Securing the Frisbee Multicast Disk Loader

Robert Ricci, Jonathon Duerig

University of Utah

What is Frisbee?

Frisbee is Emulab's tool to install whole disk images from a server to many clients using multicast

What is our goal?

Motivation

- Frisbee was developed for a relatively trusting environment
  - Existing features were to prevent accidents
- Changing Environment
  - More users
  - More sensitive experiments
  - More private images

Security Goals

- Confidentiality
- Integrity Protection
- Authentication
  - Ensure that an image is authentic
- Use cases
  - Public images
  - Private images

Our Contribution

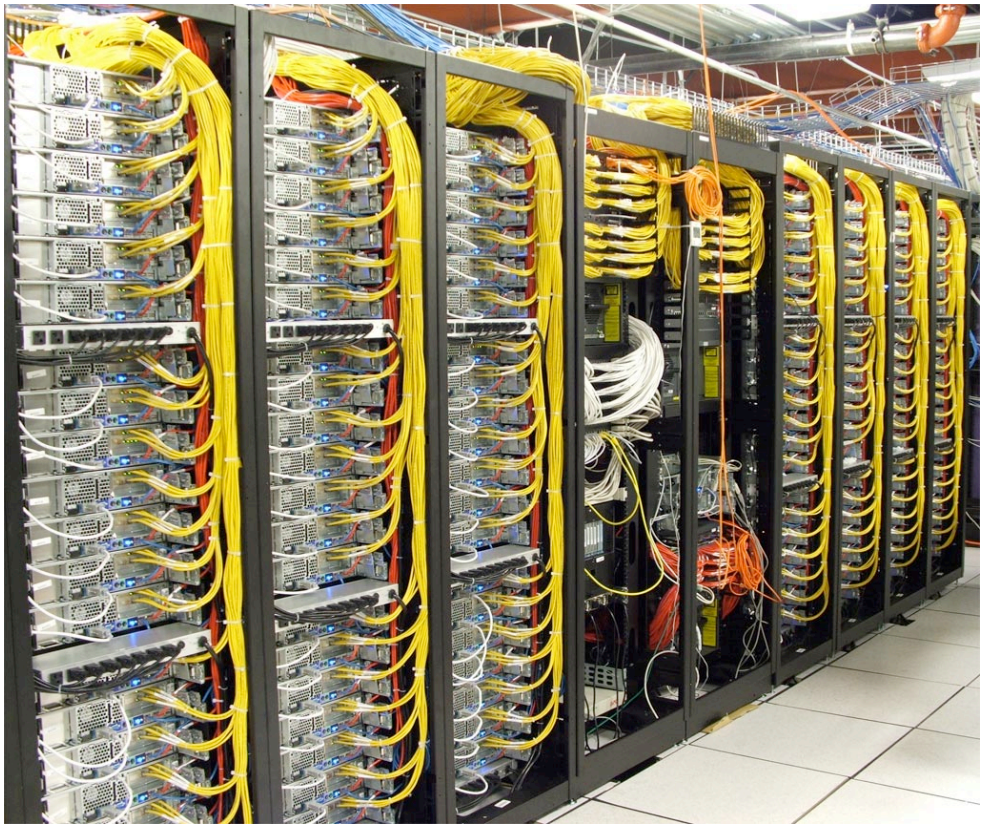
- Analyze and describe a new and interesting threat model
- Protect against those threats while preserving Frisbee's essential strengths

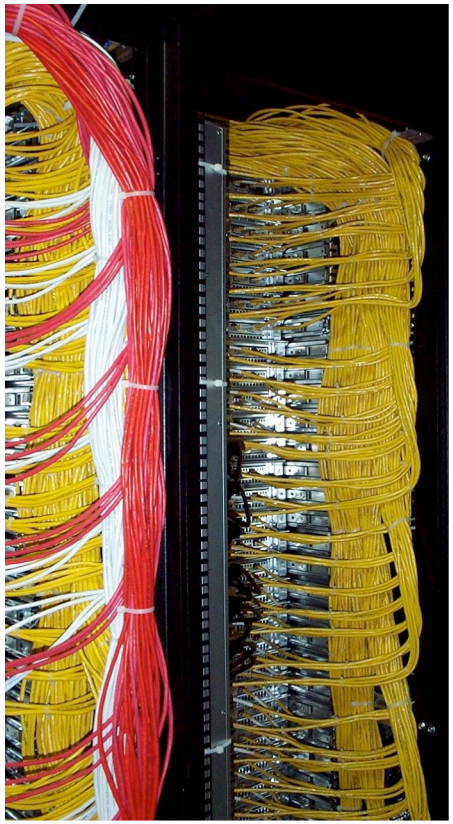
Outline

- Motivation
- **Frisbee Background**
- Threat Model
- Protecting Frisbee
- Evaluation

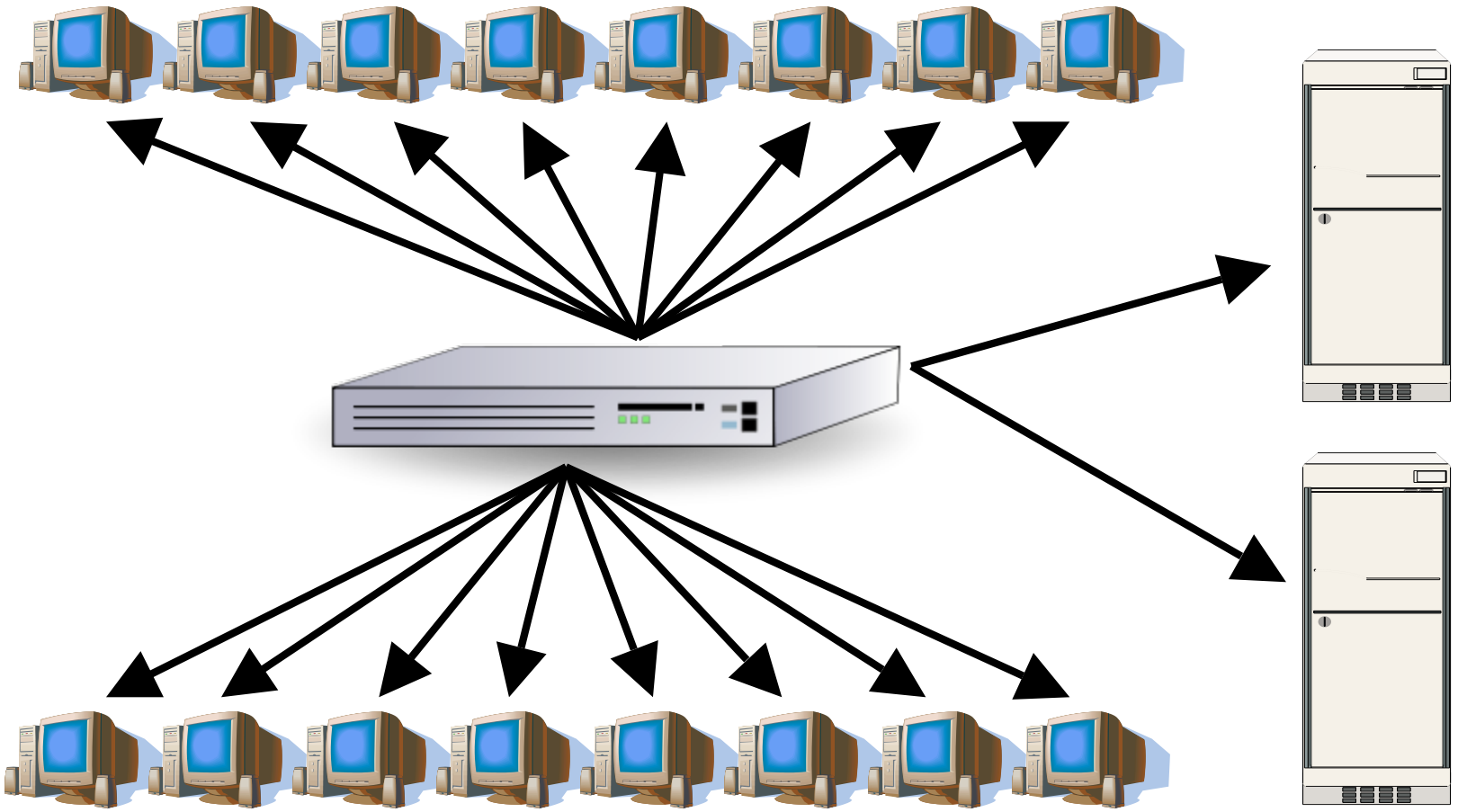
Frisbee & Emulab

Emulab





Control Plane



Frisbee’s Strengths

Frisbee's Strengths

- Disk Imaging System
  - General and versatile
  - Robust
- Fast
  - Loads a machine in 2 minutes
- Scalable
  - Loads dozens of machines in 2 minutes
- Hibler et al. (USENIX 2003)

How Does Frisbee Work?

