>
</table>

High rate persists

Can be >> T

Because T is very low and queueing delay dominates

Each sender uses a window to limit inflight bytes: \( W = \text{target\_rate} \times T \)
Challenge 1: tolerate feedback delay

Need a new way of measuring congestion:

Rate-based scheme
- Use rate mismatch

HPCC
- Rate mismatch is bad measurement
- Actual sending rate is non-constant, due to the inflight bytes limit
Estimating inflight bytes

- Use **total inflight bytes** to measure congestion
  - Each flow needs to estimate the total inflight bytes
  - Use INT to estimate the total inflight bytes

\[
\text{total inflight bytes} \approx \frac{qlen + txRate \times T}{B (\text{bottleneck bandwidth})}
\]

Available in INT

Each sender estimates independently in a distributed way.
Controlling inflight bytes

• **Use total inflight bytes** to measure congestion
  o Each flow needs to estimate the total inflight bytes
  o Use **INT** to estimate the total inflight bytes

• **Compute adjustment based inflight bytes**
  o Use **MIMD** to quickly adjust to the right rate
Challenge 2: Fast reaction without overreaction

Per-ACK reaction
- Overreaction

\[ W = W_0 / 1.5 \]

Pipe volume = 6
Challenge 2: Fast reaction without overreaction

Per-ACK reaction
- Overreaction

\[ W = W_0 / 1.5 \]

\[ W = W_0 / 1.3 \]

Overlap → overreaction

Pipe volume = 6
Challenge 2: Fast reaction without overreaction

Per-ACK reaction
- Overreaction

Per-RTT reaction
- No overreaction

$W = \frac{W_0}{1.5}$

Next to react: 6

Pipe volume = 6
Challenge 2: Fast reaction without overreaction

Per-ACK reaction
- Overreaction

Per-RTT reaction
- No overreaction

- No overlap → no overreaction

Pipe volume = 6

Next to react: 6

W = W₀

/1.5

/1.1

Ack1

Ack6
Challenge 2: Fast reaction without overreaction

Per-ACK reaction
- Overreaction
- Fast reaction to sudden bursts

Per-RTT reaction
- No overreaction
- Miss sudden bursts

$W = W_0 / 1.5$

Pipe volume = 6
Challenge 2: Fast reaction without overreaction

<table>
<thead>
<tr>
<th>Per-ACK reaction</th>
<th>Per-RTT reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Overreaction</td>
<td>- No overreaction</td>
</tr>
<tr>
<td>- Fast reaction to sudden bursts</td>
<td>- Miss sudden bursts</td>
</tr>
</tbody>
</table>

HPCC combines per-ACK and per-RTT (details in paper)
- No overreaction
- Fast reaction to sudden bursts
Theory proof

- Asynchronized flows
- Arbitrary topology

### Convergence speed

\[
Y(n) = AR(n)
\]

\[
R_j(n + 1) = \frac{R_j(n)}{\max_i \{Y_i(n)A_{ij}/C_i\}}
\]

Converge to Pareto optimal

### Queueing delay

\[
(\pi N / 8)^{0.5}
\]

Queue size is bounded

### Fairness

\[
R_i = a \left(1 - \frac{U_{target}}{U_i}\right)^{-1}
\]

Max-min/proportional fairness
Implementation

- Commodity FPGA NIC
- In a 32-server testbed
Implementation in commodity FPGA NIC

- 14,000 lines of Verilog code for the whole NIC
- 2,000 lines for CC
  - <2% of hardware resources
- **Optimization:**
  - Look-up table: 8x speedup for division
  - H/w friendly compression with <1% error: 10KB for [1, 4M]
  - Multiple parallel engines: 6x more concurrent flows
- **INT header overhead:** 4.2% (MTU=1KB)
Implementation: testbed

Each server has **two** 25G FPGA NICs
• **Setup**
  o 32-server, 5-switch Testbed (25GE)
  o 320-server, 56-switch simulation, (100GE)

• **Traffic setting**
  o Web search, with diff levels of traffic load
  o Hadoop, with diff levels of traffic load
  o Diff incast sizes & ratios
  o Many micro-benchmarks

• **Compare with DCQCN, TIMELY, DCTCP, with their recommended parameters**

• **Metrics**
  o Flow completion time (FCT)
  o Queue length distribution
HPCC achieves lower FCT and near-zero queue

- In testbed, vs. DCQCN (hardware-based, widely used in industry)
  - Web search traffic at 50% load
- Vs. other CC (unavailable in HW) in simulation. HPCC performs better

![Graph showing comparison between HPCC and DCQCN](image-url)
HPCC has the potential to remove PFC

- HPCC is persistently better, regardless of the flow control schemes
- HPCC is insensitive to the flow control schemes

![Graph showing 95 percentile FCT for different flow sizes and load conditions.](graph.png)

(a) 30% Avg. Load + incast
(b) 50% Avg. Load
Conclusion

Practical impact
- Demonstrate problems of existing CC with real production experience
- Implement HPCC with HW NIC & switch

Advance in CC
- Demonstrate using precise info from INT make CC highly performant
- Address the challenge of using INT
- Proof for convergence speed, queueing delay, and fairness

Collaborating with switch and NIC partners on deploying HPCC in Alibaba’s production networks
Summary

• DC TCP incast problem
• Sample solutions
  – DCTCP
  – HPCC