CS 349C
Reducing Network Traffic Redundancy, Riverbed patent

Debangsu Sengupta
Michael Chan
10/6/2009

Outline

- Paper
  - Motivation, Contributions
  - Architecture
  - Scheme
  - Experimental Setup
  - Results
  - Thoughts/analysis
- Patent
  - Hierarchical – fixed/variable
  - Applications – Client/server, Near line file system

Motivation

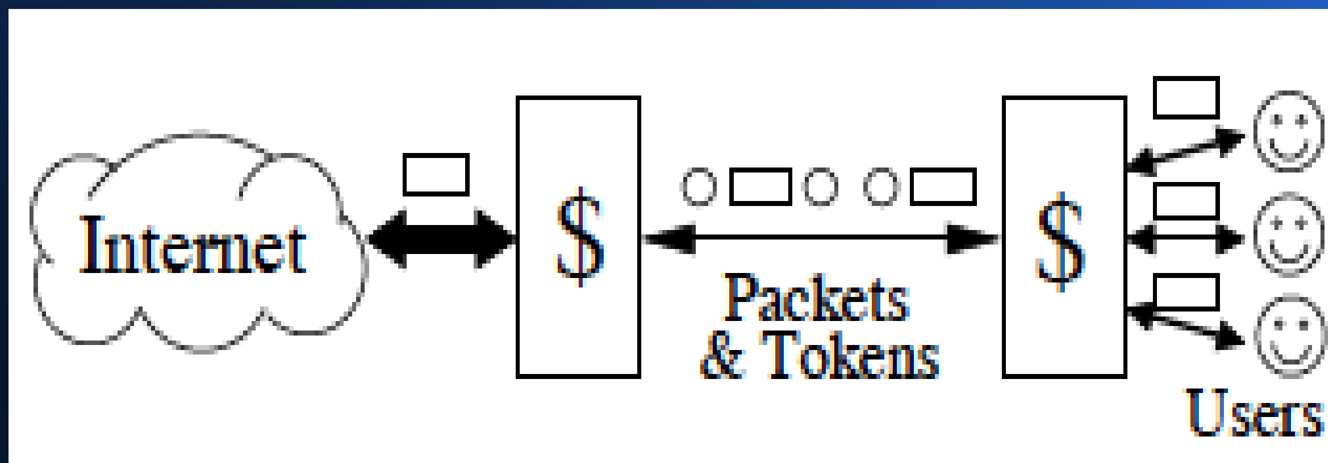
- Goal - Protect slow links
  - Data centers communicating over expensive links
  - Client/server accelerators (e.g. dial-up/mobile)
- Web caches
  - Optimize on static content. Peak hit rate ~45%
  - Name based : coarse-grained redundancy detection
- Dynamic content can have redundancy
  - Database queries, dynamic pages, searches, etc.
- Reduce traffic by exploiting content redundancy

Contributions

- Show via analysis that caching is not sufficient
- Protocol-independent scheme for reducing traffic
- Sub-packet-level redundancy suppression algorithm
  - Contrast with delta-encoding, duplicate suppression, e2e compression
- Complements caching and compression

Architecture

- Consistent cache on both sides of slow link
- Caches implement redundancy suppression
- Optional compression via zlib, etc.