Frisbee Life Cycle

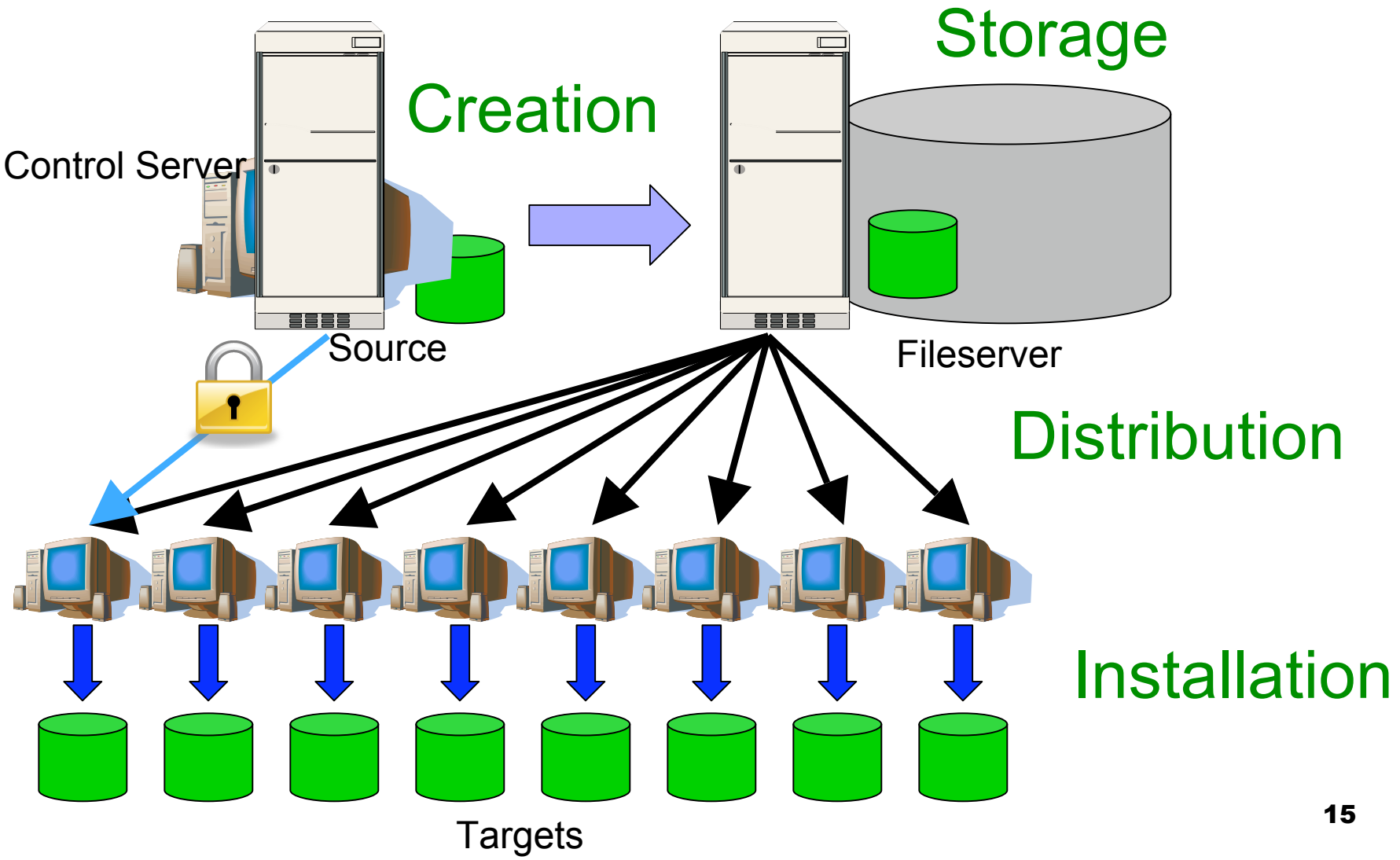
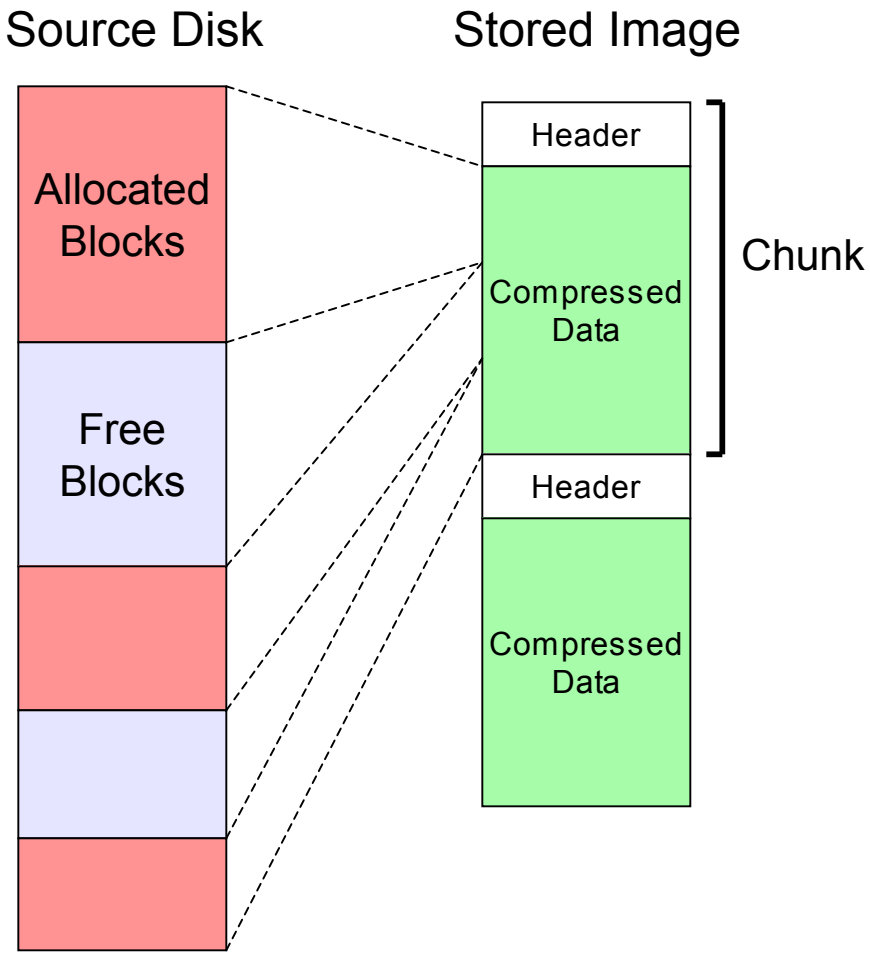


Image Layout



- Image is divide into chunks
- Each chunk is independently installable
  - Start receiving chunks at any point
  - Chunks are multicast

Outline

- Motivation
- Frisbee Background
- **Threat Model**
- Protecting Frisbee
- Evaluation

Potential Attackers

Potential Attackers

- Firewall
  - Frisbee traffic can't leave control network
  - Forged Frisbee traffic can't enter control network
- Any attackers are inside Emulab
  - Compromised Emulab node
  - Infiltrated Emulab server
  - Emulab user

Vectors for Attack in Emulab

- Space Shared
  - Multiple users on the testbed at the same time
- Shared control network
  - Frisbee runs on control network
- No software solution to limit users
  - Users have full root access to their nodes

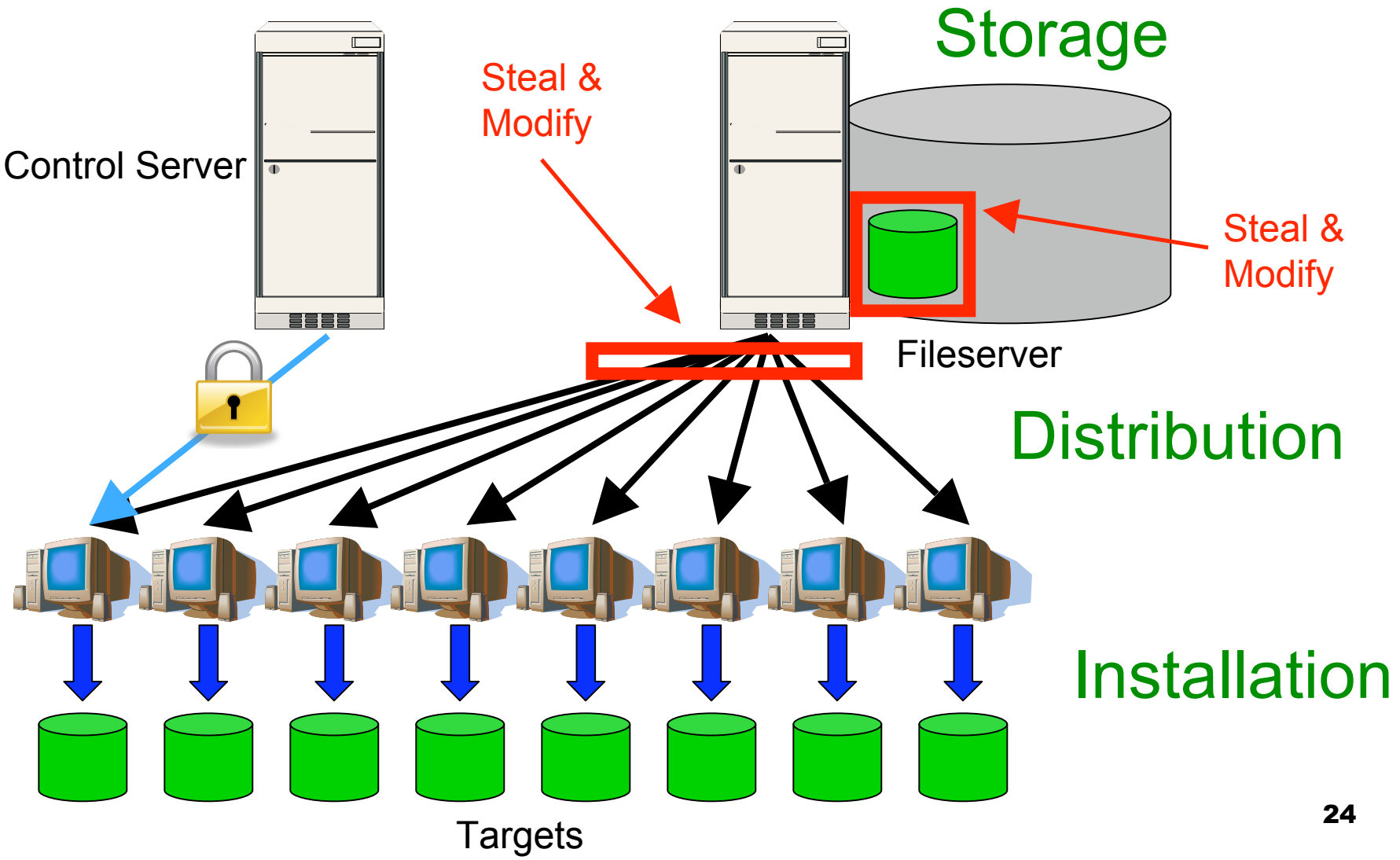
What do attackers want?

