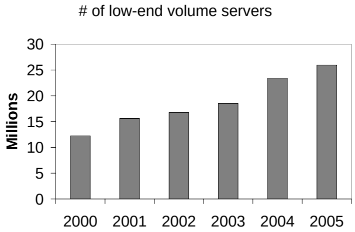
# Maelstrom: An Enterprise Continuity Protocol for Financial Datacenters

Mahesh Balakrishnan, **Ken Birman**
Tudor Marian, Hakim Weatherspoon

Cornell University, Ithaca, NY

# Datacenters

- Internet Services (90s) — Websites, Search, Online Stores
- Since then:




Installed Server Base 00-05:

- Commodity — up by 100%
- High/Mid — down by 40%

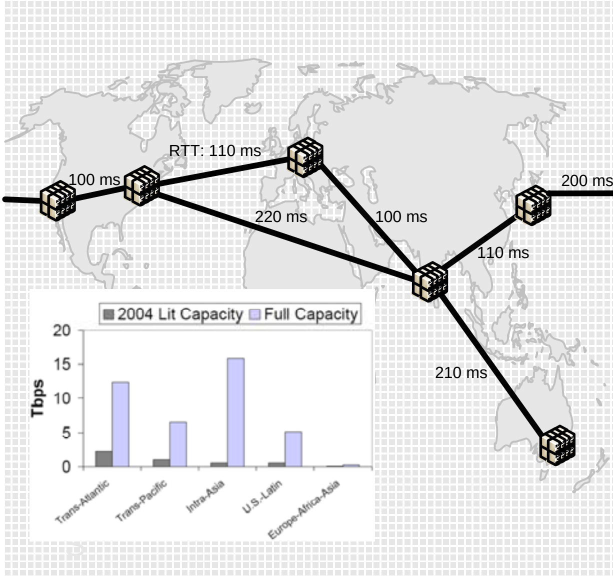
2007: $\approx$ 7 million new units

- Today: Datacenters are ubiquitous
- How have they evolved?

Data partially sourced from IDC press releases (www.idc.com)

# *Networks of* Datacenters

Why?
Client Locality,
Distributed Datasets
or Operations,
Enterprise Continuity



# *Real-Time* Enterprise Continuity

- Financial Datacenters: Real-Time, Mission-Critical
- Current State-of-the-art: Mirror from NYC to NJ
- Wanted: Arbitrarily far mirrors!

- Step Zero: How do we send data reliably across high-speed long-distance pipes?

# Reliable Communication between Datacenters

Open a TCP/IP socket from CA to NY. What happens?

- Throughput $\propto \frac{BufSize}{RTT}$
  - BufSize = Receiver Buffer Size
- Current Solution: Manually reset buffer sizes

# Reliable Communication between Datacenters

Open a TCP/IP socket from CA to NY. What happens?

- Throughput $\propto \frac{BufSize}{RTT}$
  - BufSize = Receiver Buffer Size
- Current Solution: Manually reset buffer sizes

- TCP/IP needs zero loss on high-speed long-distance links:
  - Sensitive Congestion Control
  - RTT dependence + Sequenced Delivery

# Reliable Communication between Datacenters

Open a TCP/IP socket from CA to NY. What happens?

- Throughput $\propto \frac{BufSize}{RTT}$
  - BufSize = Receiver Buffer Size
- Current Solution: Manually reset buffer sizes

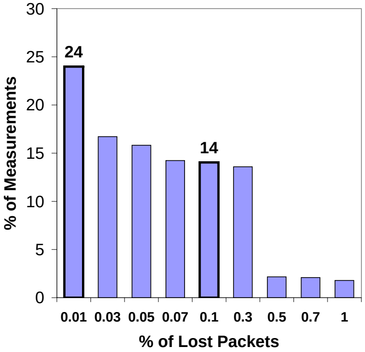
- TCP/IP needs zero loss on high-speed long-distance links:
  - Sensitive Congestion Control
  - RTT dependence + Sequenced Delivery

Current State-of-the-Art Solution: $$$!

