# Smoke and Mirrors: Shadowing Files at a Geographically Remote Location Without Loss of Performance
# - and -
# Message Logging: Pessimistic, Optimistic, Causal, and Optimal


presenter

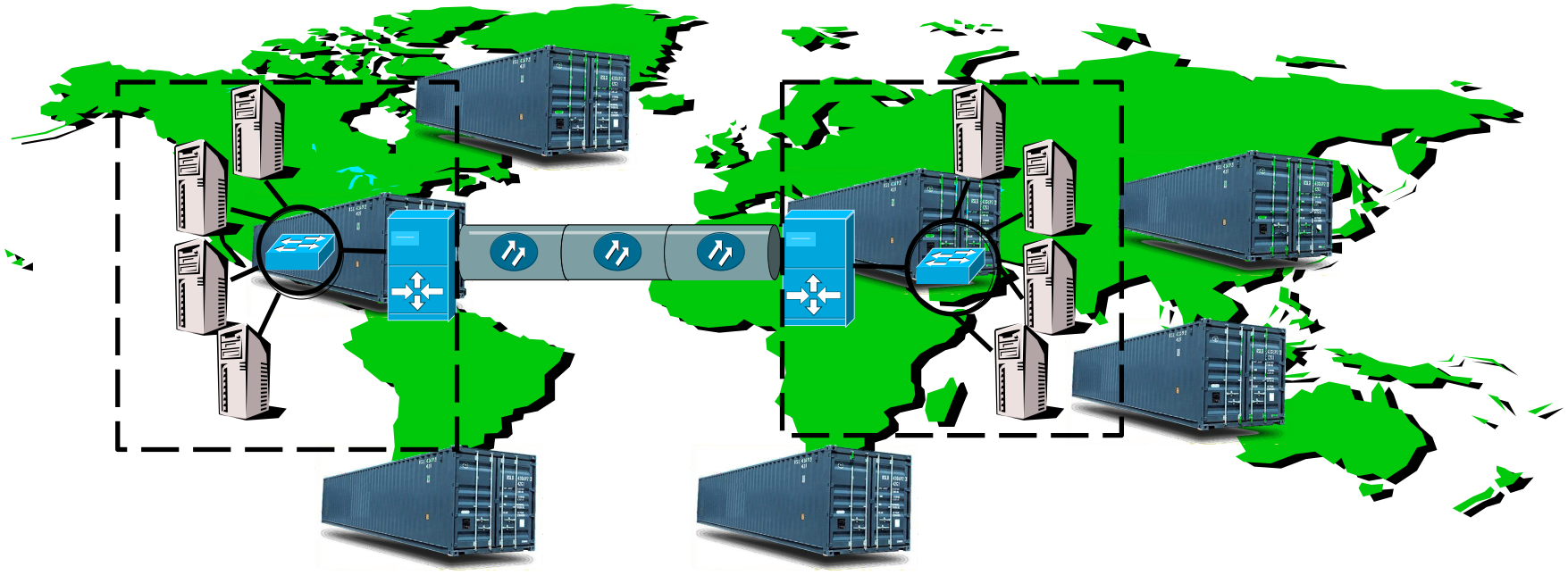
**Hakim Weatherspoon**


CS 6464

February 5th, 2009

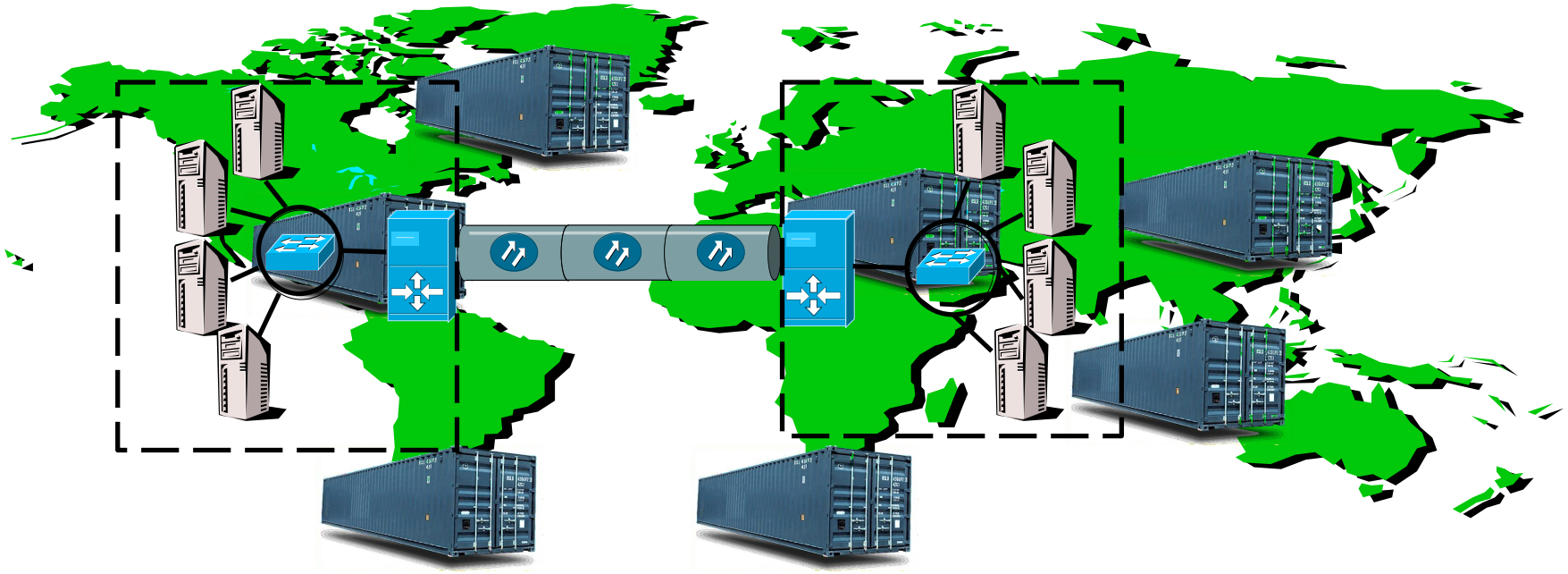
# Critical Infrastructure Protection and Compliance

- U.S. Department of Treasury Study
  - Financial Sector vulnerable to significant data loss in disaster
  - Need new technical options
- Risks are real, technology available, Why is problem not solved?

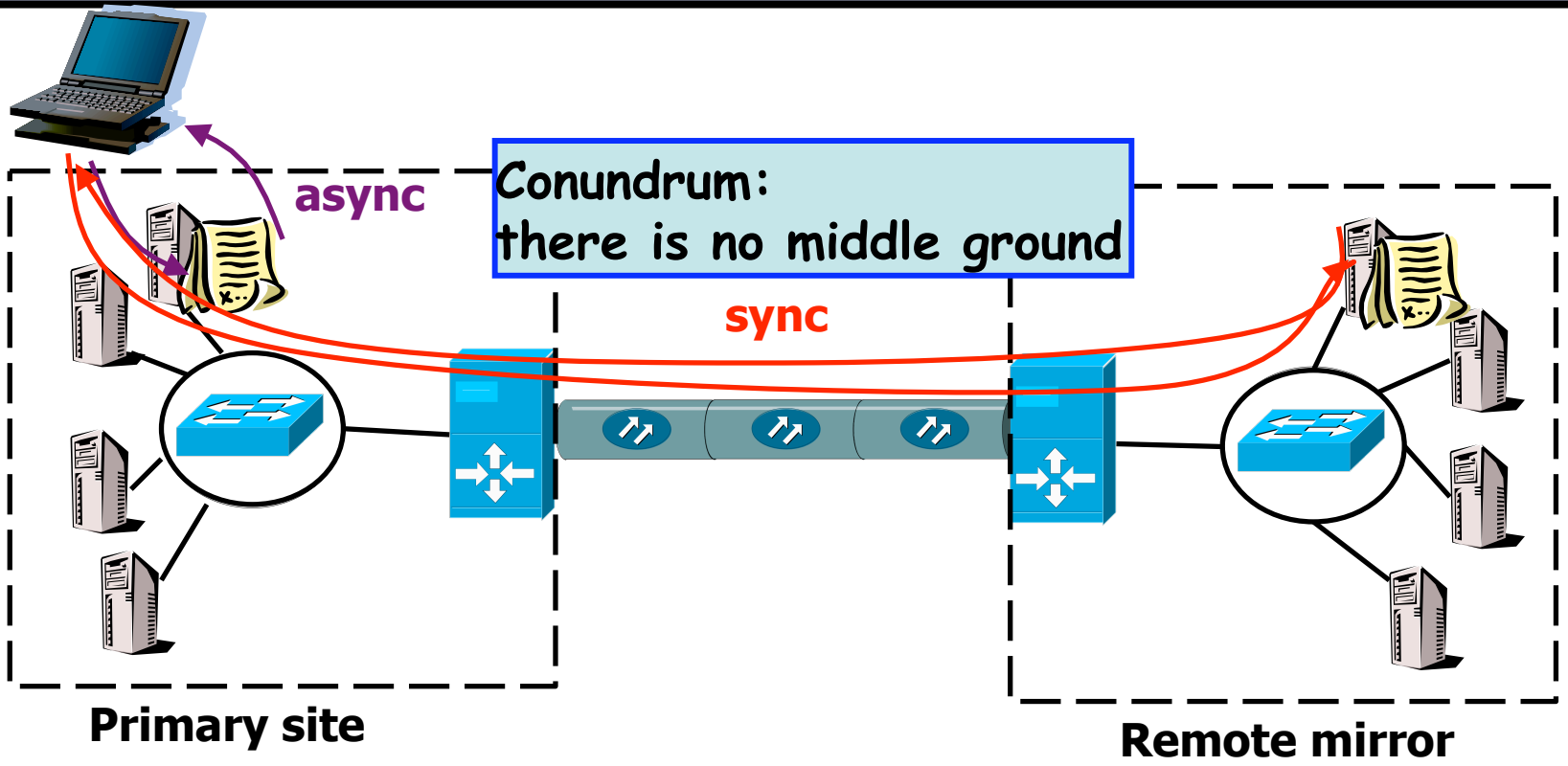


# Bold New (Distributed Systems) World…

- High performance applications coordinate over vast distances
  - Systems coordinate to cache, mirror, tolerate disaster, etc.
  - Enabled by cheap storage and cpus, commodity datacenters, and plentiful bandwidth
- Scale is amazing, but tradeoffs are old ones