What do attackers want?

- Steal your data
  - Malicious software (security research)
  - Unreleased software (trade secrets)
- Modify your image
  - Denial of Service
  - Add a backdoor
    - /etc/passwd
    - ssh daemon
  - Tainting results

Frisbee Weakpoints

Frisbee Weakpoints



How do the attacks work?

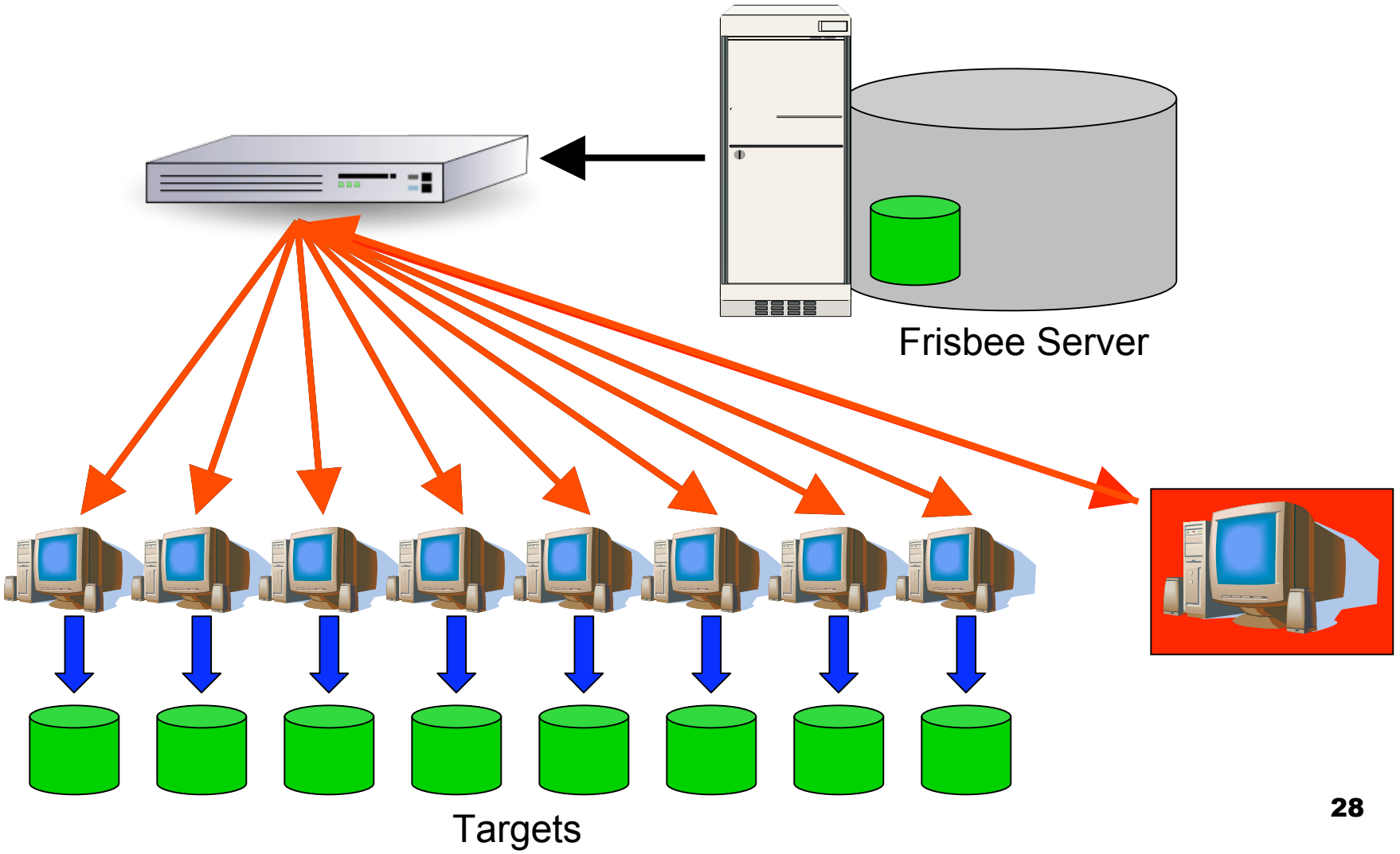
Storage Attack

- Images are stored on a common fileserver
- All users have shell access on this server
- Images are protected by UNIX permissions
- Any escalation of privilege attacks compromise images

Distribution Attack

- Emulab is space shared
- A single control network is used to communicate with all nodes
- Join multicast group
  - No security protection in IP multicast
    - Receive copies of packets
    - Inject packets into stream

Multicast



Outline

- Motivation
- Frisbee Background
- Threat Model
- **Protecting Frisbee**
- Evaluation

Storage and Distribution Attacks

- Two birds with one stone
- End-to-end encryption & authentication
  - Image creation: Encrypt & Sign
  - Image installation: Decrypt & Verify
  - Same techniques prevent both attacks
- Distribution protocol remains identical

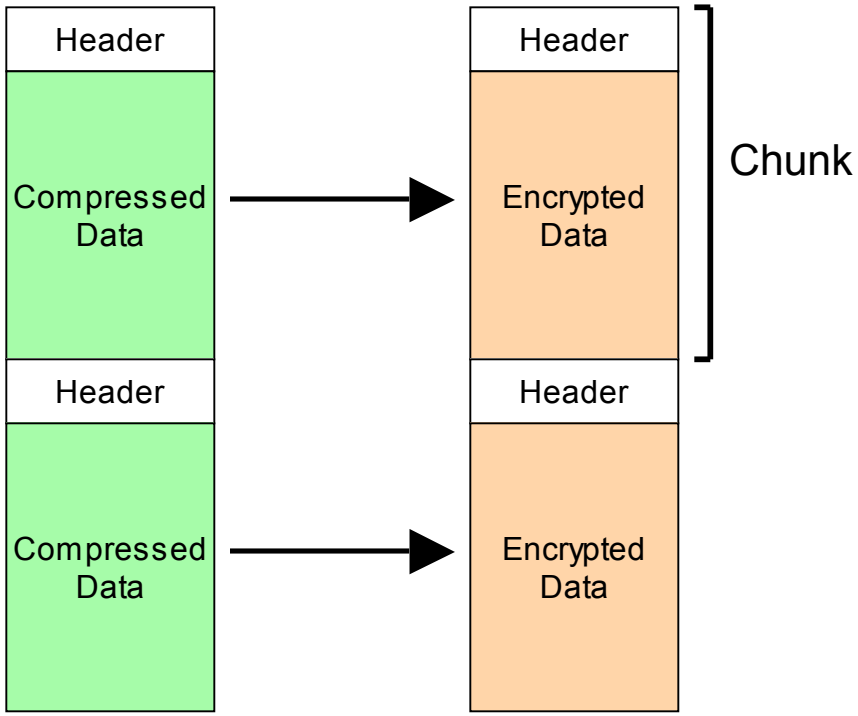
Confidentiality

- Encrypted at image creation
  - Remains encrypted on fileserver
- Decrypted only at image installation
- Details
  - Encryption algorithm: Blowfish
  - Encrypt after compression

Integrity Protection & Authentication

- Calculate cryptographic hash
  - Breaks backwards compatibility
- Sign hash using public-key cryptography (RSA)

Chunk by Chunk



- Each chunk is self-describing
- Hash & sign each chunk independently
- CBC restarts at each chunk
- Each header must have
  - Digital Signature
  - Initialization Vector

Image Authentication

- Weakness
  - Cut and paste attacks
- Give each image a unique UUID and put that in chunk headers
  - UUID is a 128 bit universal identifier
  - Can be selected randomly

Key Distribution

- Through secure control channel
  - Already part of Emulab
  - Encrypted using SSL with well-known certificate
  - TCP spoofing prevented by Utah Emulab's network setup
    - No forged MAC addresses
    - No forged IP addresses
- Key can come from user
  - Flexible policy for images
- Not yet integrated into Emulab

