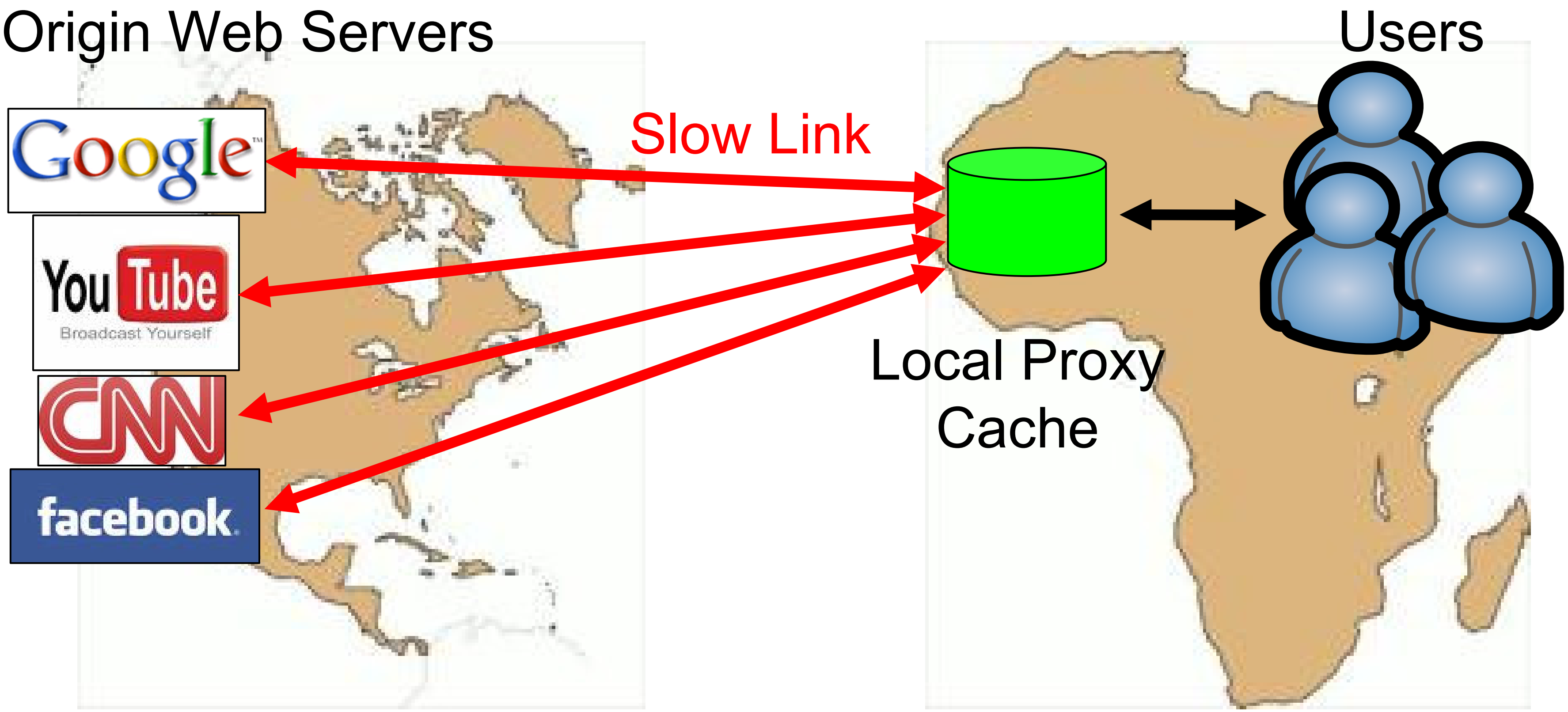
Wide-area Network Acceleration for the Developing World

Sunghwan Ihm (Princeton)
KyoungSoo Park (KAIST)
Vivek S. Pai (Princeton)