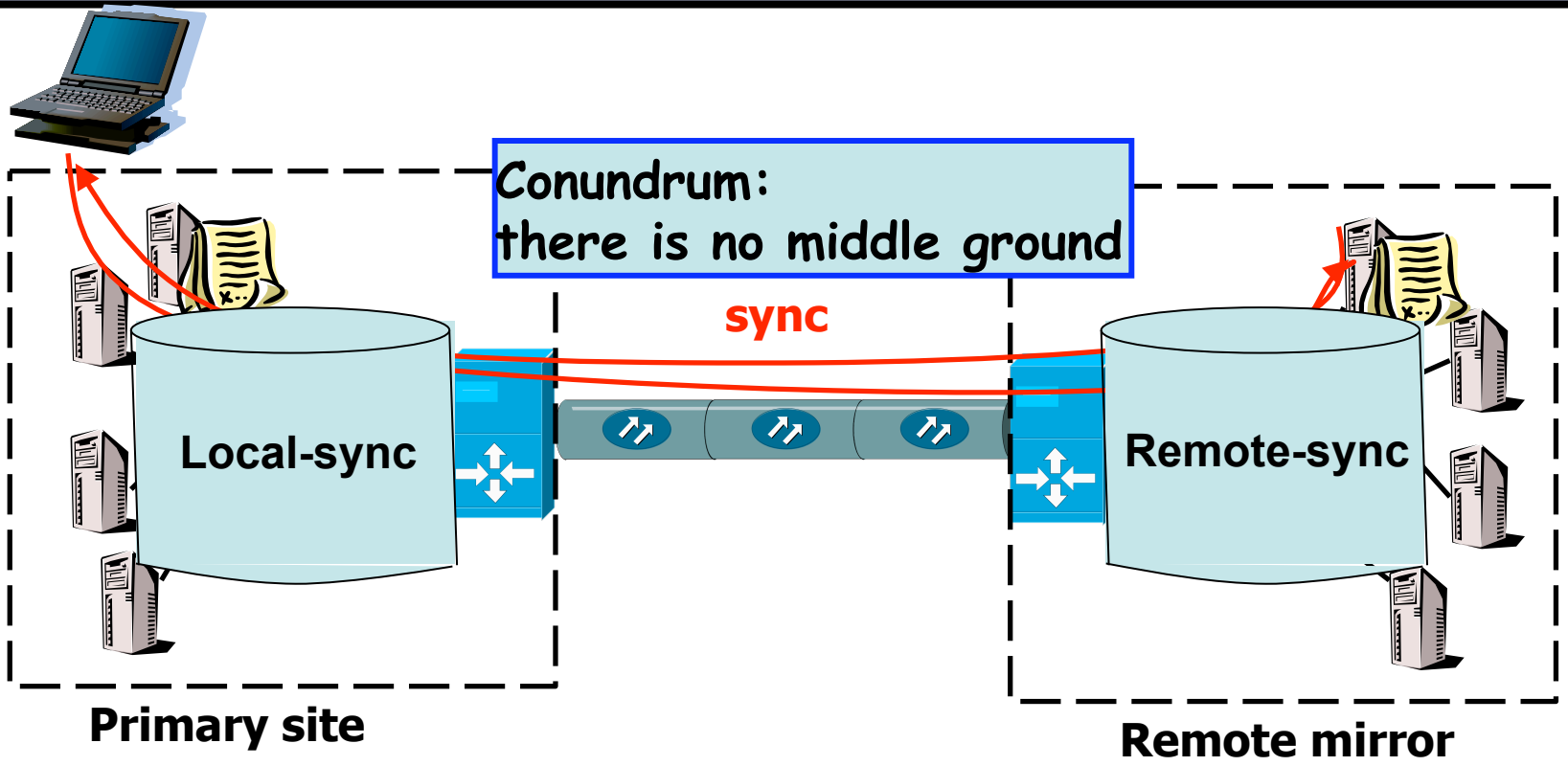


# Mirroring and speed of light dilemma…




- Want asynchronous performance to local data center
- *And* want synchronous guarantee

# Mirroring and speed of light dilemma…




- Want asynchronous performance to local data center
- *And* want synchronous guarantee

# Challenge

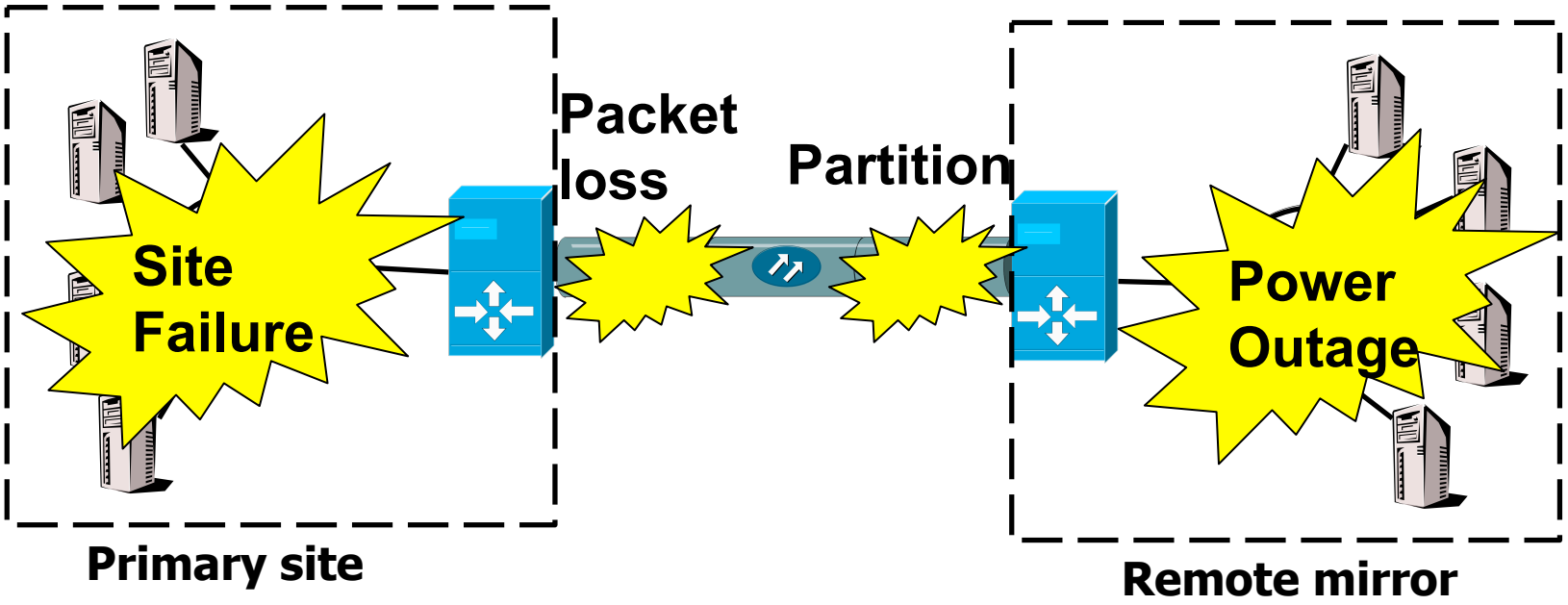
- How can we increase reliability of local-sync protocols?
  - Given many enterprises use local-sync mirroring anyways
- Different levels of local-sync reliability
  - Send update to mirror immediately
  - Delay sending update to mirror – deduplication reduces BW

# Talk Outline

- Introduction
- **Enterprise Continuity**
  - How data loss occurs
  - How we prevent it
  - A possible solution
- Evaluation
- Message Logging: Pessimisitic, Optimistic, Causal, and Optimal
- Cornell National Lambda Rail (NLR) Testbed
- Conclusion

