Better Never than Late: Meeting Deadlines in Datacenter Networks

Christo Wilson, Hitesh Ballani, Thomas Karagiannis, Ant Rowstron

Microsoft Research, Cambridge
User-facing online services

Two common underlying themes

1. Soft real-time nature
   • Online services have **SLAs** baked into their operation
   • **Example**: 300ms response time for 99.9% requests

Impact of breached SLAs:
• Amazon: extra 100ms costs 1% in sales

2. Partition-aggregate workflow

![Diagram showing a workflow with an aggregator and worker nodes](image-url)
User-facing online services

Application SLAs
SLAs for components at each level of the hierarchy

Cascading SLAs
Deadlines on communications between components

Flow Deadlines
A flow is useful if and only if it satisfies its deadline

Today’s transport protocols:
Deadline agnostic and strive for fairness
Flows get bandwidth in accordance to their deadlines. Deadline awareness ensures both flows satisfy deadlines.

Limitations of Fair Sharing

Case for unfair sharing:

Flows f1 and f2 get a fair share of bandwidth. Flow f1 misses its deadline (incomplete response to user).
Flows get bandwidth in accordance to their deadlines.
Deadline awareness ensures both flows satisfy deadlines.

Case for unfair sharing:

- Flow f1, 20ms
- Flow f2, 40ms

Status Quo:
Flows f1 and f2 are not evenly distributed in time, with f2 receiving more bandwidth before its deadline.

Deadline aware:
Both flows, f1 and f2, are given bandwidth to ensure they meet their deadlines.

Graphs showing time on the x-axis and flows on the y-axis, with f1 and f2 represented by bars of different lengths, illustrating the unfair sharing and the fair sharing scenario.
Limitations of Fair Sharing

Case for unfair sharing:

Flow f1, 20ms
Flow f2, 40ms

Status Quo

Time

Flows

f1

f2

20

40

X

Deadline aware

Time

Flows

f1

f2

20

40

Case for flow quenching:

6 flows with 30ms deadline

Flows

Time

30

X

X

X

X

X

X

Insufficient bandwidth to satisfy all deadlines
With fair share, all flows miss the deadline (empty response)
Limitations of Fair Sharing

Case for unfair sharing:

- Flow f1, 20ms
- Flow f2, 40ms

<table>
<thead>
<tr>
<th>Flow</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>20</td>
</tr>
<tr>
<td>f2</td>
<td>40</td>
</tr>
</tbody>
</table>

Status Quo

- Flows: f1, f2
- X: Missed deadline

Deadline aware

- Flows: f1, f2

Case for flow quenching:

- 6 flows with 30ms deadline

<table>
<thead>
<tr>
<th>Flow</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>

With deadline awareness, one flow can be quenched
All other flows make their deadline (partial response)
D³: Deadline-driven Delivery

Main idea: Make the network aware of flow deadlines
➢ Prioritize flows based on deadlines

A deadline-aware datacenter transport protocol that:
➢ schedules network traffic based on SLAs
➢ can double the peak load a datacenter supports
➢ performs well as a congestion control protocol

Advantages:
➢ Improve quality of responses
➢ Save resources
Challenges

• Deadlines are associated with flows, not packets
  - Packet scheduling mechanisms (e.g., EDF)
  - Reservation schemes (e.g., IntServ, DiffServ)

• Short flows (<50 KB) and minimal RTTs (~300 μsec)

• Deadlines can vary significantly

• Beyond documented TCP problems in datacenters (e.g., incast, buffer pressure)
D³ Design

Design goals
— Maximize application throughput (i.e., deadlines satisfied)
— Burst tolerance
— High utilization

Non-goals: Incremental deployment, backwards compatibility, being friendly to legacy protocols

Key Insight:
— Rate required to satisfy a flow deadline:

\[ r = \frac{s}{d} \]

s: flow size
d: deadline
1. **Application exposes** \((s, d)\)

2. **Desired rate**: \(r = \frac{s}{d}\)

3. Routers allocate rates \((\alpha)\) based on traffic load

4. **Sending rate for next RTT**: \(sr = \min(\alpha_1, \alpha_2)\)

\(s\) : flow size
\(d\) : deadline
RRQ : Rate Request
\(\alpha\) : allocated rate
D³ overview