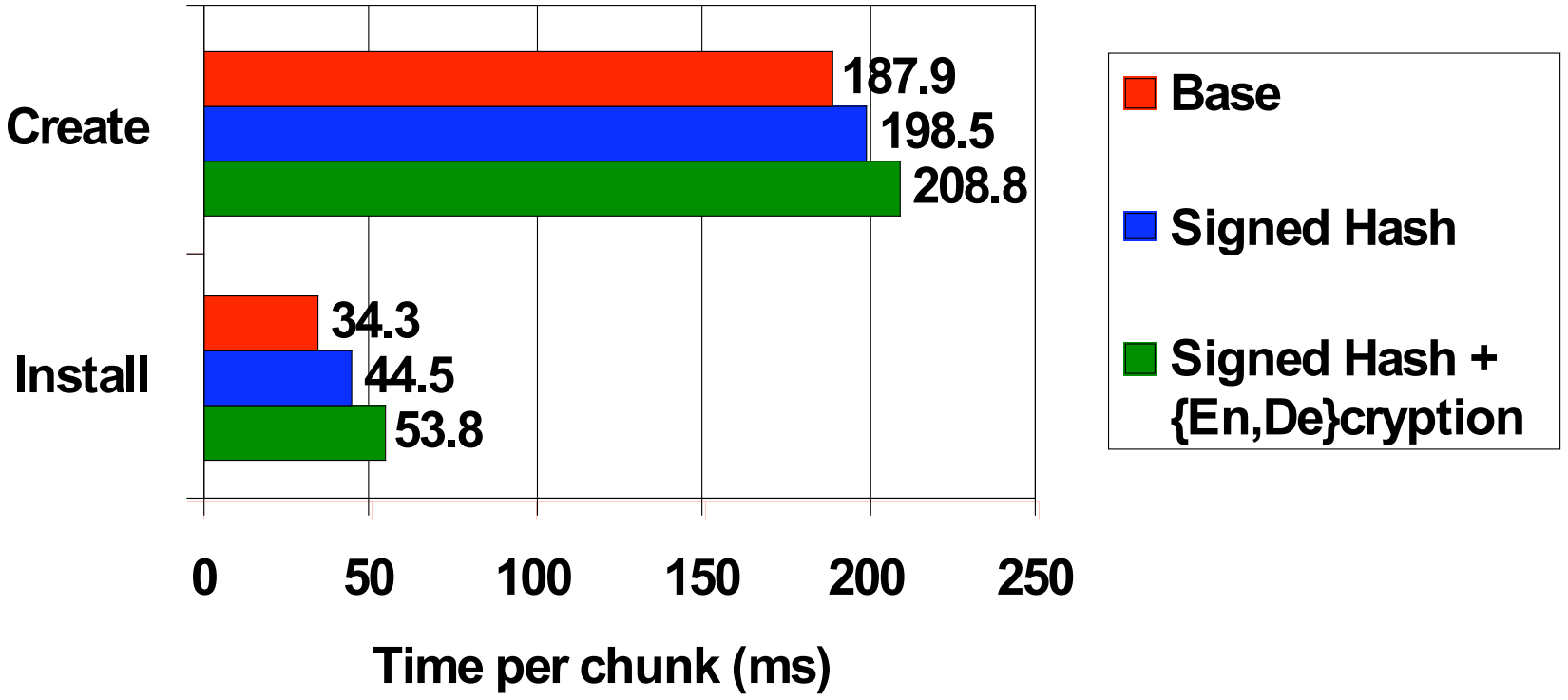
Outline

- Motivation
- Frisbee Background
- Threat Model
- Protecting Frisbee
- **Evaluation**

Experimental Procedure

- Machine Specs
  - 3 GHz Pentium IV Xeon
  - 2 GB RAM
- Measurement
  - CPU time
    - Network and disk usage unaffected
  - Per chunk
    - Typical Image has 300 chunks (300 MB)

Performance



Conclusion

Conclusion

- Frisbee faces an unusual set of attacks
  - Cause: Space sharing of infrastructure
- Frisbee can be secured against these attacks
  - Cost: An extra 6 seconds for an average image

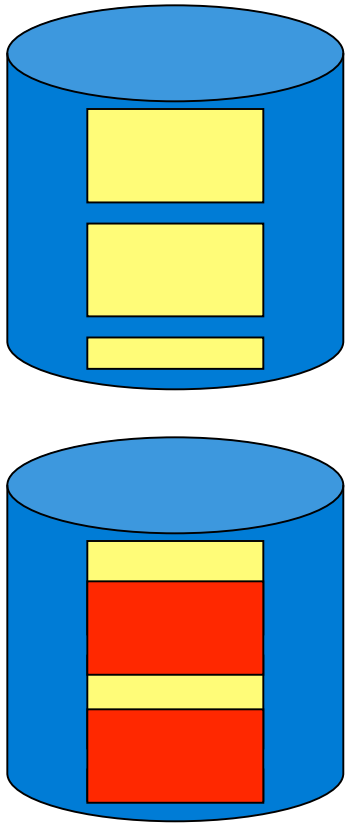
Emulab

http://www.emulab.net

Preventing Disk Leakage

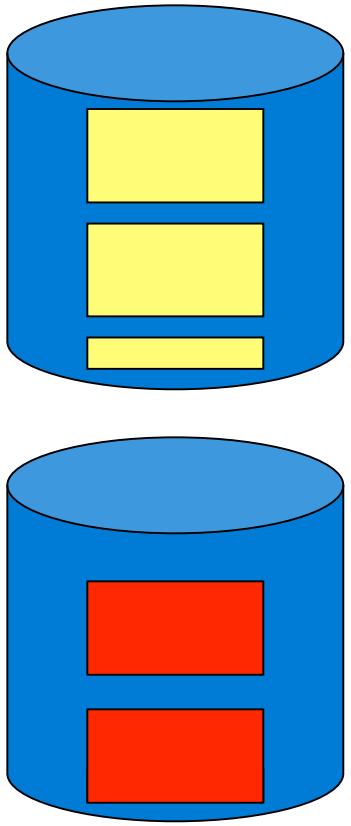
Disk Leakage

- Disks are time shared
- Frisbee is aware of filesystem
  - Does not write free blocks
  - Old image will not be completely overwritten
- Another user could read the unwritten parts



Fixing Disk Leakage

- Zero out disks on next disk load
- Implemented in Frisbee
  - Much slower



Comparison to Symantec Ghost

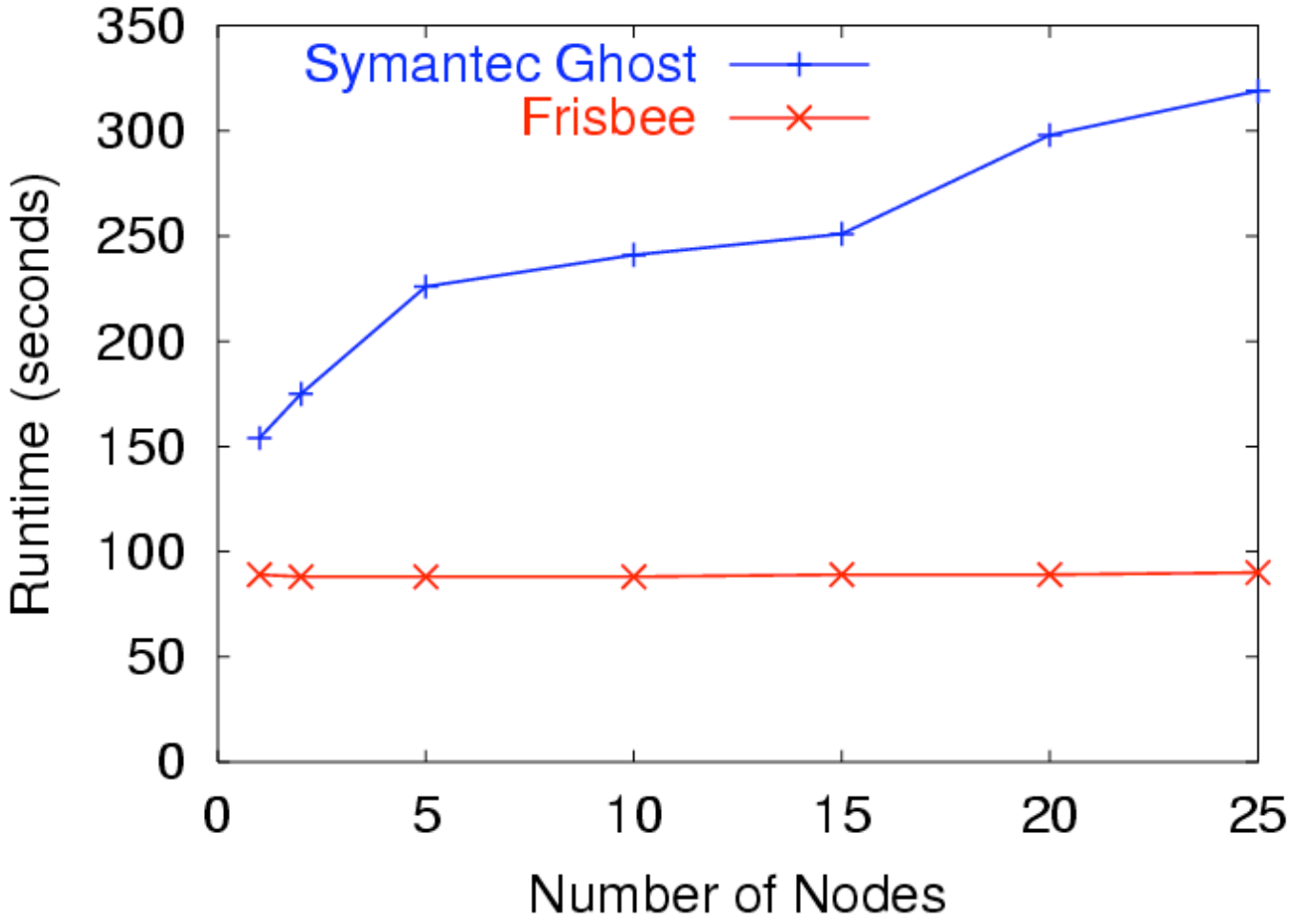


Image Creation (CPU per chunk)