POOR INTERNET ACCESS IN THE DEVELOPING WORLD

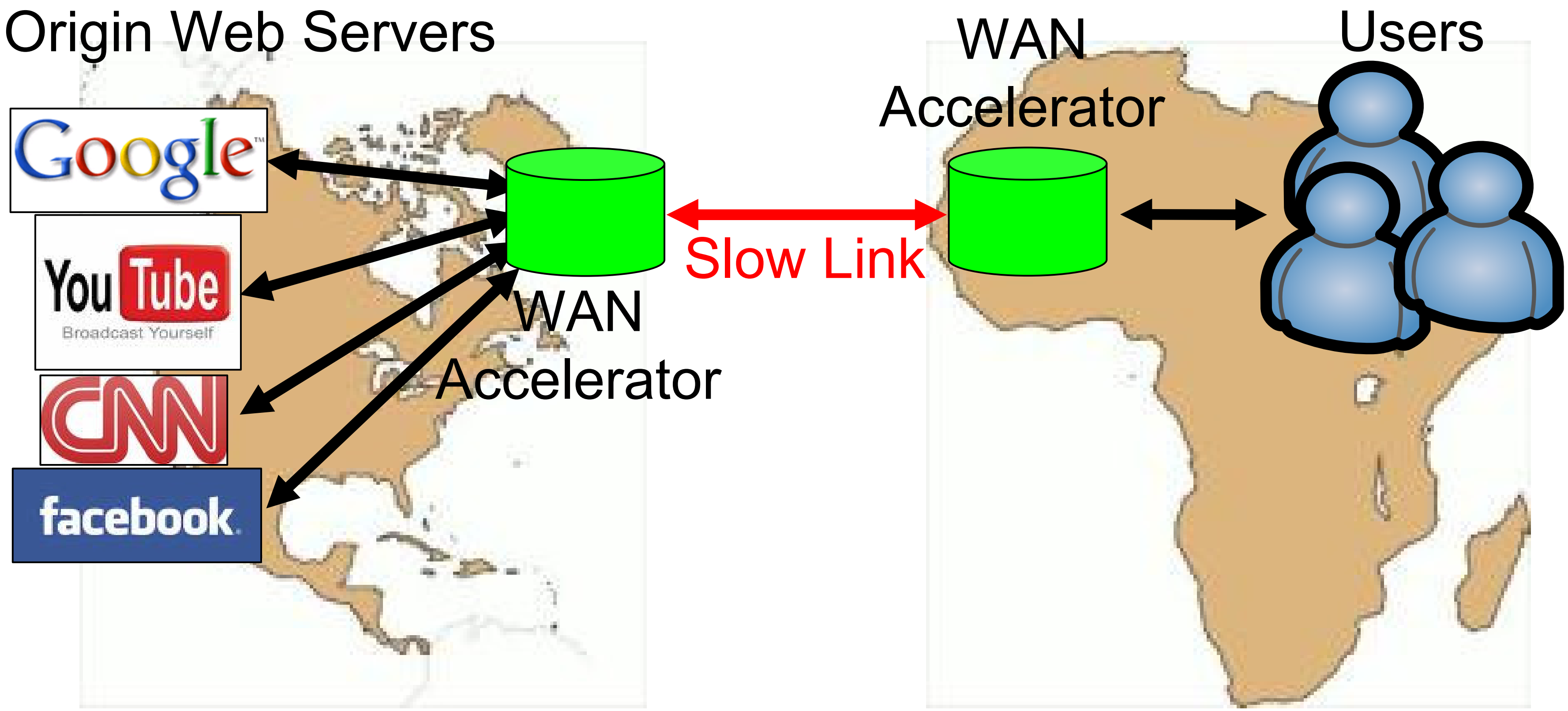
- Internet access is a scarce commodity in the developing regions
  - Expensive / slow
- Zambia example [Johnson et al. NSDR'10]
  - 300 people share 1Mbps satellite link
  - $1200 per month

WEB PROXY CACHING IS NOT ENOUGH



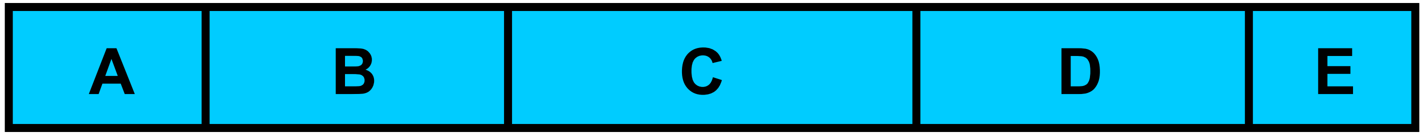
- Whole objects, designated cacheable traffic only
- Zambia: 43% cache hit rate, 20% byte hit rate

WAN ACCELERATION: FOCUS ON CACHE MISSES



- Packet-level (chunks) caching
- Mostly for enterprise
- Two (or more) endpoints, coordinated

CONTENT FINGERPRINTING



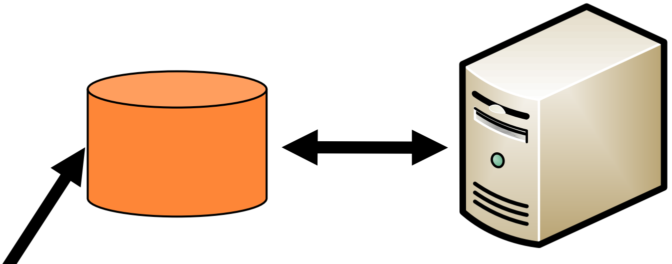
- Split content into chunks
  - Rabin's fingerprinting over a sliding window
  - Match n low-order bits of a global constant K
  - Average chunk size: 2^n bytes
- Name chunks by content (SHA-1 hash)
- Cache chunks and pass references

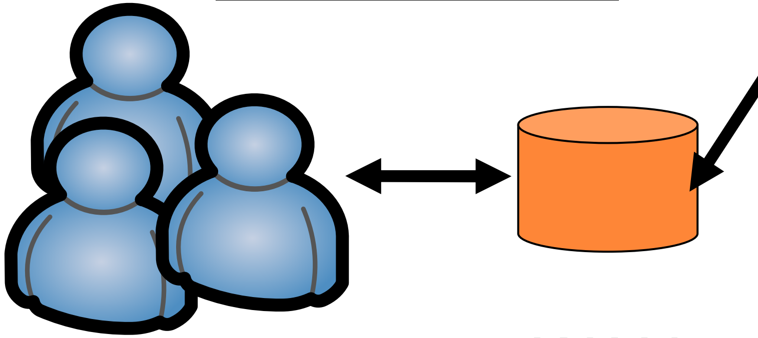
WHERE TO STORE CHUNKS

- Chunk data
  - Usually stored on disk
  - Can be cached in memory to reduce disk access
- Chunk metadata index
  - Name, offset, length, etc.
  - Partially or completely kept in memory to minimize disk accesses