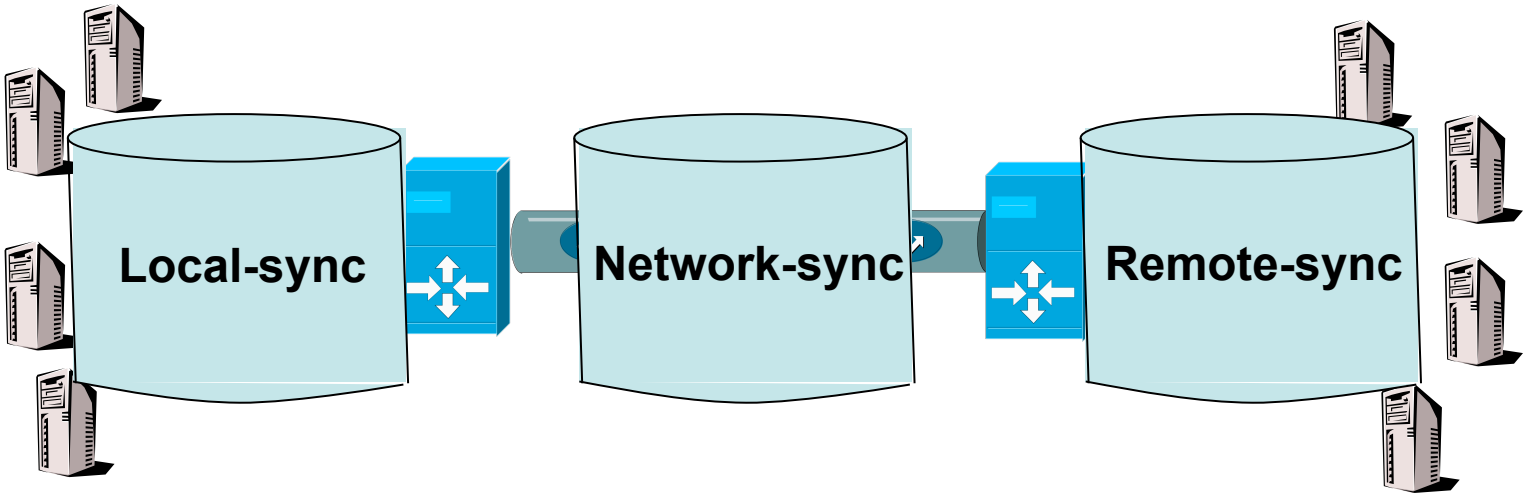
# How does loss occur?

- Rather, where do failures occur?




- Rolling disasters

# Enterprise Continuity: Network-sync

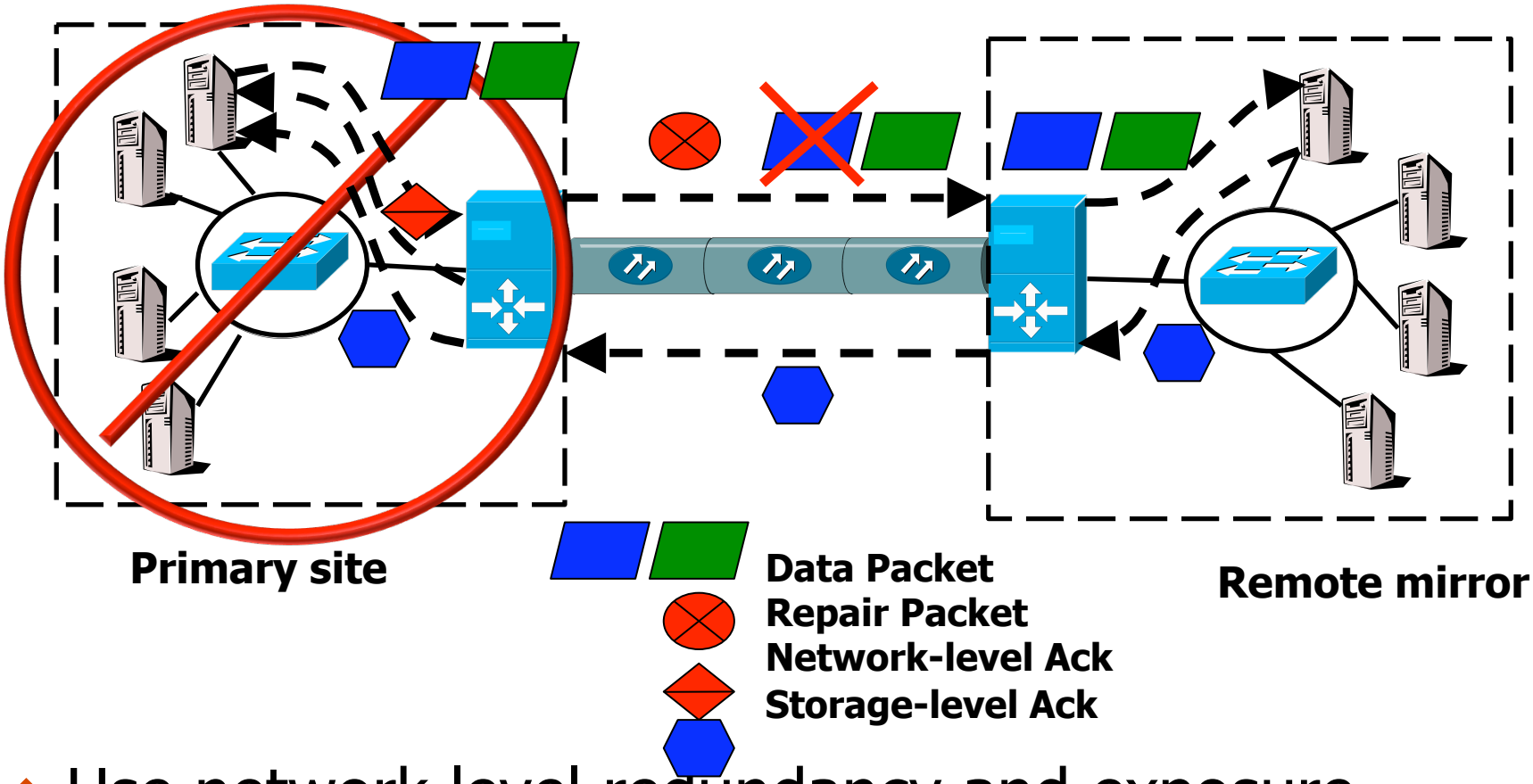



**Primary site**

**Wide-area network**

**Remote mirror**

# Enterprise Continuity Middle Ground




- Use network level redundancy and exposure
  - reduces probability data lost due to network failure

# Enterprise Continuity Middle Ground

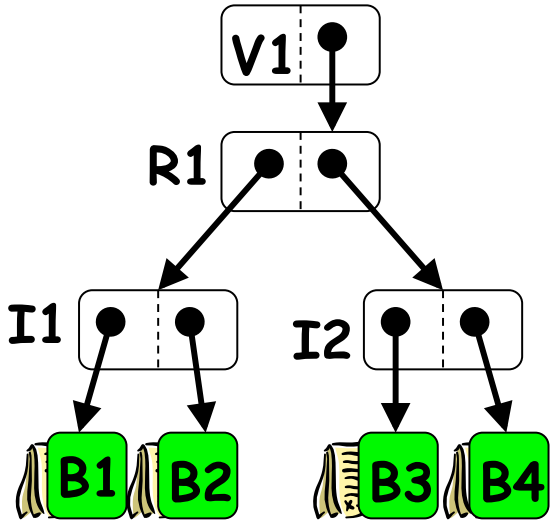
- Network-sync increases data reliability
  - reduces data loss failure modes, can prevent data loss if
  - At the same time primary site fail network drops packet
  - And ensure data not lost in send buffers and local queues
- Data loss can still occur
  - Split second(s) before/after primary site fails…
  - Network partitions
  - Disk controller fails at mirror
  - Power  outage at mirror
- Existing mirroring solutions can use network-sync

# Smoke and Mirrors File System

- A file system constructed over network-sync
  - Transparently mirrors files over wide-area
  - Embraces concept:

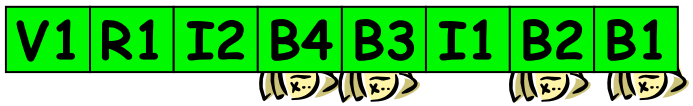
    file is in transit (in the WAN link) but with enough recovery data to ensure that loss rates are as low as for the remote disk case!
  - Group mirroring consistency

# Mirroring consistency and Log-Structured File System




*append*(B1,B2)

*append*(V1..)



