aks249

Parallel Metropolis-Hastings-Walker Sampling for LDA

Xanda Schofield

- **Topic**: probability distribution across words (P("how") = 0.05, P("cow") = 0.001).
- **Document**: a list of tokens ("how now brown cow").
- **Topic model**: a way of describing how a set of topics could generate documents (e.g. Latent Dirichlet Allocation [Blei et. al., 2003]).

Inferring topic models is **HARD**, **SLOW**, and **DIFFICULT TO PARALLELIZE.**

Goal: to parallelize an optimized inference algorithm (MHW for LDA) efficiently for one consumer-grade computer.

Parallel Metropolis-Hastings-Walker Sampling for LDA

Xanda Schofield

Gibbs Sampling [Griffiths et. al. 2004]:

O(number of iterations * number of tokens * number of topics)

Metropolis-Hastings-Walker Sampling [Li et. al. 2014]:

O(number of iterations * number of tokens * number of topics **in a token's document**)

Needed for computations:

- N_{kd}: tokens in document d assigned to topic k
- N_{wk}: tokens of word w assigned to topic k
- $\underline{q}_w$: cached sampled topics for word w
- A few user-set parameters

Parallel Metropolis-Hastings-Walker Sampling for LDA

Xanda Schofield

How we do it:

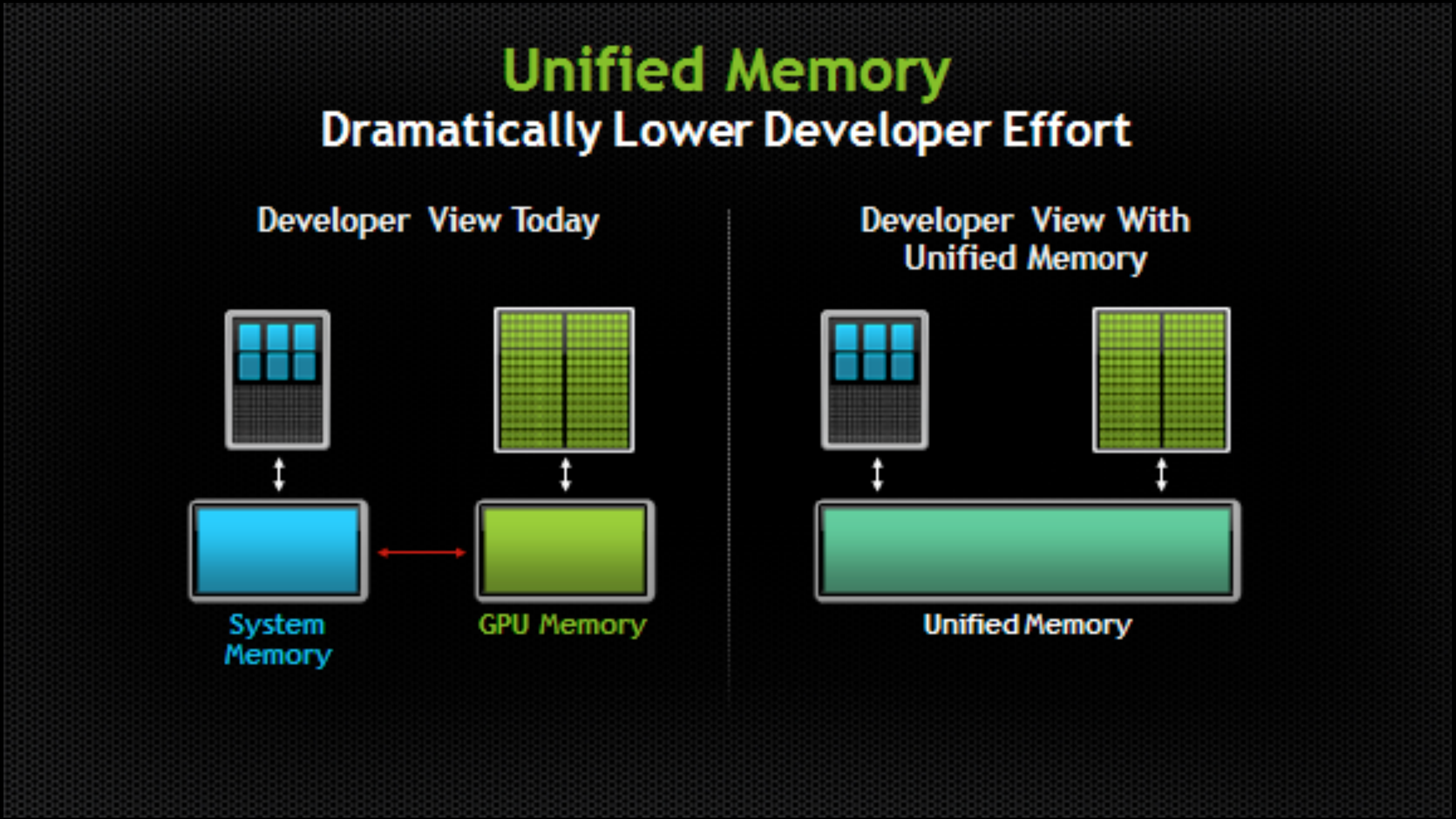
- Split documents across processors (N_{kd})
- Keep updated N_{wk}
  - Share N_{wk}
  - Synchronize N_{wk} each iteration
  - Gossip N_{wk} to a random processor each iteration
- Keep valid $\underline{q}_w$
  - Share
  - Make per-processor

Measuring comparative performance and held-out likelihood with # processors ers273

An Analysis of the CPU/GPU Unified Memory Model

Eston Schweickart

Unified Memory



3 Contexts

- Multi-stream Cross-device Mapping
    - Basic, intended use case for UM
- Big integer addition
    - Linked lists: hard to transfer
- Nonlinear Exponential Time Integration (Michels et al 2014)
    - Both GPU and CPU bound computation, nontrivial implementation

Analysis

- Ease of implementation
  - Lines of code
  - Required concepts
- Performance
  - Memory Transfer Optimizations?

Results

- UM is best as an introductory concept
  - Removes burden of explicit memory transfer
- UM is hard to optimize
  - No control over data location
  - Recommend: compiler hints, better profiling tools

fno2

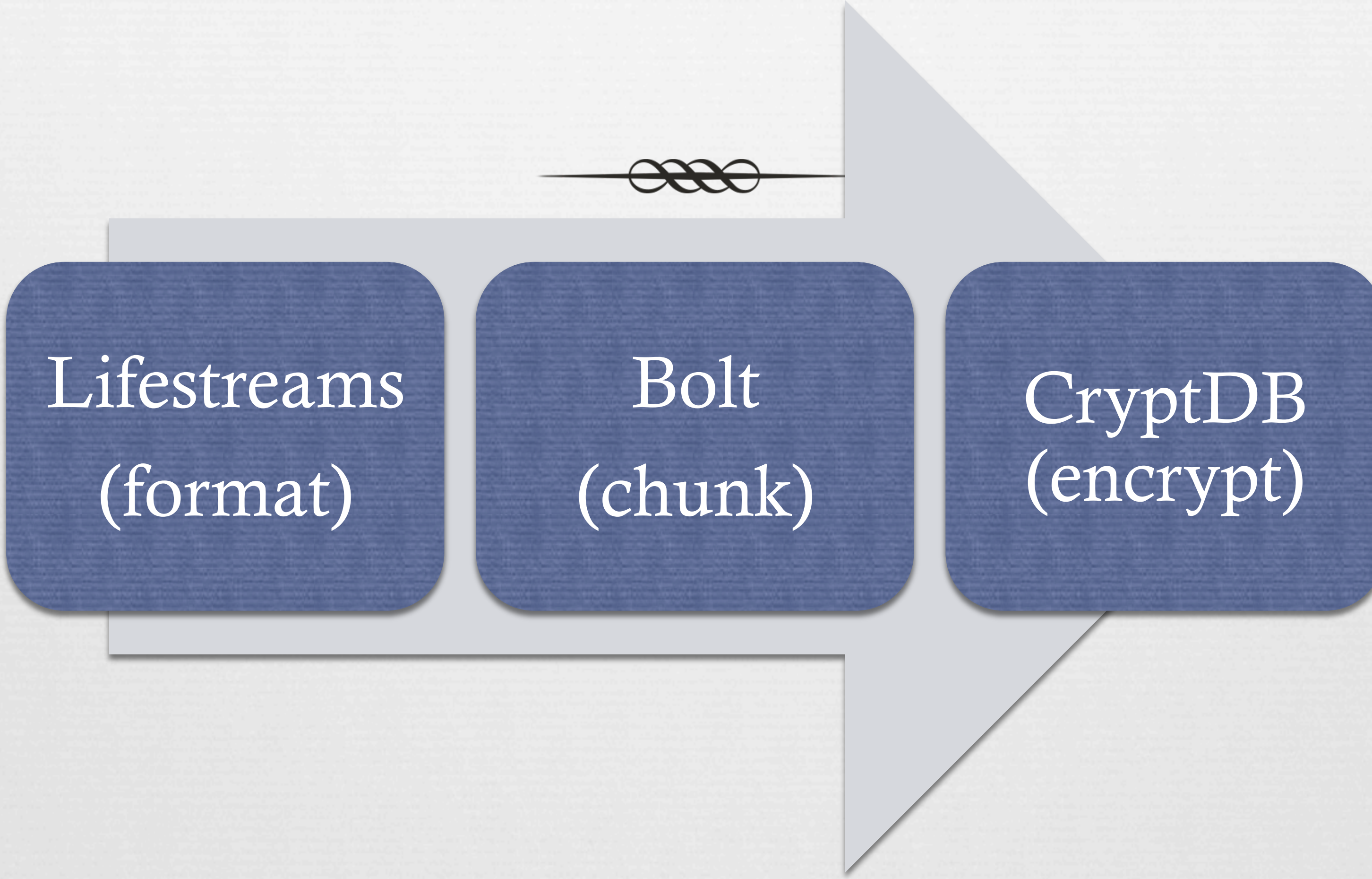
Encrypting small data

Fabian Okeke
CS 6410, Fall 2014

Provide insight

+

maximize privacy


Lifestreams (format)
Bolt (chunk)
CryptDB (encrypt)

- Built Lifestreams DPU
- Encrypted database & queries
- Demoed visualizations
- Developed 3rd party API

fz84

Timing Channel Mitigation in Scheduler

a Case Study of GHC

Fan Zhang

Dept. of Computer Science
Cornell University

December 4, 2014

Timing Channel
in Scheduler

- A timing channel is a secret channel for passing unauthorized information, which is encoded in certain timing information
  - E.g.: Cache timing: response time of a memory access can reveal information about whether the page is in cache or not
  - by observing running time of AES encryption thread, one can guess AES key.
- In OS, scheduler is an main source of secret information leakage

Timing Channel
in Scheduler

- Consider round-robin scheduling with epoch T
- Ideally, context switch happens at nT.
- However, for many reasons, context switch happens at $nT + \delta$ (where $\delta > 0$ is a random variable) as at nT,
  - thread is performing atomic operation (uninterruptable)
  - interrupt is disabled (so timer interrupt is ignored)
- δ is exploitable to pass secret information (even don't know how)
- $H = -\sum p_i \log(p_i)$

Problem

- How to measurement δ in a real world scheduler (GHC)?
  - Glasgow Haskell Compiler, is a state-of-the-art, open source compiler and interactive environment for the functional language Haskell.
- How to mitigate this timing channel, i.e. eliminate δ

Problem I

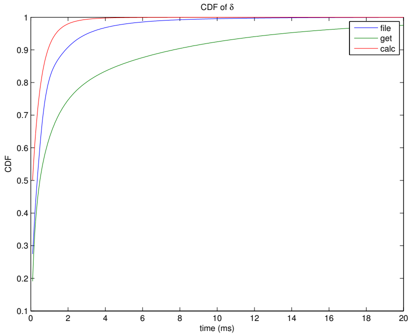
How to measurement δ in a real world scheduler (GHC)?

- Probe GHC $\rightarrow$ break GHC scheduler down $\rightarrow$ read GHC code..

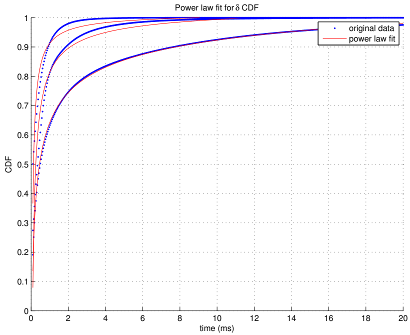
Workload dependent? Three samples:

- calc: an encryption thread which is computation intensive
- get: a networking thread retrieving files via HTTP
- file: a thread reading and writing files on disk

Results



(a) CDF of δ



(b) Power law fit ($p = ax^{-b} + c$)

	calc	file	get
Delay(δ)	6.05	14.52	49.82

Table: Timing channel capacity $H = -\sum p_i \log(p_i)$ (bit/s)

Problem

How to mitigate this timing channel, i.e. eliminate δ

Algorithm 1: Incremental round-robin schedule

Data: initial time slice T_0, and incremental value b, timer `t`

t_c = `t.expiration`

if *current thread can be switched* $\vee\ t_c \geq T_0/2$ **then**

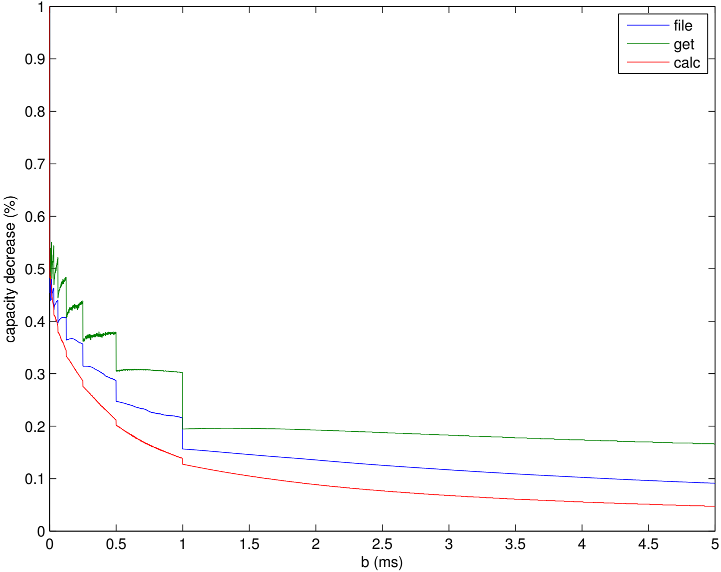
  set `context_switch` $= 1$

  reset `t.expiration` $= T_0$

else

  reset `t.expiration` $= t_c + b$

Results


file
get
calc
capacity decrease (%)
b (ms)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Conclusion

- Timing channel exists in scheduler, δ is an example
- δ can be approximated by power law distribution!
- I/O-bound threads tend to leak more information in its schedule footprint.
- Though the simulation shows that incremental round robin scheduling can effectively erase information entropy, timing channel mitigation is a difficult problem.

gjm97

PROOF OF STAKE IN THE ETHEREUM DECENTRALIZED STATE MACHINE

Galen Marchetti

Decentralized State Machine

- Ethereum
  - second-generation cryptocurrency – coins called "ether"
  - Bitcoin Blockchain + Ethereum Virtual Machine

- Stakeholders in network purchase state transitions
  - Mining fee includes cost per opcode in EVM
  - Miners provide Proof-of-Work to register transitions in blockchain

Establishing Consensus

- Proof-Of-Work
  - Consensus group is all CPU power in existence
  - Miners solve crypto-puzzles
  - Employed by Bitcoin, Namecoin, Ethereum
  - Subject to 51% outsider attack
- Proof-Of-Stake
  - Consensus group is all crypto-coins in the network
  - Miners provide evidence of coin possession

Mining Procedure

- Select parent block and “uncles” in blockchain
- Generate nonce and broadcast block header
- Nodes receiving empty header deterministically select N pseudo-random stakeholders
- Each stakeholder signs blockheader and broadcasts to network
- Last stakeholder adds state transistions, signs total block with its own signature, broadcasts to network.
- Mining profit evenly distributed among stakeholders and original node

Evaluation

- Mining now requires several broadcast steps
- Use Amazon’s EC2 with geographically separate nodes

- Measure time for pure Ethereum cluster to propagate state transitions in blockchain
- Measure time for Proof of Stake Ethereum cluster to propagate state transitions

ica23

Multicast Channelization for RDMA

Isaac Ackerman

Channelization

- Routing multicast traffic is difficult
- Share resource for highest performance

