Minimizing Flow Completion Times in Data Centers

Ihsan Ayyub Qazi, Ali Munir Zartash A. Uzmi, Aisha Mushtaq, Saad N. Ismail, M. Safdar Iqbal, and Basma Khan
Computer Science Department, LUMS, Pakistan

User Facing Online Services

- Online services becoming extremely popular
  - e.g., web search, social networking, advertisement systems

Key goal: Minimize user response time!

Why response time matters?

Every 100ms latency costs 1% in business revenue
[Speed matters, G. Lindan]

Traffic reduced by 20% due to 500ms increase in latency
[M. Mayer at Web 2.0]

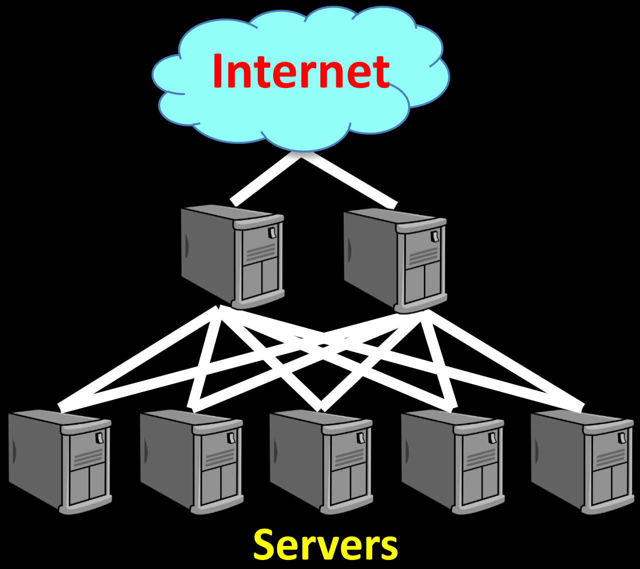
An extra 400ms reduced traffic by 9%
[YSlow 2.0, S. Stefanov]

Impacts user experience & operator revenue

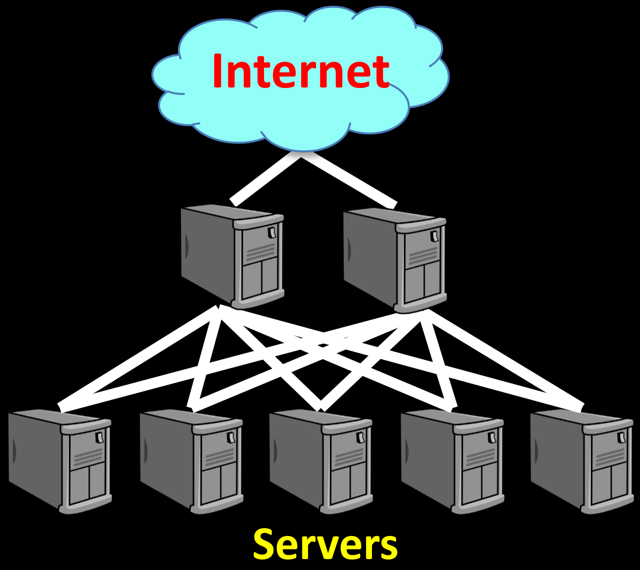
Challenges in Data Centers

Challenges in Data Centers

- Partition/Aggregate Structure
  - Leads to synchronized responses



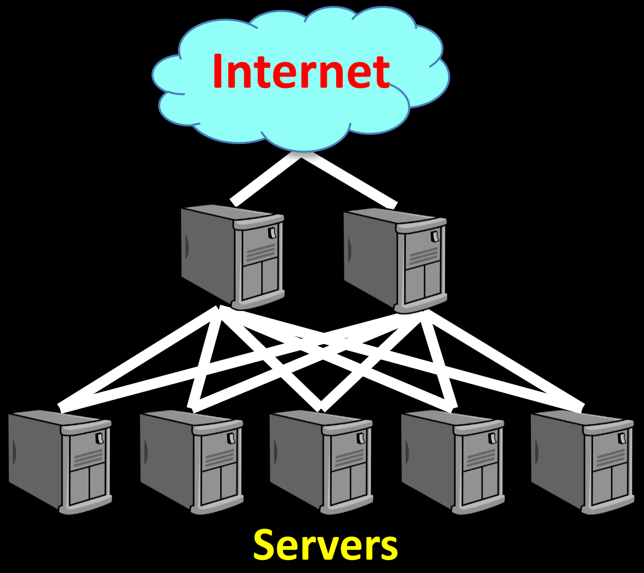
Challenges in Data Centers



- Partition/Aggregate Structure
  - Leads to synchronized responses
- Traffic workloads
  - Short flows (Query) 
    - Delay-sensitive
  - Long flows (Data Update) 
    - Throughput-sensitive

Challenges in Data Centers

- Partition/Aggregate Structure
  - Leads to synchronized responses



- Traffic workloads
  - Short flows (Query)
    - Delay-sensitive

  - Long flows (Data Update)
    - Throughput-sensitive

- TCP does not meet the demands of applications
  - Requires large queues for achieving high throughput
    - Adds significant latency

This paper proposes..