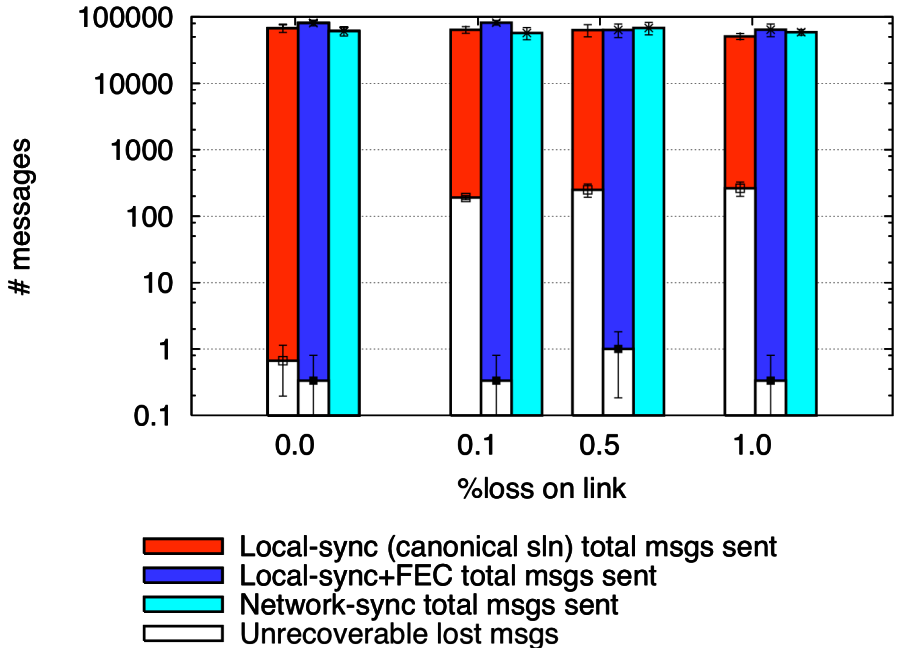
# Talk Outline

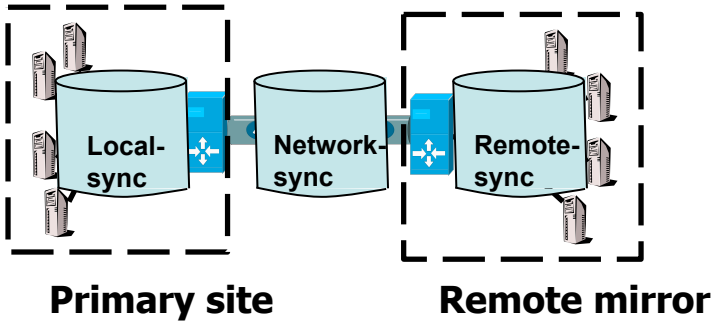
- Introduction
- Enterprise Continuity
- **Evaluation**
- Message Logging
- Conclusion

# Evaluation

- Demonstrate SMFS performance over Maelstrom
  - In the event of disaster, how much data is lost?
  - What is system and app throughput as link loss increases?
  - How much are the primary and mirror sites allowed to diverge?
- Emulab setup
  - 1 Gbps, 25ms to 100ms link connects two data centers
  - Eight primary and eight mirror storage nodes
  - 64 testers submit 512kB appends to separate logs
    - Each tester submits only one append at a time

# Data loss as a result of disaster

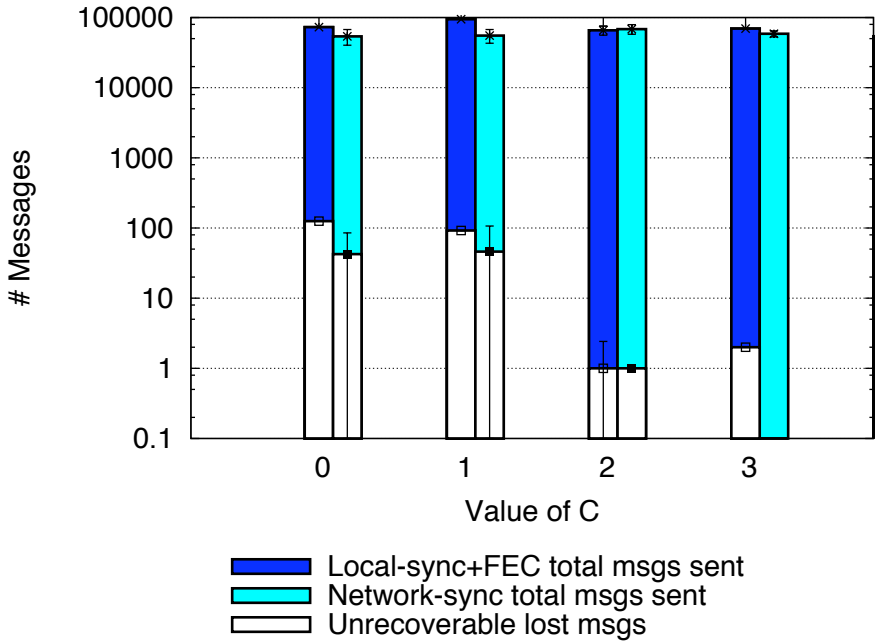


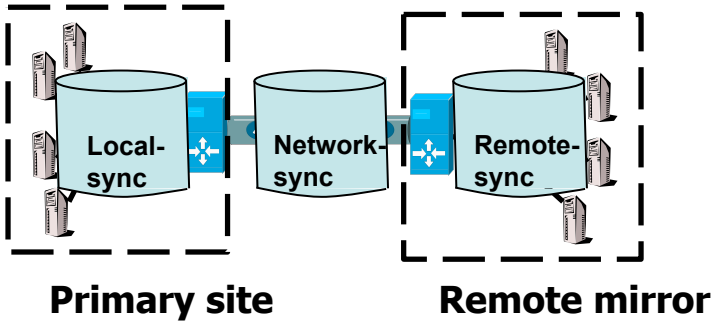




- **50 ms one-way latency**
- **FEC(r,c) = (8,3)**

- Local-sync unable to recover data dropped by network
- Local-sync+FEC lost data not in transit
- Network-sync did *not* lose any data
  - Represents a new tradeoff in design space