HOW IT WORKS

1. Users send requests
2. Get content from Web server
3. Generate chunks
4. Send cached chunk names + uncached raw content (compression)





1. Fetch chunk contents from local disk (cache hits)
2. Request any cache misses from sender-side node
3. As we assemble, send it to users (cut-through)

WAN ACCELERATOR PERFORMACE

- Effective bandwidth (throughput)
  - Original data size / total time
- Transfer: send data + refs
  - Compression rate **High**
- Reconstruction: hits from cache
  - Disk performance **High**
  - Memory pressure **Low**

DEVELOPING WORLD CHALLENGES/OPPORTUNITES

Enterprise

- Dedicated machine with ample RAM
- High-RPM SCSI disk
- Inter-office content only
- Star topology

Developing World

- Shared machine with limited RAM
- Slow disk
- All content
- Mesh topology

Poor Performance!

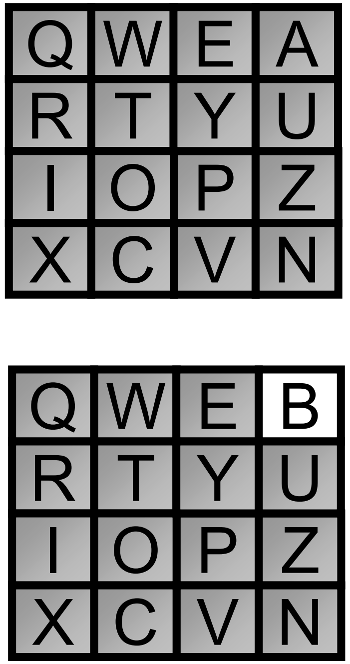
vs.

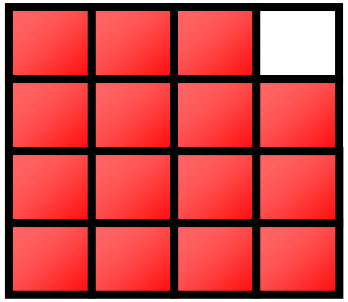
WANAX: HIGH-PERFORMANCE WAN ACCELERATOR

- Design Goals
  - Maximize compression rate
  - Minimize memory pressure
  - Maximize disk performance
  - Exploit local resources

- Contributions
  - Multi-Resolution Chunking (MRC)
  - Peering
  - Intelligent Load Shedding (ILS)

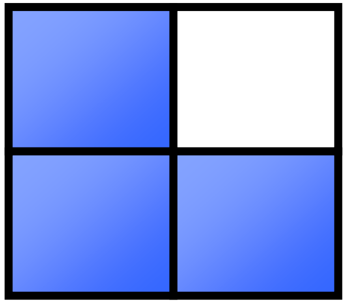
SINGLE RESOLUTION CHUNKING (SRC) TRADEOFFS





93.75% saving
15 disk reads
15 index entries

High compression rate
High memory pressure
Low disk performance

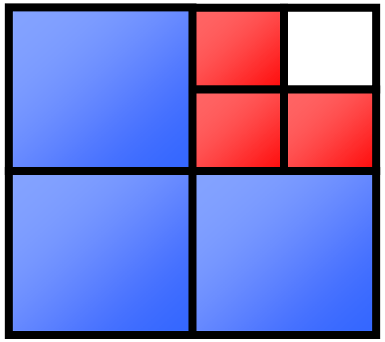


75% saving
3 disk reads
3 index entries

Low compression rate
Low memory pressure
High disk performance

MULTI-RESOLUTION CHUNKING (MRC)