- L^2DCT (Low Latency Data Center Transport)
  - A data center transport protocol that targets minimizing average flow completion times (AFCT)
  - Uses insights from scheduling theory
- L^2DCT reduces AFCT by 50% over DCTCP & 95% over TCP
  - Improves tail latency (99^{th} percentile) by 37% over DCTCP
  - Requires no changes in switch hardware or applications
  - Can co-exist with TCP and is incrementally deployable

Outline

- Background
- L^2DCT Design
- Evaluation
  - At-Scale Simulations
  - Small-Scale Real Implementation
- Related Work and Conclusion

Outline

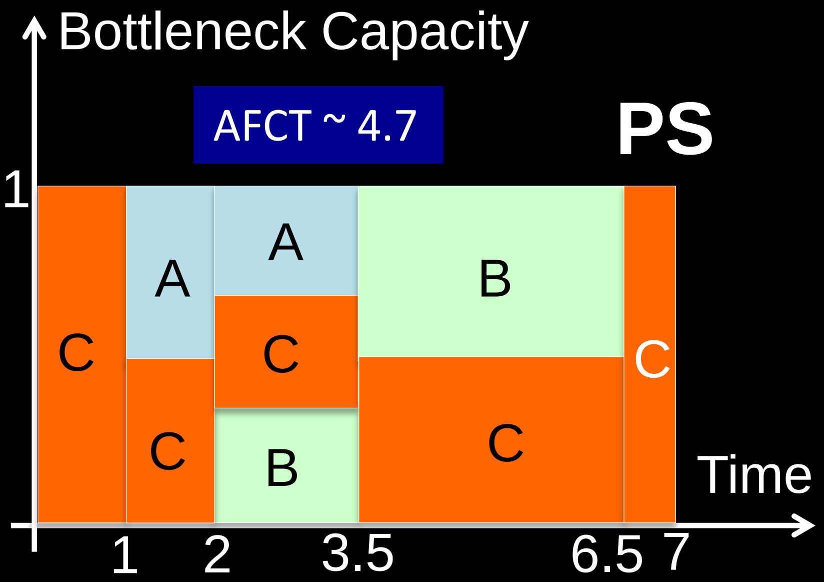
- Background
- L^2DCT Design
- Evaluation
  - At-Scale Simulations
  - Small-Scale Real Implementation
- Related Work and Conclusion

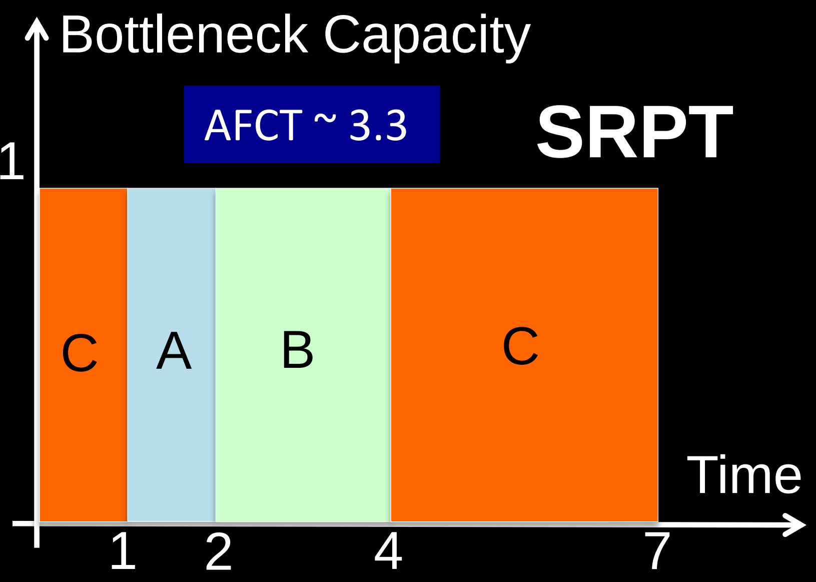
Background

- Shortest Remaining Processing Time (SRPT) is known to be optimal for minimizing AFCT [Schrage, OR'68]
  - Always process the job with least remaining work
- Brings significant improvements in completion times over Processor Sharing (PS) [Bansal et al., SIGMETRICS' 01]
  - Especially if jobs follow a heavy-tailed distribution
  - Shown to hold in data centers environments [Alizadeh et al., SIGCOMM'10, Greenberg et al., SIGCOMM'09]

Example

Flow ID	Size	Start Time
A	1	1
B	2	2
C	4	0





SRPT improves over PS (in this case) by ~30%

SRPT Challenges

- Requires knowledge of flow sizes
- A centralized scheduler
- Deployment challenges