Duplicate Packet Suppression

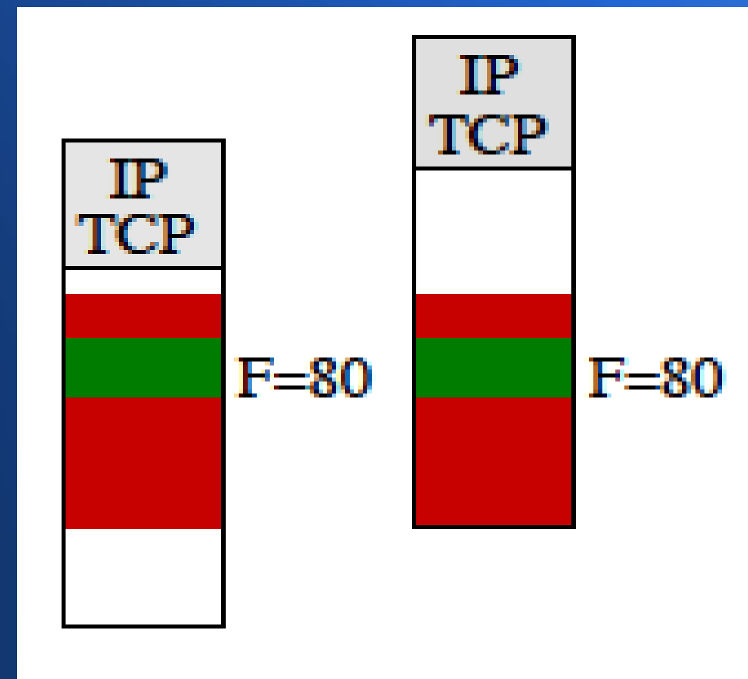
- Identify duplicates using fingerprints
- Sender
  - Computes packet payload's fingerprint
  - If match found in cache, transmit fingerprint in place of redundant packet payload
- Receiver has consistent copy of cache
  - Decodes fingerprints
  - Reassembles packet payload

Duplicate Fragment Suppression

- Identify 64-byte packet fragments using fingerprints
- Sender (for each packet payload)
  - Computes fingerprints of 64-byte fragments
  - For each fingerprint, if match found in cache, transmit fingerprint in place of fragment
- Receiver
  - Decodes fingerprints to get fragments
  - Reassembles packet payload from fragments

Fingerprint Matching

- Search cache for matching fingerprint
- Find largest match region and union
- Update cache
  - FIFO
  - LRU

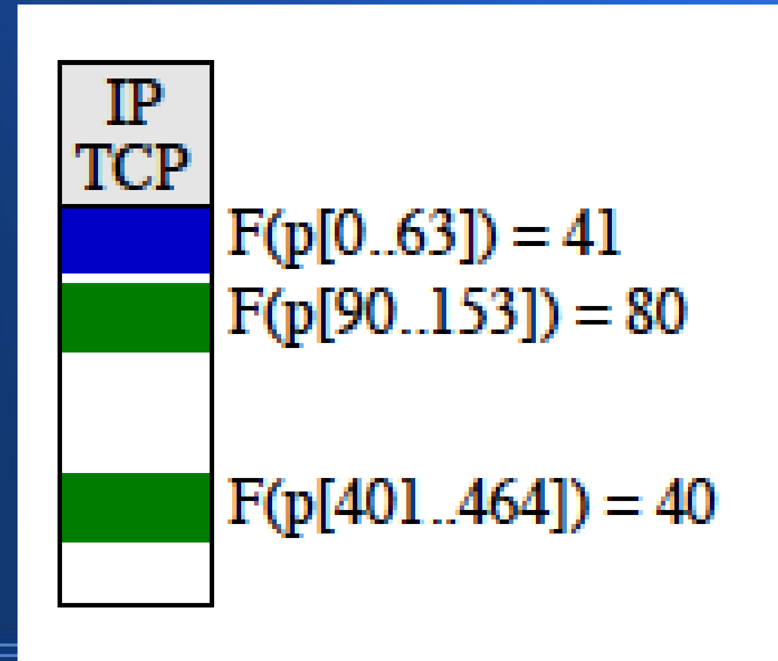


Fingerprint Computation

- Rabin fingerprint: $F_i = (t_i p^\beta + t_{i+1} p^{\beta-1} + \ldots + t_{i+\beta}) \bmod M$
- Incremental calculation:

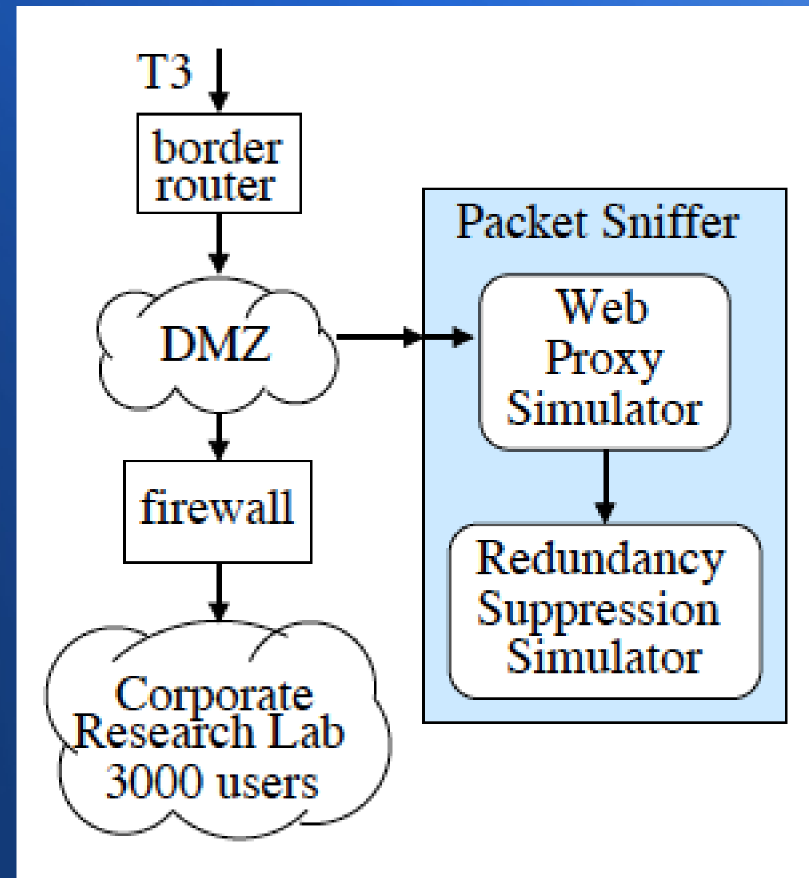
$$F_{i+1} = [(F_i - t_i p^\beta) * p + t_{i+\beta+1}] \bmod M$$

- To save cache space, index only fingerprints ending with γ zeroes (e.g. γ = 5)



Experimental Setup

- 100 – 200MB cache
- 576-byte MTU
- 12-byte encoding for fingerprints
- Byte savings = redundancy - encoding cost
- Primarily offline analysis

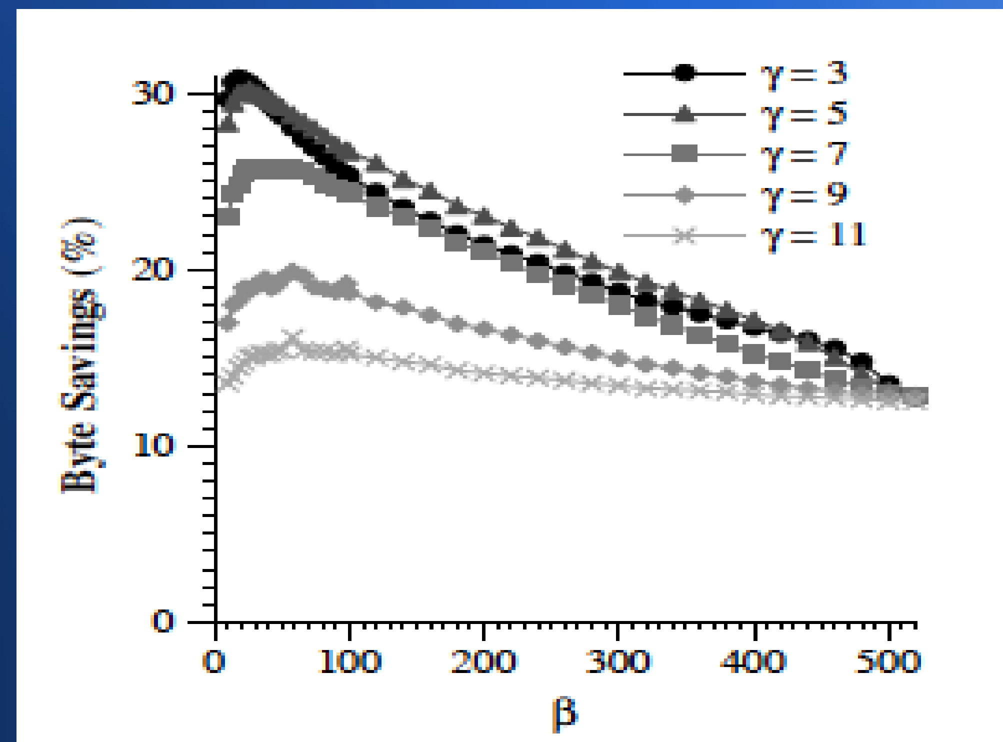


Experimental Setup (2)