# TeraGrid: Supercomputer Network

## SDSC, PSC, CTC, IU, **NCSA** ...

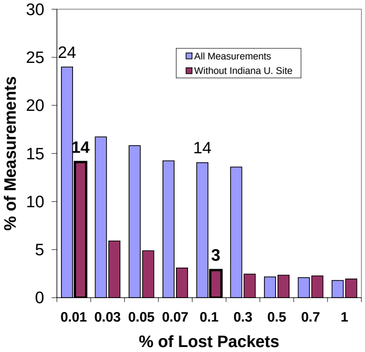
- End-to-End UDP Probes: Zero Congestion, Non-Zero Loss!
- Possible Reasons:
  - transient congestion
  - degraded fiber
  - malfunctioning HW
  - misconfigured HW
  - switching contention
  - low receiver power
  - end-host overflow
  - ...



# TeraGrid: Supercomputer Network

## SDSC, PSC, CTC, IU, **NCSA** ...

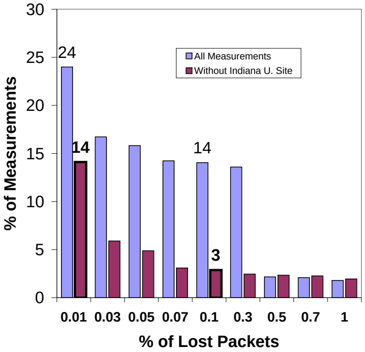
- End-to-End UDP Probes: Zero Congestion, Non-Zero Loss!
- Possible Reasons:
  - transient congestion
  - degraded fiber
  - malfunctioning HW
  - misconfigured HW
  - switching contention
  - low receiver power
  - end-host overflow
  - ...



# TeraGrid: Supercomputer Network

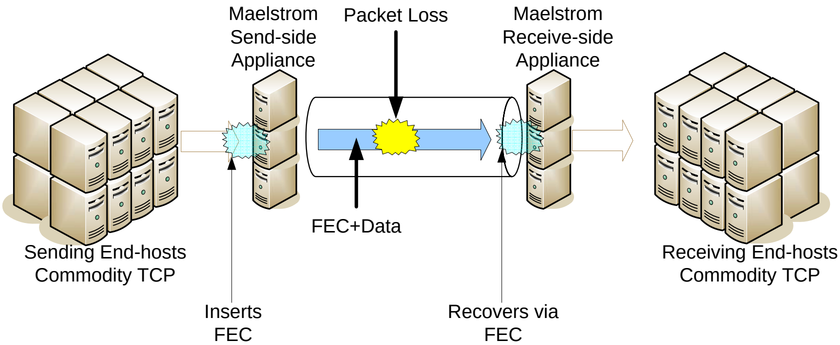
## SDSC, PSC, CTC, IU, **NCSA** ...

- End-to-End UDP Probes: Zero Congestion, Non-Zero Loss!
- Possible Reasons:
  - transient congestion
  - degraded fiber
  - malfunctioning HW
  - misconfigured HW
  - switching contention
  - low receiver power
  - end-host overflow
  - ...




Problem Statement: Run *unmodified* TCP/IP over *lossy* high-speed long-distance networks

# The Maelstrom Network Appliance




FEC = Forward Error Correction

Transparent: No modification to end-host or network

# What is FEC?


3 repair packets from every 5 data packets

Receiver can recover from any 3 lost packets

Rate[1]: $(r, c)$ — $c$ repair packets for every $r$ data packets.

- Pro: No Feedback Loop, Constant Overhead
- Why not run FEC at end-hosts?
- Con: Recovery Latency dependent on channel data rate

[1] Rateless codes are popular, but inapplicable to real-time streams

# What is FEC?


3 repair packets from every 5 data packets

Receiver can recover from any 3 lost packets

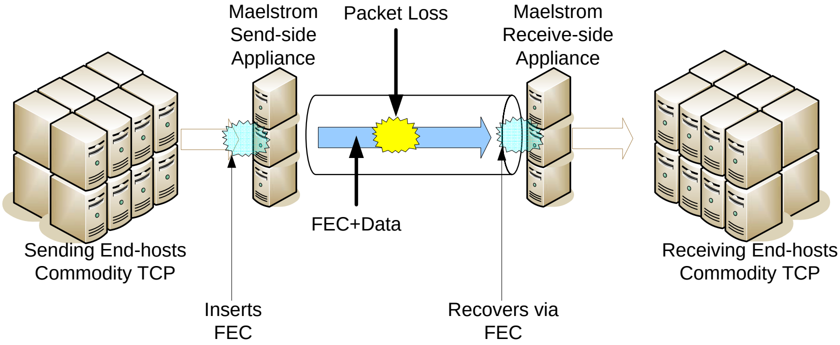
Rate[1]: $(r, c)$ — $c$ repair packets for every $r$ data packets.