RDMA

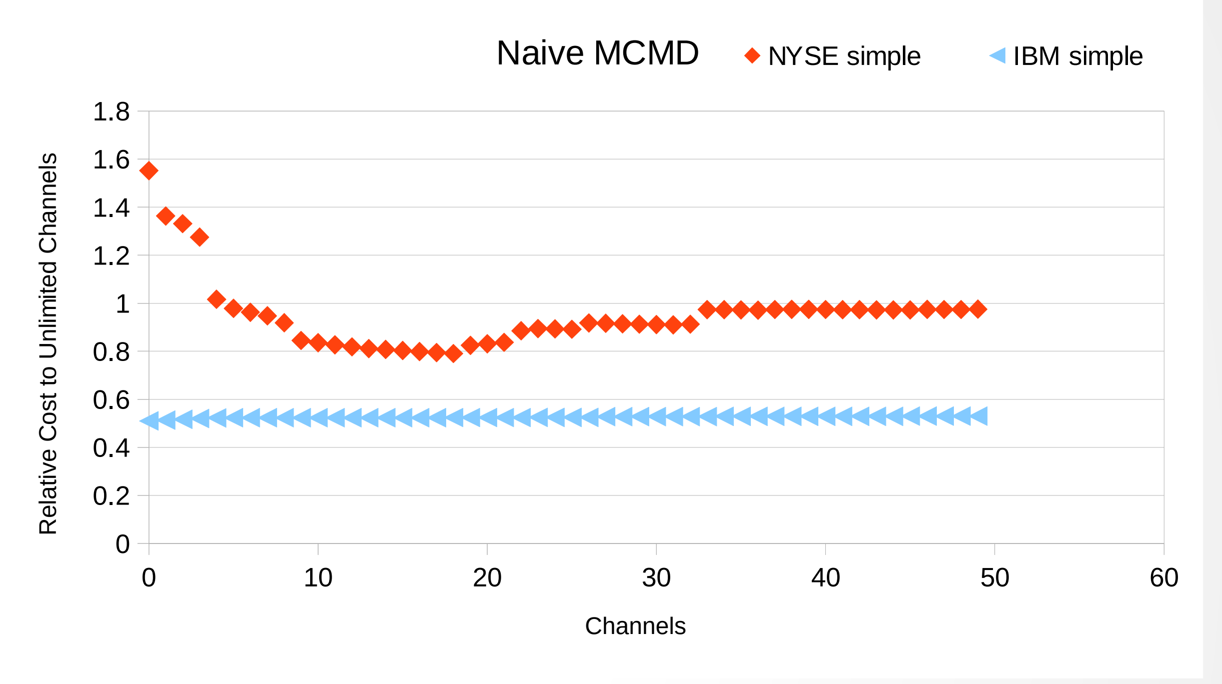
- Verbs interface
- Pre-allocate buffers
- Sender needs buffer descriptor

Model

- Cost for each send/receive
- Cost for client to hold buffers open
- Cost to coordinate senders

Existing Solutions

- Clustering
- MCMD

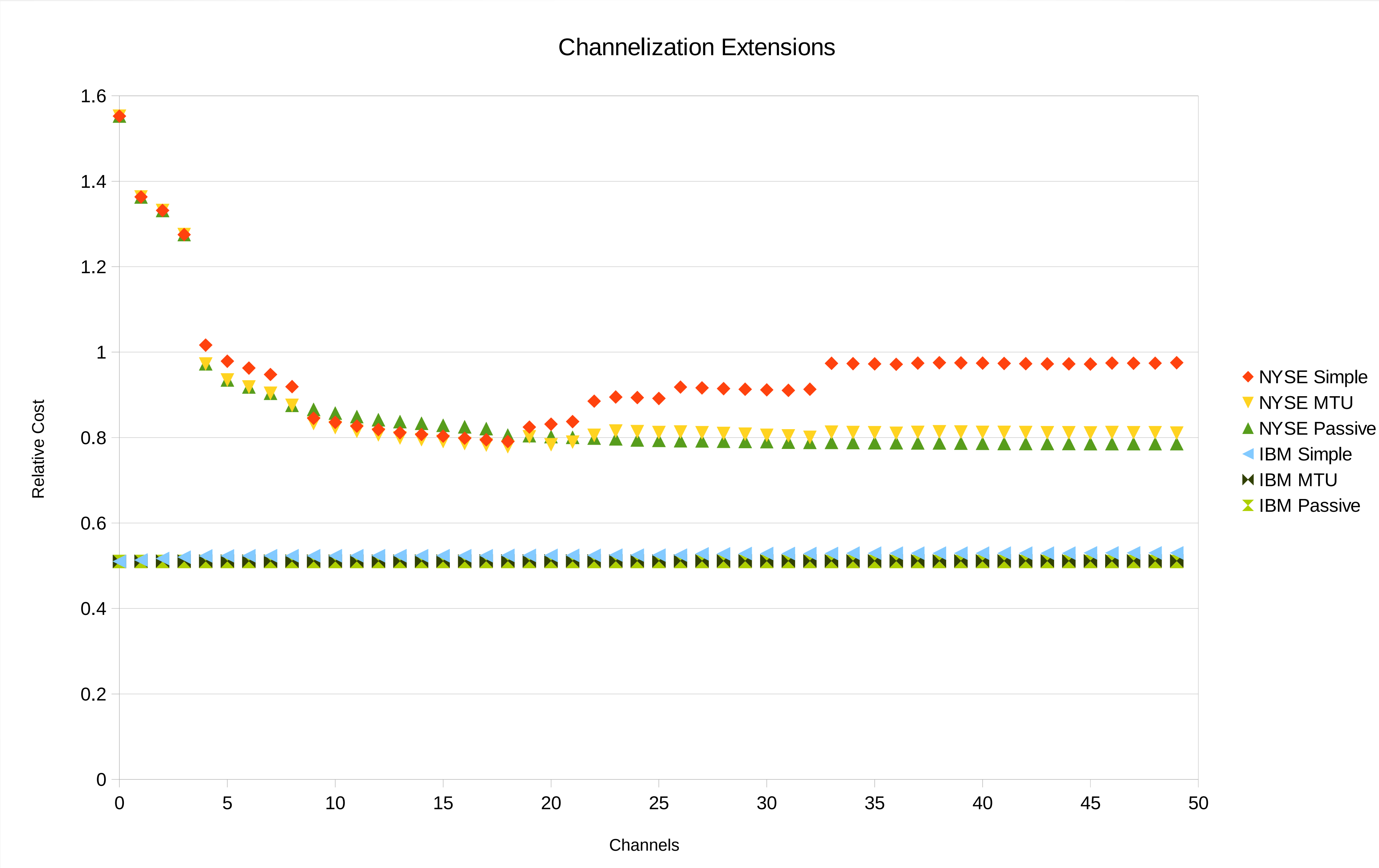


Doesn't consider memory consumption

Fixes

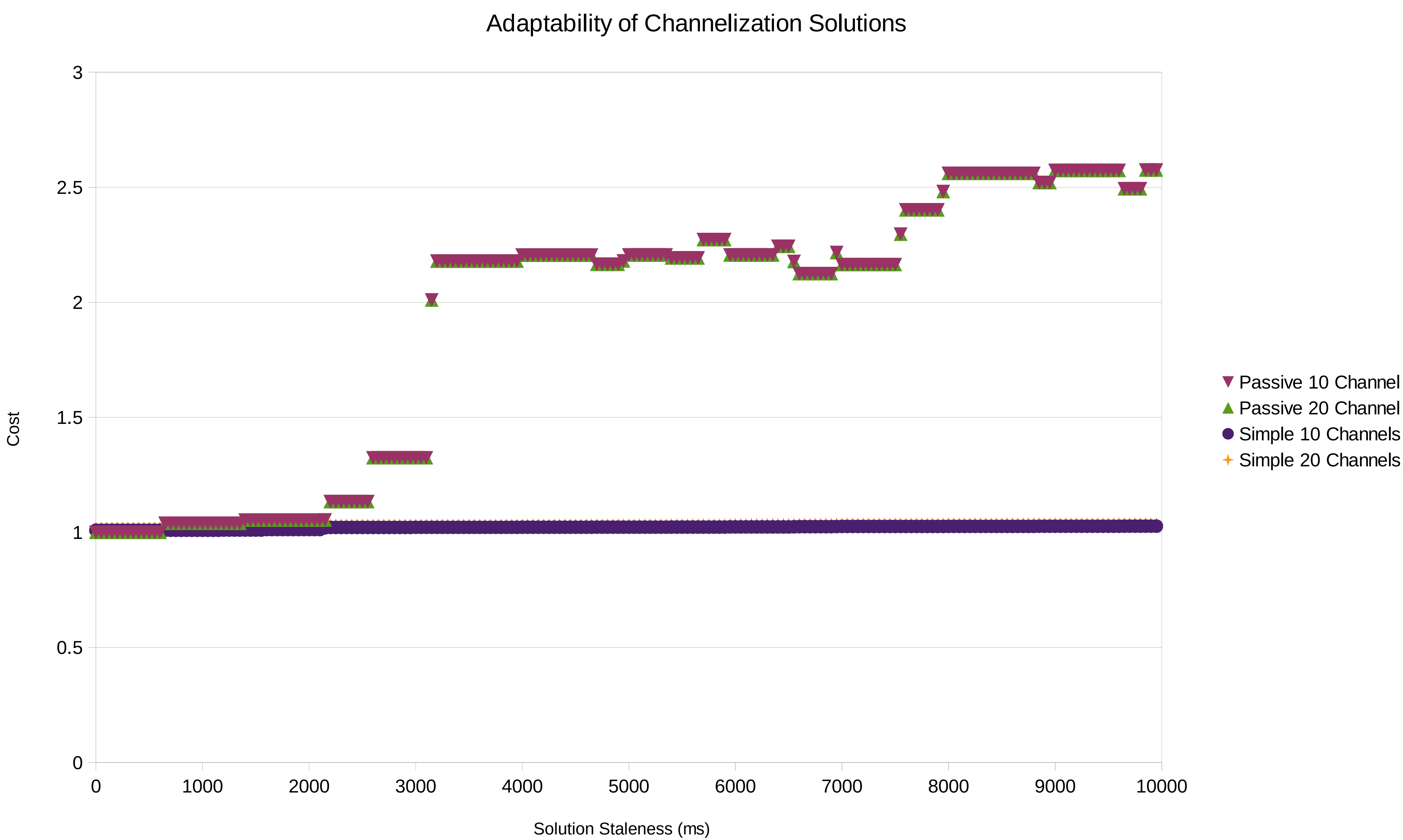
- Consider different MTU
  - Pseudopolynomial
  - Still slow
- Using channels incurs memory cost
  - Cautiously introduce new channels

Results



Channelization Extensions
Relative Cost
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
35
40
45
50
Channels
NYSE Simple
NYSE MTU
NYSE Passive
IBM Simple
IBM MTU
IBM Passive

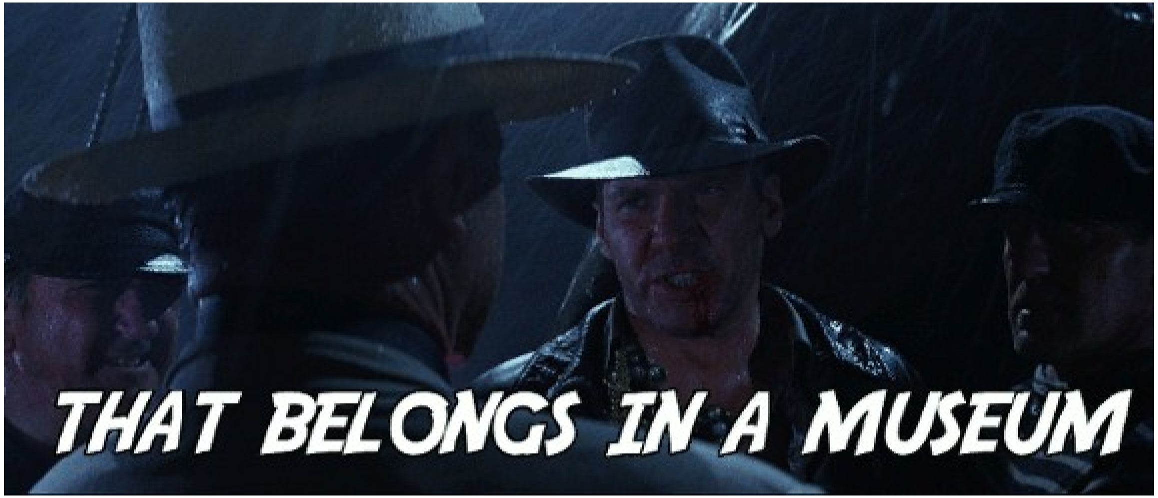
Adaptivity


Adaptability of Channelization Solutions
Cost
3
2.5
2
1.5
1
0.5
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Solution Staleness (ms)
Passive 10 Channel
Passive 20 Channel
Simple 10 Channels
Simple 20 Channels

Future Work

- Making use of unused channels
- Incorporating UDP for low rate flows
- Reliability, Congestion Control

km676



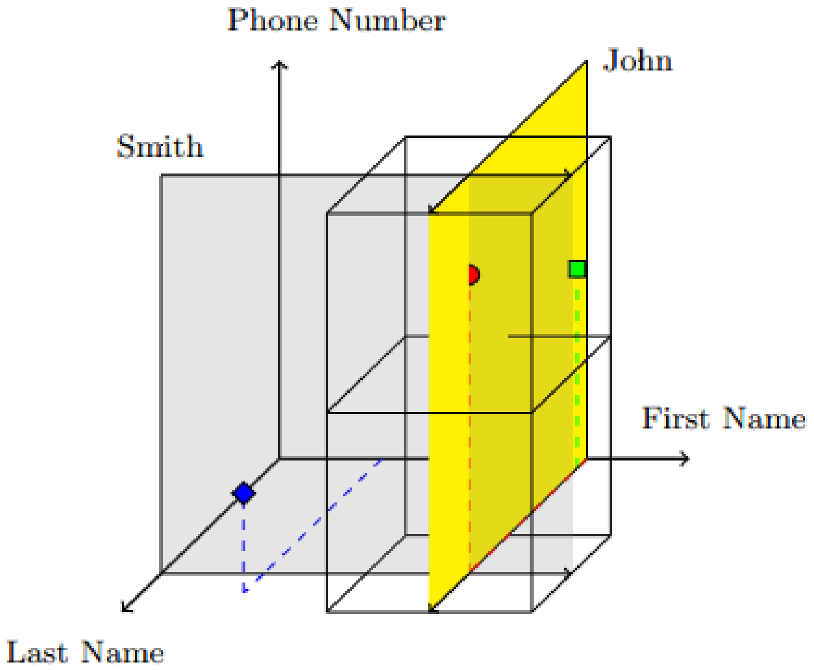
Kai Mast

The Brave New World of NoSQL

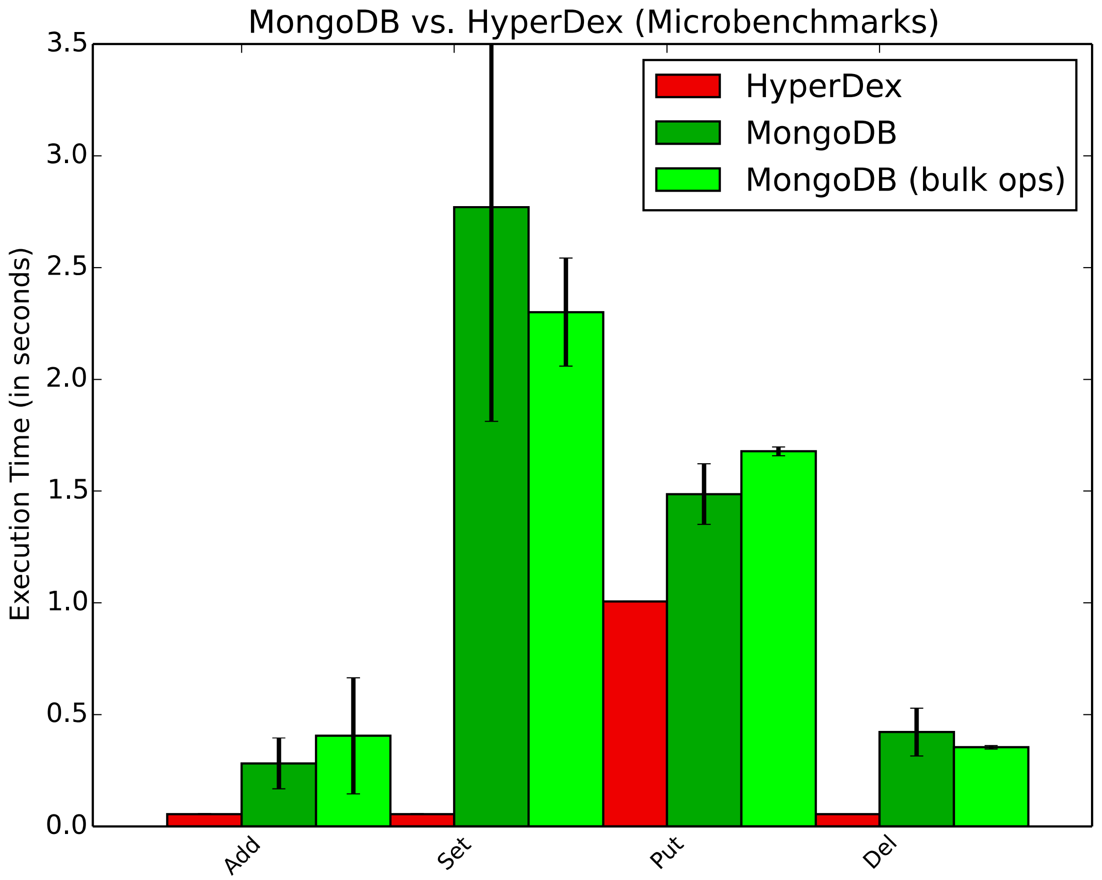
- Key-Value Store – Is this Big Data?
- Document Store – The solution?
- Eventual Consistency – Who wants this?