# Data loss as a result of disaster

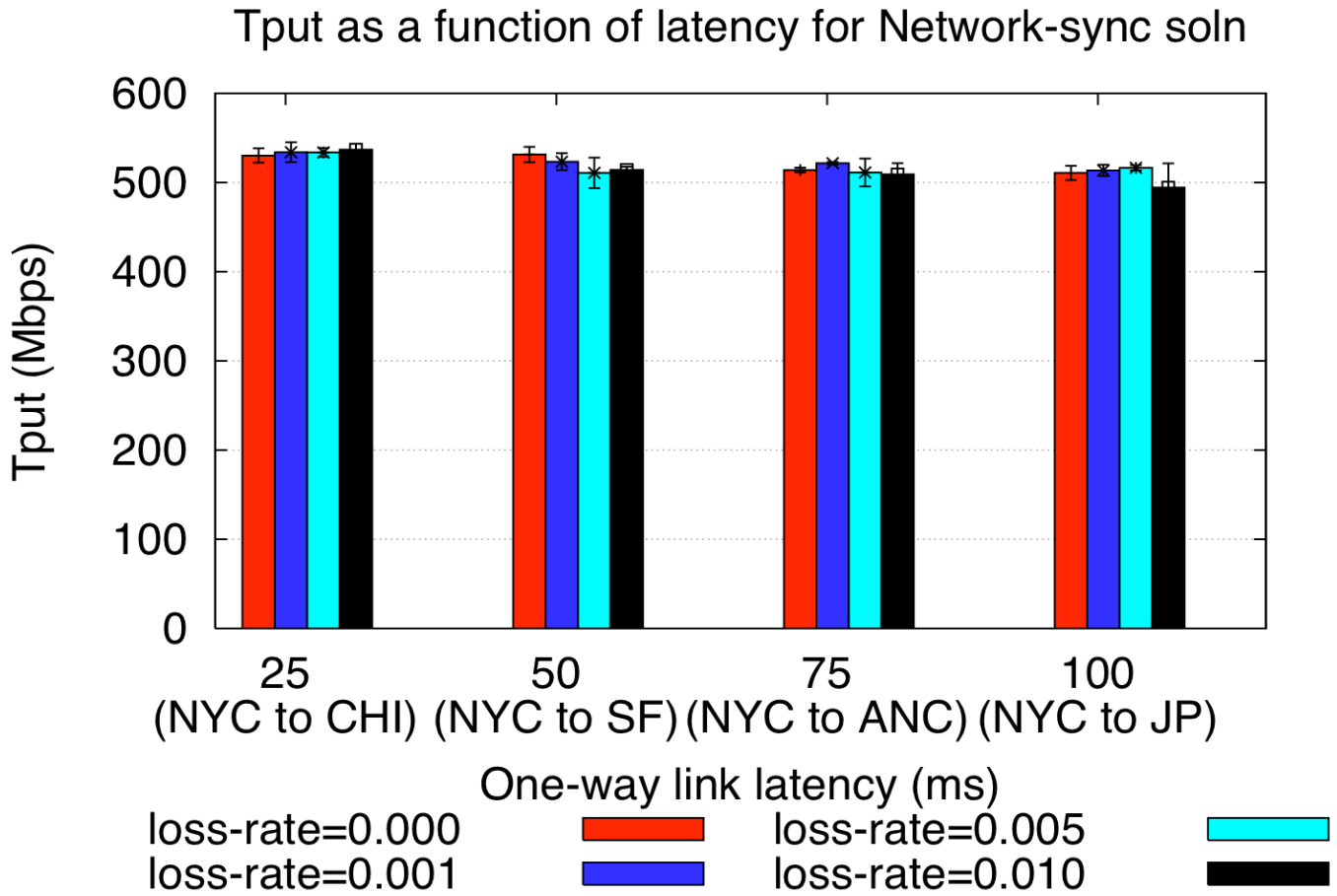


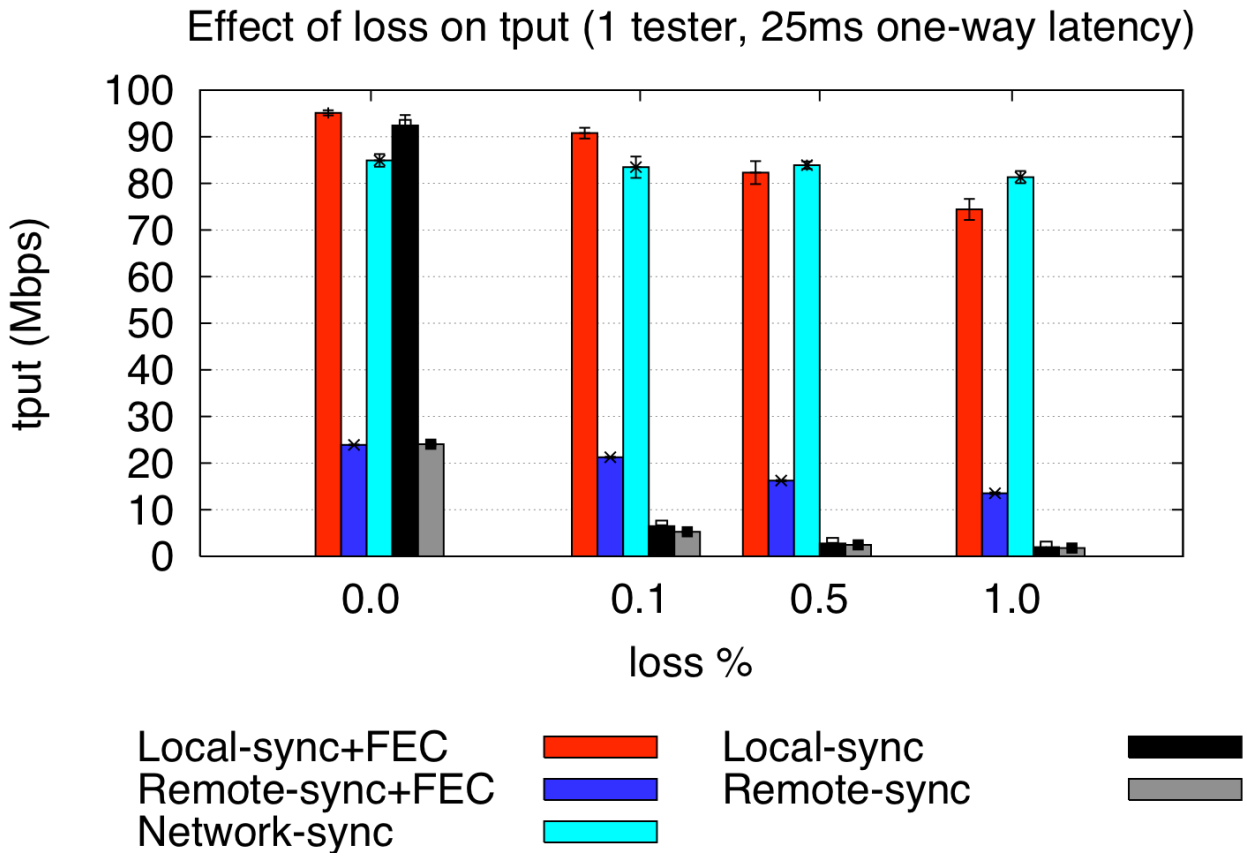

- **50 ms one-way latency**
- **FEC(r,c) = (8,*varies*)**
- **1% link loss**

- *c* = 0, No recovery packets: data loss due to packet loss
- *c* = 1, not sufficient to mask packet loss either
- *c* > 2, can mask most packet loss
- *Network-sync can prevent loss in local buffers*