- “Penalty” of 12 bytes
  - 60-bit fingerprint
  - 11-bit offset in packet
  - 11-bit redundant byte-count before fingerprint
  - 11-bit redundant byte-count after fingerprint

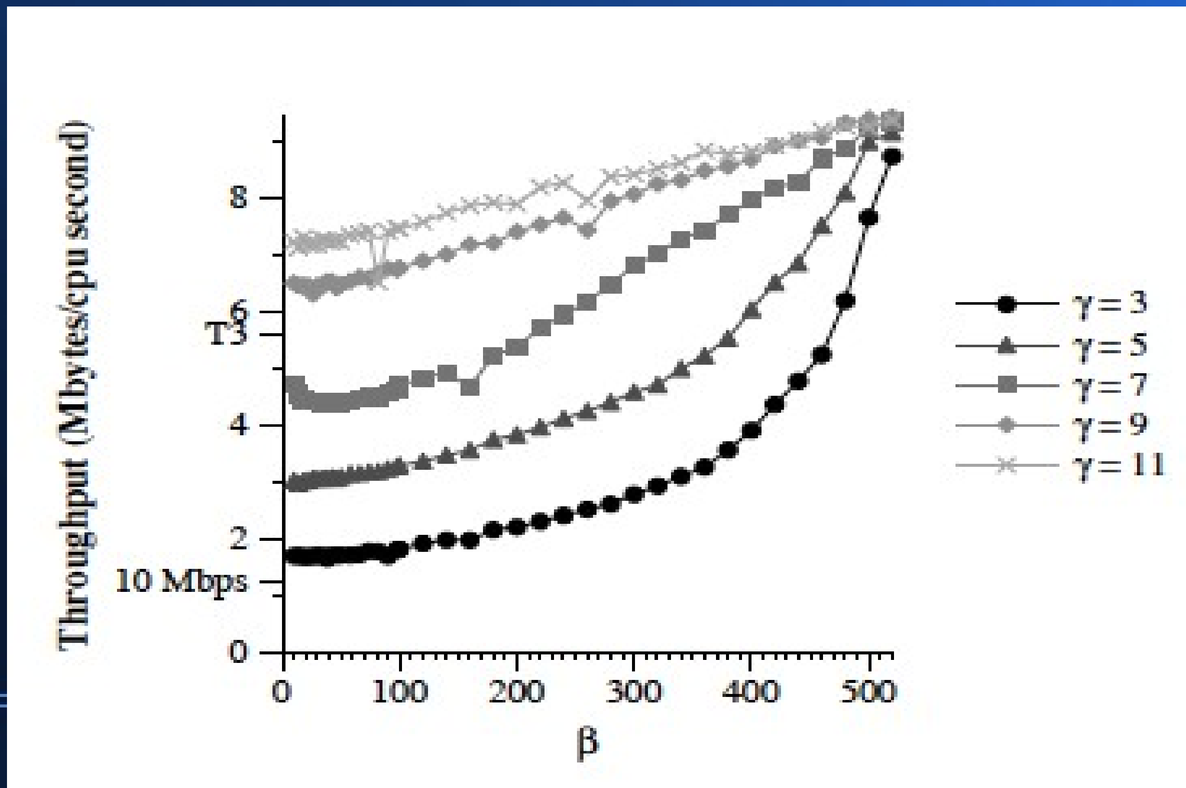
Results

- Byte-saving sensitivity to Beta and Gamma
- Hi Beta
  - Approaches duplicate match
  - Lo Byte Saving
- Lo Beta
  - Peaks just after encoding penalty
  - Hi Byte saving