HyperDex

- Hyperspace Hashing
- Chain-Replication
- Fast & Reliable
- Imperative API
- But...Strict Datatypes

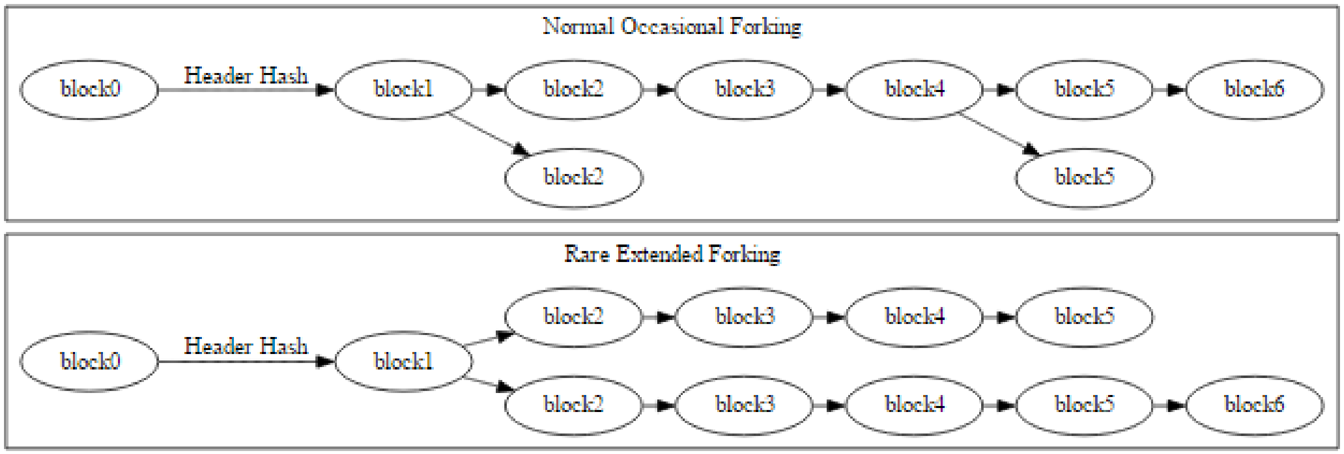



WHY DONT WE TAKE SCHEMALESS DOCUMENTS
AND PUT THEM INTO HYPERDEX
memegenerator.net


MongoDB vs. HyperDex (Microbenchmarks)
Execution Time (in seconds)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
HyperDex
MongoDB
MongoDB (bulk ops)
Add
Set
Put
Del

ktc34

Transaction Rollback in Bitcoin



- Forking makes rollback unavoidable, but can we minimize the loss of valid transactions?

Source: Bitcoin Developer Guide

Motivation

CVE-2010-5139

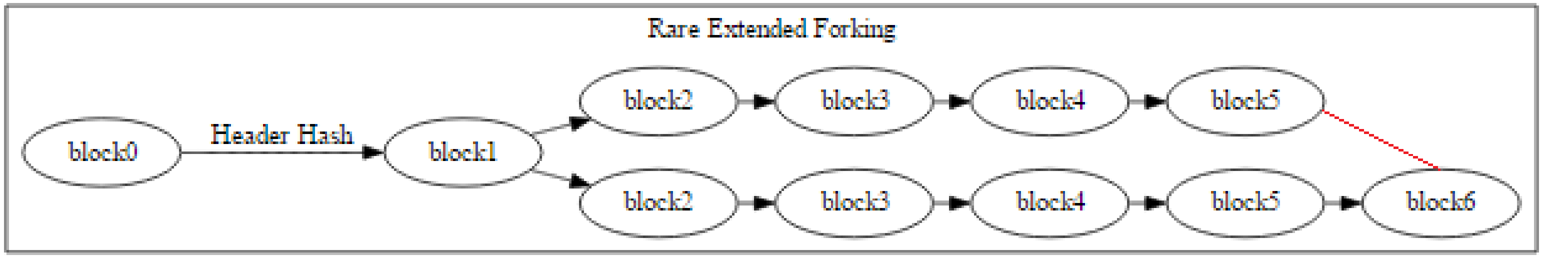
1d5e512a9723cbef373b970eb52f1e9598ad67e7408077a82fdac194b65333c9

237fe8348fc77ace11049931058abb034c99698c7fe99b1cc022b1365a705d39-000 0.5 BTC

92,233,720,368.54275808 BTC
1Hk51V49a58fC2r471hScXopEQpioDEuqx
92,233,720,368.54275808 BTC
12vRJXnnA21YAaLacWXpNshy7MBAwrigtQ
0.51 BTC *Miner's fee*

- Extended Forks
  - August 2010 Overflow bug (>50 blocks)
  - March 2013 Fork (>20 blocks)
- Partitioned Networks
- Record double spends

Merge Protocol



- Create a new block combining the hash of both previous headers
- Add a second Merkle tree containing invalidated transactions
  - Any input used twice
  - Any output of an invalid transaction used as input

Practicality
(or: Why this is a terrible idea)

- Rewards miners who deliberately fork the blockchain
- Cascading invalidations

- Useful for preserving transactions when the community deliberately forks the chain
  - Usually means something else bad has happened
- Useful for detecting double spends

ml2255

Topology Prediction for Distributed Systems

Moontae Lee

Department of Computer Science
Cornell University

December 4th, 2014

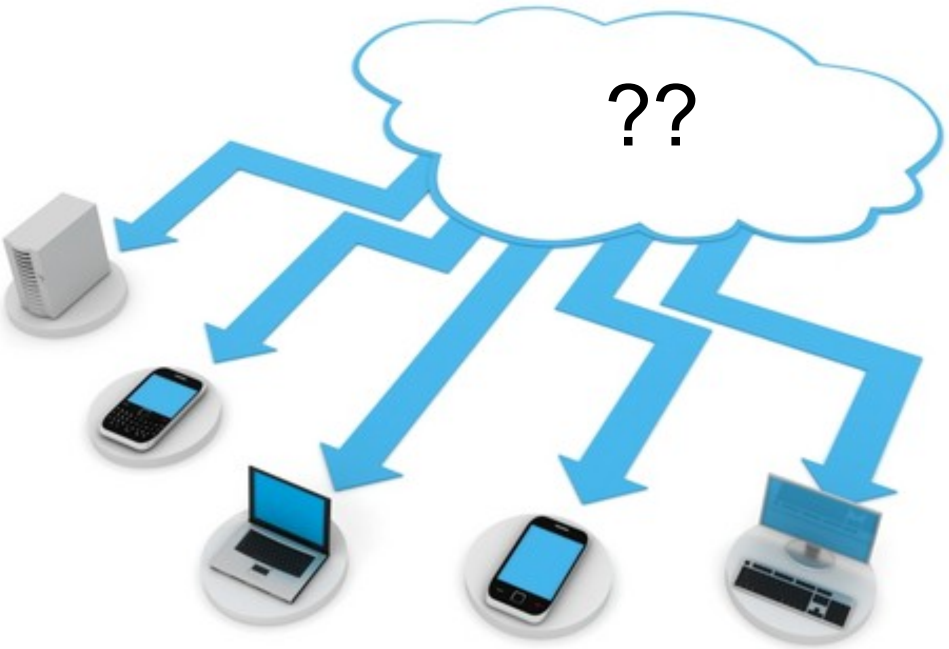
Introduction

- People use various cloud services
  - Amazon / VMWare / Rackspace
  - Essential for big-data mining and learning

Introduction

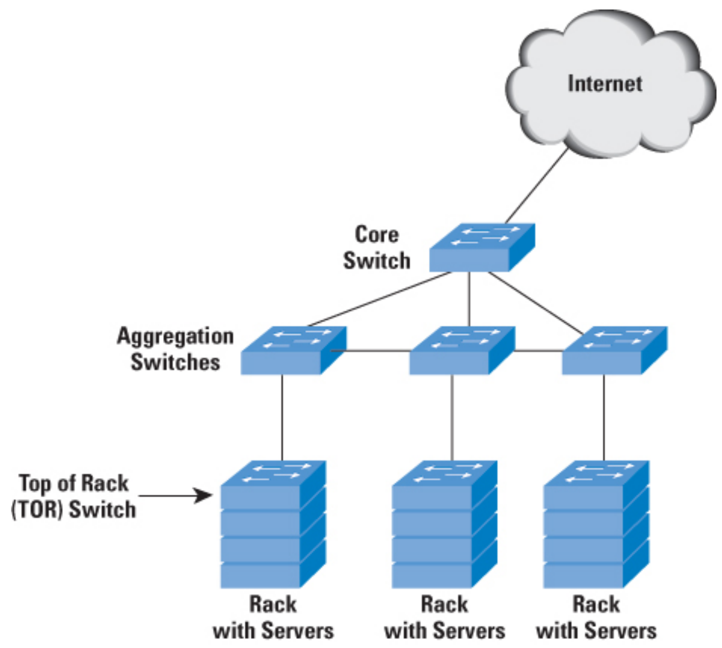
- People use various cloud services
  - Amazon / VMWare / Rackspace
  - Essential for big-data mining and learning

without knowing
how computer nodes
are interconnected!



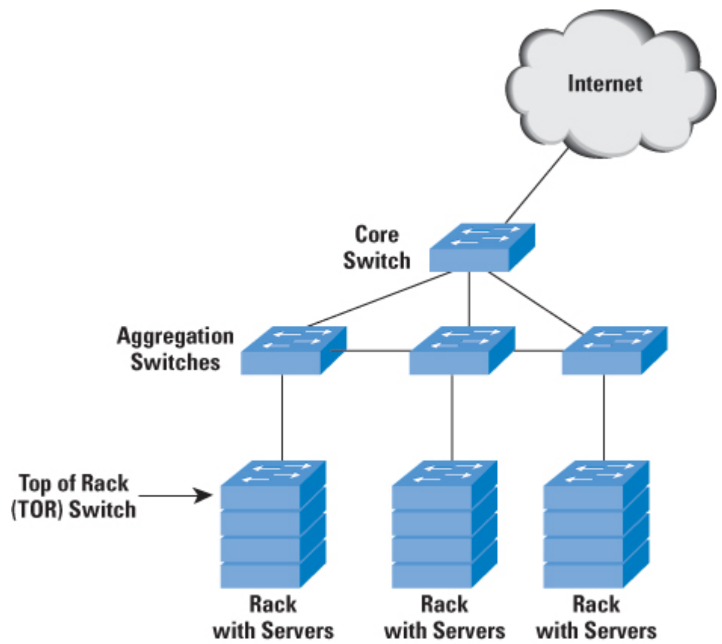
Motivation

- What if we can predict underlying topology?



Motivation

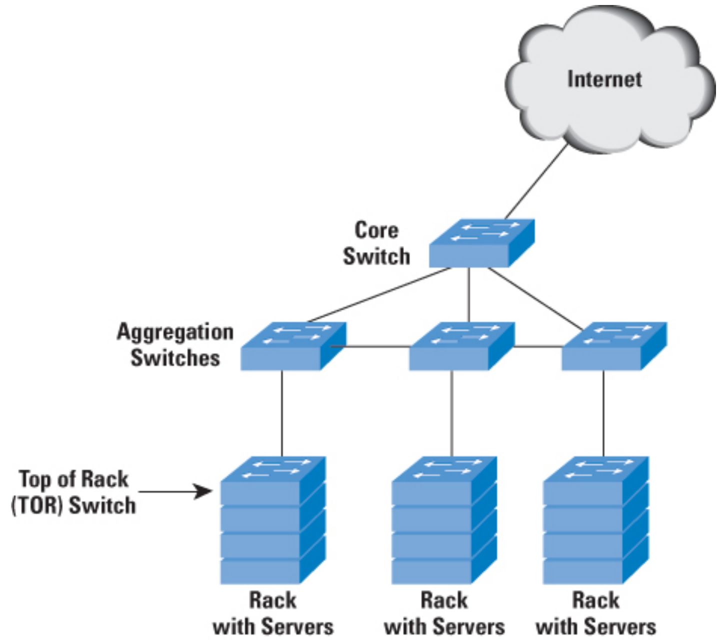
- What if we can predict underlying topology?



- For computer system (e.g., rack-awareness for Map Reduce)

Motivation

- What if we can predict underlying topology?



- For computer system (e.g., rack-awareness for Map Reduce)
- For machine learning (e.g., dual-decomposition)

How?

- Let's combine ML technique with computer system!

latency info → Black Box → topology

- Assumptions
  - Topology structure is tree (even simpler than DAG)
  - Ping can provide useful pairwise latencies between nodes

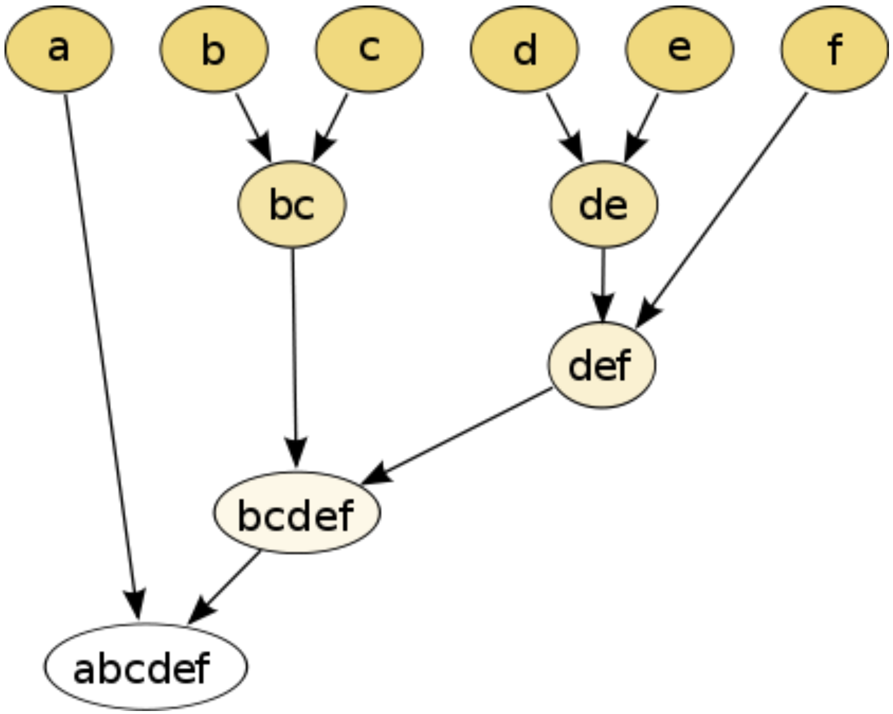
- Hypothesis
  - Approximately knowing the topology is beneficial!

Method

- Unsupervised hierarchical agglomerative clustering

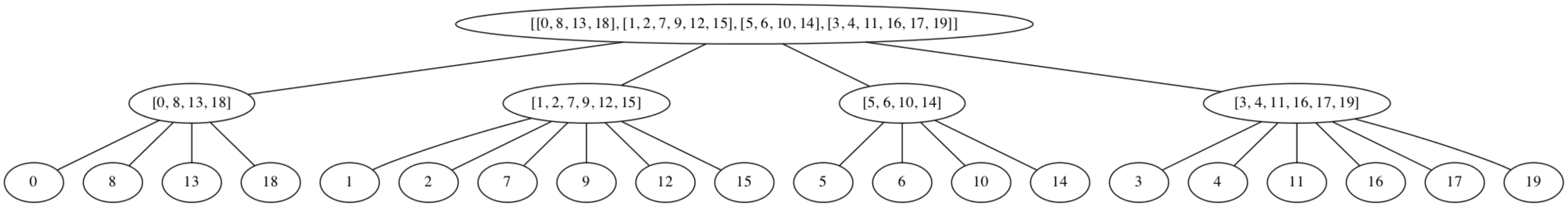
	a	b	c	d	e	f
a	0	7	8	9	8	11
b	7	0	1	4	4	6
c	8	1	0	4	4	5
d	9	5	5	0	1	3
e	8	5	5	1	0	3
f	11	6	5	3	3	0

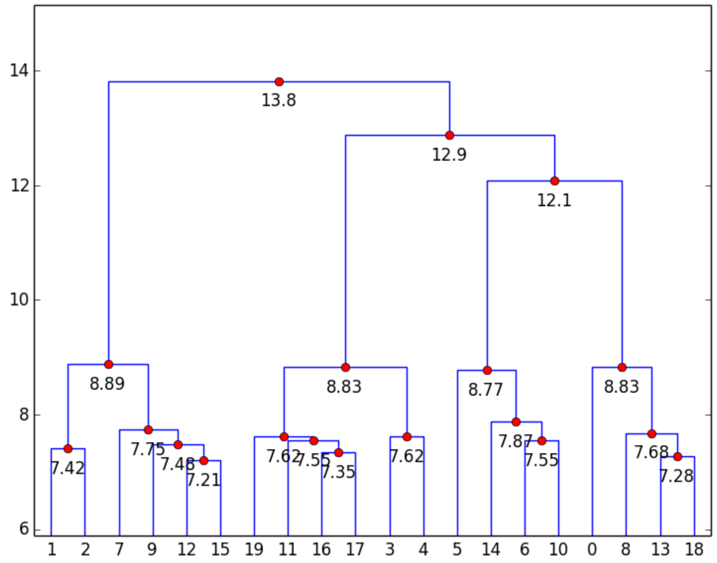
→

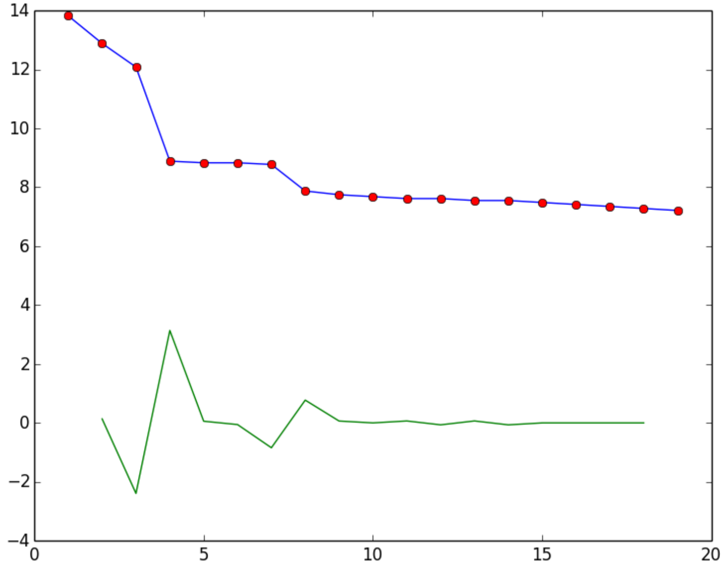


Merge the closest two nodes every time!