- Use multiple chunk sizes simultaneously

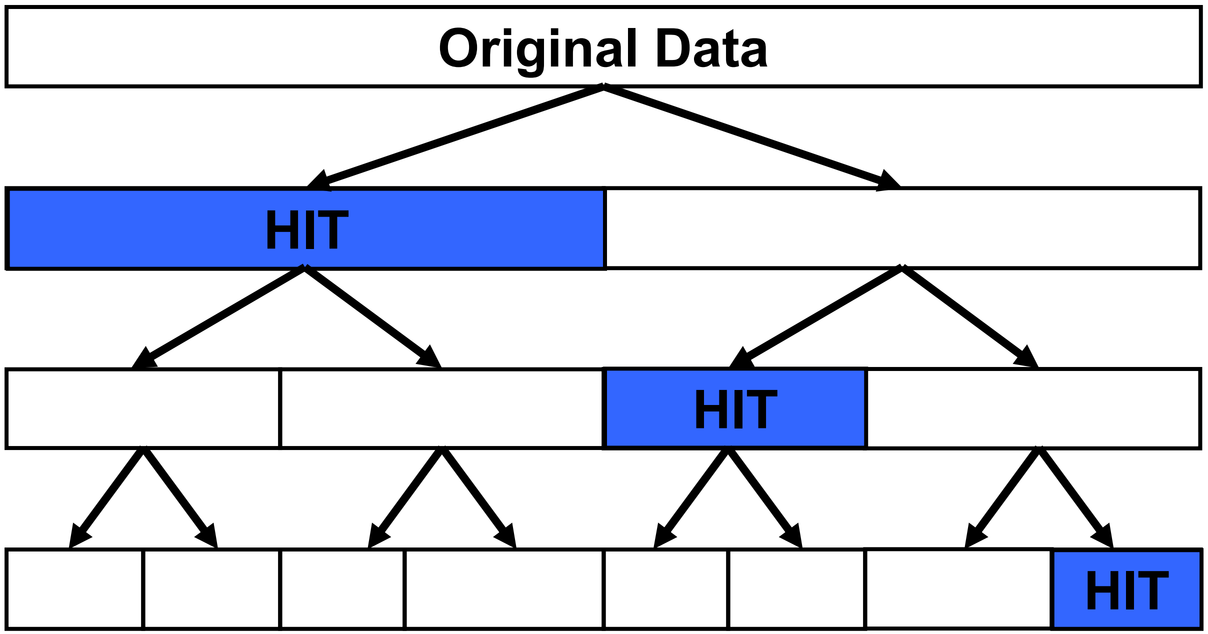


93.75% saving
6 disk reads
6 index entries

- Large chunks for low memory pressure and disk seeks

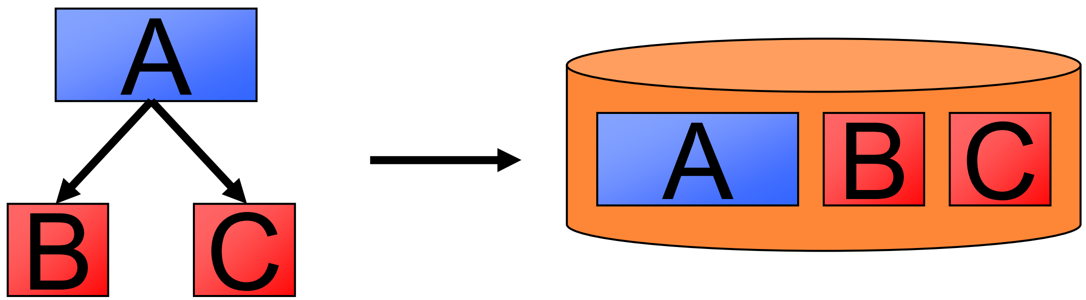
- Small chunks for high compression rate

GENERATING MRC CHUNKS



- Detect smallest chunk boundaries first
- Larger chunks are generated by matching more bits of the detected boundaries
- Clean chunk alignment + less CPU

STORING MRC CHUNKS



- Store every chunk regardless of content overlaps
  - No association among chunks
  - One index entry load + one disk seek
  - Reduce memory pressure and disk seeks
  - Disk **space** is cheap, disk **seeks** are limited

*Alternative storage options in the paper

PEERING

- Cheap or free local networks (ex: wireless mesh, WiLDNet)
  - Multiple caches in the same region
  - Extra memory and disk
- Use Highest Random Weight (HRW)
  - Robust to node churn
  - Scalable: no broadcast or digests exchange
  - Trade CPU cycles for low memory footprint

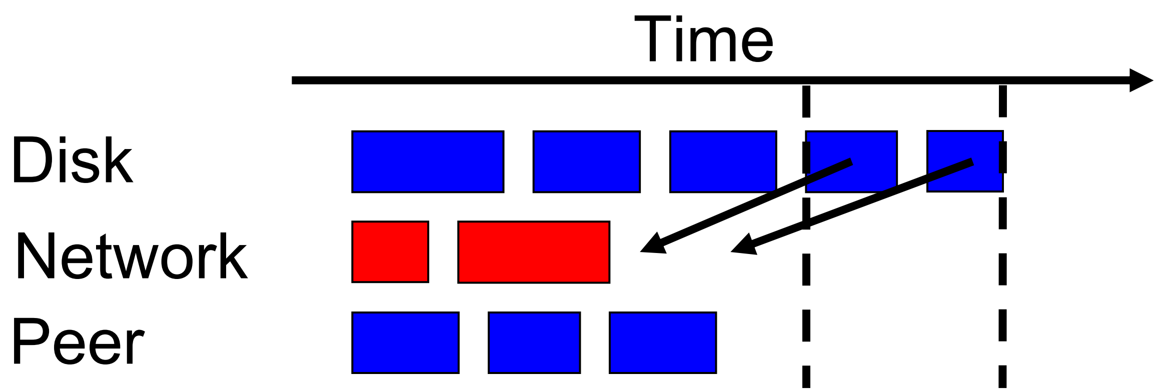
INTELLIGENT LOAD SHEDDING (ILS)

- Two sources of fetching chunks
  - Cache hits from disk
  - Cache misses from network

- Fetching chunks from disks is not always desirable
  - Disk heavily loaded (shared/slow)
  - Network underutilized