- Pro: No Feedback Loop, Constant Overhead
- Why not run FEC at end-hosts?
- Con: Recovery Latency dependent on channel data rate

- FEC in the Network:
  - Where and What: At the appliance, across multiple channels

[1] Rateless codes are popular, but inapplicable to real-time streams

# The Maelstrom Network Appliance




FEC = Forward Error Correction

Transparent: No modification to end-host or network

Where: at the appliance, What: aggregated data

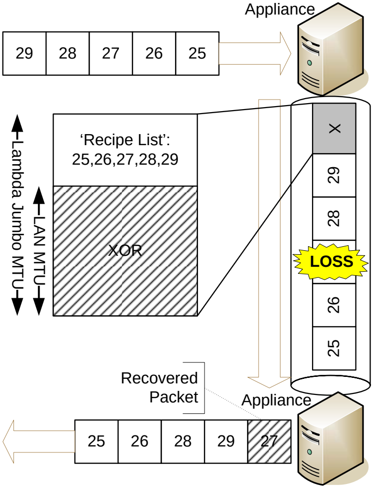
# Maelstrom Mechanism

Send-Side Appliance:

- Intercept IP packets
- Create repair packet = XOR + 'recipe' of data packet IDs

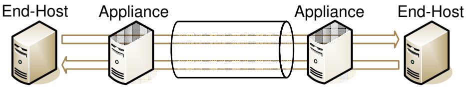
Receive-Side Appliance:

- Lost packet recovered using XOR and other data packets
- At receiver end-host: out of order, no loss



# Flow Control

- Two Flow Control Modes for TCP/IP Traffic:



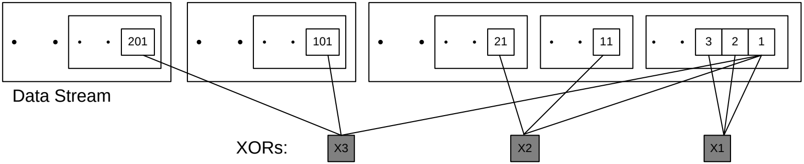

A) End-to-End Flow Control


B) Split Flow Control

- Recall: Two problems with TCP/IP — loss and buffering
  - End-to-end mode hides loss
  - Split mode buffers data (standard PeP)

# Layered Interleaving for Bursty Loss

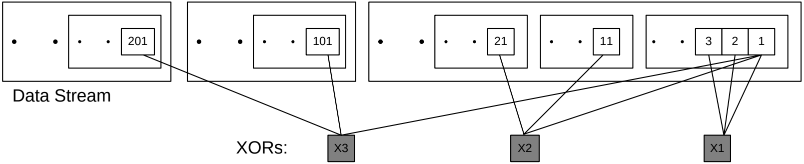
Recovery Latency $\propto$ Actual Burst Size, not Max Burst Size



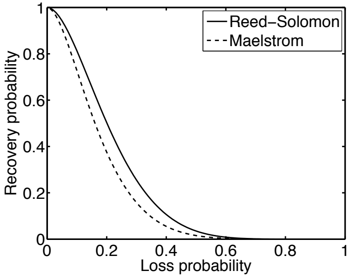

- XORs at different interleaves
- Recovery latency degrades gracefully with loss burstiness:
  X1 catches random singleton losses
  X2 catches loss bursts of 10 or less
  X3 catches bursts of 100 or less

# Layered Interleaving for Bursty Loss

Recovery Latency $\propto$ Actual Burst Size, not Max Burst Size




- XORs at different interleaves
- Recovery latency degrades gracefully with loss burstiness:
  X1 catches random singleton losses
  X2 catches loss bursts of 10 or less
  X3 catches bursts of 100 or less