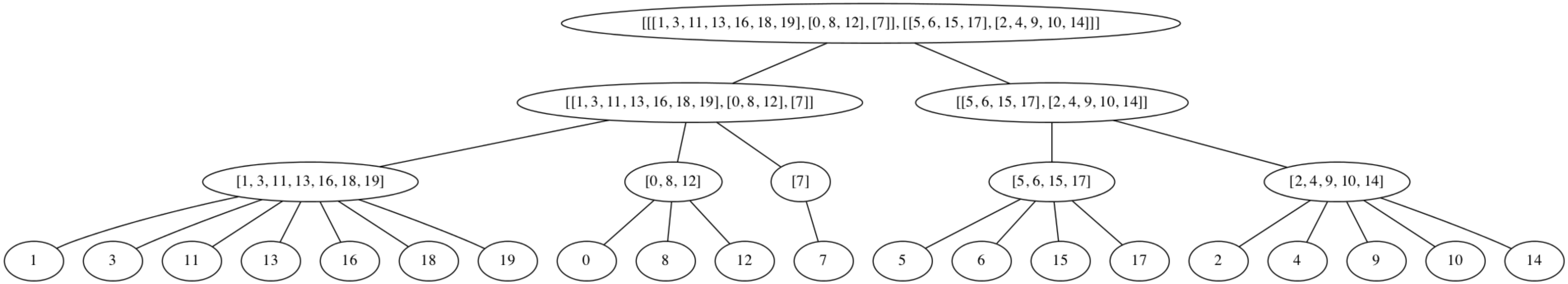
Sample Results (1/2)

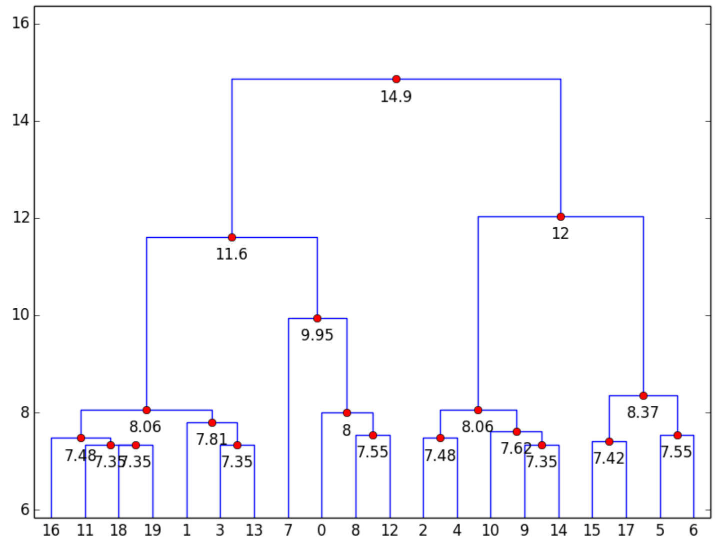


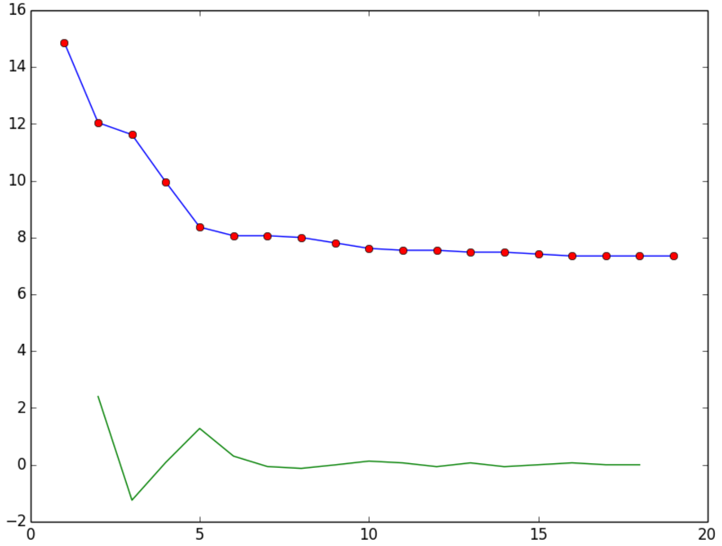




Sample Results (2/2)







Design Decisions

- How to evaluate distance? (Euclidean vs other)
- What is the linkage type? (single vs complete)

- How to determine cutoff points? (most crucial)
- How to measure the closeness of two trees?
  - Average hops two the lowest common ancestor

- What other **baselines**?
  - K-means clustering / DP-means clustering
  - Greedy partitioning

Evaluation

- Intrinsically (within simulator setting)
  - Compute the similarity with the ground-truth trees

- Extrinsically (within real applications)
  - Short-lived (e.g., Map Reduce)
    - Underlying topology does not change drastically while running
    - Better performance by configuring with the initial prediction

  - Long-lived: (e.g., Streaming from sensors to monitor the powergrid)
    - Topology could change drastically when failures occur
    - Repeat prediction and configuration periodically
    - Stable performance even if the topology changes frequently

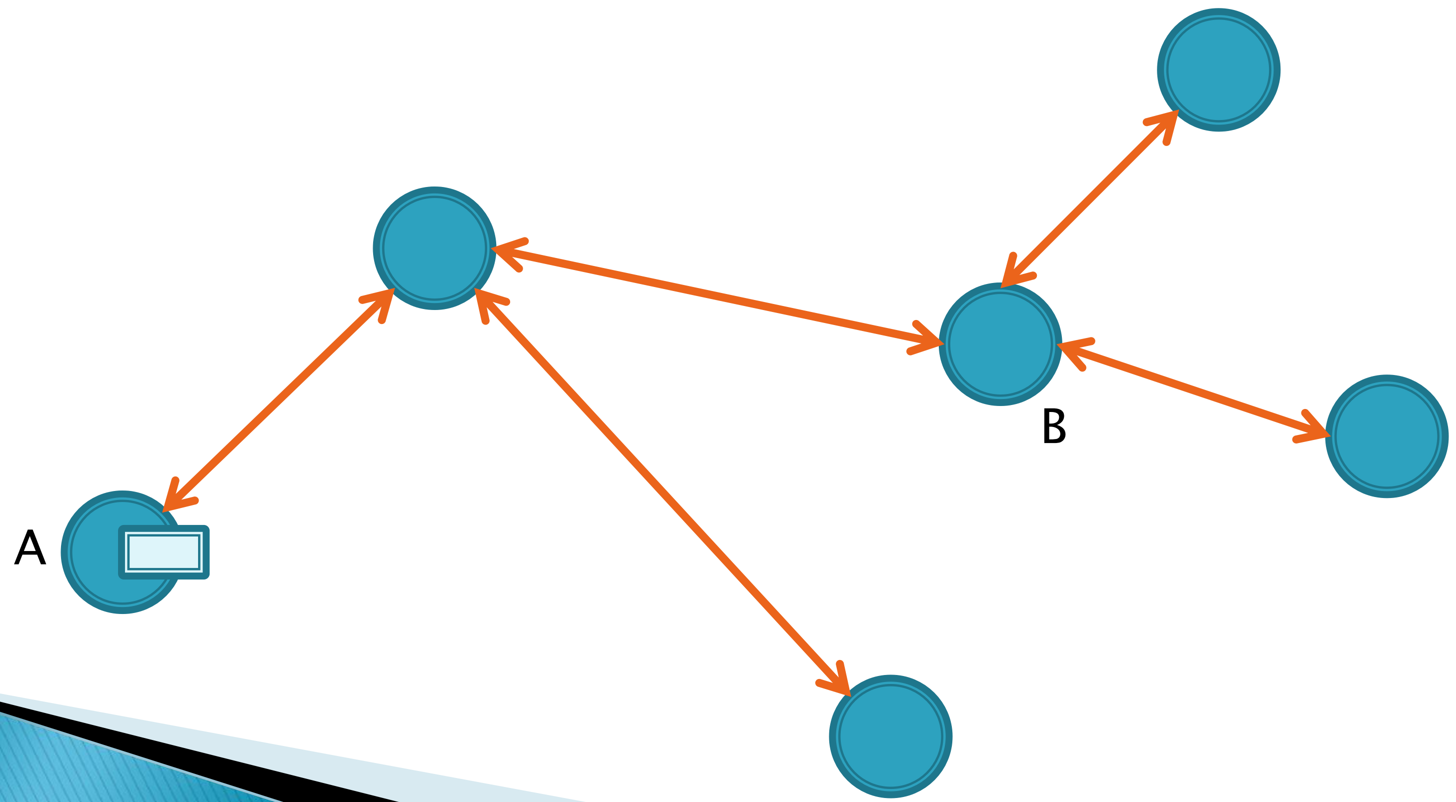
nja39

Distributed Network Topology Detection for the IronStack OpenFlow Controller

Noah Apthorpe

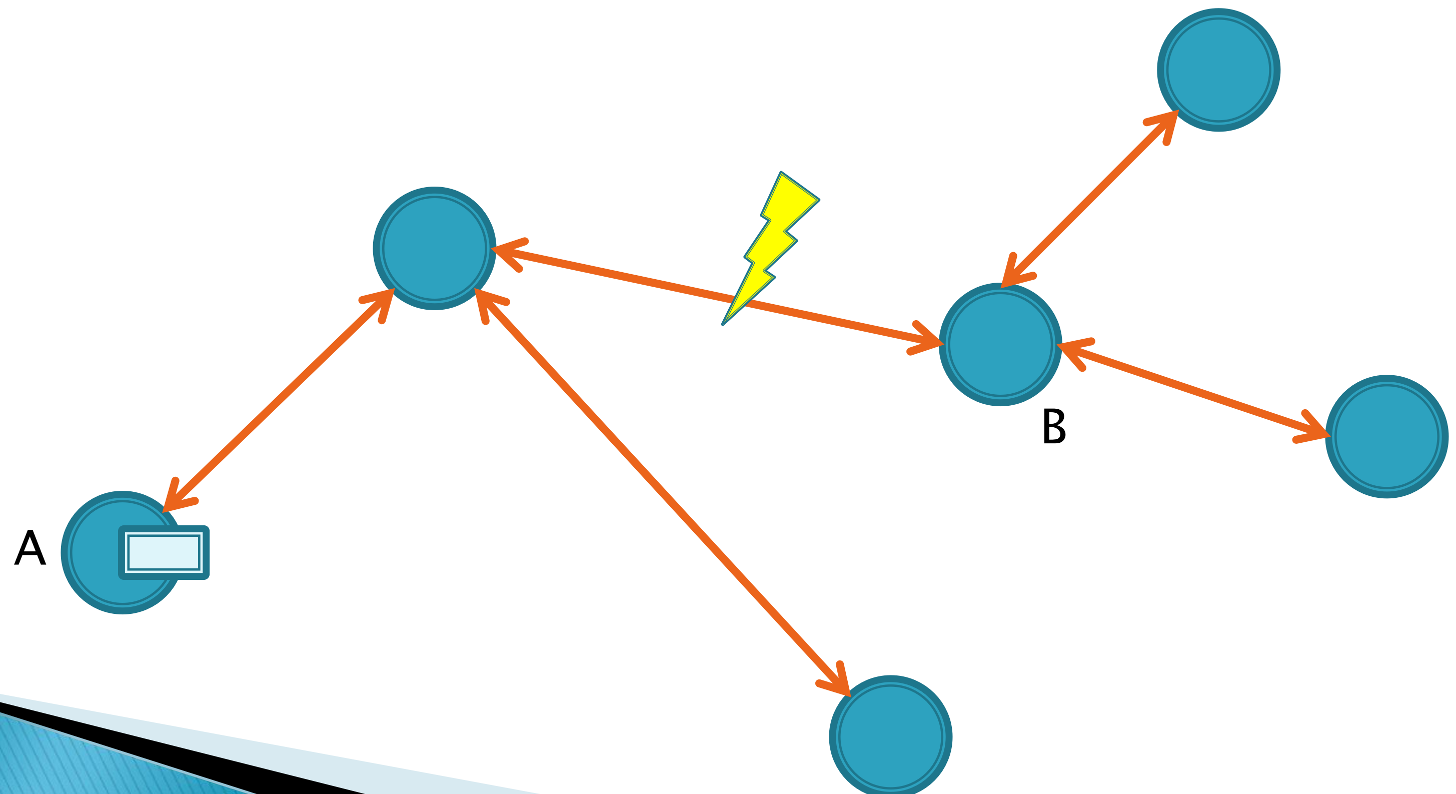
IronStack: RAID for Networks

- Commodity Ethernet
  - Spanning tree topologies
  - No link redundancy



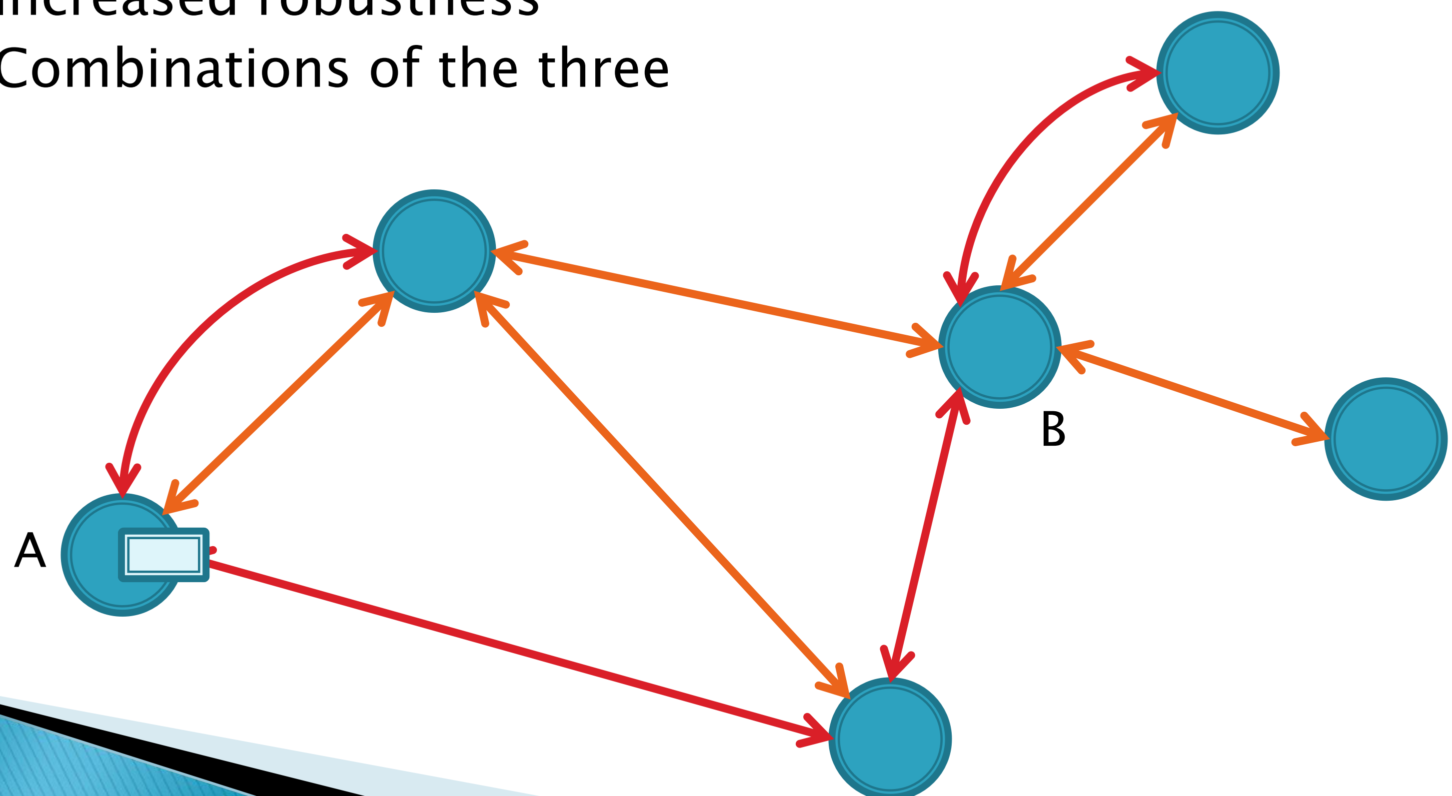
IronStack: RAID for Networks

- Commodity Ethernet
  - Spanning tree topologies
  - No link redundancy



IronStack: RAID for Networks

- IronStack spreads packet flows over disjoint paths
  - Improved bandwidth
  - Stronger security
  - Increased robustness
  - Combinations of the three