- Solution: adjust network and disk usage dynamically to maximize throughput

SHEDDING SMALLEST CHUNK

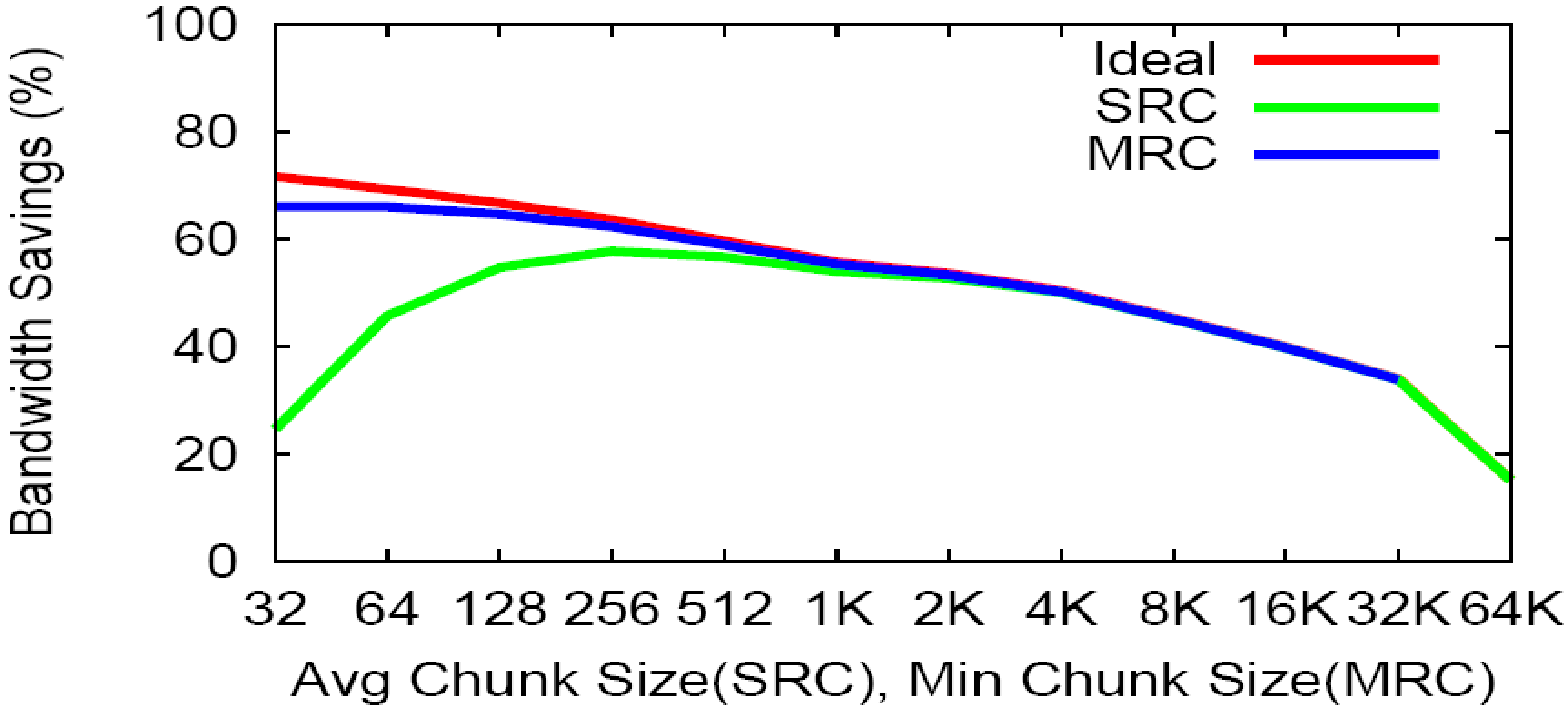


- Move the **smallest** chunks from disk to network, until network becomes the bottleneck
- Disk → one seek regardless of chunk size
- Network → proportional to chunk size
- Total latency → max(disk, network)

SIMULATION ANALYSIS

- News Sites
  - Web browsing of dynamically generated Web sites (1GB)
  - Refresh the front pages of 9 popular Web sites every five minutes
  - CNN, Google News, NYTimes, Slashdot, etc.
- Linux Kernel
  - Large file downloads (276MB)
  - Two different versions of Linux kernel source tar file
- Bandwidth savings, and # disk reads

