CS 514: Transport Protocols for Datacenters

Mahesh Balakrishnan

Department of Computer Science
Cornell University

Outline

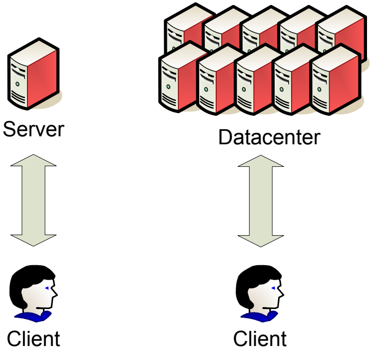
Commodity Datacenters

- Blade-servers, Fast Interconnects
- Different Apps:
  - Google -> Search
  - Amazon -> Etailing
  - Computational Finance, Aerospace, Military C&C, e-Science...
  - ... YouTube?

The Datacenter Paradigm

Extreme Scale-out

- More Nodes, More Capacity
- Services distributed / replicated / partitioned over multiple nodes



Building Datacenter Apps

- High-level Abstractions:
  - Publish/Subscribe
  - Event Notification
  - Replication (Data/Functionality)
  - Caching
- BEA Weblogic, JBoss, IBM Websphere, Tibco, RTI DDS, Tangosol, Gemfire...
- What's under the hood?
  Multicast!

Properties of a Multicast Primitive

Rapid Delivery

- ... when failures occur (reliable)
- ... at extreme scales (scalable)

Questions:

- What technology do current systems use?
- Is it truly 'reliable' and 'scalable'?
- Can we do better?

A Brief History of Multicast

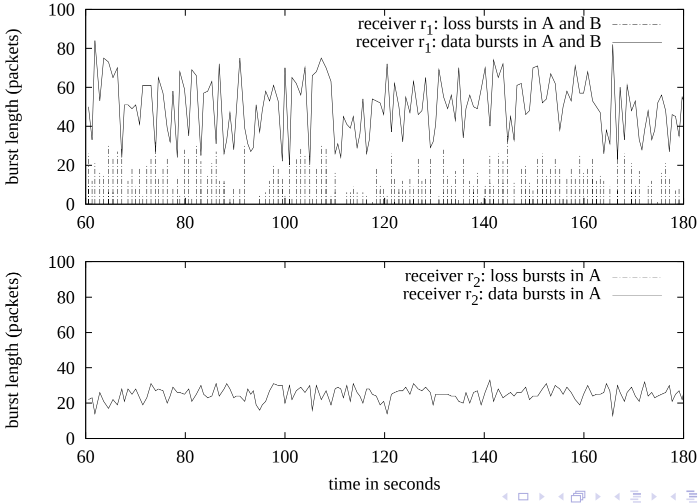
- IP Multicast - Deering, et al., 1988.
  Limited Deployment — the Mbone.
- Two Divergent Directions:
  - Overlay Multicast *instead of* IP Multicast (e.g, BitTorrent)
  - Reliable Multicast *over* IP Multicast (e.g, TIBCO)
- Datacenters have IP Multicast support...

Multicast Research

- Many different *reliable*, *scalable* protocols
- Designed for streaming video/TV, file distribution
- Reliable:
  - Packet Loss at WAN routers
- Scalable:
  - Single group with massive numbers of receivers
- Not suited for datacenter multicast!
  - Different failure mode
  - Different scalability dimensions

(Reliable) Multicast in the Datacenter

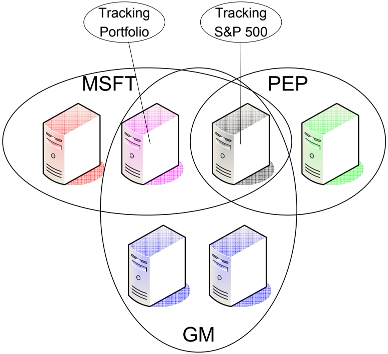
Packet Loss occurs at end-hosts: independent and bursty



(Scalable) Multicast in the Datacenter

Financial Datacenter Example:

- Each equity is mapped to a multicast group.
- Each Node is interested in a different set of equities...
- ... each Node joins a different set of groups.



Lots of overlapping groups $\Longrightarrow$ Low per-group data rate.