L2 Topology Detection: The Problem

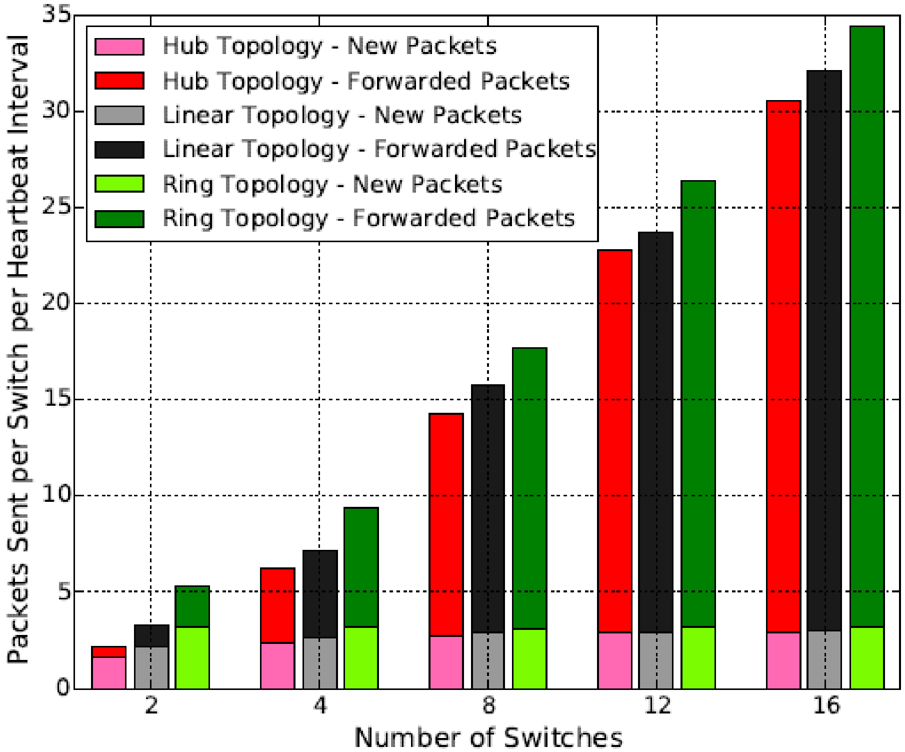
- IronStack controllers must learn and monitor network topology to determine disjoint paths
- One controller per OpenFlow switch
- No centralized authority
- Must adapt to switch joins and failures
- Learned topology must reflect actual physical links
  - No hidden non-IronStack bridges

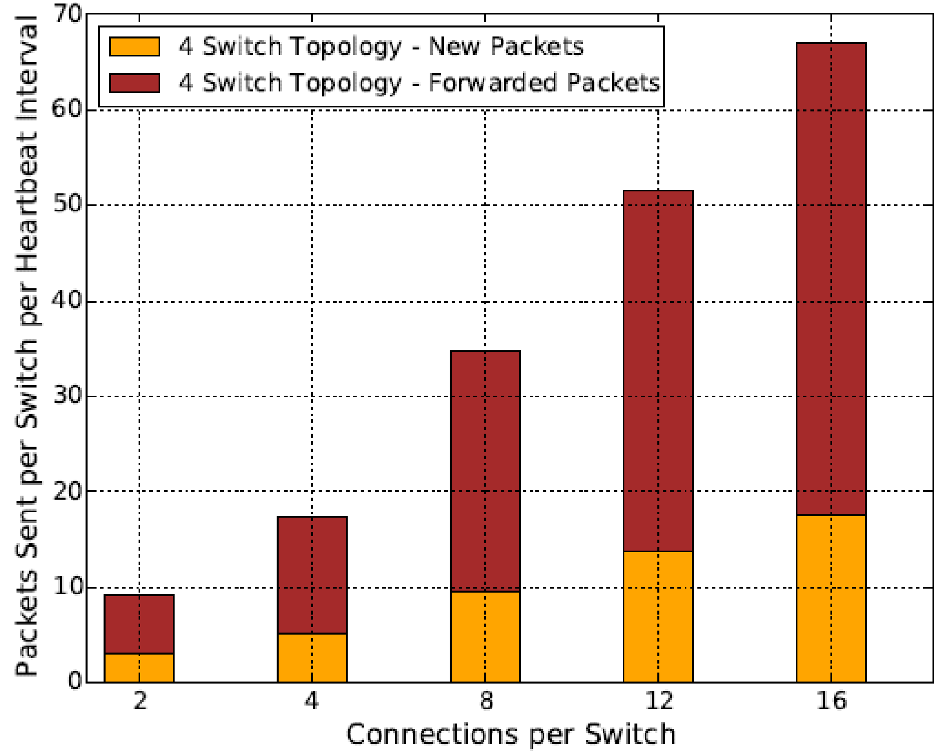
A Distributed Solution

- Protocol reminiscent of IP link-state routing
- Each controller broadcasts adjacent links and port statuses to all other controllers
  - Provides enough information to reconstruct network topology
  - Edmonds-Karp maxflow algorithm for disjoint path detection
- A "heartbeat" of broadcasts allows failure detection
- Uses OpenFlow controller packet handling to differentiate bridged links from individual wires
- Additional details to ensure logical update ordering and graph convergence

Protocol Efficiency Benchmarks

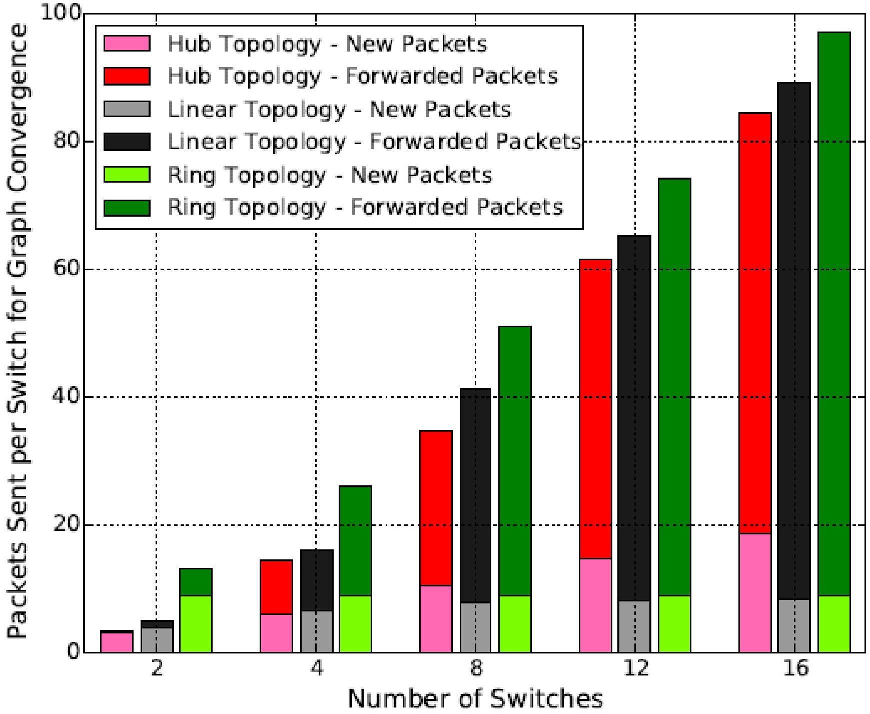
- Traffic at equilibrium

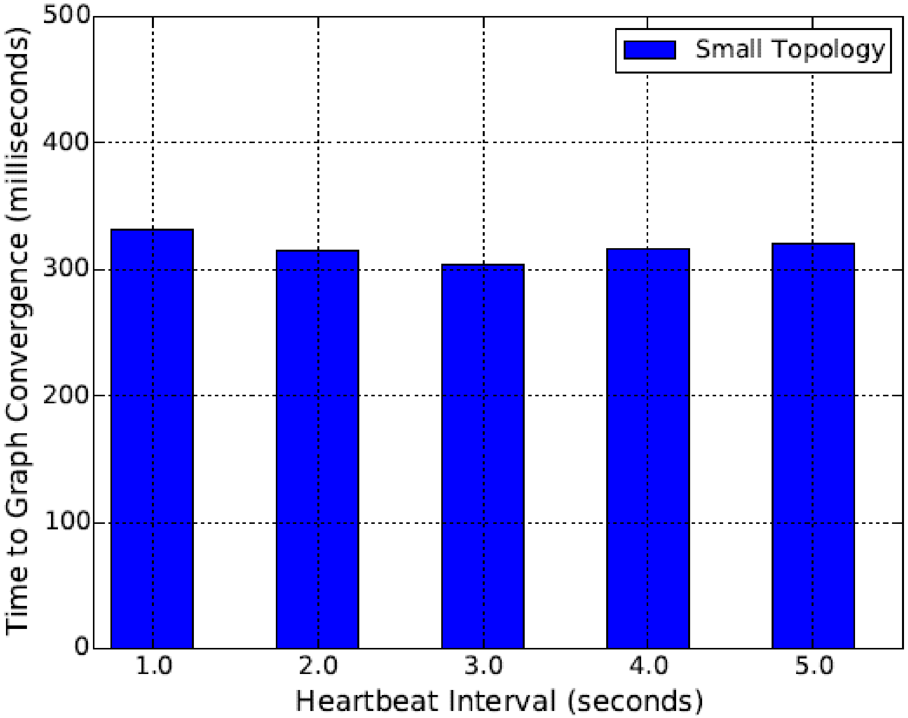




Protocol Efficiency Benchmarks

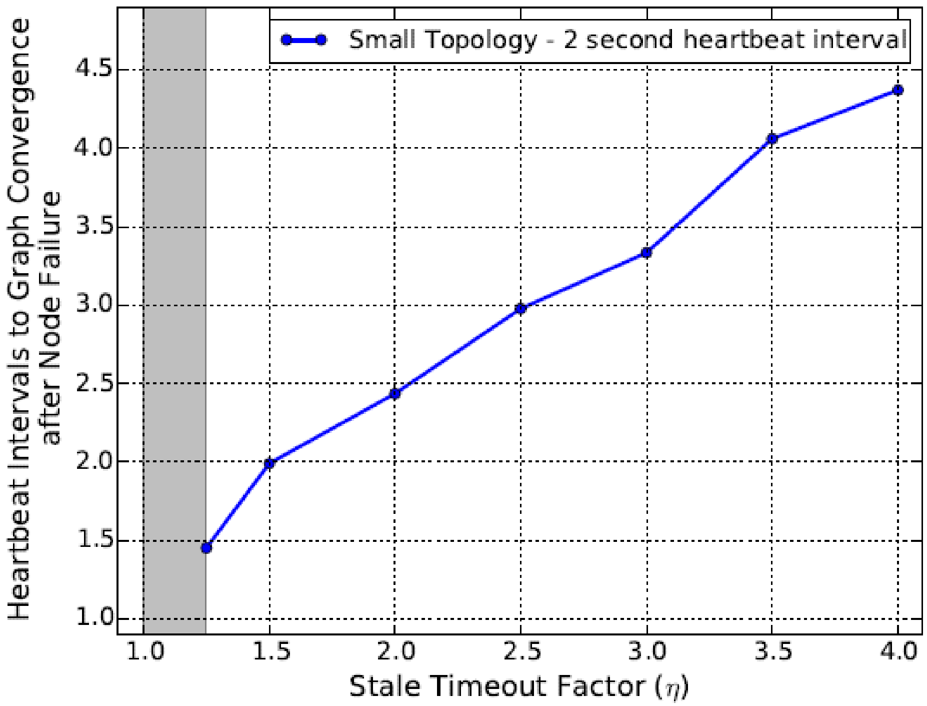
- Traffic and time to topology graph convergence

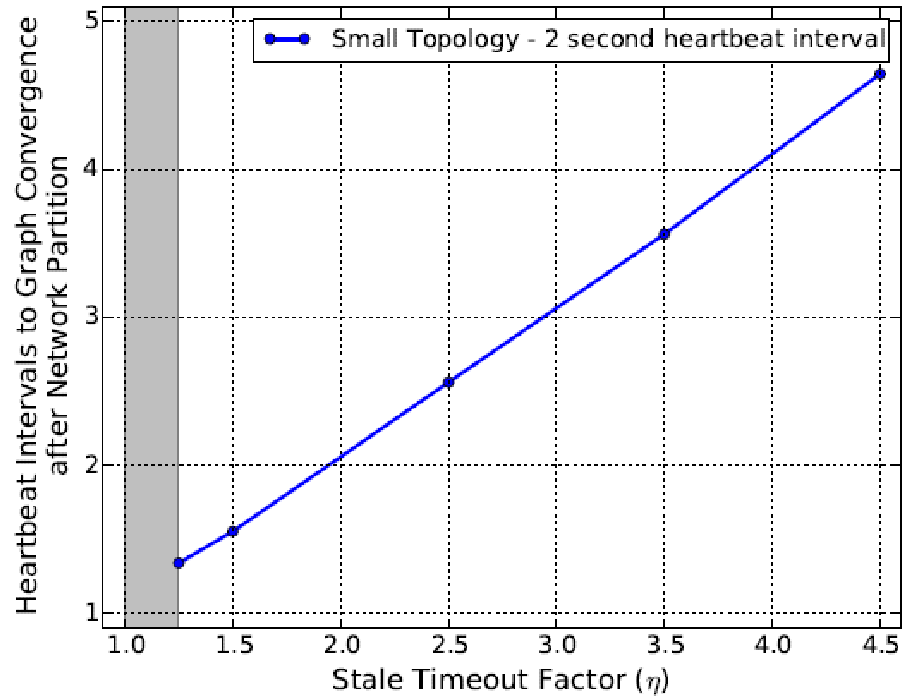




Protocol Efficiency Benchmarks

- Node failure and partition response rates





Thanks for listening!

- Questions?

pj97

Fast response scheduling with machine partitioning

Soroush Alamdari

Pooya Jalaly

Fast response scheduling

Fast response scheduling

- Distributed schedulers

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second
- No time to waste
  - Assign the next job to a random machine.

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second
- No time to waste
  - Assign the next job to a random machine.
- Two choice method
  - Choose two random machines
  - Assign the job to the machine with smaller load.

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second
- No time to waste
  - Assign the next job to a random machine.
- Two choice method
  - Choose two random machines
  - Assign the job to the machine with smaller load.
- Two choice method works exponentially better than random assignment.

Partitioning

Partitioning

- Partitioning the machines among the schedulers

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks
- Allows for locality respecting assignment
  - Smaller communication time, faster decision making.