SRPT Challenges

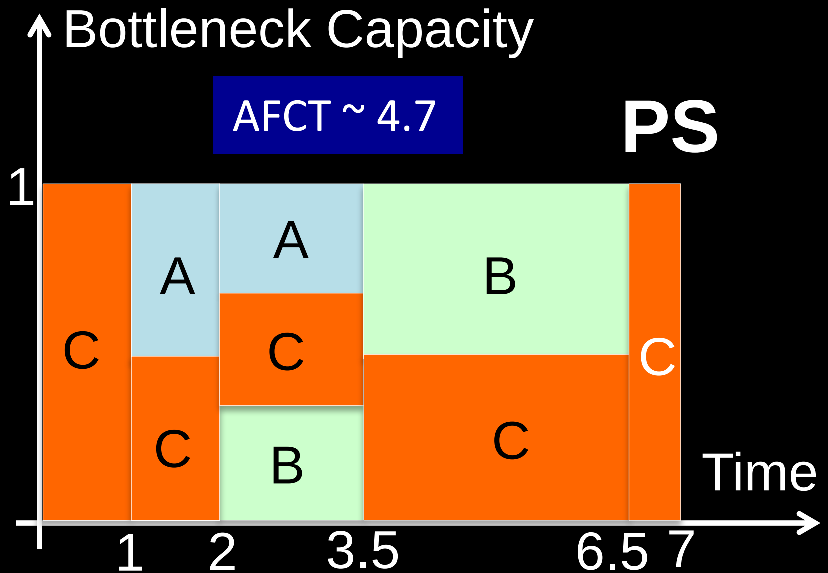
- Requires knowledge of flow sizes
- A centralized scheduler
- Deployment challenges

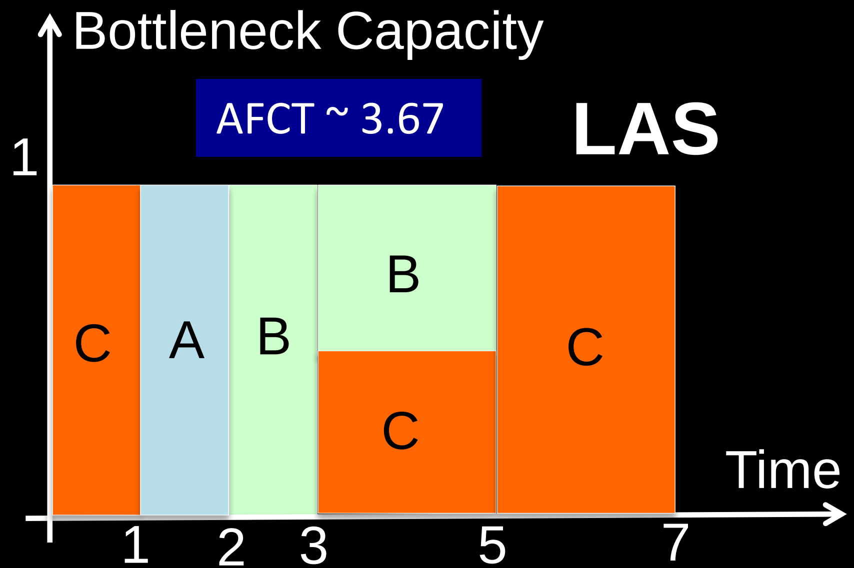
SRPT Challenges

- Requires knowledge of flow sizes
  - Use LAS (Least Attained Service) scheduling
    - Uses the data sent so so far for scheduling
    - Closely approximates SRPT for heavy-tailed distributions
- A centralized scheduler
- Deployment challenges

LAS vs PS

Flow ID	Size	Start Time
A	1	1
B	2	2
C	4	0





LAS improves over PS (in this case) by ~22%

SRPT Challenges

- Requires knowledge of flow sizes
- A centralized scheduler
- Deployment challenges

SRPT Challenges

- Requires knowledge of flow sizes
- A centralized scheduler
  - Incorporate LAS in distributed congestion control protocol
- Deployment challenges

SRPT Challenges

- Requires knowledge of flow sizes
- A centralized scheduler
- Deployment challenges

SRPT Challenges

- Requires knowledge of flow sizes
- A centralized scheduler
- Deployment challenges
  - The designed protocol should not require changes in switches or applications
  - It should be able to co-exist with TCP

Outline

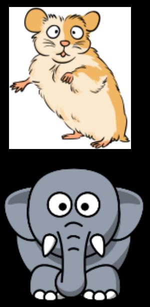
- Background
- L^2DCT Design
- Evaluation
  - At-Scale Simulations
  - Small-Scale Real Implementation
- Related Work and Conclusion

L^2DCT addresses these challenges

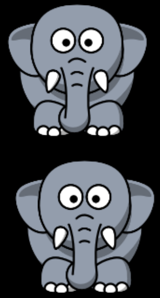
- Uses LAS
  - Flows modulate window sizes based on priority (data sent so far) and network congestion
  - Does not require flow size information
- Requires no changes in applications or switches
- Can co-exist with TCP

Goals

- Long flows should allow a greater short term share of the bandwidth to short flows



- When only long flows are present, they should be able to achieve high throughput and not be penalized any more than TCP