Designing a Time-Critical Multicast Primitive

- Wanted: A *reliable*, *scalable* multicast protocol.
- Reliable:
  - can tolerate end-host loss bursts
- Scalable:
  - the size of the group
  - the number of senders to a group
  - the number of groups per node

Designing a Time-Critical Multicast Primitive

- Wanted: A *reliable*, *scalable* multicast protocol.
- Reliable:
  - can tolerate end-host loss bursts
- Scalable:
  - the size of the group
  - the number of senders to a group
  - the number of groups per node

Designing a Time-Critical Multicast Primitive

- Wanted: A *reliable*, *scalable* multicast protocol.
- Reliable:
  - can tolerate end-host loss bursts
- Scalable:
  - the size of the group
  - the number of senders to a group
  - the number of groups per node

Outline

1. Motivation
2. Systems
   - Ricochet
   - Maelstrom
3. Conclusion

Design Space for Reliable Multicast

How does latency scale?

Two Phases: *Discovery* and *Recovery* of Lost Packets

- ACK/timeout: RMTP/RMTP-II
- Gossip-based: Bimodal Multicast, lpbcast
- NAK/sender-based sequencing: SRM
- Forward Error Correction

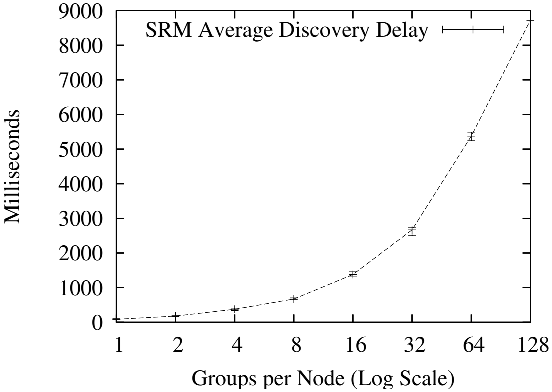
Fundamental Insight: $latency \alpha \frac{1}{datarate}$

NAK/Sender-Based Sequencing: SRM

Scalable Reliable Multicast - Developed 1998

- Loss *discovered* on next packet from same sender in same group
- *latency* $\alpha \frac{1}{datarate}$

data rate: at a single sender, in a single group



Forward Error Correction

Pros:

- Tunable, Proactive Overhead
- *Time-Critical*: Eliminates need for retransmission

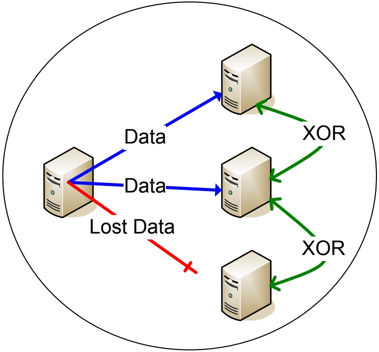
Cons:

- FEC packets are generated over a stream of data
  - Have to wait for *r* data packets before generating FEC
  - $latency \alpha \frac{1}{datarate}$

data rate: at a single sender, in a single group

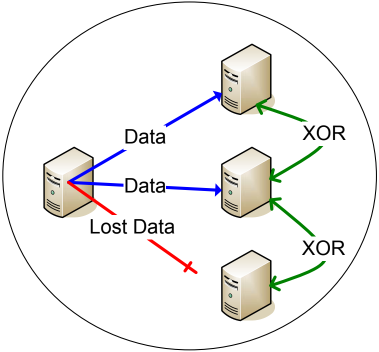
Receiver-Based Forward Error Correction

- Randomness: Each Receiver picks another Receiver randomly to send XOR to
- Tunability: Percentage of XOR packets to data is determined by *rate-of-fire* (r, c)
- *latency* $\alpha \frac{1}{datarate}$
  data rate: across all senders, in a single group



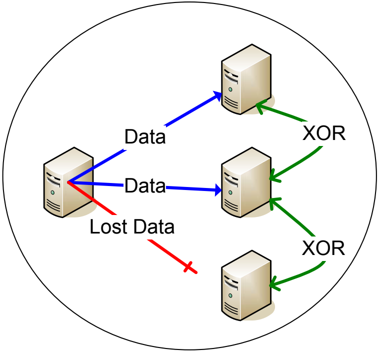
Receiver-Based Forward Error Correction

- Randomness: Each Receiver picks another Receiver randomly to send XOR to
- Tunability: Percentage of XOR packets to data is determined by *rate-of-fire* (r, c)
- $latency \alpha \frac{1}{datarate}$
  data rate: across all senders, in a single group