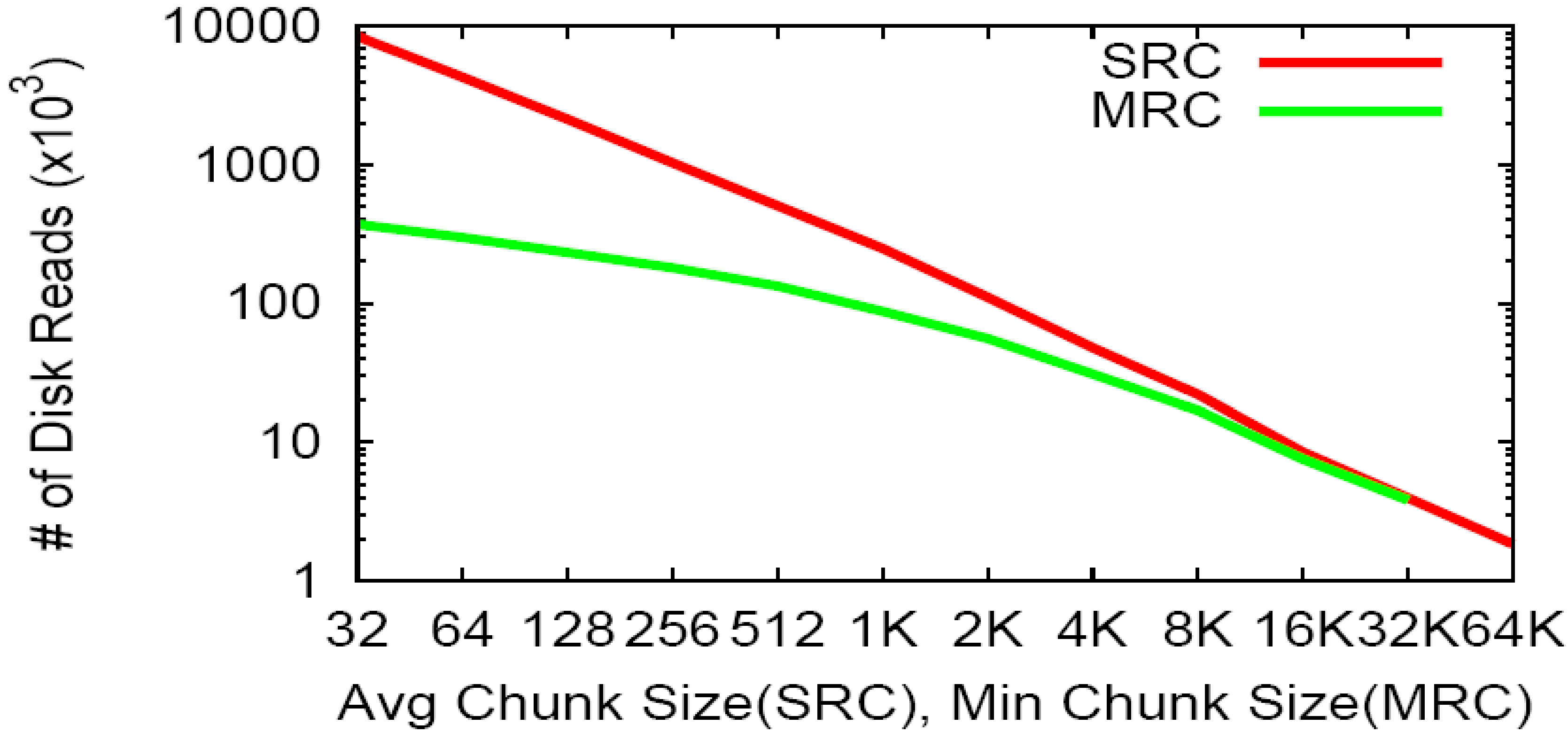
BANDWITH SAVINGS



(a) News Sites

- SRC: high metadata overhead
- MRC: close to ideal bandwidth savings

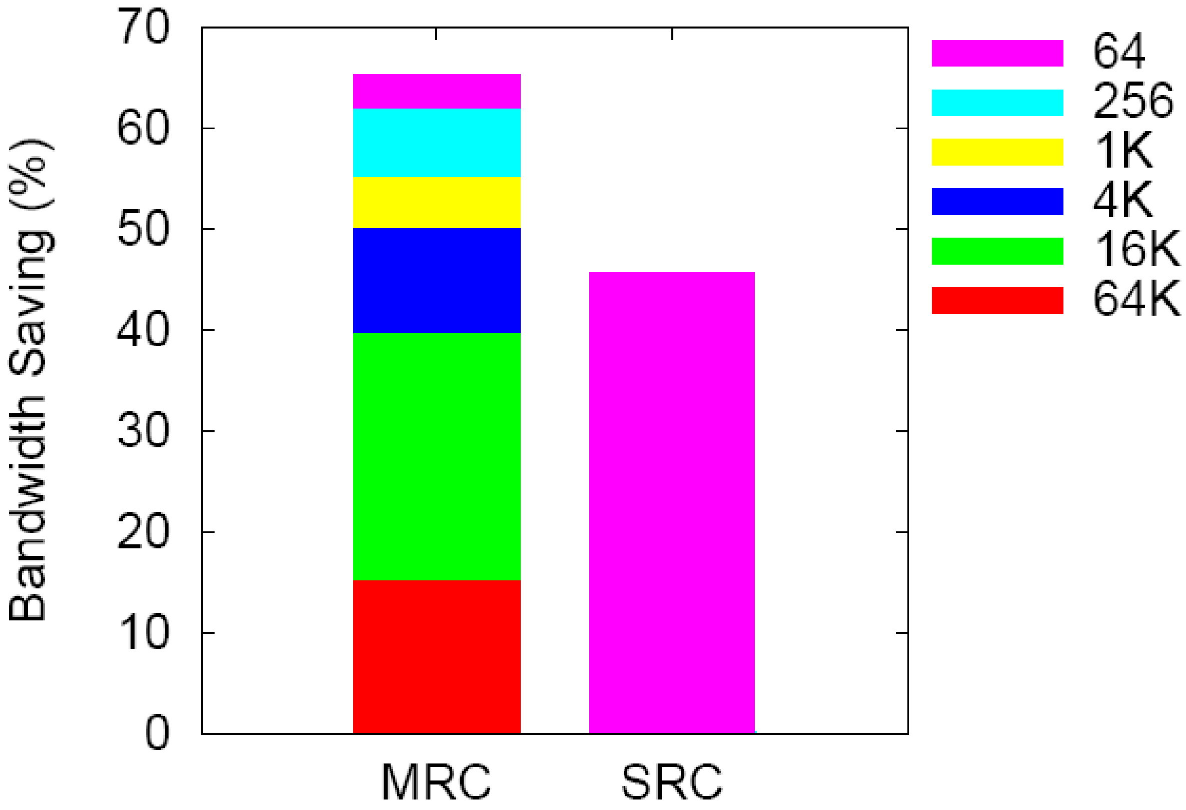
DISK FETCH COST



(b) Linux Kernel