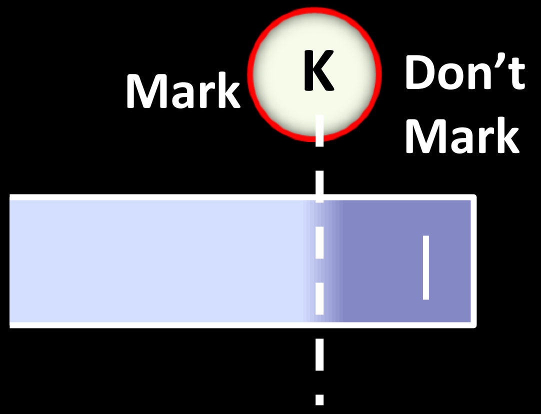
- When congestion becomes severe, all flows should converge to applying full backoff, similar to TCP, to prevent congestion collapse

L^2DCT Sender

- Flows adapt AIMD parameters based on
  - Priorities: captured by w_c = f(data sent so far)
  - Network congestion: captured by the alpha parameter
- L^2DCT meets the desirable goals by dynamically adapting the AIMD parameters

Marking at the Switches

- Queue length based marking



  - Similar to DCTCP [Alizadeh et al., SIGCOMM'10]

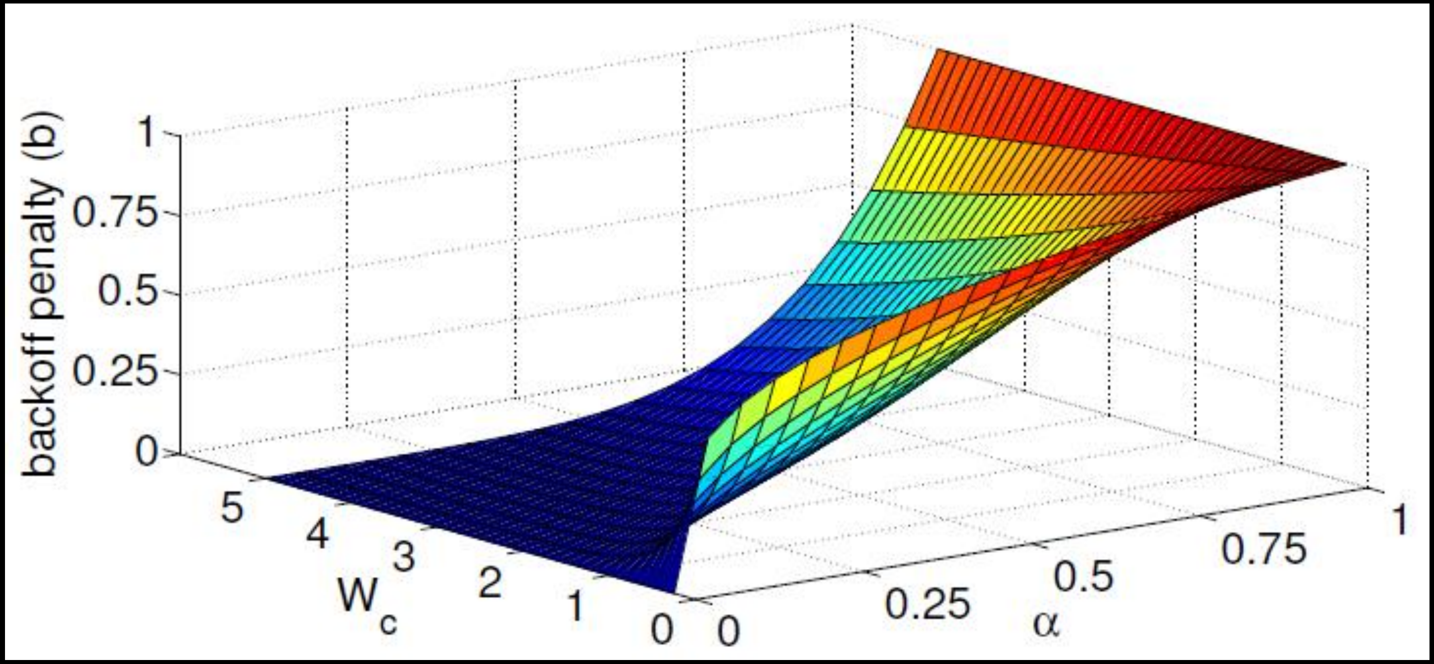
- Senders compute the fraction of marked packets F

$$\text{Each RTT}: F = \frac{\#\text{ of marked ACKs}}{\text{Total }\#\text{ of ACKs}} \Rightarrow \alpha \leftarrow (1-g)\alpha + gF$$

- Allows senders to react the extent of congestion

Decrease Rule

- Decrease: cwnd = cwnd x (1-b/2), where b= α^{w_c}



- Small increase in congestion causes long flows ($0<w_c<1$) to backoff more than short flows
  - But no more than TCP when congestion becomes severe

Increase Rule

- Increase: cwnd = cwnd + k, where $k=w_c/w_{max}$
  - New flows set $w_c = w_{max}$
  - Short flows start with k=1 and as flow progresses w_c decreases causing k to decrease