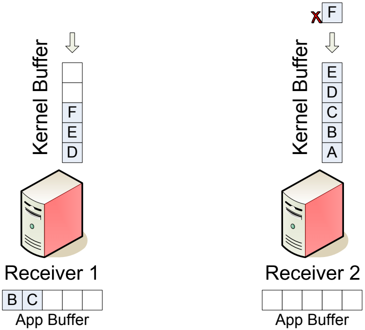
Receiver-Based Forward Error Correction

- Randomness: Each Receiver picks another Receiver randomly to send XOR to
- Tunability: Percentage of XOR packets to data is determined by *rate-of-fire* (r, c)
- $latency \alpha \frac{1}{datarate}$
  data rate: across all senders, in a single group



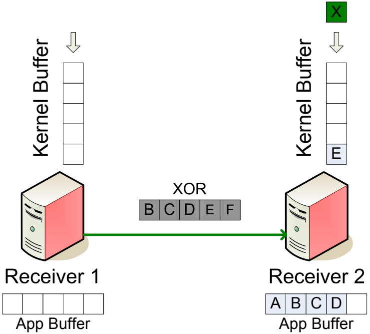
Receiver-Based Forward Error Correction

- Randomness: Each Receiver picks another Receiver randomly to send XOR to
- Tunability: Percentage of XOR packets to data is determined by *rate-of-fire* (r, c)
- *latency* $\alpha \frac{1}{datarate}$
  data rate: across all senders, in a single group



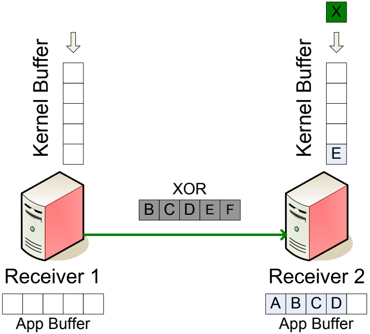
Receiver-Based Forward Error Correction

- Randomness: Each Receiver picks another Receiver randomly to send XOR to
- Tunability: Percentage of XOR packets to data is determined by *rate-of-fire* (r, c)
- *latency* $\alpha \frac{1}{datarate}$
  data rate: across all senders, in a single group



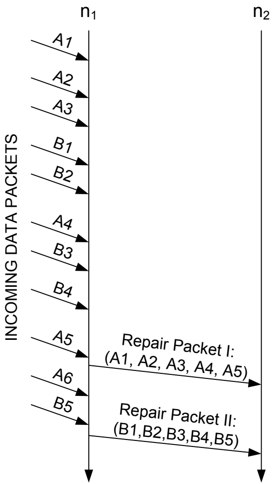
Receiver-Based Forward Error Correction

- Randomness: Each Receiver picks another Receiver randomly to send XOR to
- Tunability: Percentage of XOR packets to data is determined by *rate-of-fire* (r, c)
- $latency \alpha \frac{1}{datarate}$
  data rate: across all senders, in a single group



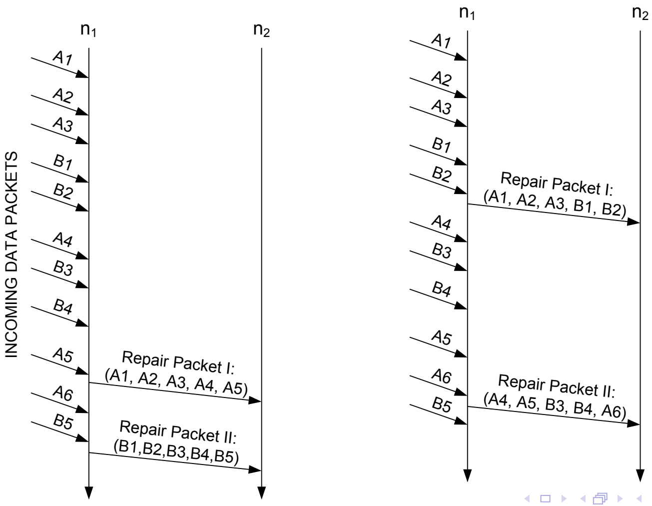
Lateral Error Correction: Principle

Nodes n_1 and n_2 are both in groups A and B.