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks
- Allows for locality respecting assignment
  - Smaller communication time, faster decision making.
- Irregular patterns of incoming jobs
  - Soft partitioning

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks
- Allows for locality respecting assignment
  - Smaller communication time, faster decision making.
- Irregular patterns of incoming jobs
  - Soft partitioning
- Modified two choice model
  - Probe a machine from within, one from outside

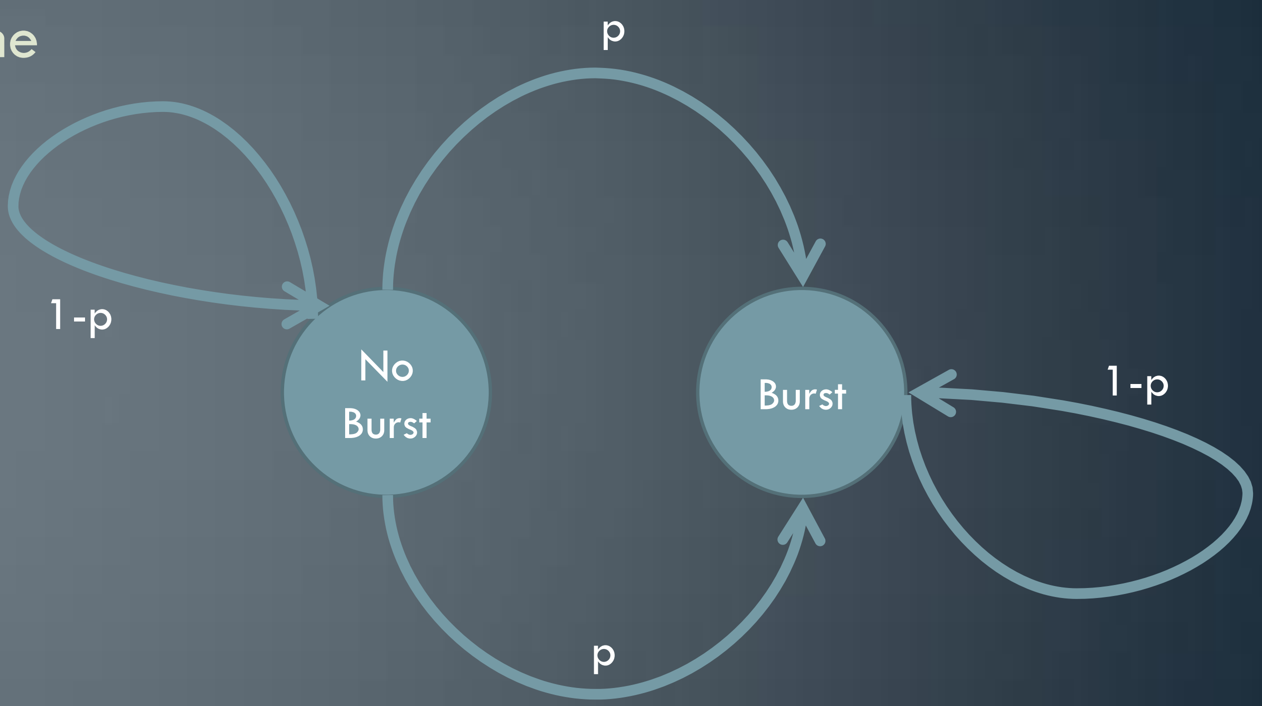
Testing

Testing

- Simulated timeline

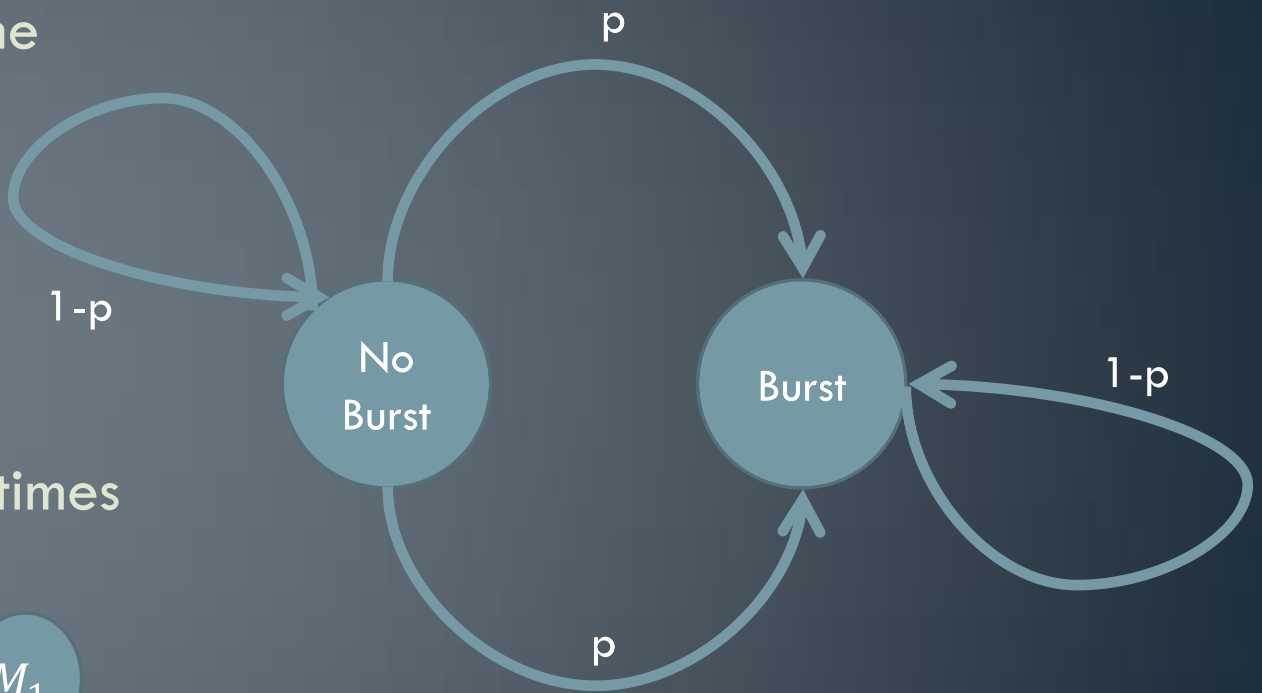
Testing

- Simulated timeline
- Burst of tasks

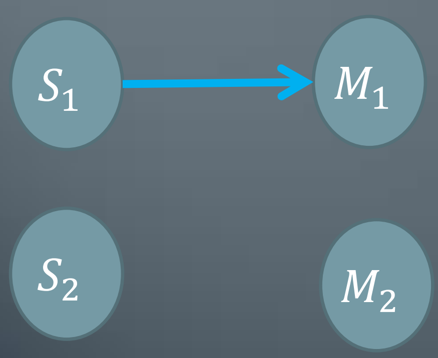


Testing

- Simulated timeline
- Burst of tasks

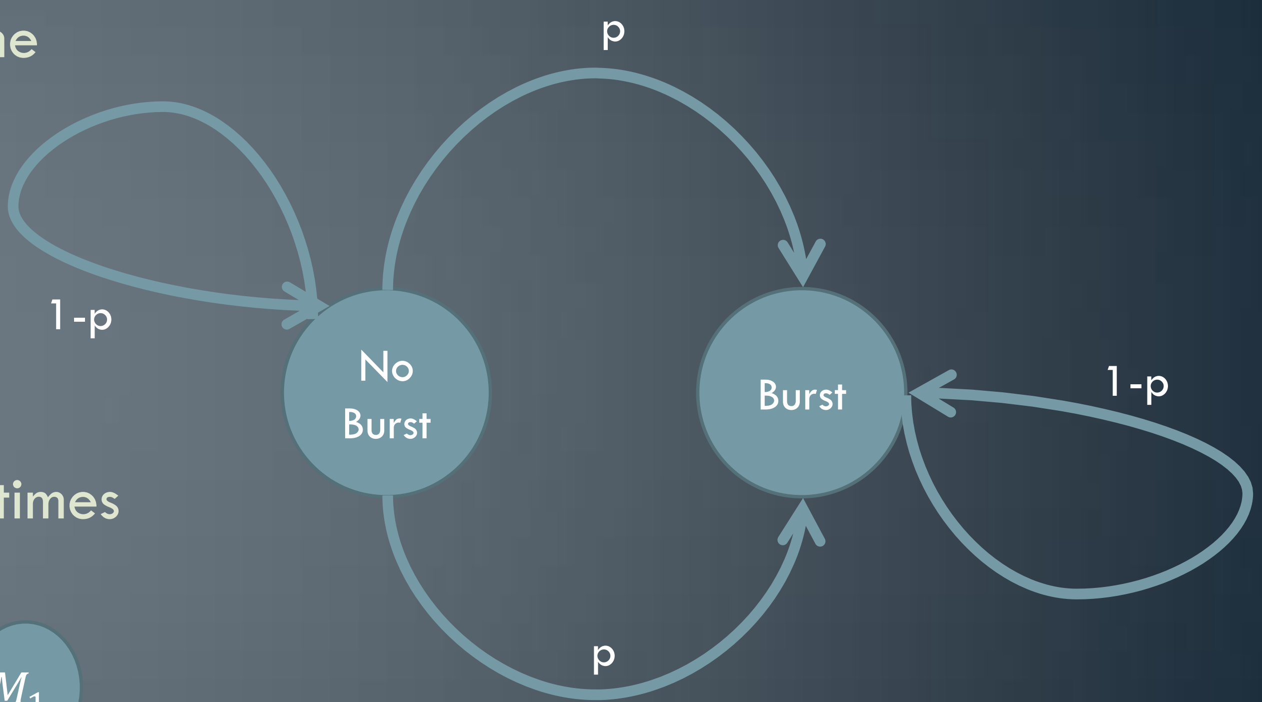


- Metric response times

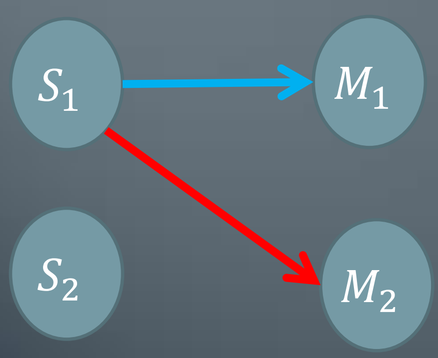


Testing

- Simulated timeline
- Burst of tasks

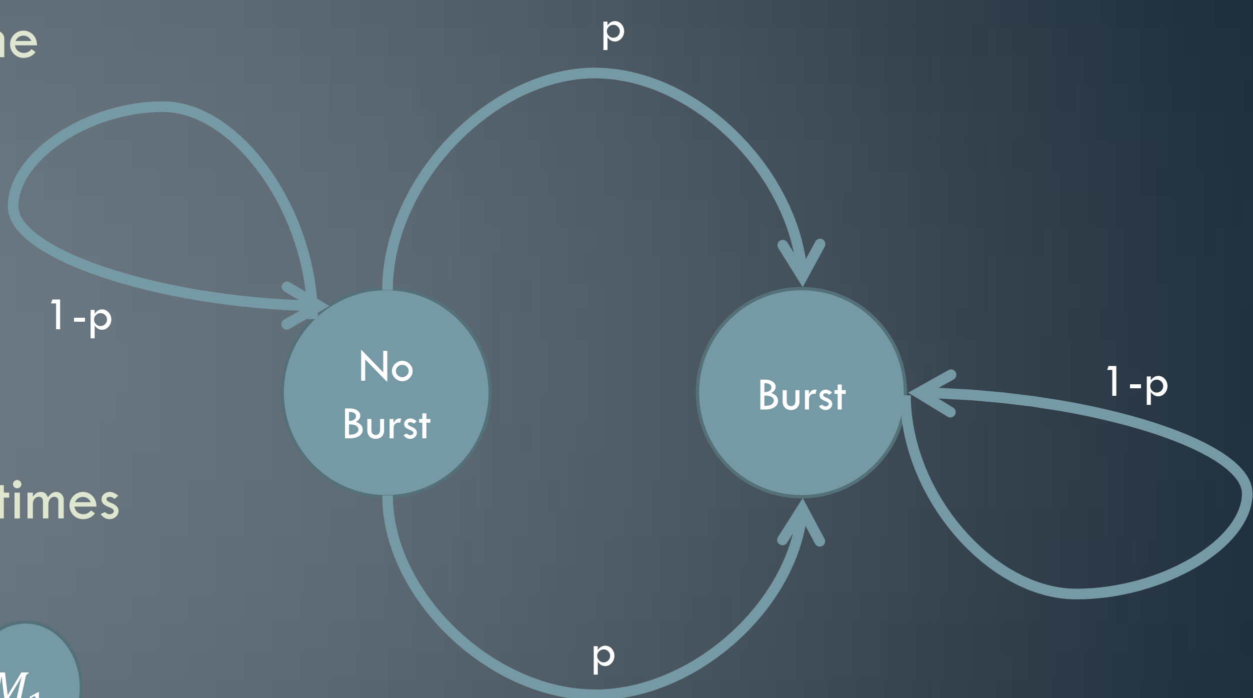


- Metric response times

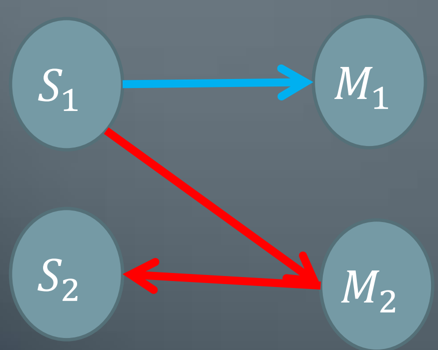


Testing

- Simulated timeline
- Burst of tasks

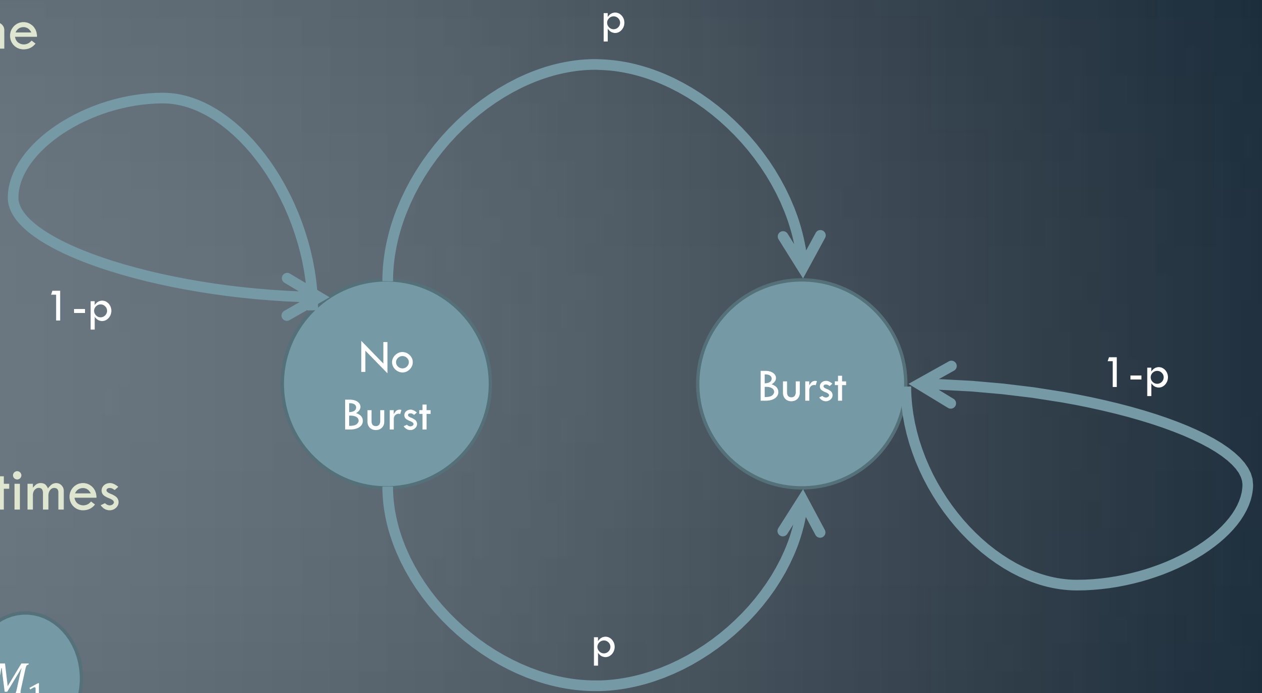


- Metric response times

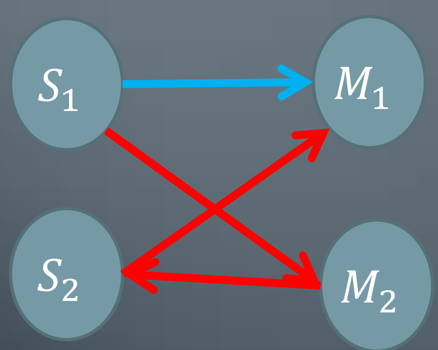


Testing

- Simulated timeline
- Burst of tasks



- Metric response times



pk467

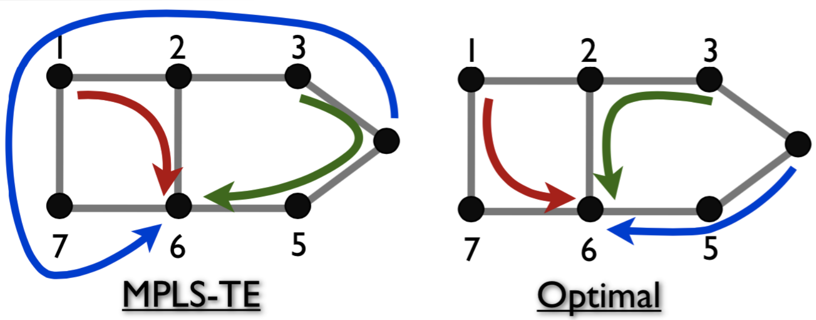
Software-Defined Routing for Inter-Datacenter Wide Area Networks

Praveen Kumar

Problems

1. Inter-DC WANs are critical and highly expensive
2. Poor efficiency - average utilization over time of busy links is only 30-50%
3. Poor sharing - little support for flexible resource sharing

MPLS Example: Flow arrival order: A, B, C; each link can carry at most one flow



* Make smarter routing decisions - considering the link capacities and flow demands

Source: Achieving High Utilization with Software-Driven WAN, SIGCOMM 2013

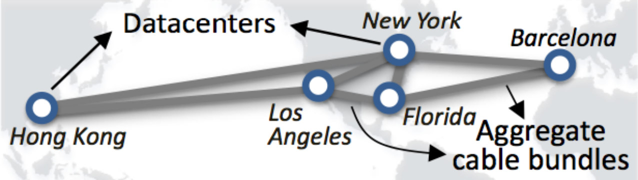
Merlin: Software-Defined Routing

- Merlin Controller
  - MCF solver
  - RRT generation

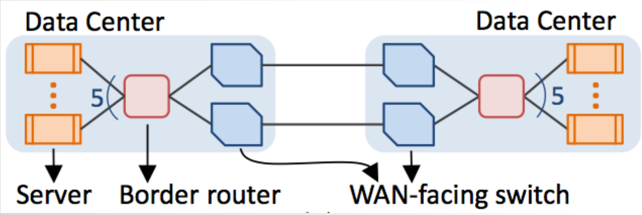
- Merlin Virtual Switch (MVS) - A modular software switch
  - Merlin
    - Path: ordered list of pathlets (VLANs)
    - Randomized source routing
    - Push stack of VLANs
  - Flow tracking
  - Network function modules - pluggable
  - Compose complex network functions from primitives