	Time (ms)	Overhead (ms)	Overhead (%)
Base	187.9		
Signed Hash	198.5	10.5	5.6%
Signed Hash + Encryption	208.8	20.9	11.1%

Image Installation (CPU per chunk)

	Time (ms)	Overhead (ms)	Overhead (%)
Base	34.3		
Signed Hash	44.5	10.2	29.5%
Signed Hash + Decryption	53.8	19.5	56.8%

Disk Imaging Matters

- Data on a disk or partition, rather than file, granularity
- Uses
  - OS installation
  - Catastrophe recovery
- Environments
  - Enterprise
  - Clusters
  - Utility computing
  - Research/education environments

Key Design Aspects

- Domain-specific data compression
- Two-level data segmentation
- LAN-optimized custom multicast protocol
- High levels of concurrency in the client

Image Creation

- Segments images into self-describing "chunks"
- Compresses with zlib
- Can create "raw" images with opaque contents
- Optimizes some common filesystems
  - ext2, FFS, NTFS
  - Skips free blocks

Image Distribution Environment

- LAN environment
  - Low latency, high bandwidth
  - IP multicast
  - Low packet loss
- Dedicated clients
  - Consuming all bandwidth and CPU OK

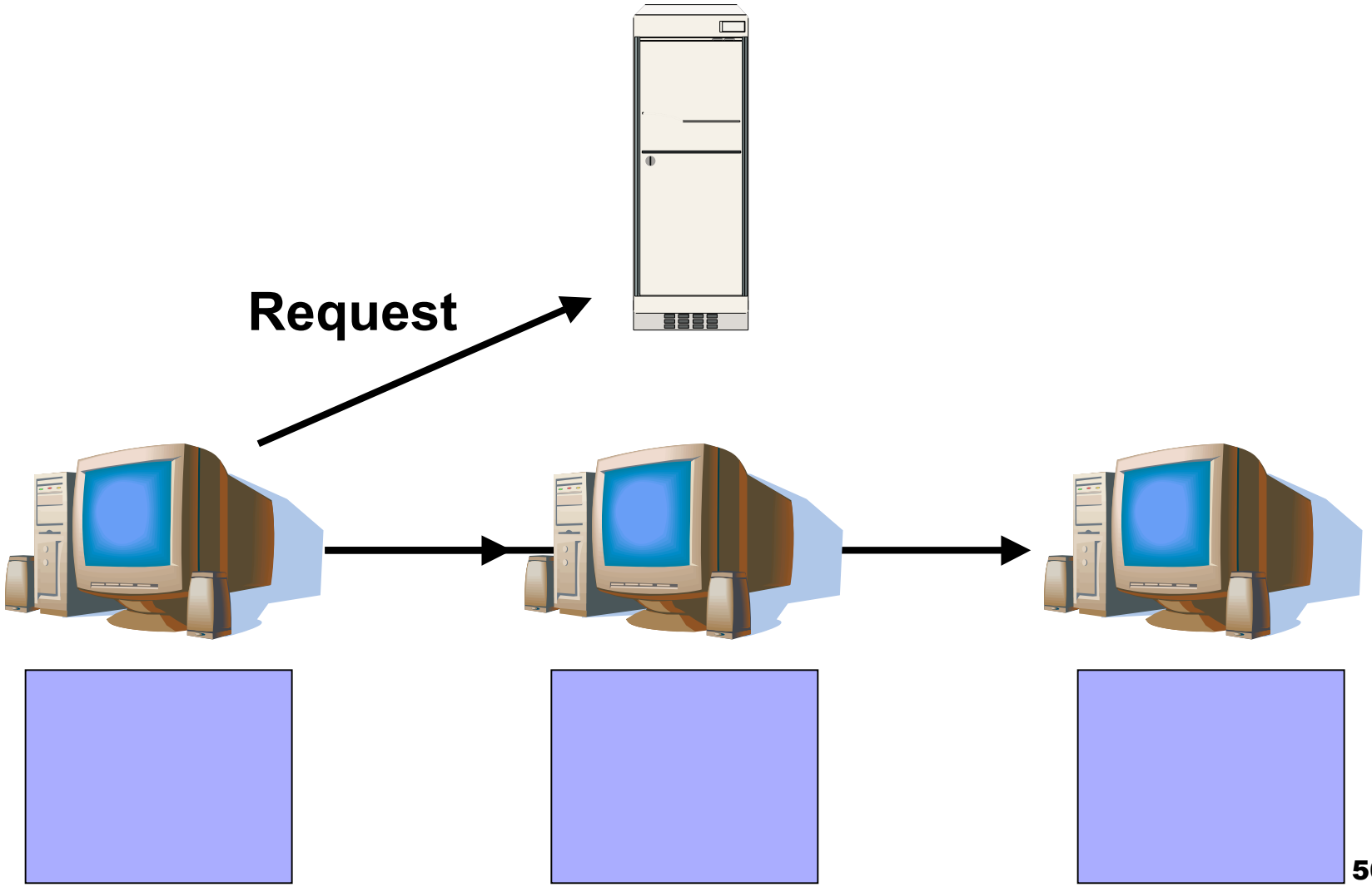
Custom Multicast Protocol

- Receiver-driven
  - Server is stateless
  - Server consumes no bandwidth when idle
- Reliable, unordered delivery
- “Application-level framing”
- Requests block ranges within 1MB chunk

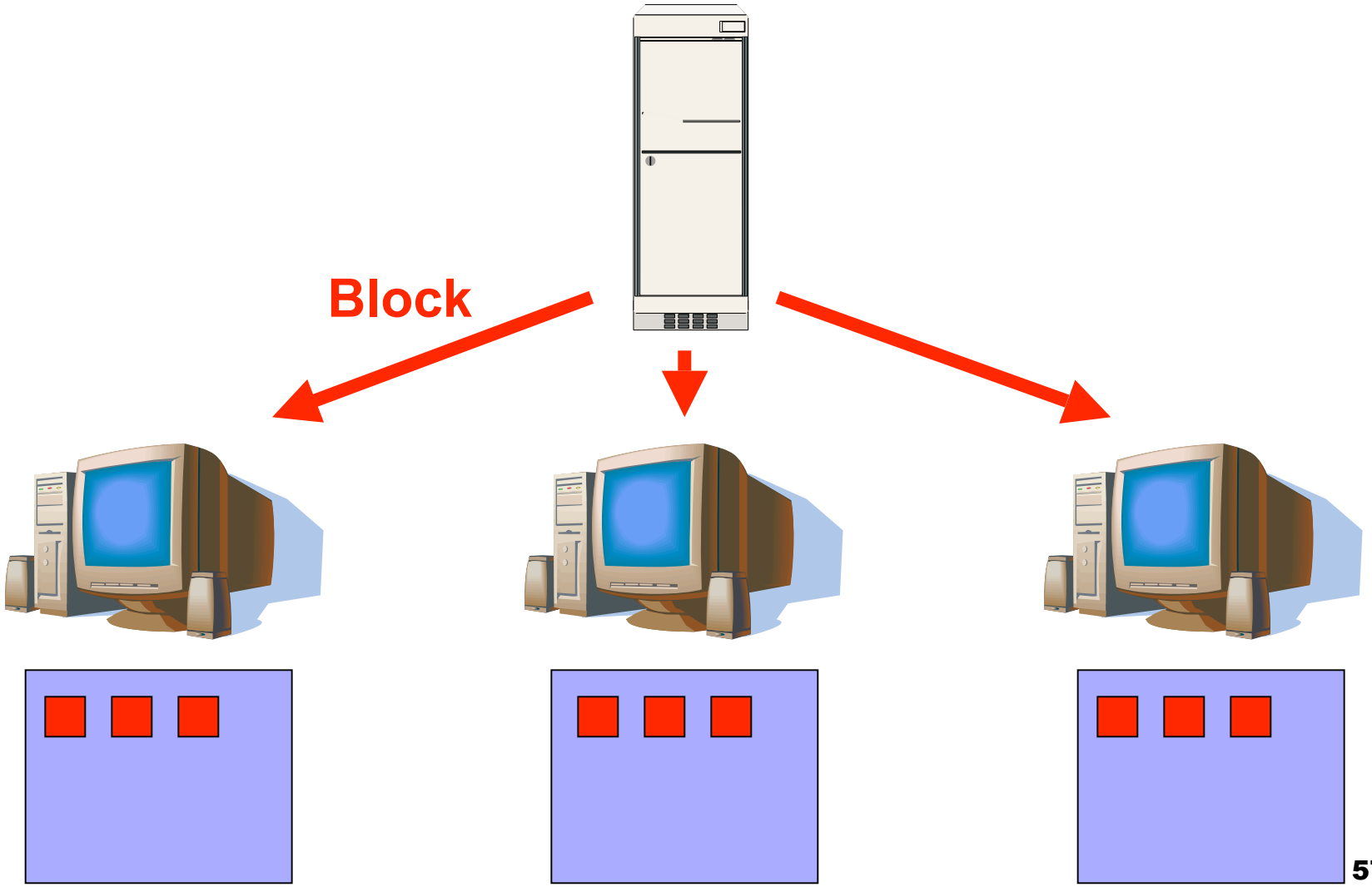
Client Operation

- Joins multicast channel
  - One per image
- Asks server for image size
- Starts requesting blocks
  - Requests are multicast
- Client start not synchronized