Lateral Error Correction: Principle

Nodes n_1 and n_2 are both in groups A and B.



Lateral Error Correction

Combine error traffic for multiple groups within intersections, while conserving:

- Coherent, tunable per-group overhead: Ratio of data packets to repair packets in the system is $r : c$
- Fairness: Each node receives on average the same ratio of repair packets to data packets
- $latency \alpha \frac{1}{datarate}$
  data rate: across all senders, in intersections of groups

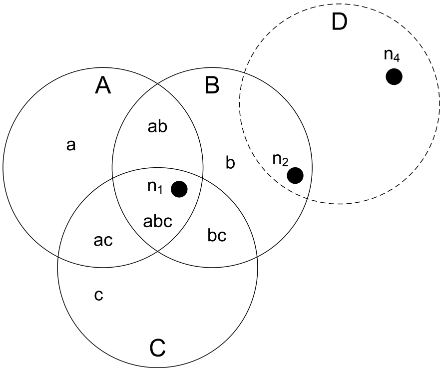
Lateral Error Correction

Combine error traffic for multiple groups within intersections, while conserving:

- Coherent, tunable per-group overhead: Ratio of data packets to repair packets in the system is $r : c$
- Fairness: Each node receives on average the same ratio of repair packets to data packets
- $latency \alpha \frac{1}{datarate}$
  data rate: across all senders, in intersections of groups

Lateral Error Correction: Mechanism

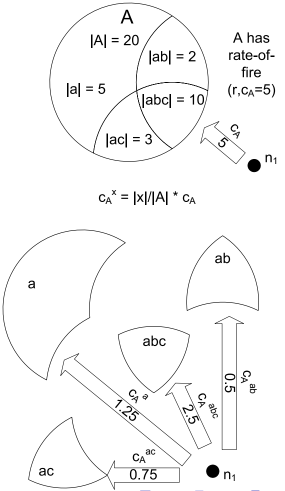
Divide overlapping groups into *regions*



n_1 belongs to groups A, B, C:
It divides them into regions abc, ab, ac, bc, a, b, c

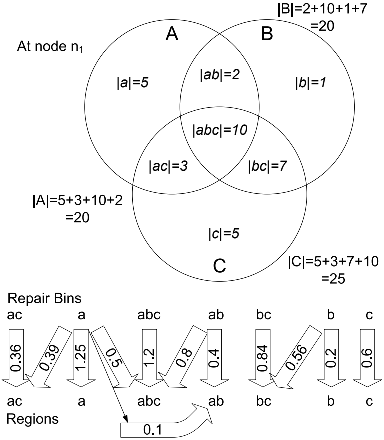
Lateral Error Correction: Mechanism

- n_1 selects proportionally sized chunks of c_A from the regions of A
- Total number of targets selected, across regions, is equal to the c value of a group



Repair Bin Structure

- Repair Bins:
- Input: Data Packets in *union* of Groups
- Output: Repair Packets to *region*
- Expectation: Avg # of targets chosen from region

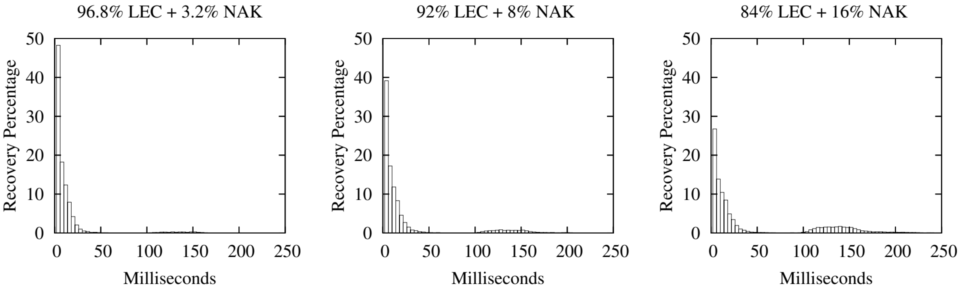


Experimental Evaluation

- Cornell Cluster: 64 1.3 Ghz nodes
- Java Implementation running on Linux 2.6.12
- Three Loss Models: {Uniform, Burst, Markov}
- Grouping Parameters: $g * s = d * n$
  - g: Number of Groups in System
  - s: Average Size of Group
  - d: Groups joined by each Node
  - n: Number of Nodes in System
- Each node joins *d* randomly selected groups from *g* groups