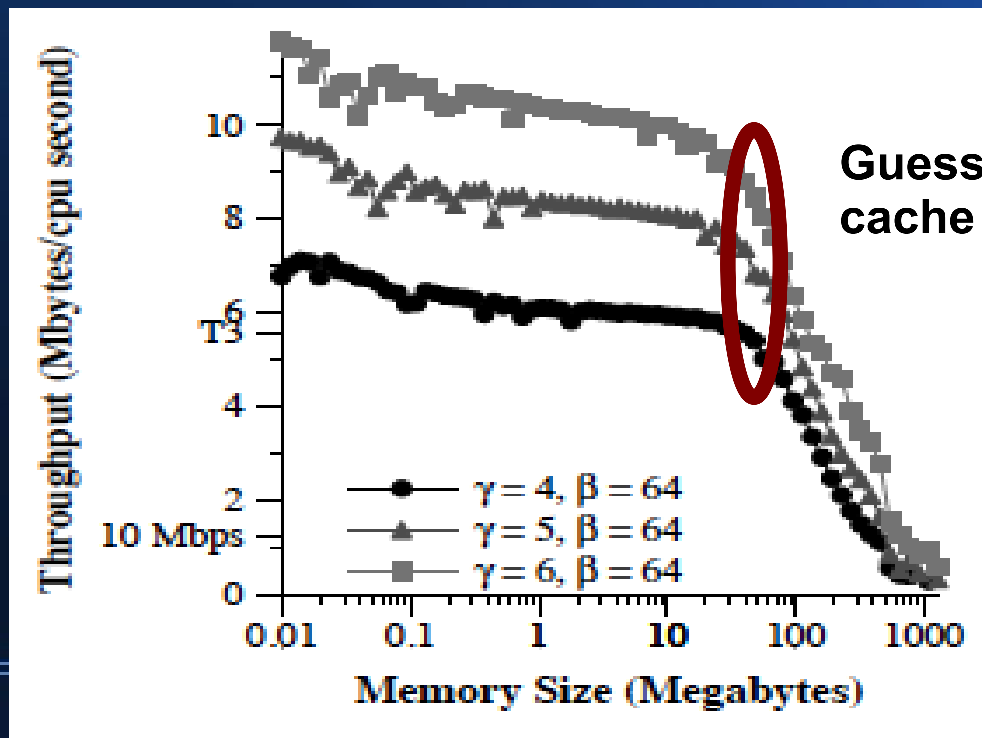
Results (2)

- Throughput sensitivity to Beta and Gamma
- Hi Beta => less windows => hi xput
- Hi Gamma => less fingerprints => hi xput



Results (3)

- Throughput sensitivity to memory availability
- Lo Gamma => more fingerprints => Lo xput
- More memory => more fingerprints => Lo xput



Guess: Thrashing in processor cache

Results (4): Compare w/ Web proxy

- Web 64.3% traffic, 30% byte savings
- Uncacheable documents
  - Request characteristics: Pragma/Cache Control directives (no-cache) Question mark, CGI, Auth.
  - Response characteristics: Pragma/Cache Control directives (no-cache), Set Cookie, Redirect HTTP status, Push.
- Technique exploits redundancy in uncacheable documents.

Doc type	Traffic Vol (% total bytes)	Redundancy (% of category)
Uncached Web	87.4%	23.3%
...Cacheable	63.4%	15.6%
...Uncacheable	24.0%	41.4%

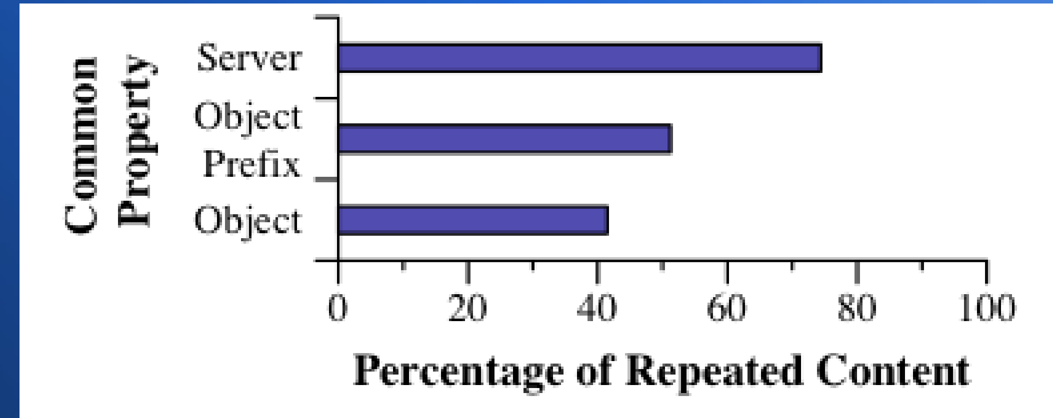
Results (5): Compare w/ Web proxy ctd.