1. Application exposes \((s, d)\)

2. Desired rate: \(r = \frac{s}{d}\)

3. Routers allocate rates (\(\alpha\)) based on traffic load

4. Sending rate for next RTT: \(sr = \min(\alpha_1, \alpha_2)\)

5. Send data at rate \(sr\)

6. One of the packets contains and updated RRQ based on the remaining flow size and deadline

\(s\) : flow size \\
\(d\) : deadline \\
RRQ : Rate Request \\
\(\alpha\) : allocated rate
D³ overview

No priority queuing at routers
Routers only provide allocated rates

1. Application exposes \((s, d)\)

2. Rate control is performed at the end host which enforces the minimum of the allocated rates

3. 

4. Sending rate for next RTT: \(sr = \min(\alpha_1, \alpha_2)\)

5. Send data at rate \(sr\)

6. One of the packets contains an updated RRQ based on the remaining flow size and deadline
Rate allocation

Goals:
— Maximize the number of deadlines satisfied
— Fully utilize the network

Allocated rate ($\alpha$):
— Available capacity $\geq \sum$ (desired rates)
  — Deadline flow: $\alpha = r + fs$
  — Non-deadline flow ($r=0$): $\alpha = fs$
— Available capacity not enough to satisfy all requests
  — greedily satisfy requests
  — remaining flows are assigned a base_rate (header only packet)

fs: fair-share after satisfying flow requests
r: desired rate
Rate allocation

Router needs to track:
1. Sum of desired rates (demand)
2. Available Capacity
3. Fair-share (fs)

Three aggregate values
(no per-flow state)

Allocations : \( A = \sum \alpha \)
Demand : \( D = \sum r \)
Number of flows : \( N \)

\( \Rightarrow \) \( fs = \frac{C-D}{N} \)
available capacity = C-A

\[ A = A - \alpha_{t-1} \]
\[ D = D - r_{t-1} + r_t \]

If available capacity \( > D \):
\[ \alpha_t = r + fs \]
\[ A = A + \alpha_t \]

\( r \) : desired rate
\( \alpha \) : allocated rate
\( C \) : link capacity
RRQ : Rate Request
Implementation

End host:
- Complete transport protocol
- Sockets-like API where applications expose flow length and deadlines

Router:
- User-space PC-based implementation
- Able to sustain four links at full duplex line rate
- Packet processing overhead $< 1\mu$sec
**Evaluation: Testbed and metrics**

**Metric:**
- Application throughput
  - Number of flows satisfying deadlines
- Operational regime
  - Application throughput > 99%

**Protocols:**
1. TCP
2. RCP_{dc} : D^3 in fair-share mode only (best-case for fair-sharing protocols)
3. TCP_{pr} : TCP with priority queuing
4. D^3
1. Flow microbenchmarks: Synthetic workload

Experiment: Multiple workers sending traffic

How many workers can be supported while satisfying >99% flow deadlines?
1. Flow microbenchmarks

Experiment: Multiple workers sending traffic
How many workers can be supported while satisfying >99% deadlines?

D³ can support roughly *twice as many workers* while satisfying application deadlines.
2. Datacenter-like traffic dynamics

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>TCP (optimized)</th>
<th>Fair Share Protocols</th>
<th>D³</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peak Load Supported</strong> Load Supported (flows/sec)</td>
<td>100</td>
<td>1100</td>
<td>1300</td>
<td>2000</td>
</tr>
</tbody>
</table>

- D³ maintains low flow completion times
- completion times similar to RCP
- Long background flows are not penalized
- throughput similar to TCP
Results so far...

Case for unfair sharing:

Flow f1, 20ms
Flow f2, 40ms

Case for flow quenching:

6 flows with 30ms deadline

Status Quo

Deadline aware

6 flows with 30ms deadline

Time

Time
3. Flow quenching

Terminate “useless flows” when:
- Desired rate exceeds link capacity
- Deadline has expired

Allows for graceful degradation of performance
Conclusions

Case for deadline-aware allocation of bandwidth

— tension between today’s offered functionality and application requirements

$D^3$:

— schedules network traffic based on SLAs
— can double the peak load a datacenter supports
— design based on challenges and luxuries of datacenter environment
Thank you!