Distribution of Recovery Latency

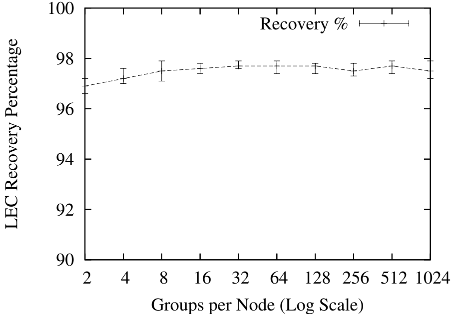
16 Nodes, 128 groups per node, 10 nodes per group, Uniform 1% Loss

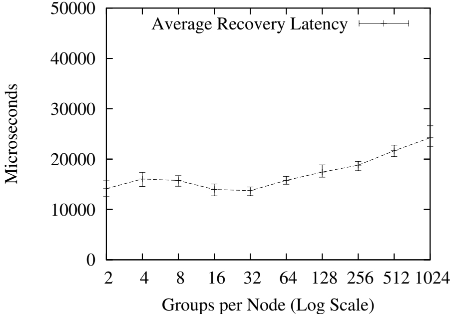


(a) 10% Loss Rate **(b) 15% Loss Rate** **(c) 20% Loss Rate**

Scalability in Groups

64 nodes, * groups per node, 10 nodes per group, Loss Model: Uniform 1%

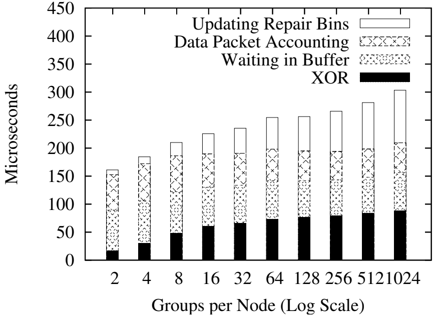


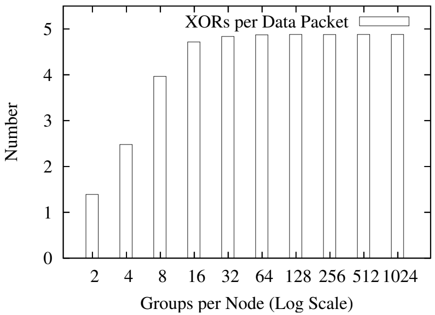


Ricochet scales to hundreds of groups. Comparison: at 128 groups, SRM latency was 8 seconds. 400 times slower!