- MRC greatly reduces # of disk seeks
- 22.7x at 32 bytes

CHUNK SIZE BREAKDOWN

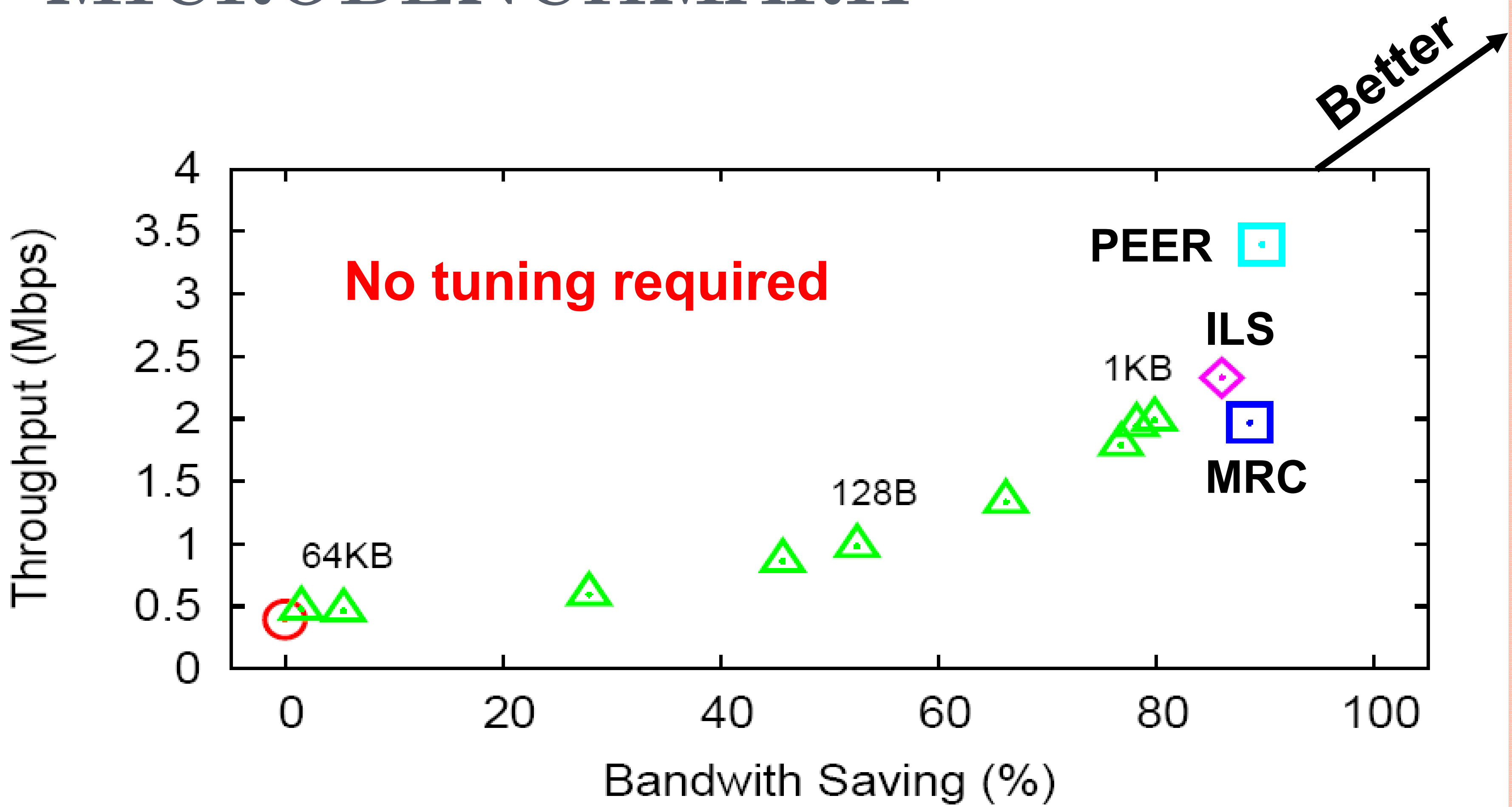


- SRC: high metadata overhead
- MRC: much less # of disk seeks and index entries (40% handled by large chunks)