Comparison of Recovery Probability: r=7, c=2

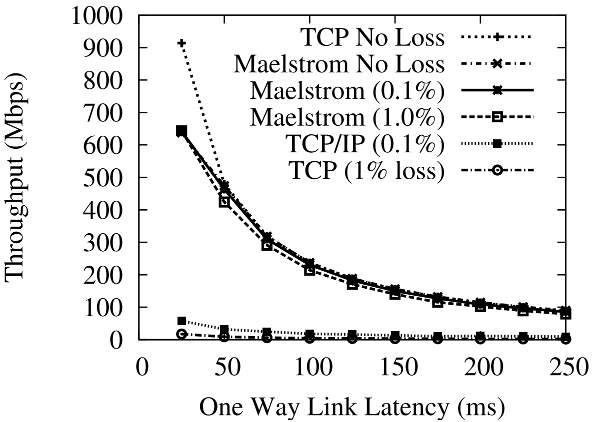
# Implementation Details

- In Kernel — Linux 2.6.20 Module
- Commodity Box: 3 Ghz, 1 Gbps NIC ($\approx$ 800$)
- Max speed: 1 Gbps, Memory Footprint: 10 MB
- 50-60% CPU $\rightarrow$ NIC is the bottleneck (for $c = 3$)

- How do we efficiently store/access/clean a gigabit of data every second?
- Scaling to Multi-Gigabit: Partition IP space across proxies