- Helps short flows to attain a greater short term share of bandwidth

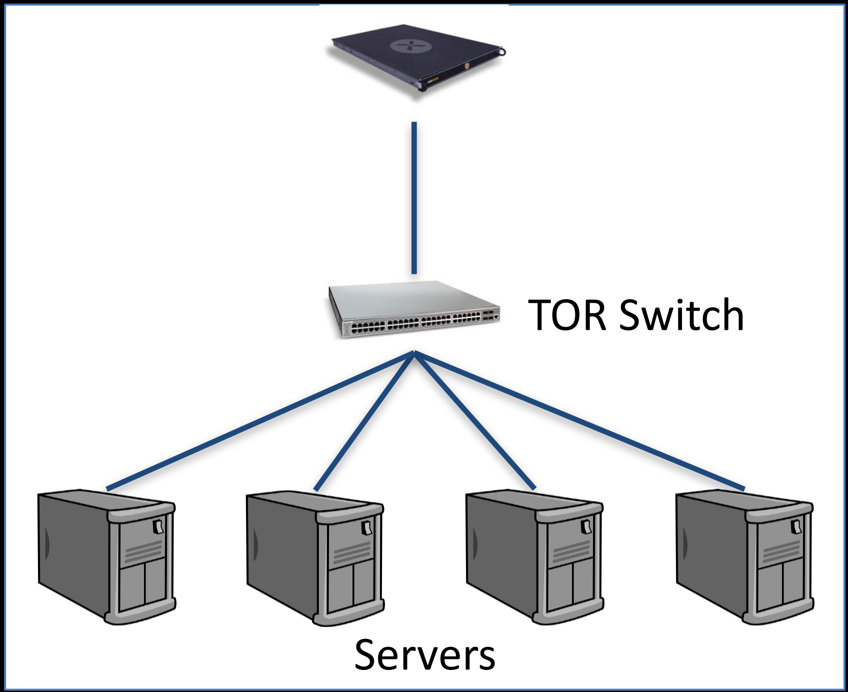
Outline

- Background
- L^2DCT Design
- Evaluation
  - At-Scale Simulations
  - Small-Scale Real Implementation
- Related Work and Conclusion

At-Scale Simulations

- Implemented L^2DCT in ns2
  - Comparison with TCP SACK, DCTCP, and D^2TCP

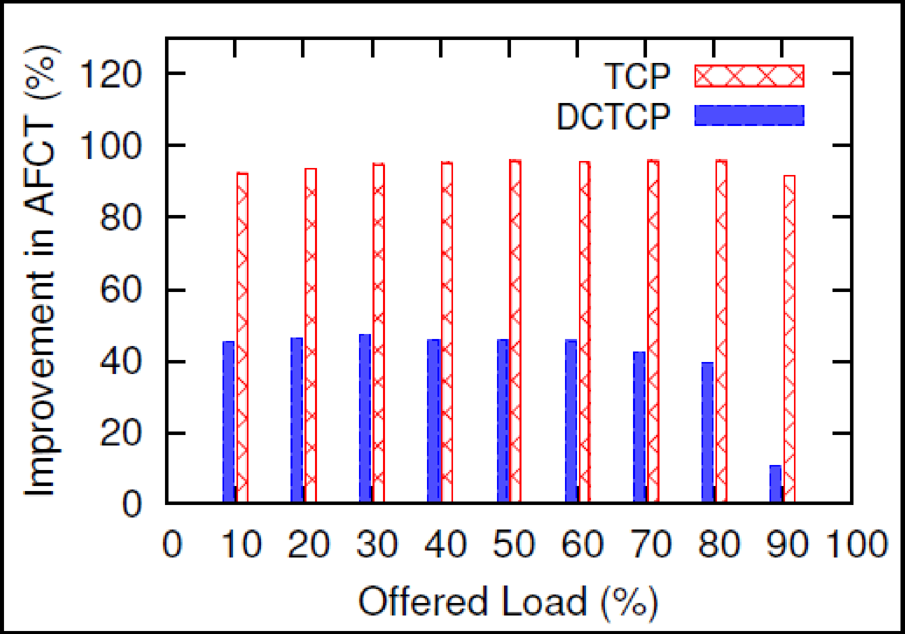
- Basic Topology: Single-rooted tree
- 1Gbps interfaces
- Round-trip delay: 300us
- Buffer Size: 250KB



Evaluation Scenarios

- Data center specific scenarios
  - Incast, Benchmark settings, Pareto distributed traffic
    - Impact of number of senders, flow size
- L^{2}DCT evaluation as a congestion control protocol
  - Single and multiple bottleneck scenarios, effect of sudden short flow bursts, impact of the weight function
- Deadline constrained flows
- Co-existence with TCP
- Realizing other scheduling policies with L^{2}DCT

AFCT Improvement



Settings:

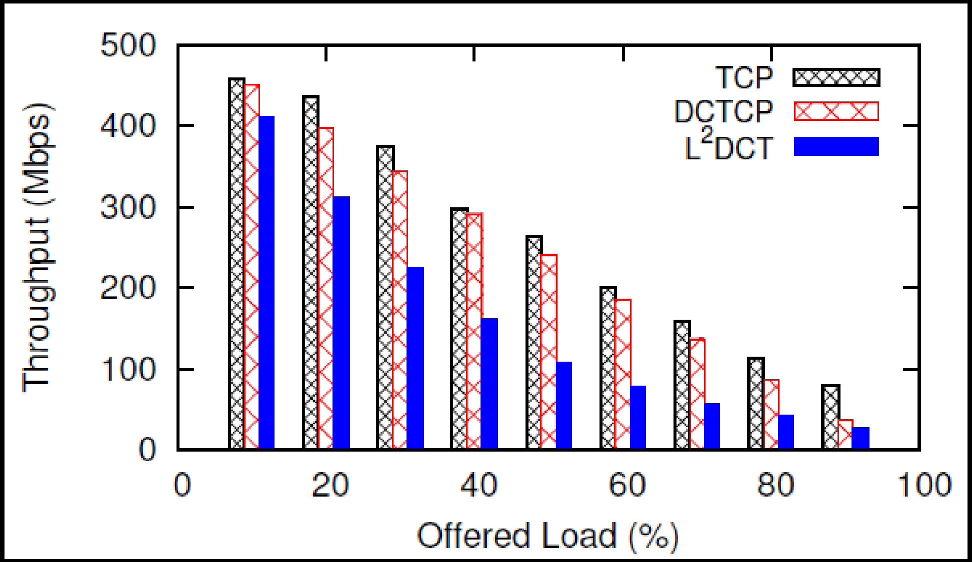
- Short Query Traffic: Uniformly distributed in [2KB, 98KB]

- Two Long-Lived Flows

  - 75th percentile of concurrent large flows [Alizadeh et al., SIGCOMM'10]

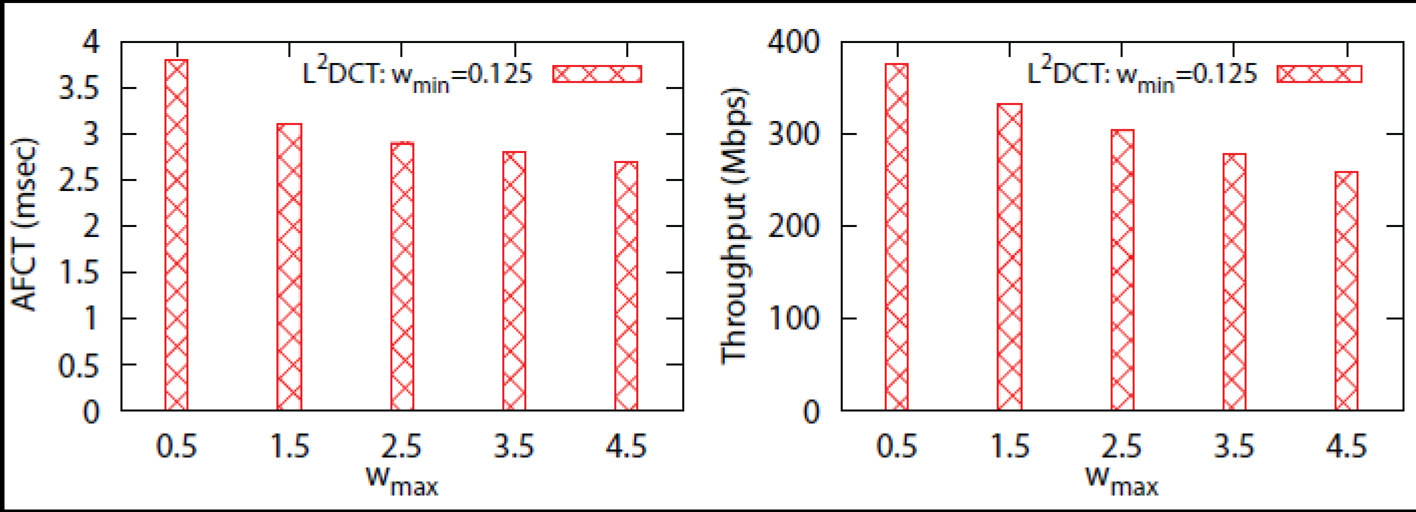
- At least 85% improvement in AFCT over TCP and at least 35% (for up to 80% load) over DCTCP
- 99th percentile of completion time improves by 37% over DCTCP
  - Similar results with Pareto distributed flow sizes