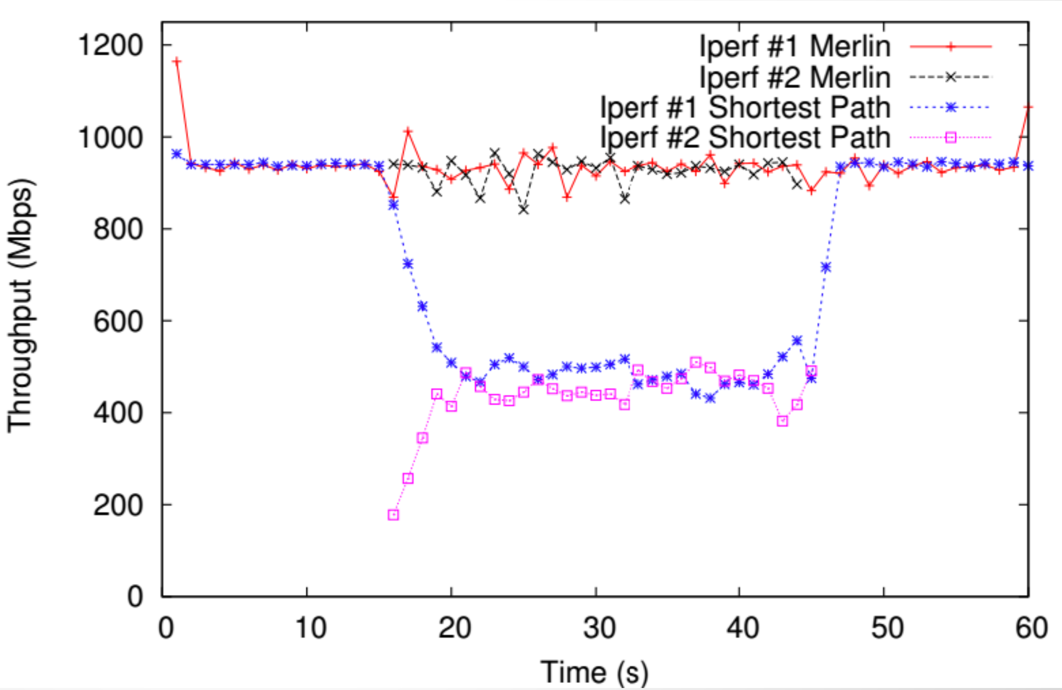
Some results

No VLAN Stack	Open vSwitch	CPqD ofsoftswitch13	MVS
941 Mbps	0 (N/A)	98 Mbps	925 Mbps



SWAN topology *





- Source: Achieving High Utilization with Software-Driven WAN, SIGCOMM 2013 rmo26

AMNESIA-FREEDOM AND EPHEMERAL DATA

CS6410 December 4, 2014

Ephemeral Data

- “Overcoming CAP” describes using soft-state replication to keep application state in the first-tier of the cloud.
- Beyond potential performance advantages, this architecture may be the basis for “ephemerality” wherein data is intended to disappear.
- “Subpoena-freedom”
- No need to wipe disks, just restart your instances.

Cost and Architecture

- “Overcoming CAP” does not address questions of cost.
- Using reliable storage to preserve state has significant cost consequences.
- First goal of this project is to produce a model of the cost with cloud architecture choices.
- Key cost drivers: compute hours, data movement, storage.

Performance Numbers

- "Overcoming CAP" claims but does not demonstrate superior performance with the amnesia-free approach.
- Second goal of this project is to compare performance in live systems.
- A cost-determined amnesia-free architecture compared against architectures that rely on reliable storage.

sh954

Fast response scheduling with machine partitioning

Soroush Alamdari

Pooya Jalaly

Fast response scheduling

Fast response scheduling

- Distributed schedulers

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second
- No time to waste
  - Assign the next job to a random machine.

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second
- No time to waste
  - Assign the next job to a random machine.
- Two choice method
  - Choose two random machines
  - Assign the job to the machine with smaller load.

Fast response scheduling

- Distributed schedulers
- E.g. 10,000 16-core machines, 100ms average processing times
  - A million decisions per second
- No time to waste
  - Assign the next job to a random machine.
- Two choice method
  - Choose two random machines
  - Assign the job to the machine with smaller load.
- Two choice method works exponentially better than random assignment.

Partitioning

Partitioning

- Partitioning the machines among the schedulers

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks
- Allows for locality respecting assignment
  - Smaller communication time, faster decision making.

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks
- Allows for locality respecting assignment
  - Smaller communication time, faster decision making.
- Irregular patterns of incoming jobs
  - Soft partitioning

Partitioning

- Partitioning the machines among the schedulers
- Reduces expected maximum latency
  - Assuming known rates of incoming tasks
- Allows for locality respecting assignment
  - Smaller communication time, faster decision making.
- Irregular patterns of incoming jobs
  - Soft partitioning
- Modified two choice model
  - Probe a machine from within, one from outside

