Tail-Tolerance as a Systems Principle not a Metric

Marios Kogias, Edouard Bugnion

APNet20
Latency-Critical Datacenter Systems

- Economic aspect of latency
  - 100ms delay drops Amazon sales by 1%

- μs-scale computing

- Large fan-out/fan-in applications
  - Tail@Scale problem

How to design low-latency systems with tail-latency guarantees?
Understanding Tail-Latency

- RPC Latency = Communication delays + queueing time + service time

Existing systems approach tail-latency as a metric

- System benefits e.g. IX
- Better scheduling e.g. ZygOS
Tail-Tolerance is the ability of a system to predictably perform within the SLO
- SLO Violation Eliminator

- SLO-aware RPC admission control

- Application-agnostic

- In-network compute

- Orthogonal to scheduling
Outline

- SVEN design and internals
- Evaluation
- Open Questions
Trading Throughput for Latency Predictability

- Drop requests that will likely violate the SLO
- Early client notifications
  - Application of the end-to-end argument
- Client choices:
  - Ignore the dropped request and leverage the latency-completeness tradeoff
    - “Better never than late”
  - Retry to another server
Design Requirements

1. Application-agnostic
   - Implement SVEN inside the transport layer

2. Zero server overhead

3. Proactive design
R2P2 [ATC2019]

- Transport protocol for datacenter RPCs
- Expose request-response abstraction
- Break traditional point to point RPC semantics
- Designed for in-network policy enforcement
1. Application-agnostic
   - Implement SVEN inside the transport layer

2. Zero server overhead
   - Use programmable switches and implement SVEN as an R2P2 policy

3. Proactive design
Responsibilities
- Estimate RPC latency distribution
- Reject new requests that are likely to violate the latency SLO

Responsibility split
- Data plane: latency measurement, request admission control
- Control plane: drop rate estimation

Hardware implementation
- Tofino programmable switch
Latency Estimation
- Timestamps in the dataplane
- Match REQ0 and FEEDBACK timestamps
- Latency histogram in hardware

Admission Control
- Probabilistic drops
- Early rejects to the client
SVEN Control Plane

- Output: next drop rate
- Input: target SLO and current latency estimation

- Periodic execution
  - 100 ms

- Proof of concept implementation:
  - Additive increase additive decrease
  - Python
Design Requirements

1. Application-agnostic
   - Implement SVEN inside the transport layer

2. Zero server overhead
   - Use programmable switches and implement SVEN as an R2P2 policy

3. Proactive design
   - Drop requests in the network → No need for server cancellations
   - Return early rejects to clients → Retry elsewhere or ignore
Putting it all together

Clients

SVEN Middlebox

drop? (p)

get_ts

REQ0

REJECT

Server
Putting it all together

Clients → (REQ0) → SVEN Middlebox → Server

- get_ts
- drop? (p)
- update lat_stats

Feedback loop
Putting it all together

Clients

SVEN Middlebox
- drop? (p)
- update lat_stats

Control Plane

FEEDBACK

Server
Q: Can SVEN identify the load that violates the SLO and keep throughput below that point across service times?

- Synthetic microbenchmarks with different service time distributions
- Step load pattern beyond the system capacity
Setting Expectations

AVG[service_time] = 10µs
16-core machine
SLO: 300µs@99th

SLO is violated at different load levels across service time distributions
Evaluation

![Graphs showing MRPS over time with offered load highlighted.](image)
Evaluation

Offered Load
System Capacity
Throughput@SLO

Time (sec)

MRPS

0 25 50 75 100 125 150 175

0 25 50 75 100 125 150 175

0 25 50 75 100 125 150 175
SVEN identifies the load that violates the SLO and controls throughput

SVEN reduces SLO violations even when offered load is beyond capacity

SLO violations are due to the simplicity of the control loop
Open Questions

- What is a better implementation of the control plane?

- Could we instead of dropping requests use SVEN to first degrade them and only drop if absolutely necessary?

- How can we provide even stricter guarantees similarly to the fault tolerance line of research?
SLO Violation EliminNator (SVEN)
- SLO-aware RPC admission control
- R2P2 in-network policy
- P4 implementation

First step towards systematic approach to tail-tolerance

Thank you!