# High throughput at high latencies

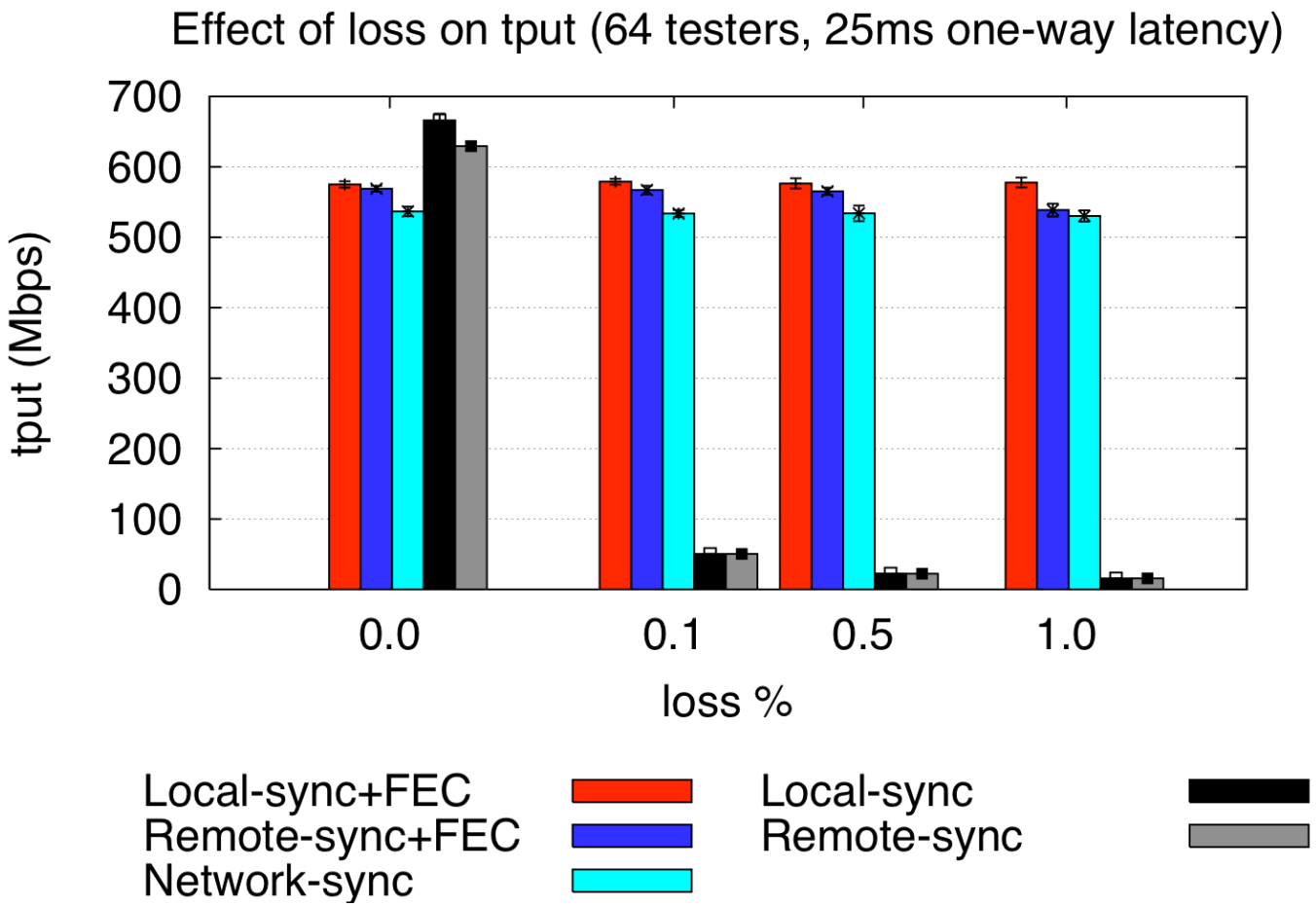


# Application Throughput

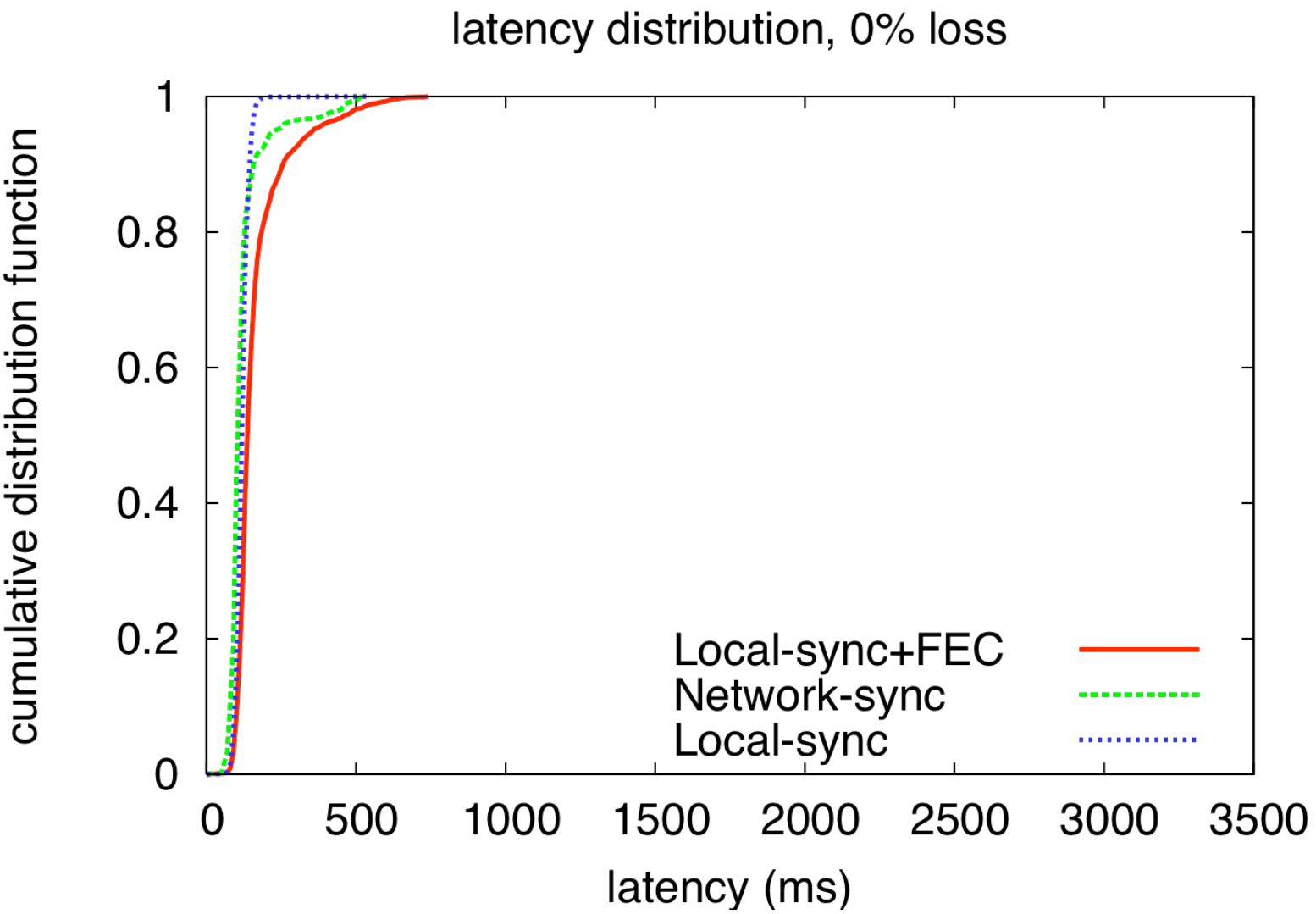



- App throughput measures application perceived performance
- Network and Local-sync+FEC tput significantly greater than Remote-sync(+FEC)