CPU time and XORs per data packet

64 nodes, * groups per node, 10 nodes per group, Loss Model: Uniform 1%

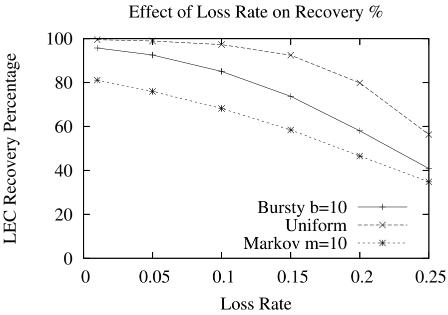


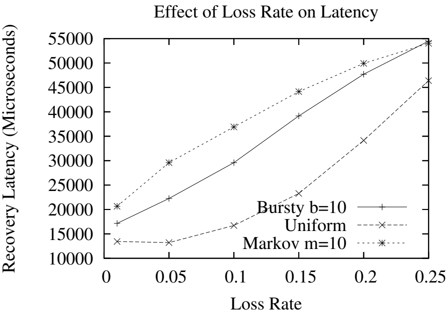


Ricochet is lightweight $\Longrightarrow$ Time-Critical Apps can run over it

Impact of Loss Rate on LEC

64 nodes, 128 groups per node, 10 nodes per group, Loss Model: *

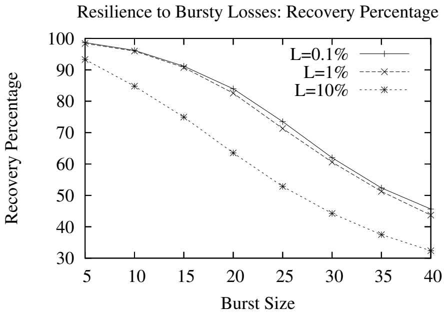


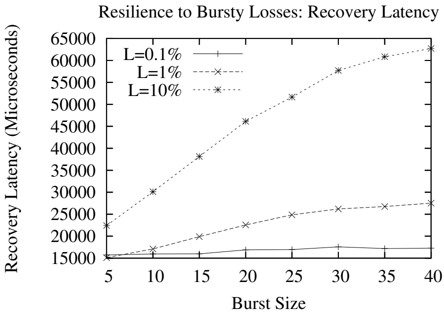


Works well at typical datacenter loss rates: 1-5%

Resilience to Burstiness

64 nodes, 128 groups per node, 10 nodes per group, Loss Model: Bursty 1%

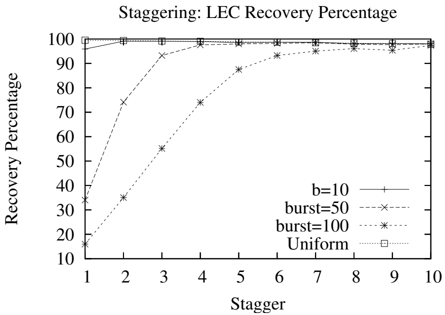


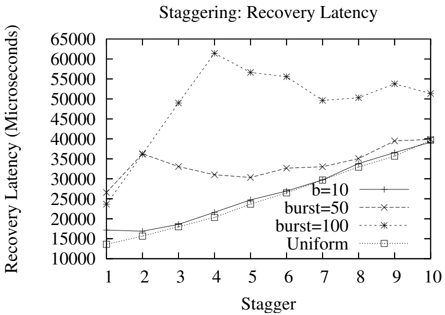


... can handle short bursts (5-10 packets) well. Good enough?

Staggering

64 nodes, 128 groups per node, 10 nodes per group, Loss Model: Bursty 1%





Stagger of *i*: Encode every *i*th packet

Stagger 6, burst of 100 packets $\Longrightarrow$ 90% recovered at 50 ms!

Ricochet: Overview

- Time-Critical Datacenters:
  - large numbers of low-rate groups
  - bursty end-host loss patterns
- Ricochet is the first protocol to scale in the *number of groups* in the system

Outline

1. Motivation
2. Systems
   - Ricochet
   - Maelstrom
3. Conclusion

Open Problem: LambdaNets

- The Lambda Internet: A collection of geographically dispersed datacenters...
- ... connected by optical 'lambda' links
- Applications need to run over LambdaNets:
  - Financial services operating in different markets
  - MNCs with operations in different countries
  - High-volume e-tailers

Why is this hard?

- Speed of Light!
- Existing systems are not designed for very high communication latencies:
  - Try executing a Java RMI call on a server sitting in India...
  - Or mirroring your Oracle database to Kentucky...
- Need for fundamental redesign of software stack

Data Transport for the Lambda Internet

- TCP/IP uses RTT-based timeouts and retransmissions...
  ... hundreds of milliseconds to recover lost packets!
- FEC: Perfect technology for long-distance transfer...
  ... but useless if loss is bursty.
- Maelstrom: Decorrelated FEC — Constructs repair packets from across multiple outgoing channels from one datacenter to another

Datacenters are the present (and future)

- The applications you build will run on Datacenters
- Current technology works... barely.
- Next-generation applications will push the limits of scalability:
  - What if all TV is IP-based (YouTube on steroids)?
  - What if all your data/functionality is remote? (AJAX-based Apps...)
  - What if *everything* is remote? (Web-based Operating Systems...)