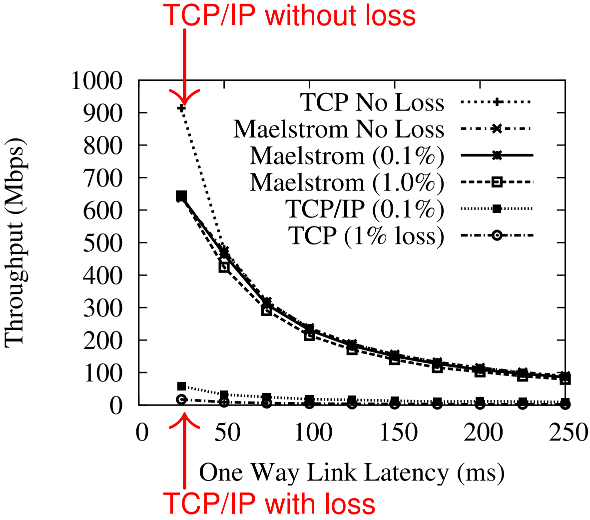
# Evaluation: FEC mode and loss

## Claim: Maelstrom effectively hides loss from TCP/IP



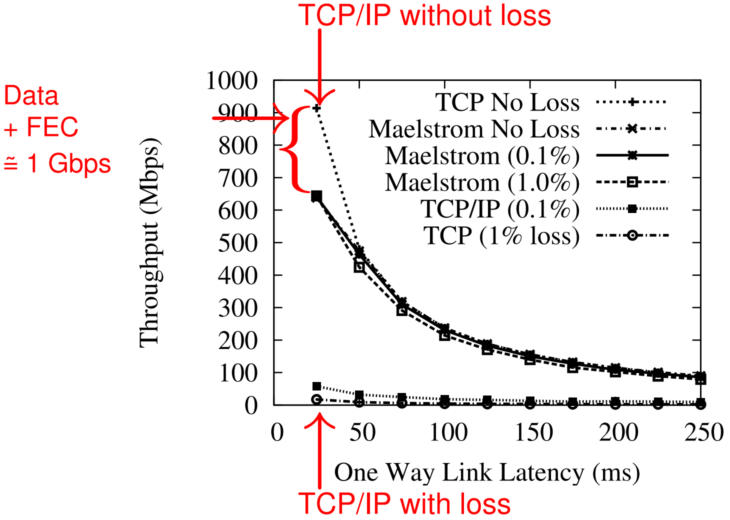
# Evaluation: FEC mode and loss

## Claim: Maelstrom effectively hides loss from TCP/IP



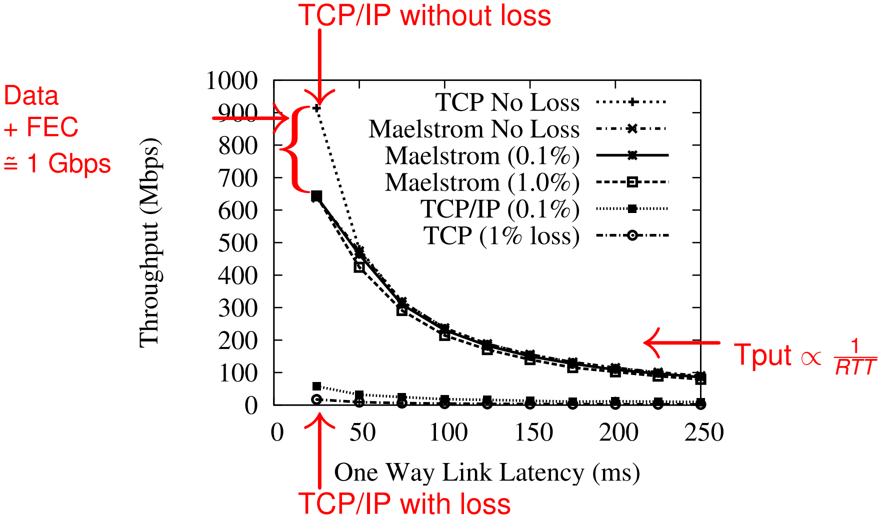
# Evaluation: FEC mode and loss

## Claim: Maelstrom effectively hides loss from TCP/IP



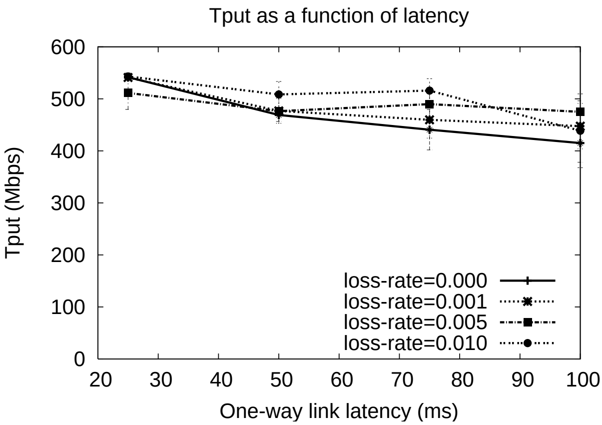
# Evaluation: FEC mode and loss

## Claim: Maelstrom effectively hides loss from TCP/IP



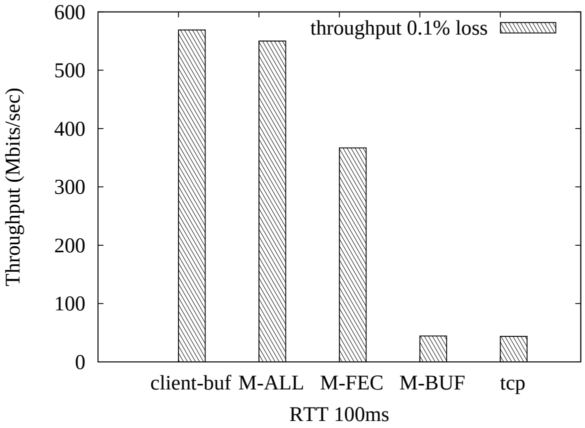
# Evaluation: Split Mode and buffering

## Claim: Maelstrom performance is independent of link length



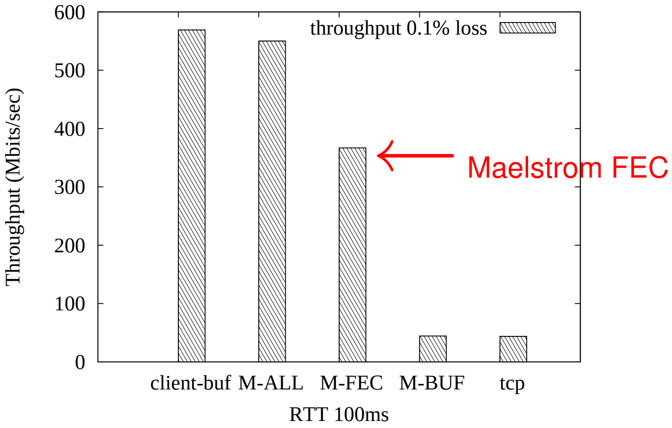
# Evaluation: Split mode and buffering

## Claim: Maelstrom split mode is as good as hand tuning of buffer sizes



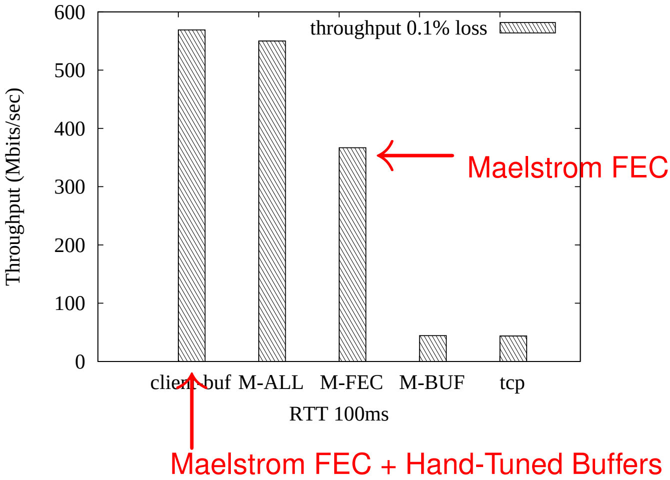
# Evaluation: Split mode and buffering

Claim: Maelstrom split mode is as good as hand tuning of buffer sizes



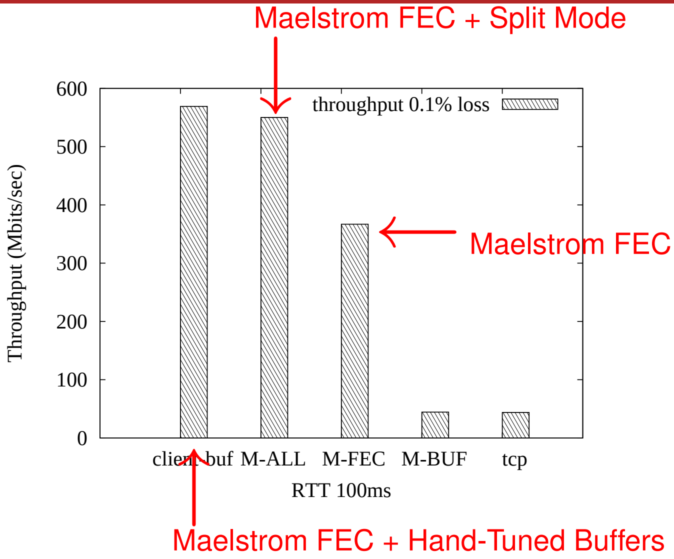
# Evaluation: Split mode and buffering

Claim: Maelstrom split mode is as good as hand tuning of buffer sizes



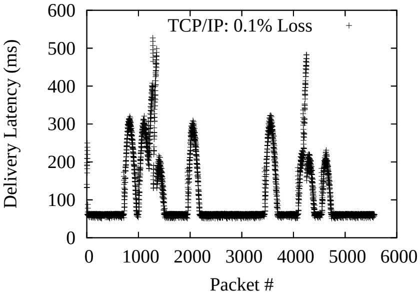
# Evaluation: Split mode and buffering

Claim: Maelstrom split mode is as good as hand tuning of buffer sizes



# Evaluation: Delivery Latency

## Claim: Maelstrom eliminates TCP/IP's loss-related jitter




- Receive-side buffering due to sequencing
- Send-side buffering due to congestion control

# SMFS: The Smoke and Mirrors Filesystem

- Classic Mirroring Trade-off:
  - Fast — return to user after sending to mirror
  - Safe — return to user after ACK from mirror
- Maelstrom: Lossy Network $\rightarrow$ Lossless Network $\rightarrow$ Disk!
- SMFS — return to user after sending enough FEC
- Result: Fast, Safe Mirroring independent of link length!
- General Principle: Gray-box Exposure of Protocol State

# Future Work: User-Mode Packet Processors

- Common need: packet processing at gigabit speeds on commodity HW/OS
- Examples: overlay routing, deep packet inspection, application/protocol acceleration (Maelstrom)...
- Current option: write in-kernel code
- Proposed solution: Lightweight User-Mode Processes
  - Narrow packet-specific interface to kernel
  - Developer writes C code against custom library
  - Pinned memory, circular buffers

# Conclusion

- How do enterprises recover from failures in real-time?
- Enabling networks of remote datacenters
- Step 0: Reliable Communication — Maelstrom
- Step 1: Data Mirroring — SMFS
- What's next?