Client Requests



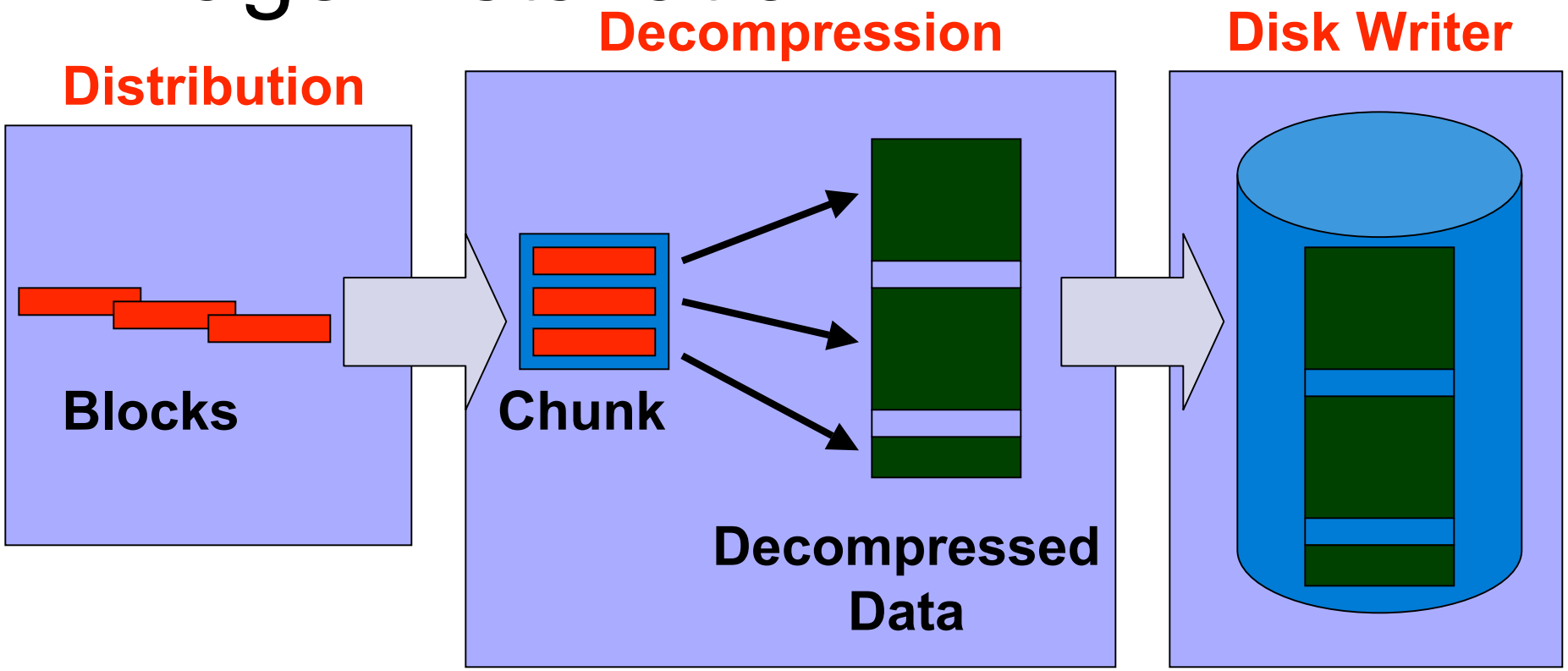
Client Requests



Tuning is Crucial

- Client side
  - Timeouts
  - Read-ahead amount
- Server side
  - Burst size
  - Inter-burst gap

Image Installation



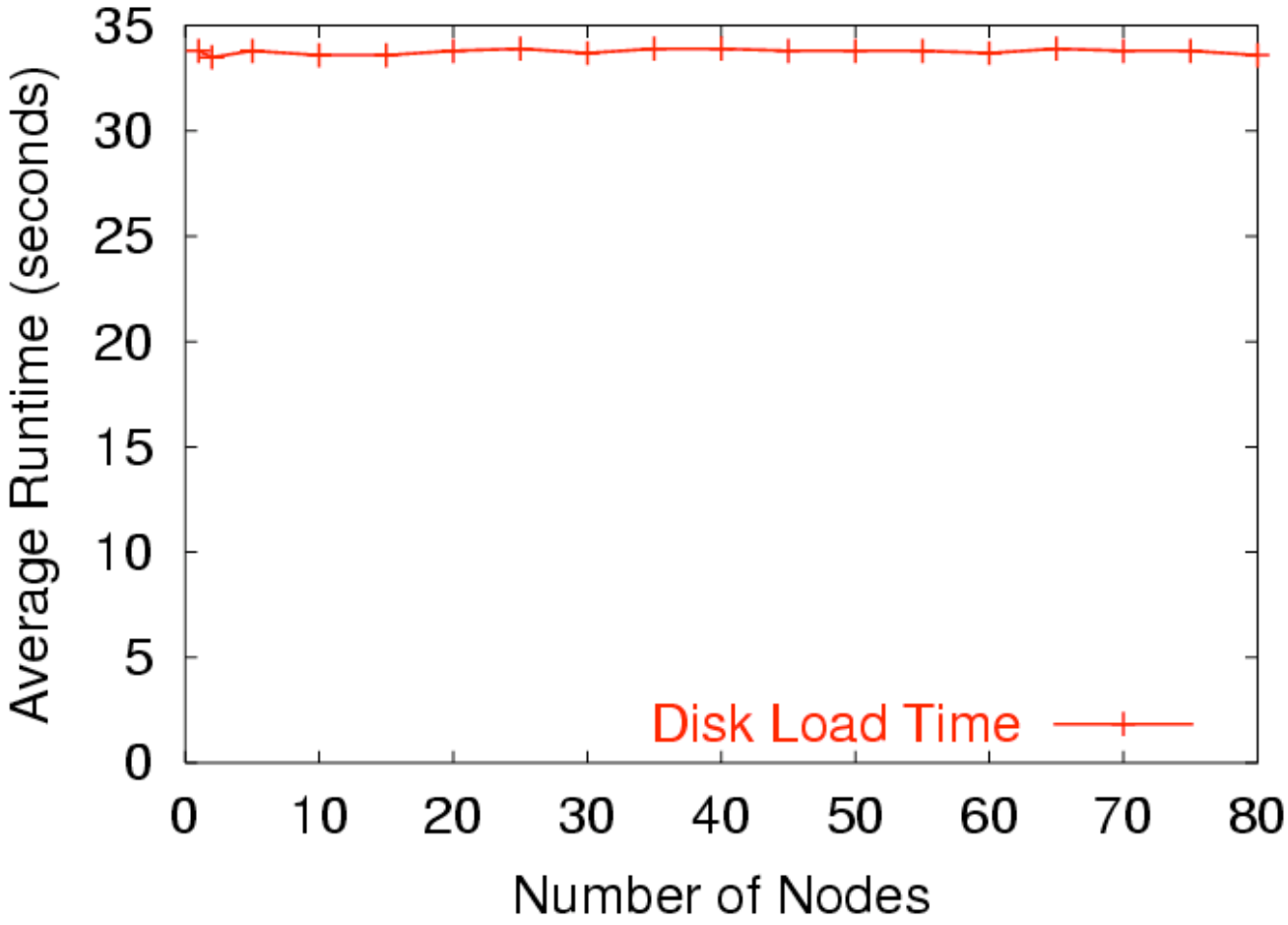
- Pipelined with distribution
  - Can install chunks in any order
  - Segmented data makes this possible
- Three threads for overlapping tasks
- Disk write speed the bottleneck
- Can skip or zero free blocks