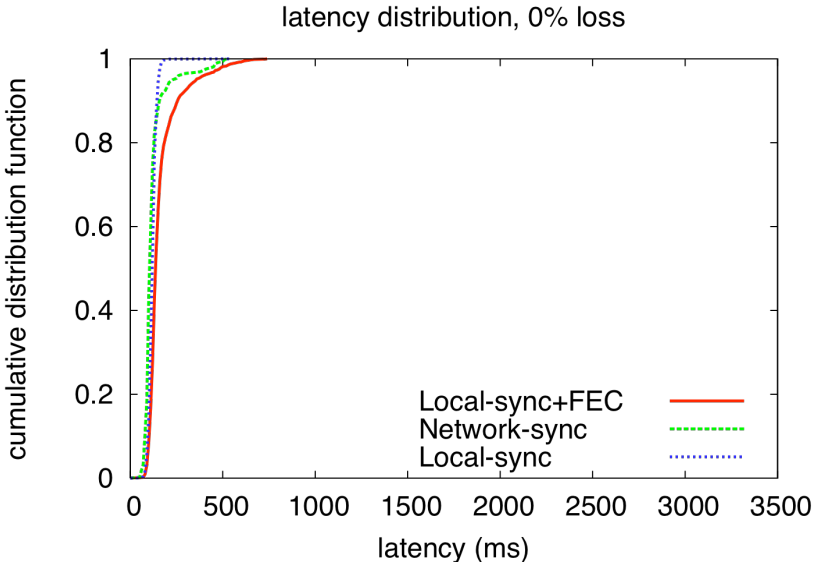
# …There is a tradeoff

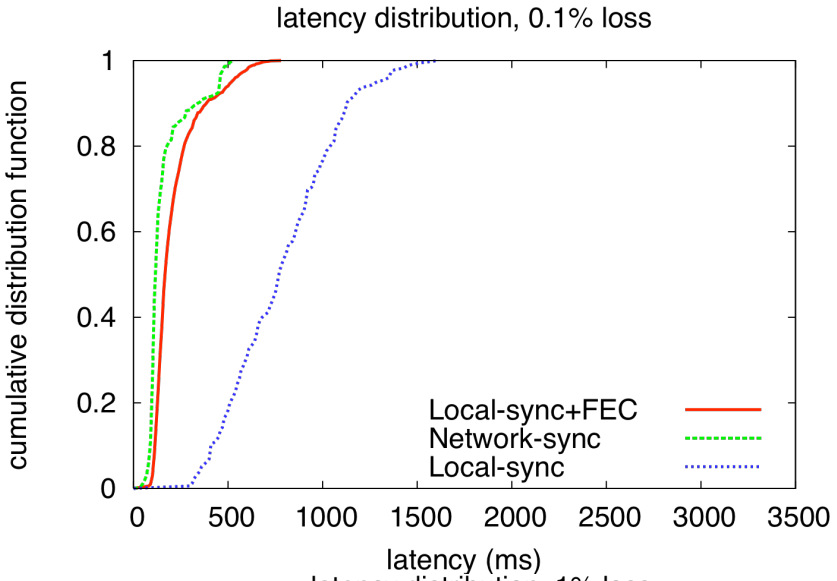


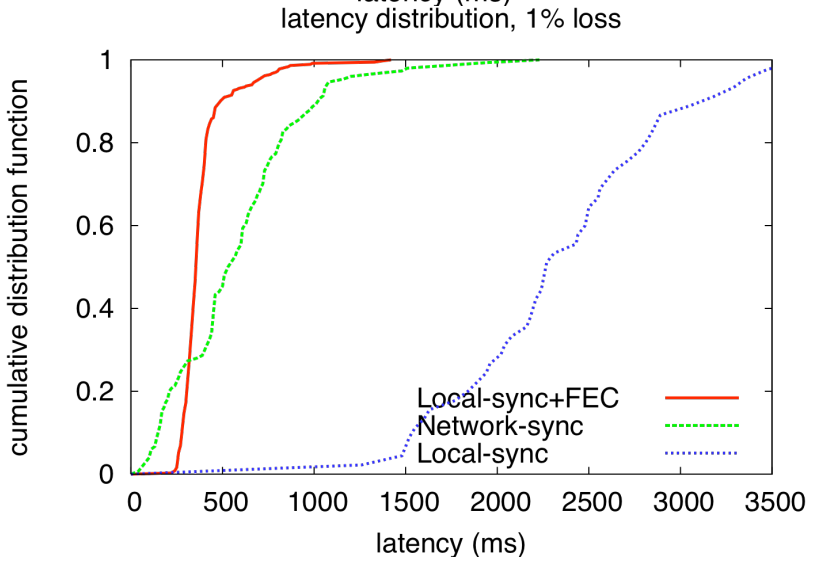
# Latency Distributions



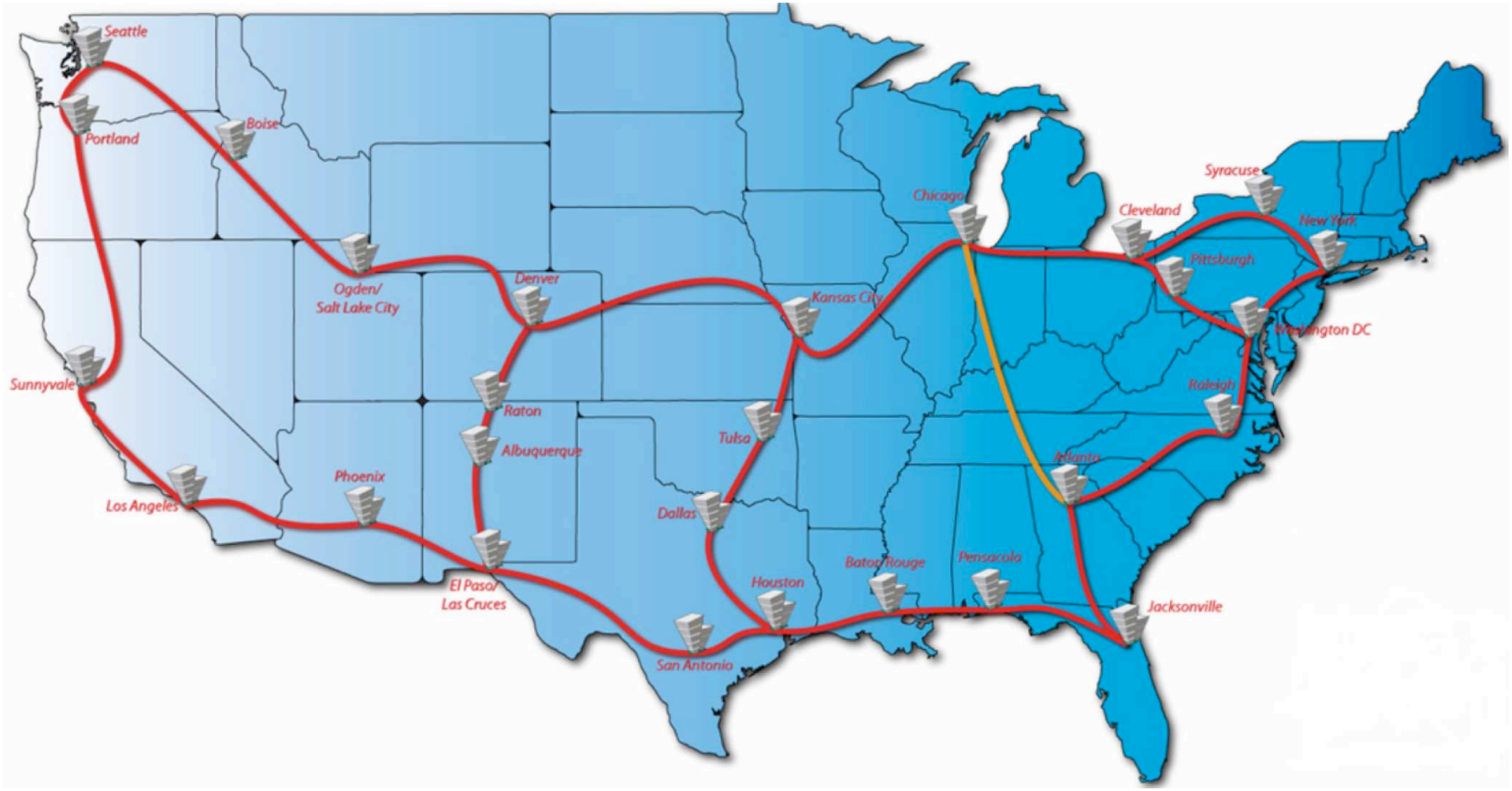
# Latency Distributions



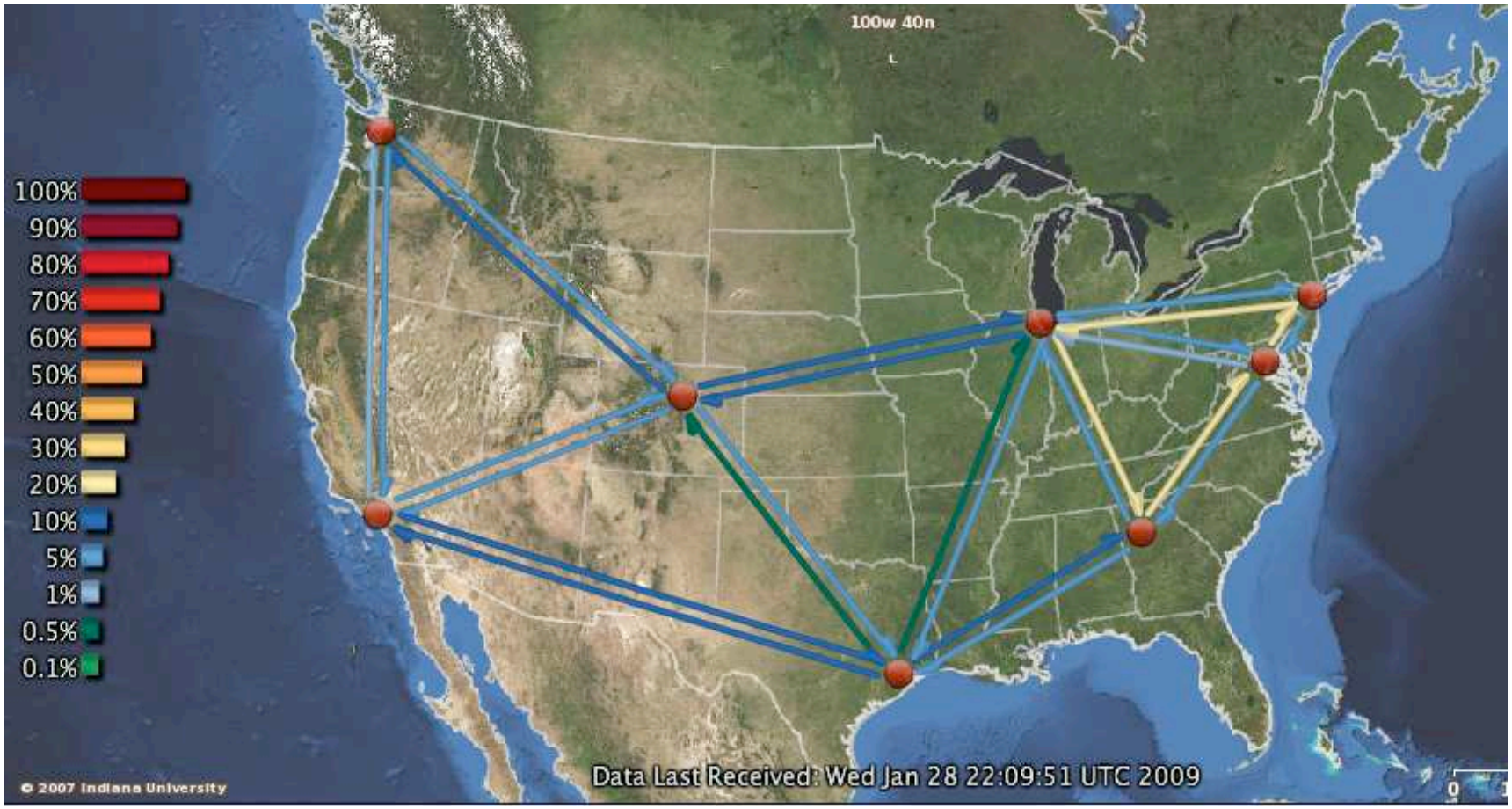



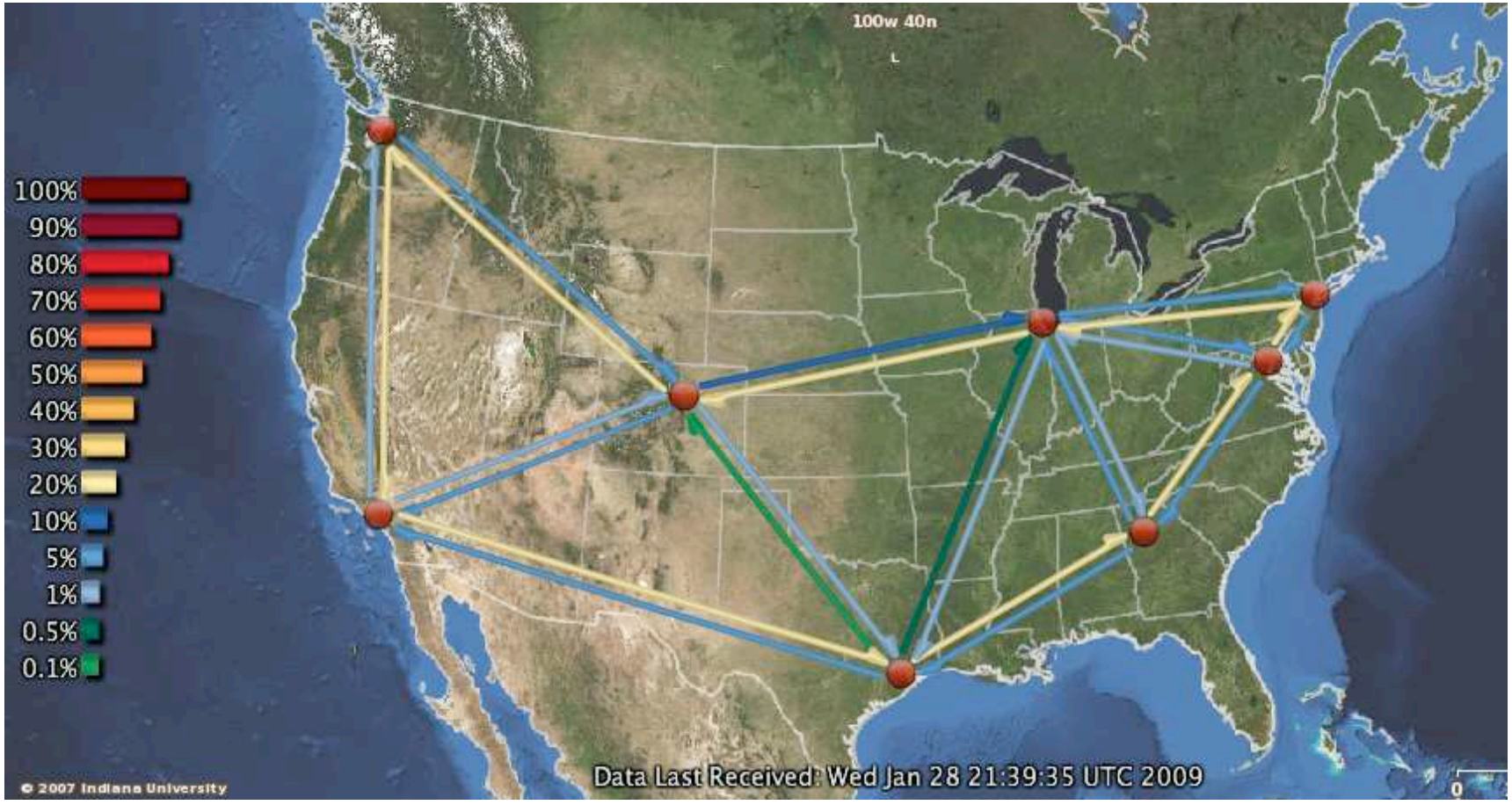
# Cornell National Lambda Rail (NLR) Rings