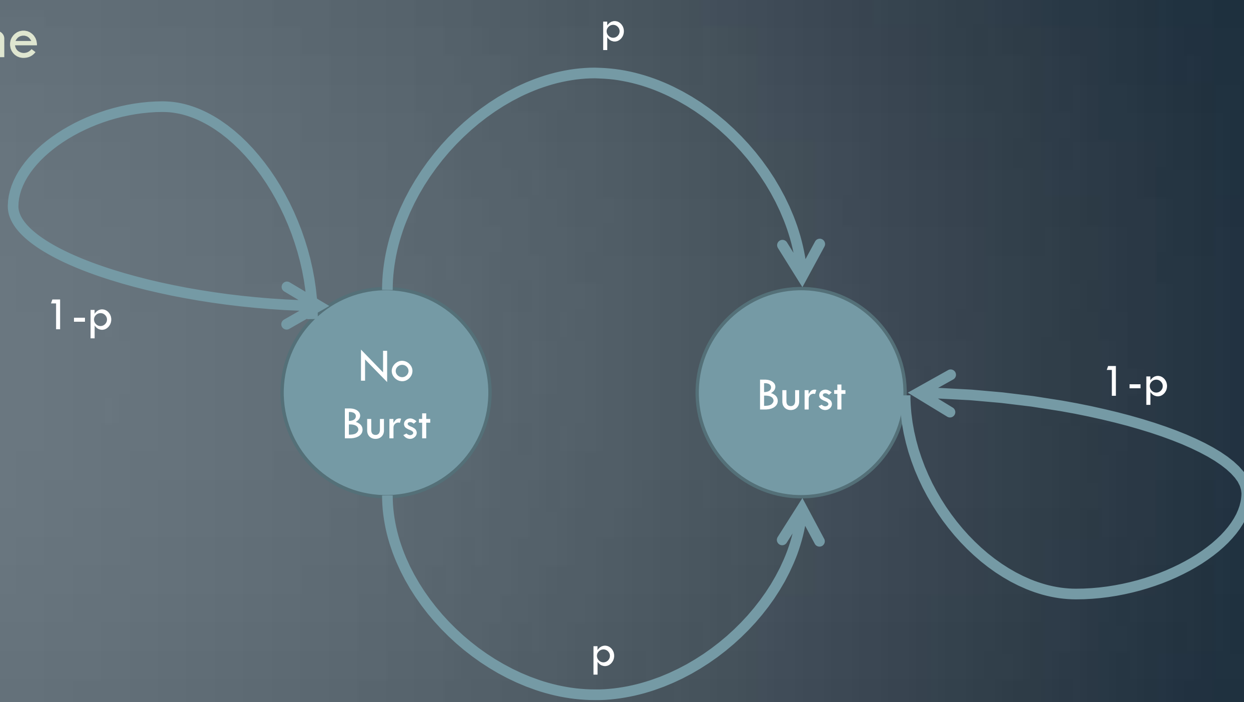
Testing

Testing

- Simulated timeline

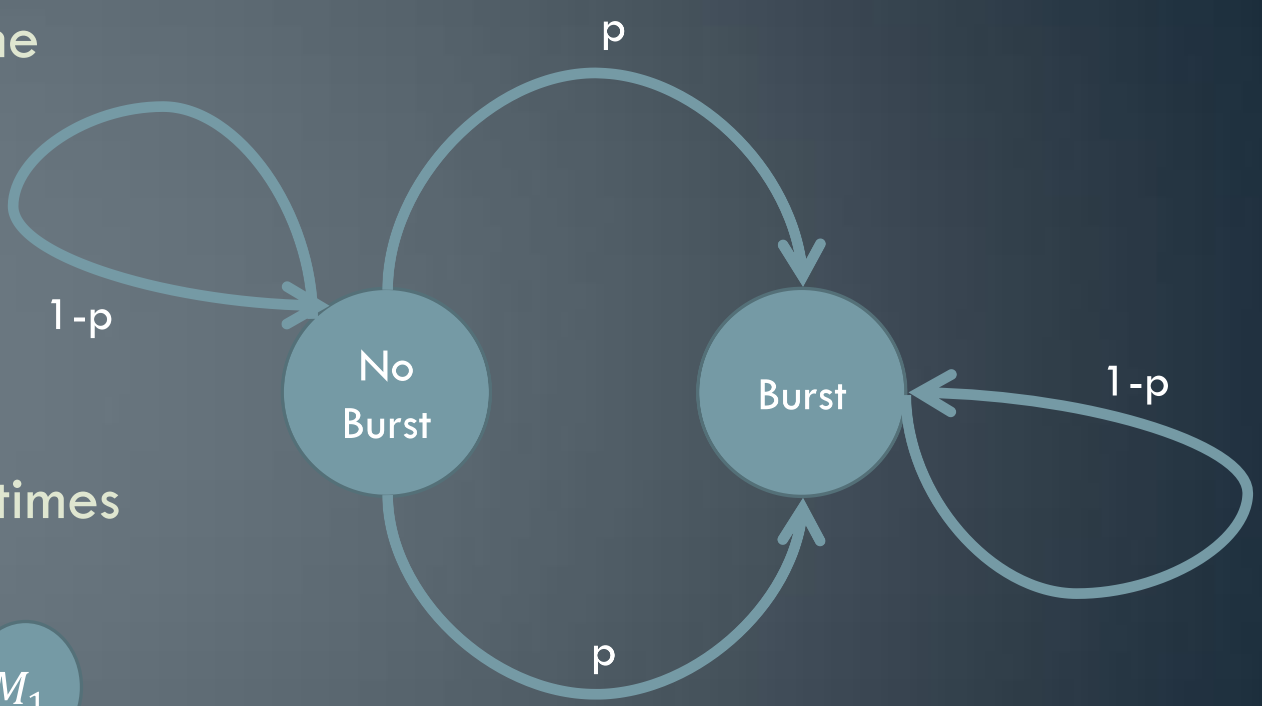
Testing

- Simulated timeline
- Burst of tasks

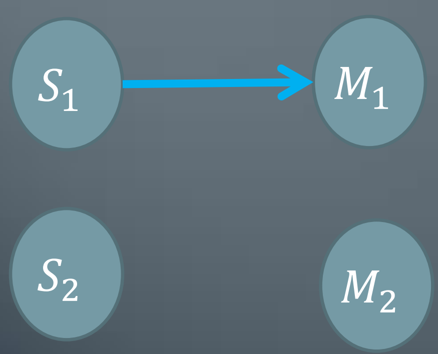


Testing

- Simulated timeline
- Burst of tasks

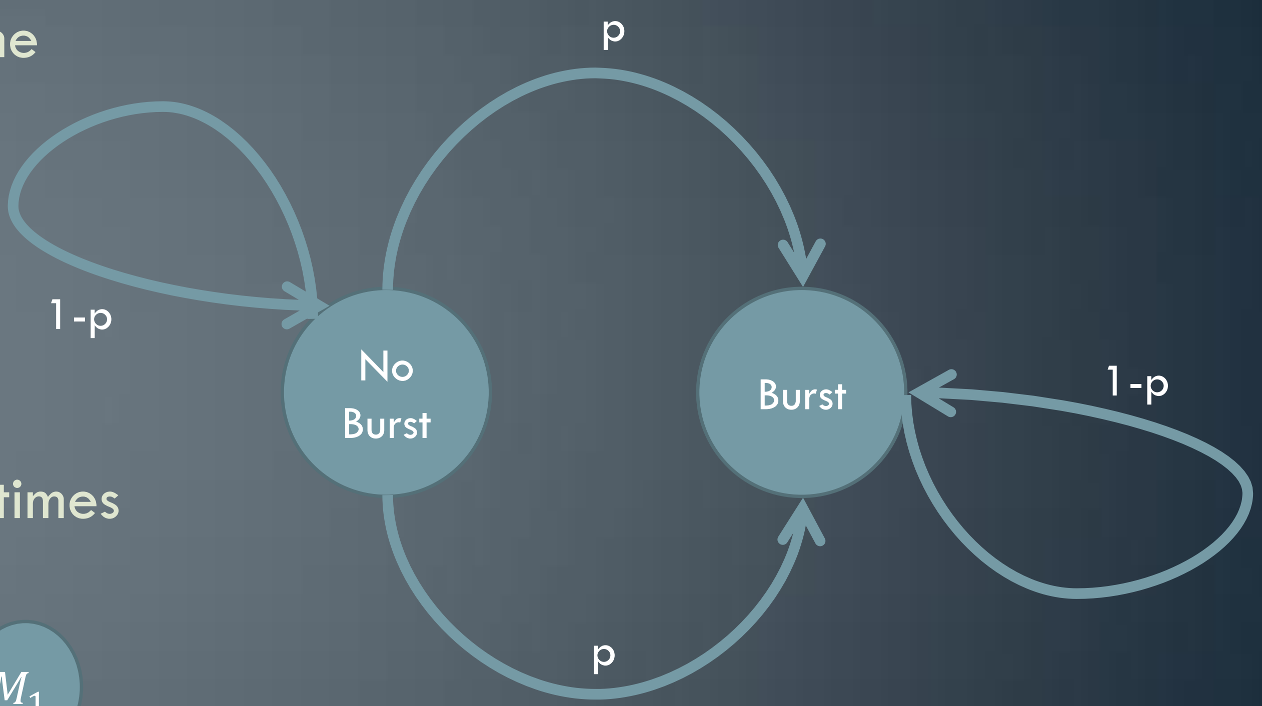


- Metric response times

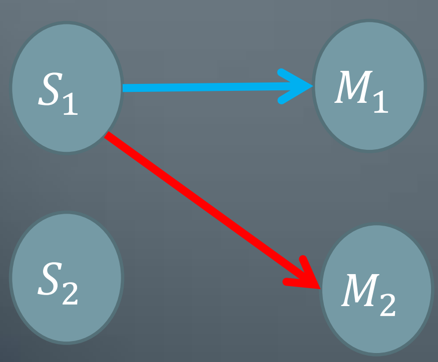


Testing

- Simulated timeline
- Burst of tasks

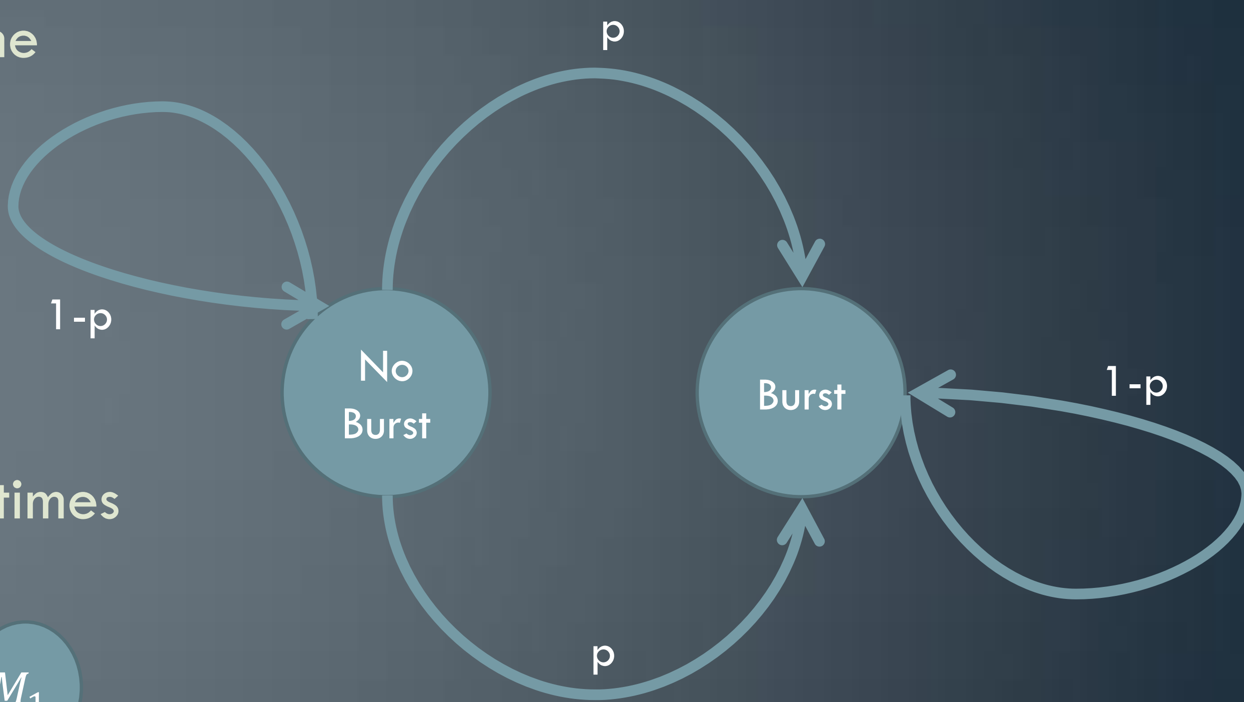


- Metric response times

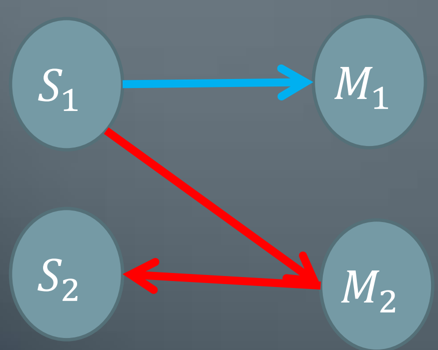


Testing

- Simulated timeline
- Burst of tasks

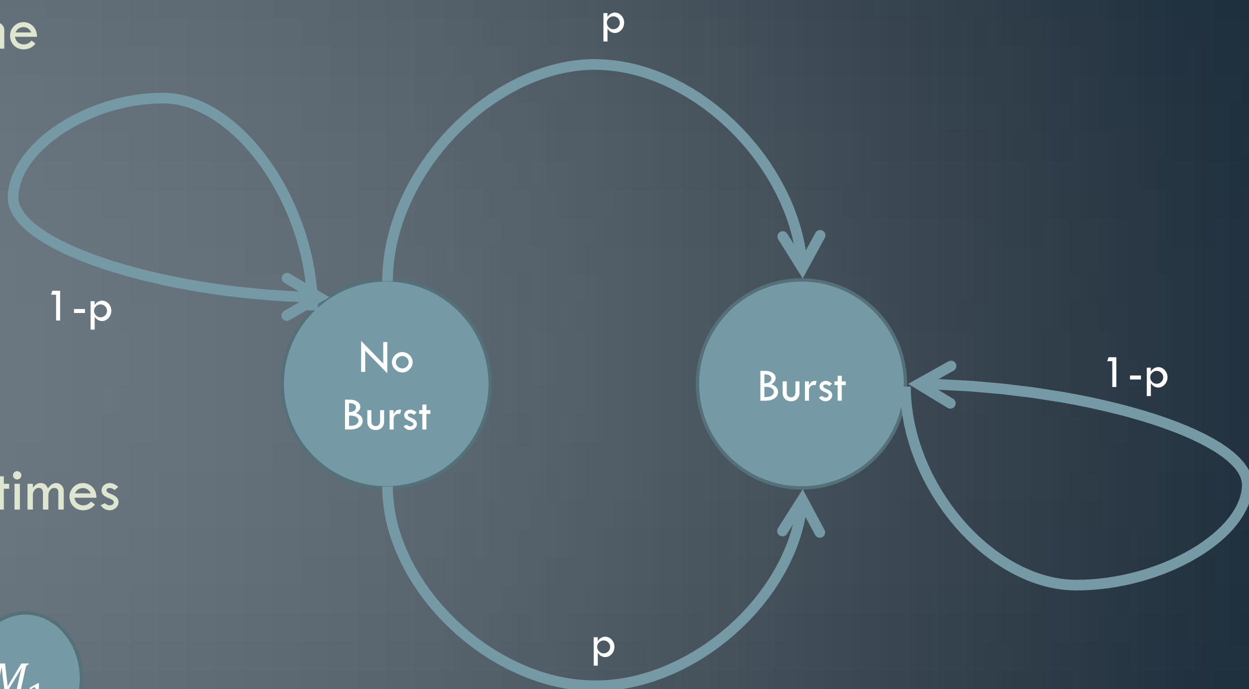


- Metric response times

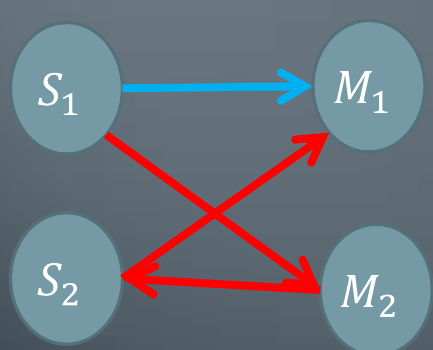


Testing

- Simulated timeline
- Burst of tasks



- Metric response times



vdk23

FPGA Packet Processor

For IronStack

Vasily Kuksenkov

Problem

- Power grid operators use an intricate feedback system for stability
- Run using microwave relays and power cable signal multiplexing
- Data network issues
  - Vulnerable to attacks
  - Vulnerable to disruptions
  - Low capacity links
- Solution: switch to simple Ethernet

Problem

- Ethernet employs a loop-free topology
  - Hard to use link redundancies
  - Failure recovery takes too long
- Solution: IronStack SDN
  - Uses redundant network paths to improve
    - Bandwidth/Latency
    - Failure Recovery
    - Security

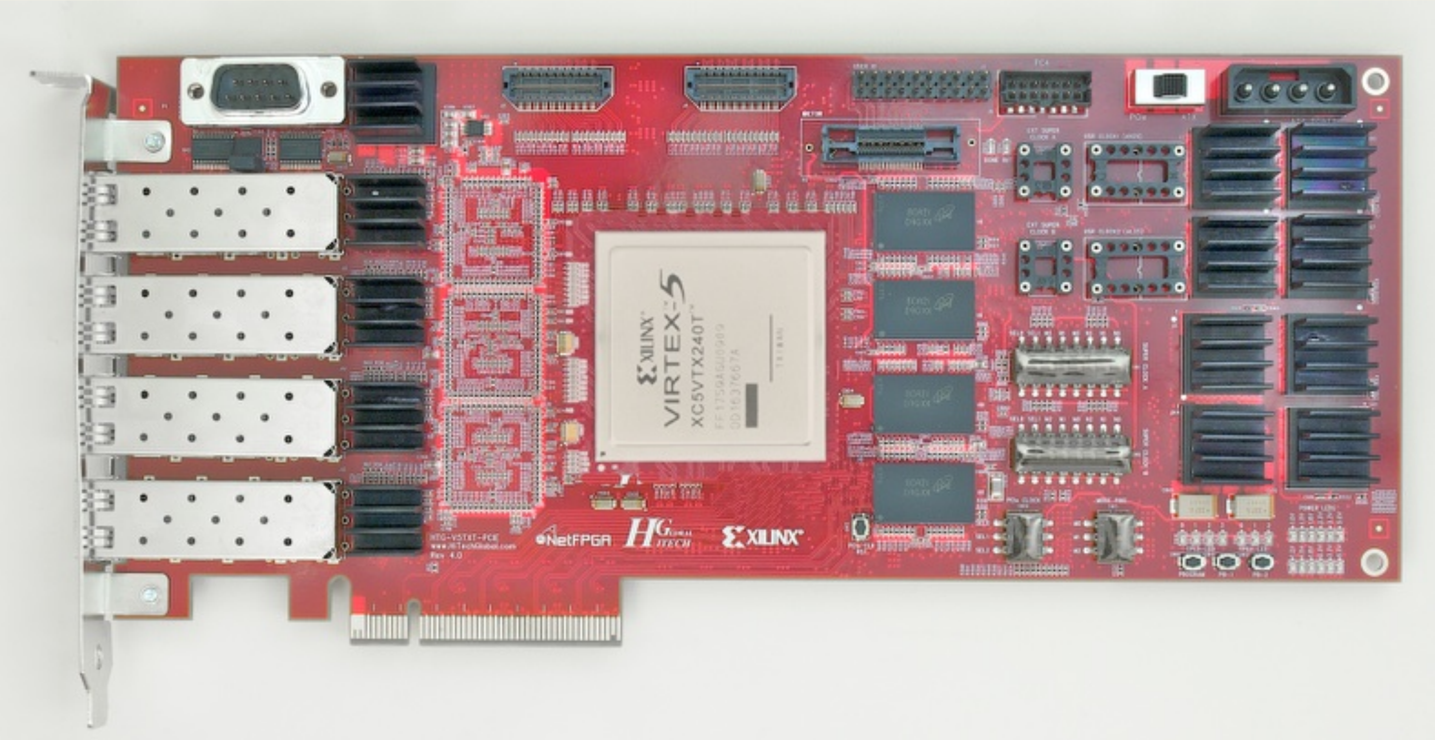
Problem

- Packet Processing
  - Cannot be done on the switch
  - Cannot be done at line rate (1-10Gbps) on the controller
- Solution: NetFPGA as a middle-man
  - Controller sets up routing rules and signals NetFPGA
  - Programmed once, continues to work
  - Scalable, efficient, cost-effective

Implementation/Analysis

- Improvements in
  - Bandwidth (RAIL 0)
  - Latency (RAIL 1)
  - Tradeoffs (RAIL 6)
- Future
  - Security
  - Automatic tuning

Questions?



vs442

Studying the effect of traffic pacing on TCP throughput

Vishal Shrivastav

Dec 4, 2014

Motivation

Burstiness: clustering of packets on the wire
Pacing: making the inter packet gaps uniform

Motivation

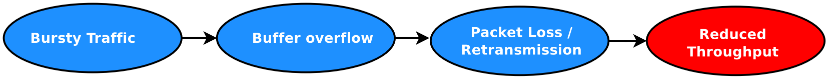
Burstiness: clustering of packets on the wire
Pacing: making the inter packet gaps uniform

TCP traffic tends to be inherently bursty

Motivation

Burstiness: clustering of packets on the wire
Pacing: making the inter packet gaps uniform

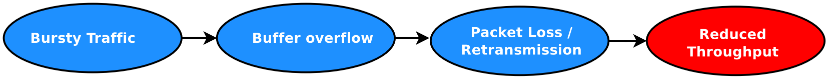
TCP traffic tends to be inherently bursty



Motivation

Burstiness: clustering of packets on the wire
Pacing: making the inter packet gaps uniform

TCP traffic tends to be inherently bursty



Other potential benefits of pacing

- Better short-term fairness among flows of similar RTTs
- May allow much larger initial congestion window to be used safely

Previous works

Focused on implementing pacing at the transport layer

Some major limitations of that approach

- Less precision - Not very fine granular control of the flow
- NIC features like TCP segment offload lead to batching and short-term packet bursts

Key Insight

Implement pacing at the PHY layer

Key Insight

Implement pacing at the PHY layer

Problem: commodity NICs do not provide software access to PHY layer

Key Insight

Implement pacing at the PHY layer

Problem: commodity NICs do not provide software access to PHY layer

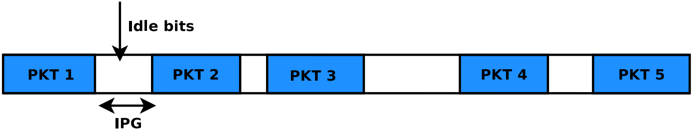
Solution: SoNIC [NSDI 2013]

Key Insight

Implement pacing at the PHY layer

Problem: commodity NICs do not provide software access to PHY layer

Solution: SoNIC [NSDI 2013]

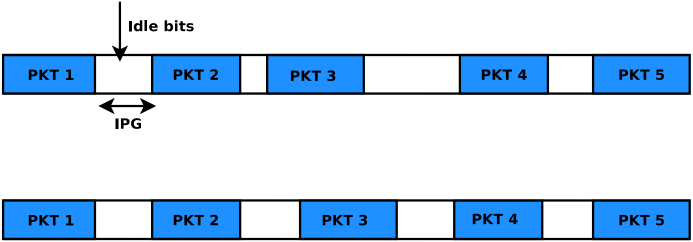


Key Insight

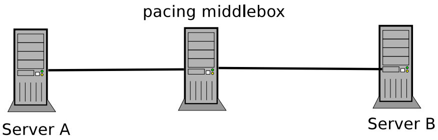
Implement pacing at the PHY layer

Problem: commodity NICs do not provide software access to PHY layer

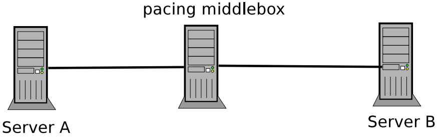
Solution: SoNIC [NSDI 2013]



Implementation Challenges

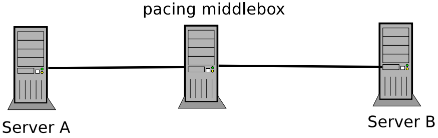


Implementation Challenges



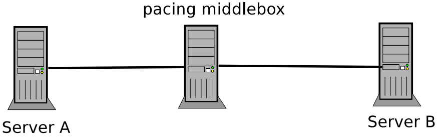
- Online Algorithm - No batching, one packet at a time

Implementation Challenges



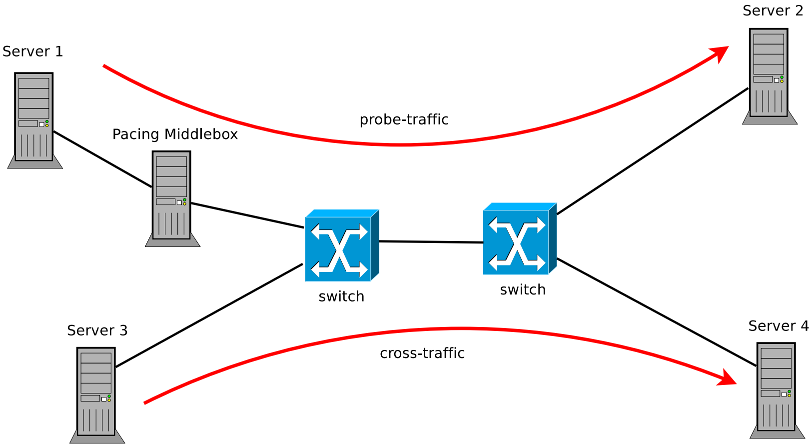
- Online Algorithm - No batching, one packet at a time
- Very small packet processing time - simple algorithm, extremely fast implementation

Implementation Challenges

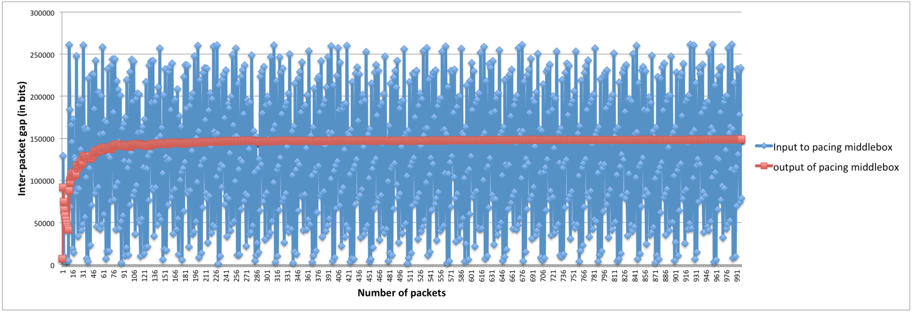


- Online Algorithm - No batching, one packet at a time
- Very small packet processing time - simple algorithm, extremely fast implementation
- Where to place pacing middleboxes in the network
  - Given a maximum of k pacing middleboxes, where should we place them in the network to achieve optimal throughput?

Network topology for experiments



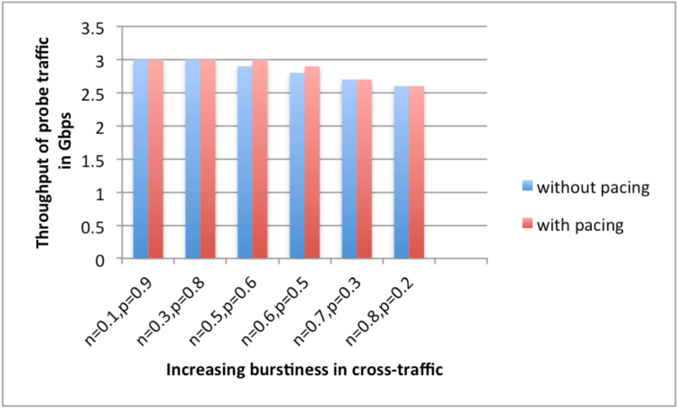
Testing the behavior of pacing algorithm



Experimental results

n : a value within [0,1], parameter for number of pkt bursts in a flow.

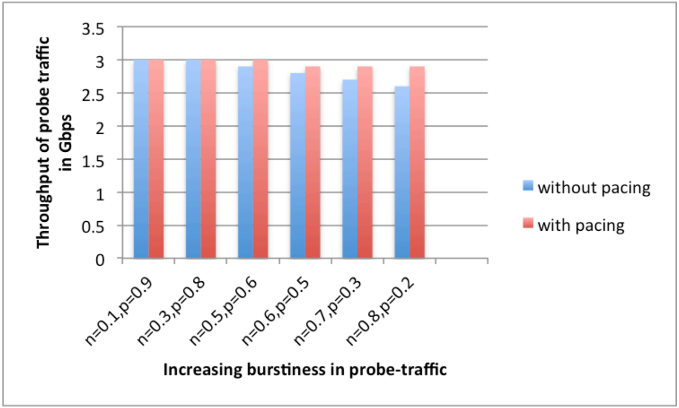
p : a value within [0,1], parameter for the geometric dist. used to generate the number of packets within a pkt burst.



Experimental results

n : a value within [0,1], parameter for number of pkt bursts in a flow.

p : a value within [0,1], parameter for the geometric dist. used to generate the number of packets within a pkt burst.




Thank You