Evaluation

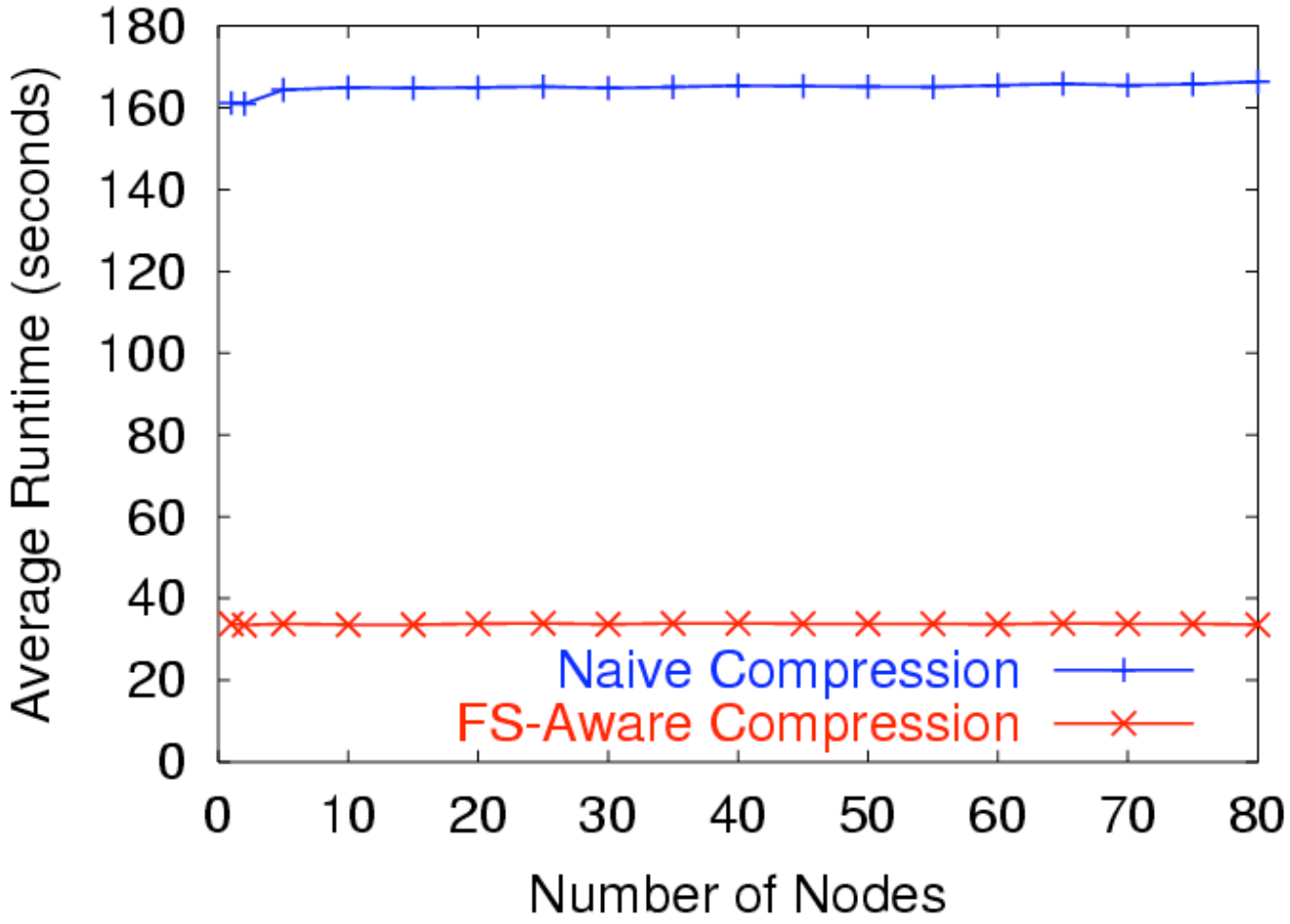
Performance

- Disk image
  - FreeBSD installation used on Emulab
  - 3 GB filesystem, 642 MB of data
  - 80% free space
  - Compressed image size is 180 MB
- Client PCs
  - 850 MHz CPU, 100 MHz memory bus
  - UDMA 33 IDE disks, 21.4 MB/sec write speed
  - 100 Mbps Ethernet, server has Gigabit

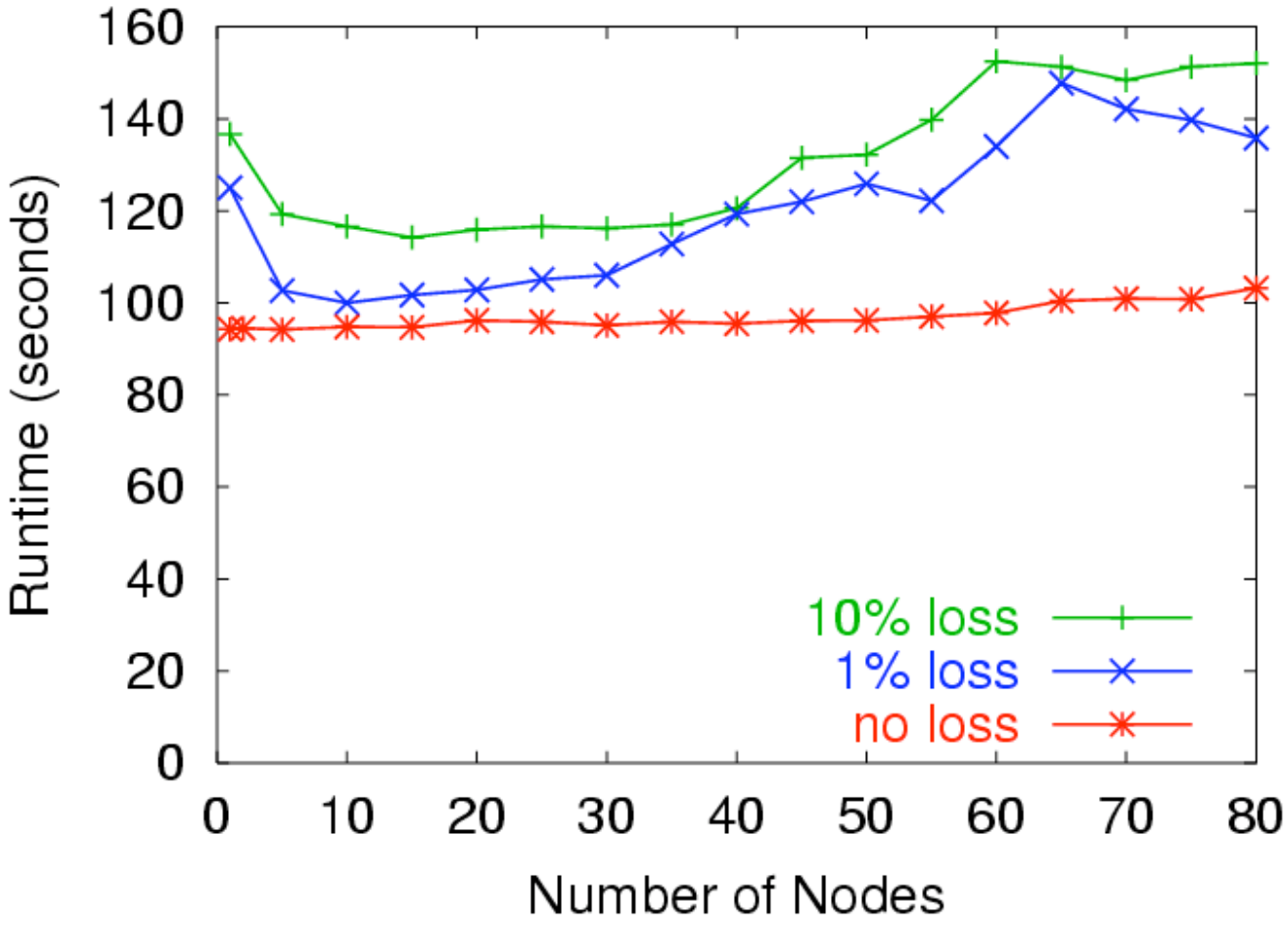
Speed and Scaling



FS-Aware Compression



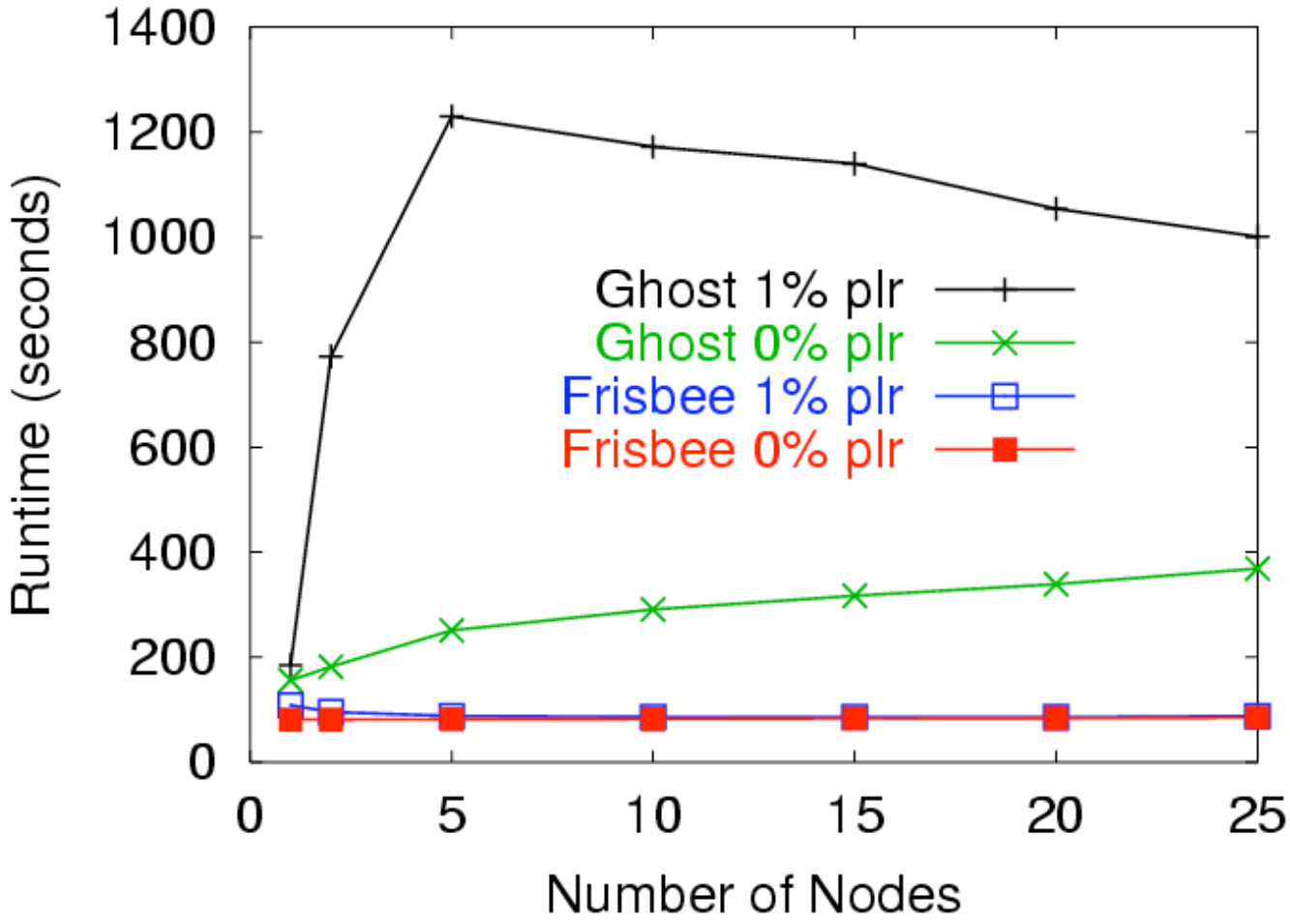
Packet Loss


Runtime (seconds)
Number of Nodes
0
20
40
60
80
100
120
140
160
0
10
20
30
40
50
60
70
80
10% loss
1% loss
no loss