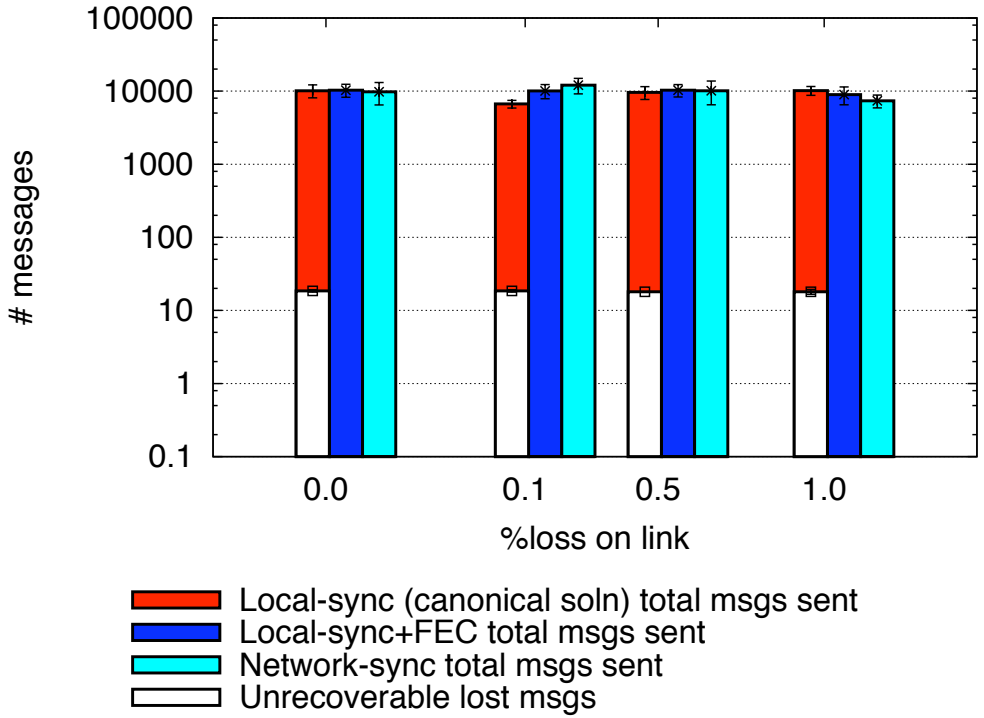
# Cornell National Lambda Rail (NLR) Rings

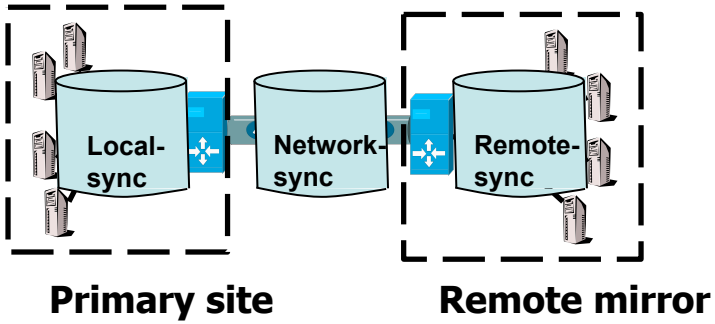


# Cornell National Lambda Rail (NLR) Rings

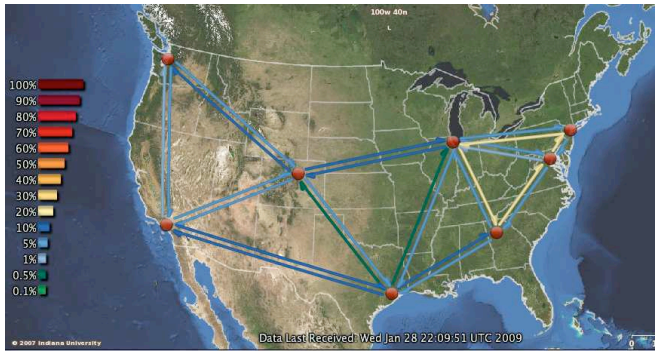


# Cornell NLR Rings Testbed: Data loss as a result of disaster




- **37 ms one-way latency**
- **FEC(r,c) = (8,3)**




- Local-sync unable to recover data dropped by network
- Both Local-sync+FEC and Network-sync did not lost data

# Talk Outline

- Introduction
- Enterprise Continuity
- Evaluation
- **Message Logging**
- Conclusion

# Message Logging

- Optimistic
  - Return result to client before recording to stable storage
- Pessimistic
  - Record result to stable storage first, then return result
- Question, how to prevent *orphaned* processes?
  - After crash, How does a crash-recovery node recover state of system before crash?
  - Pessimistic case is easy, just use log
  - Optimistic case?....
    - Either data lost
    - **Possibly can *scrape* state from clients – (possible project idea)**

# Message Logging

- e.g. “Lehmen Brothers Network Survives”
  - http://www.networkworld.com/research/2001/1126feat.html

- Pharmacy prescriptions after hurricane Katrina

# Talk Outline

- Introduction
- Enterprise Continuity
- Evaluation
- Message Logging
- **Conclusion**

# Conclusion

- Technology response to critical infrastructure needs
- When does the filesystem return to the application?
  - Fast — return after sending to mirror
  - Safe — return after ACK from mirror
- SMFS — return to user after sending enough FEC
- Network-sync:

  Lossy Network➔Lossless Network➔Disk!
- Result: Fast, Safe Mirroring independent of link length!

# Next Time

- Read *NFS* and write review. *Turn into CMS before class*:
  - *Wide-area cooperative storage with CFS, Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Appears in Proceedings of 18th ACM SIGOPS Symposium on Operating Principles (SOSP), October, 2001.*
  - *Chord: A Peer-to-Peer Lookup Service for Internet Applications, Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, Hari Balakrishnan. Appears in Proceedings of the ACM SIGCOMM Conference, September, 2001.*
- Do Lab 1---**Due next Tuesday, February 10th**

- Questions?

# Future Work

- Apply Network-sync technologies to large "DataNets"
- High-speed online processing of massive quantities of real-time data:
  - Performance enhancement proxies (PEP)
  - Deep packet inspection (DPI)
  - Sensor network monitoring
  - Malware / worm signature detection
  - Software switch, IPv4 – IPv6 gateway
- Current state of the art – solution in the kernel
- User process cannot keep up with kernel/NIC

- Backup slides

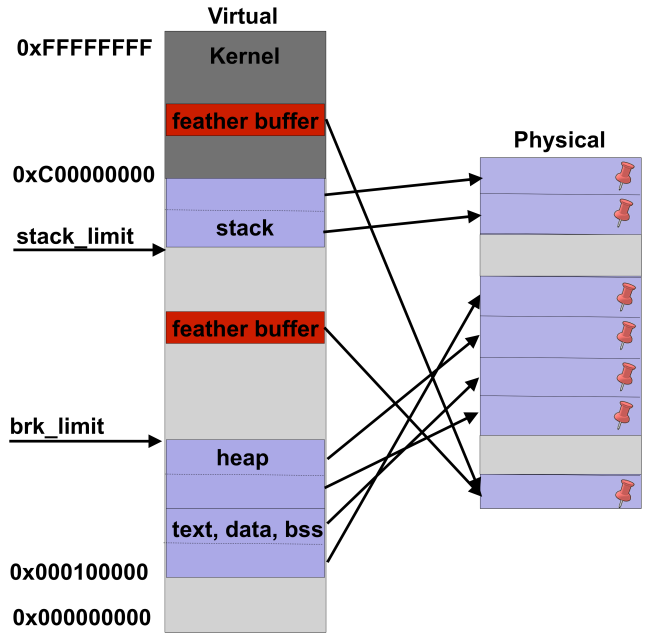
# Future Work

- Apply Network-sync technologies to large "DataNets"
  - NSF focus on global-scale scientific computing
  - DARPA interested in unification of globally distributed military data management subsystems and centers
  - AFRL interested in high speed eventing (e.g. Distributed Missions Operations – DMO)

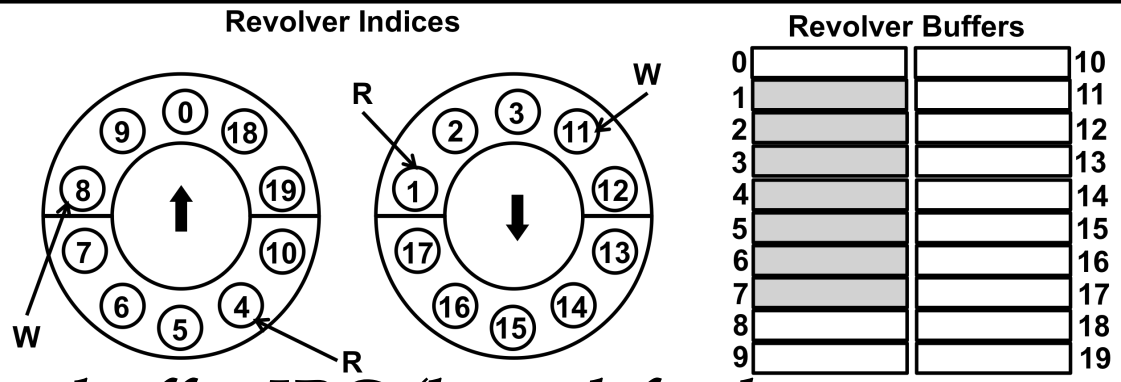
# Featherweight Processes

- *User-Mode Processes with Fast Reflexes*

- User-mode process abstraction
- Reduce copies between protection domains
- Thin syscall interface
  - Restrict blocking calls
- Proactive resource request
  - Enforce limits
- Pin down memory
- Kernel IPC – feather buffer



# Revolver Buffers




- *Feather buffer IPC (kernel-feather process communication)*
- Pair of unidirectional revolving doors in each direction
- Lock free, hold packet slots
- Writing to the buffer
  - From the kernel – while in softirq
- Reading from the buffer
  - While in the kernel – on a workqueue
- Turn buffer around – in place processing

# Technical Details

- Thin syscall interface: ffork, mmap, await
  - FFORK(feather buffer size, heap limit)
  - MMAP – maps the feather buffer
  - AWAIT – blocks feather process
- Feather buffer interface
  - put / get / peek / turnaround / available
- Peg kernel and feather process threads