Impact on the Throughput of Long Flows



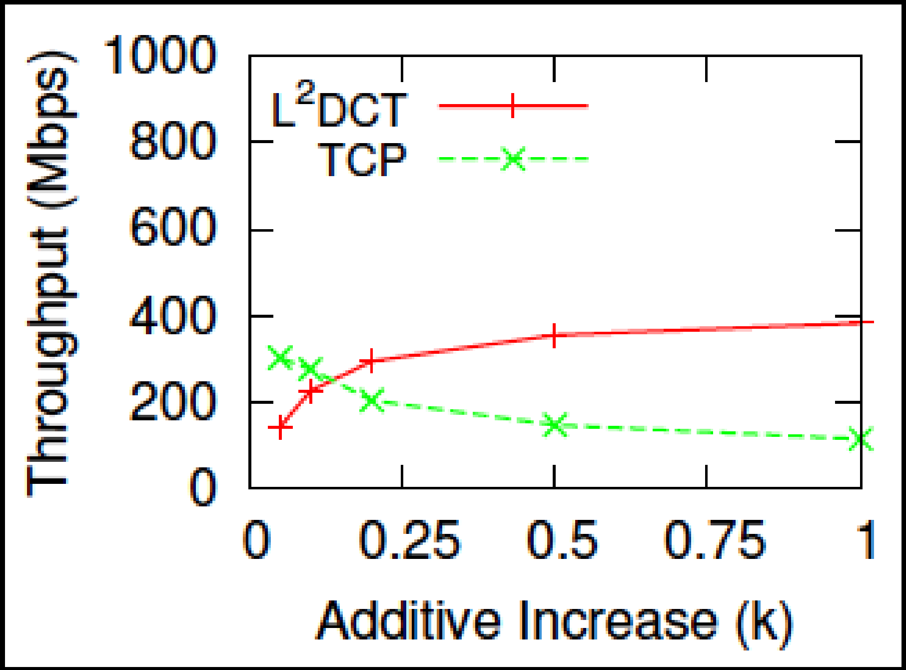
- At low load, throughput difference is small
- At high loads, more short flows arrive per sec, which increases the difference in throughput
- Different throughput/completion time tradeoff can be achieved by adjusting the cap on w_c

Impact of adjusting the cap on w_c



- As w_{max} increases, short flows become more aggressive
  - Leads to improvement in AFCT of short flows but also reduces the throughput of long flows
- Impact of changing w_{min} is similar

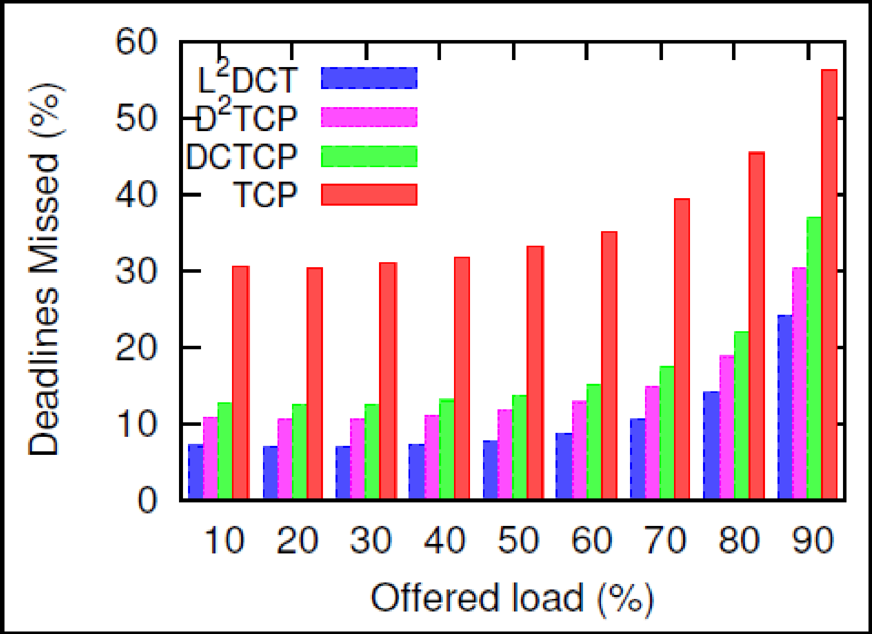
Co-existence with TCP



2 long-lived L^{2}DCT flows competing with 2 long-lived TCP flows

- L^{2}DCT can co-exist with TCP
  - Small values of k can yield similar throughput

Meeting Deadlines



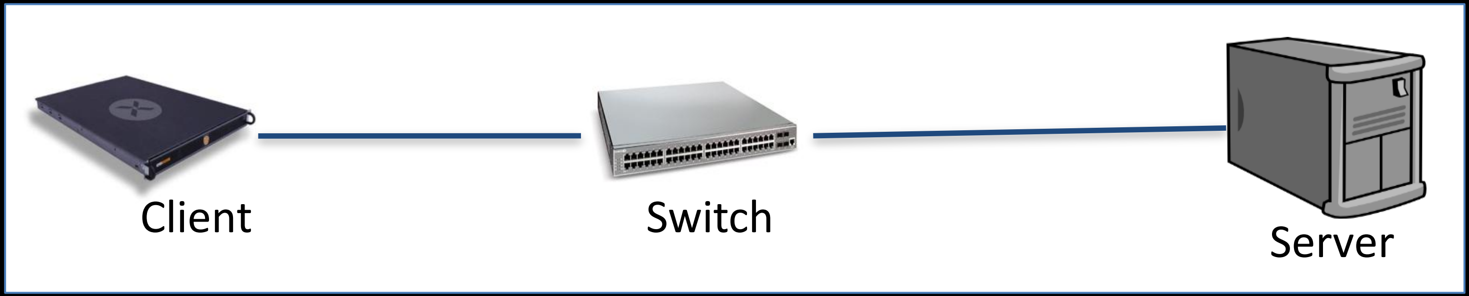
Settings:

- Flow deadlines ~ exponential with mean 40ms [Wilson et al., SIGCOMM'11]
- Short Query Traffic: Uniformly distributed in [2KB, 98KB]
- Two Long-Lived Flows

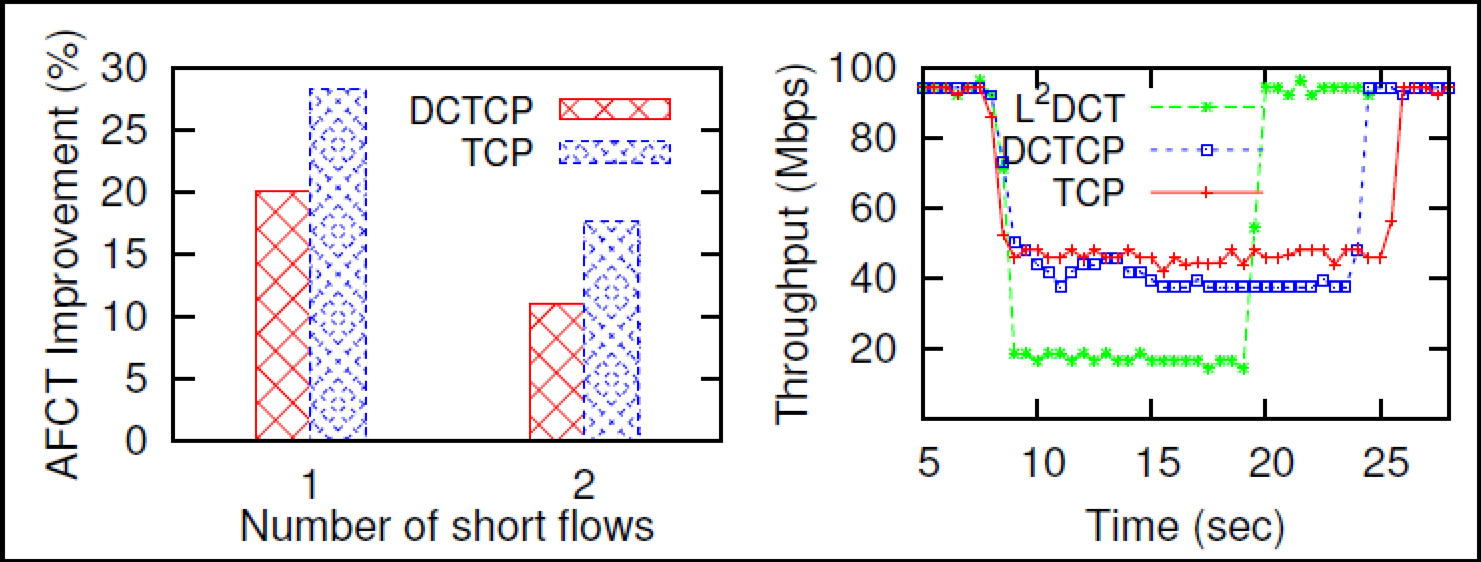
- L^{2}DCT consistently outperforms TCP, DCTCP and D^{2}TCP across a range of offered load
- Insight: A deadline agnostic protocol, which minimizes completion times, can achieve better or comparable performance than deadline-aware protocols

Testbed Evaluation

- Implemented L^2DCT in Linux
- We use the RED queue implementation in Linux for realizing the L^2DCT switch
- 3 Linux Machines (Client, Server, Switch)
  - 100Mbps interfaces



Results



- Single long-lived flow in the background
  – Multiple short flows are started
- L^2DCT improves AFCT by 20% and 29% over DCTCP and TCP, respectively

Outline

- Background
- L^2DCT Design
- Evaluation
  - At-Scale Simulations
  - Small-Scale Real Implementation
- Related Work and Conclusion

Prior Work

- PDQ [Hong et al., SIGCOMM'12]
  - Allows approximation of SRPT in a distributed manner
  - Requires changes in switch hardware and applications
  - Co-existence with TCP is challenging
- D^3 [Wilson et al., SIGCOMM'11], D^2CTCP [Alizadeh et al., SIGCOMM'12]
  - Deadline aware protocols
    - May not necessarily optimize AFCT
  - Require changes in applications (D^3 requires hardware changes)
  - We show that deadline-agnostic protocols can provide comparable performance
- HULL [Alizadeh et al., NSDI'12], DeTail [Zats et al., SIGCOMM'12]
  - Trades off some link bandwidth for latency – L^2DCT is complementary to HULL
  - DeTail: Focuses on tail latency and not AFCT

Conclusion

- L^2DCT reduces flow completion times by approximating LAS
  - Leads AFCT by up to 95% over TCP
  - Also reduces the tail latency
- It is incrementally deployable
  - Requires no changes in switches or applications
  - It can co-exist with TCP
- Can achieve comparable performance to deadline-aware protocols

Thank you!