Related Work

- Disk imagers without multicast
  - Partition Image [www.partimage.org]
- Disk imagers with multicast
  - PowerQuest Drive Image Pro
  - Symantec Ghost
- Differential Update
  - rsync 5x slower with secure checksums
- Reliable multicast
  - SRM [Floyd ’97]
  - RMTP [Lin ’96]

Ghost with Packet Loss



How Frisbee Changed our Lives (on Emulab, at least)

- Made disk loading between experiments practical
- Made large experiments possible
  - Unicast loader maxed out at 12
- Made swapping possible
  - Much more efficient resource usage

The Real Bottom Line

"I used to be able to go to lunch while I loaded a disk, now I can't even go to the bathroom!"

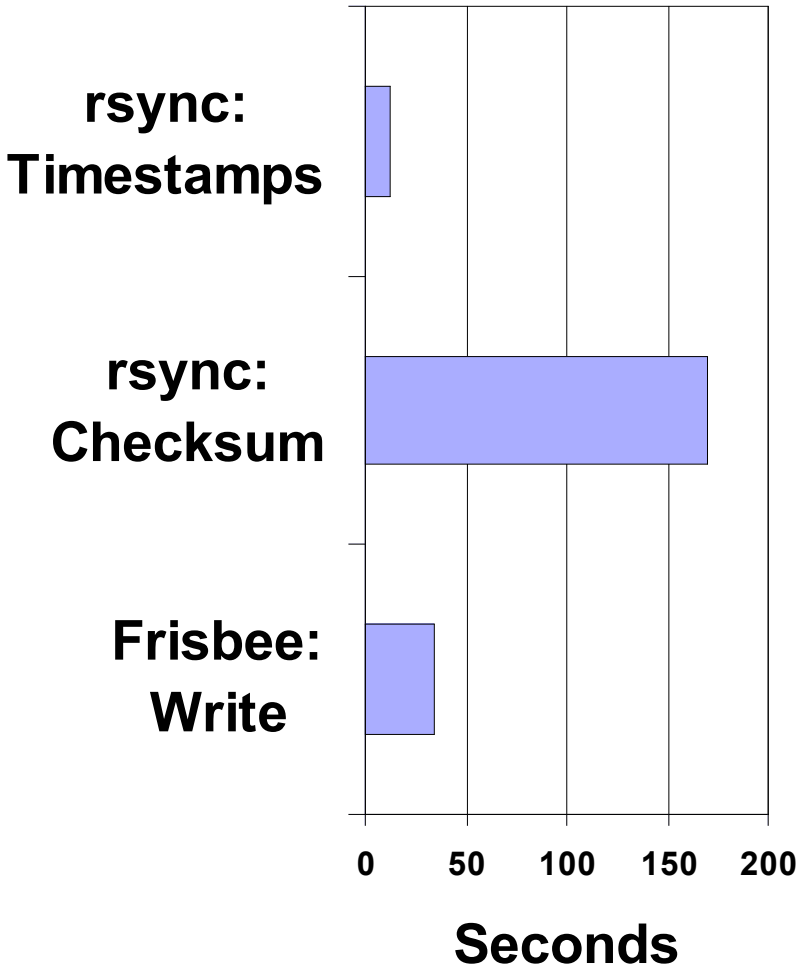
- Mike Hibler (first author)

Conclusion

- Frisbee is
  - Fast
  - Scalable
  - Proven
- Careful domain-specific design from top to bottom is key

Source available at www.emulab.net

Comparison to rsync



- Timestamps not robust
- Checksums slow
- Conclusion: Bulk writes beat data comparison

How to Synchronize Disks

- Differential update - rsync
  - Operates through filesystem
  - + Only transfers/writes changes
  - + Saves bandwidth
- Whole-disk imaging
  - Operates below filesystem
  - + General
  - + Robust
  - + Versatile
- Whole-disk imaging essential for our task

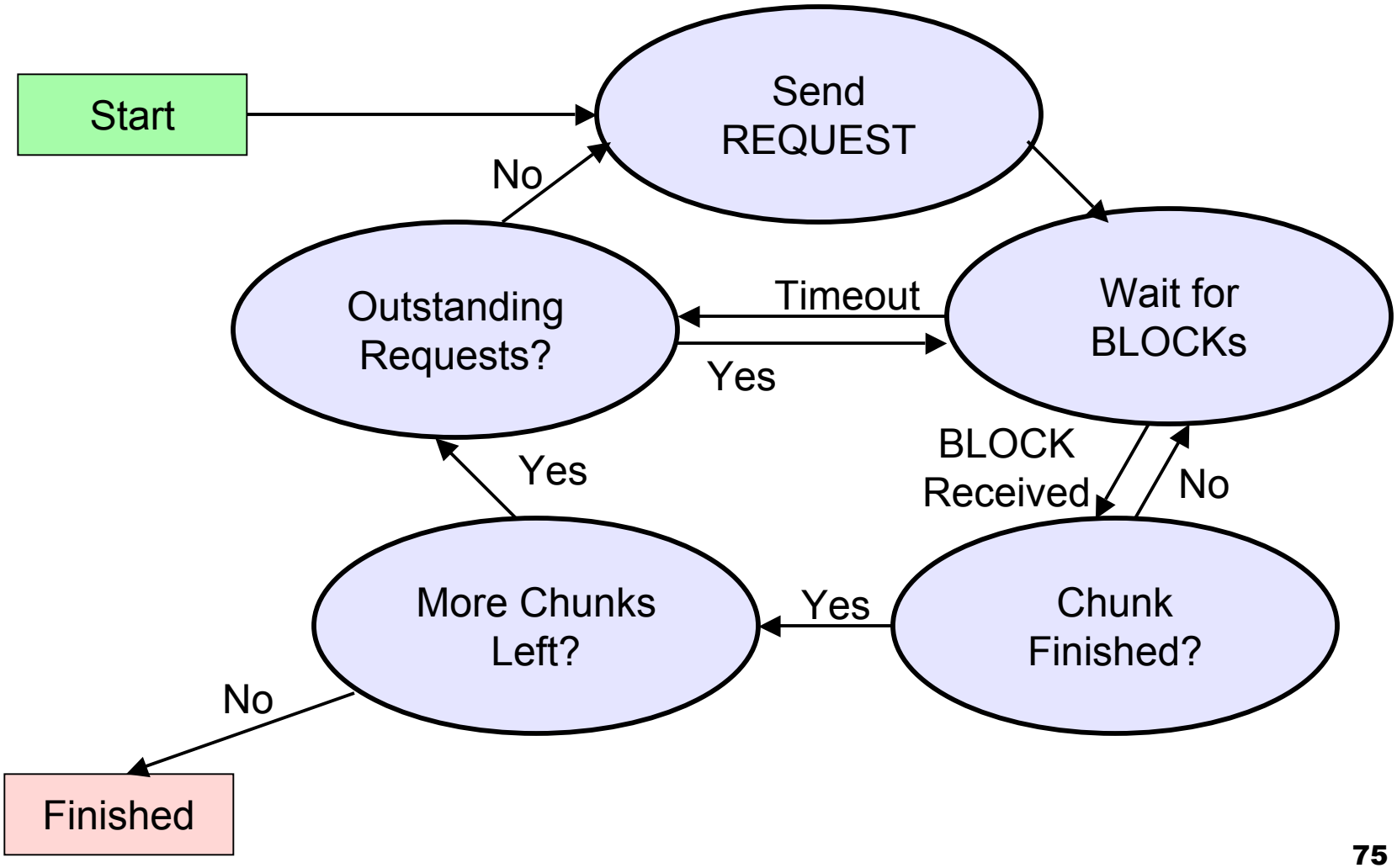
Image Distribution Performance: Skewed Starts

Startup Scenario	Runtime (s) Ave	Range	Client msgs	Dup Data
Small Image				
Simultaneous	33.6	32.9–34.7	2753	3.2%
Clustered	35.6	33.2–40.3	4561	46%
Uniform	40.0	34.5–51.0	7875	59%
Large Image				
Simultaneous	100.2	100–101	12772	7.3%
Clustered	113.3	106–126	17266	26%
Uniform	132.4	120–147	23842	37%

Future

- Server pacing
- Self tuning

The Frisbee Protocol



The Evolution of Frisbee

- First disk imager: Feb, 1999
- Started with NFS distribution
- Added compression
  - Naive
  - FS-aware
- Overlapping I/O
- Multicast

30 minutes down to 34 seconds!