EVALUATION

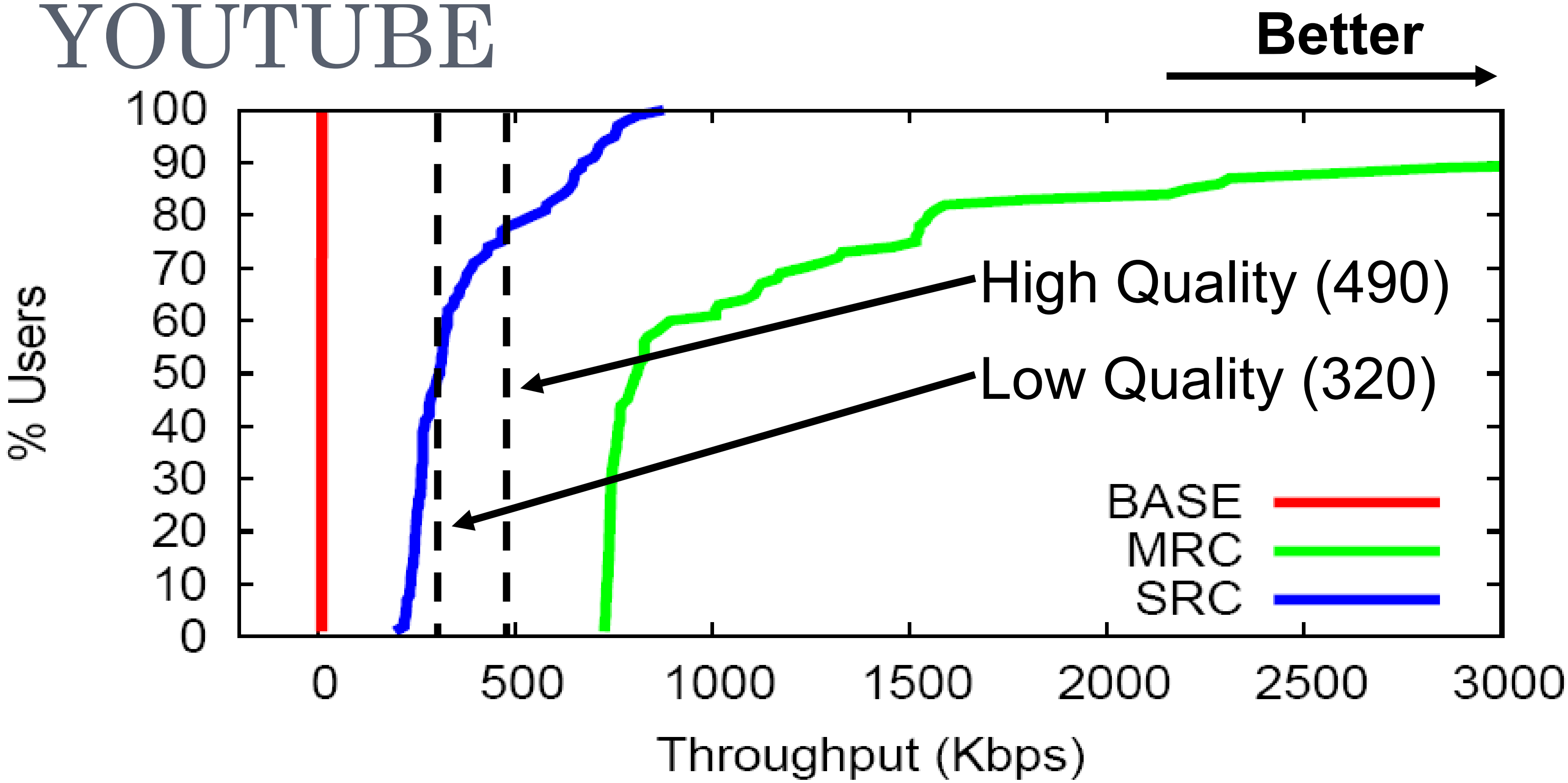
- Implementation
  - Single-process event-driven architecture
  - 18000 LOC in C
  - HashCache (NSDI'09) as chunk storage
- Intra-region bandwidths 100Mbps
- Disable in-memory cache for content
  - Large working sets do not fit in memory
- Machines
  - AMD Athlon 1GHz / 1GB RAM / SATA
  - Emulab pc850 / P3-850 / 512MB RAM / ATA

MICROBENCHMARK



- 90% similar 1 MB files
- 512kbps / 200ms RTT

REALISTIC TRAFFIC: YOUTUBE



- Classroom scenario
- 100 clients download 18MB clip
- 1Mbps / 1000ms RTT

IN THE PAPER

- Enterprise test
  - MRC scales to high performance servers
- Other storage options
  - MRC has the lowest memory pressure
  - Saving disk space increases memory pressure
- More evaluations
  - Overhead measurement
  - Alexa traffic

CONCLUSIONS

- **Wanax**: scalable / flexible WAN accelerator for the developing regions
- **MRC**: high compression / high disk performance / low memory pressure
- **ILS**: adjust disk and network usage dynamically for better performance
- **Peering**: aggregation of disk space, parallel disk access, and efficient use of local bandwidth

sihm@cs.princeton.edu

http://www.cs.princeton.edu/~sihm/