Reason Uncacheable	Traffic Volume (% total bytes)	Redundant (% category)
Question	10.7%	52.2%
Method	5.0%	17.4%
CGI	4.2%	53.7%
Pragma Response	2.3%	76.0%
Pragma Request	1.6%	39.6%

- Highest repetition – pragma response directives
- Can cache application-layer headers, e.g. HTTP, SIP, SMTP, POP, etc.
- Can still honor privacy directives – probably can't cache questions then

Results (6): Source locality

- Content-based caching
  - Objects from (same) server
  - 78% redundancy!
  - Controlled for DNS round robin, Souder's rule (split DNS)?
  - Access pattern across Web



- http://server:port/path/object1.php?attrib1=foo&attrib2=bar
- Name based caching:
  - Object prefix – Match till before ?
  - Object: Match whole string.
  - Don't work with nonce based naming.

Results (7): Compare Web proxy, Compression and Redundancy Sup.

Cache / Compression Algorithm	Byte Savings
Web Proxy	14%
Web proxy + zlib compression	28%
Redundancy suppression	54%
Web Proxy + Redundancy suppression	56%

- Complements: Web Proxy + Redundancy Suppression + E2E Compression
  - Integrated solution recommendation untested
- Redundancy Suppression + Web Proxy
  - Slightly better than just Web Proxy.
  - Orthogonality?
- Zlib + Redundancy Suppression
  - Orthogonality: 14-36% v 54%

Zlib v Redundancy Suppression - Details

Feature	zlib	Redundancy Suppression
Compresses	streams	packet payloads
History Window	32KB	100MB +
Fingerprints	15b of 4B	60b of 64B
References	offset	fingerprint
Indexing	dense	probabilistic
Runs	after	before and after
Byte Savings	14-36%	54%

Our Thoughts (1)

- Traffic data types need to be “friendly” for redundancy suppression
  - Performs great in HTTP
  - How about in FTP (header/data)?
  - How about BT?
- Web Traffic peak estimates 2009:
  US: HTTP: 55%, BT: 20% (2009)
  SW Europe: HTTP: 23% BT 55% (Ipoque 2009)

– How about streaming data?

- Video streams, VOIP streams?

Our Thoughts (2)

- Alternatives for cache synchronization
  - No synchronization? Impossible to reconstruct packet payload from fingerprints.
  - One-way hash? Similar problem.
  - Receiver request for missing fingerprints?
    - No strict consistency required, but...
    - Extra RTT per cache miss

Our Thoughts (3)

- No consideration for communication layer problems
  - Packet loss, delay, reordered packets
  - Queues on either end can fill up while waiting for packet to be resent
  - Dependencies on segments make it worse
- System complexity a lot higher than described

Our Thoughts (4)

- 550MHz CPU
- T3: 45Mbps ~= 5.6MBytes/s
- ~500-byte packet payload, window = 64 bytes
- Max. packets/s = 11,200
- Max. # windows per packet = 437 (=500-64+1)
- # cycles for 1 packet = 550M/11,200/437 ~= 110
- Memory access typically 10x slower than processor cycle. 110 cycles ~= 10 mem. ops

Our Thoughts (5)

- 10 mem. ops to run an iteration of algorithm
  - Generate fingerprint
  - Search cache for matching fingerprint
  - Find largest match region and union
  - Update cache
- Seems really hard to do this online!
- What about today's “bottleneck links”?
  - Links: 45Mbps => 10Gbps (222x)
  - CPU: 550MHz => 3GHz (5.45x)
- Seems REALLY HARD to do this online!

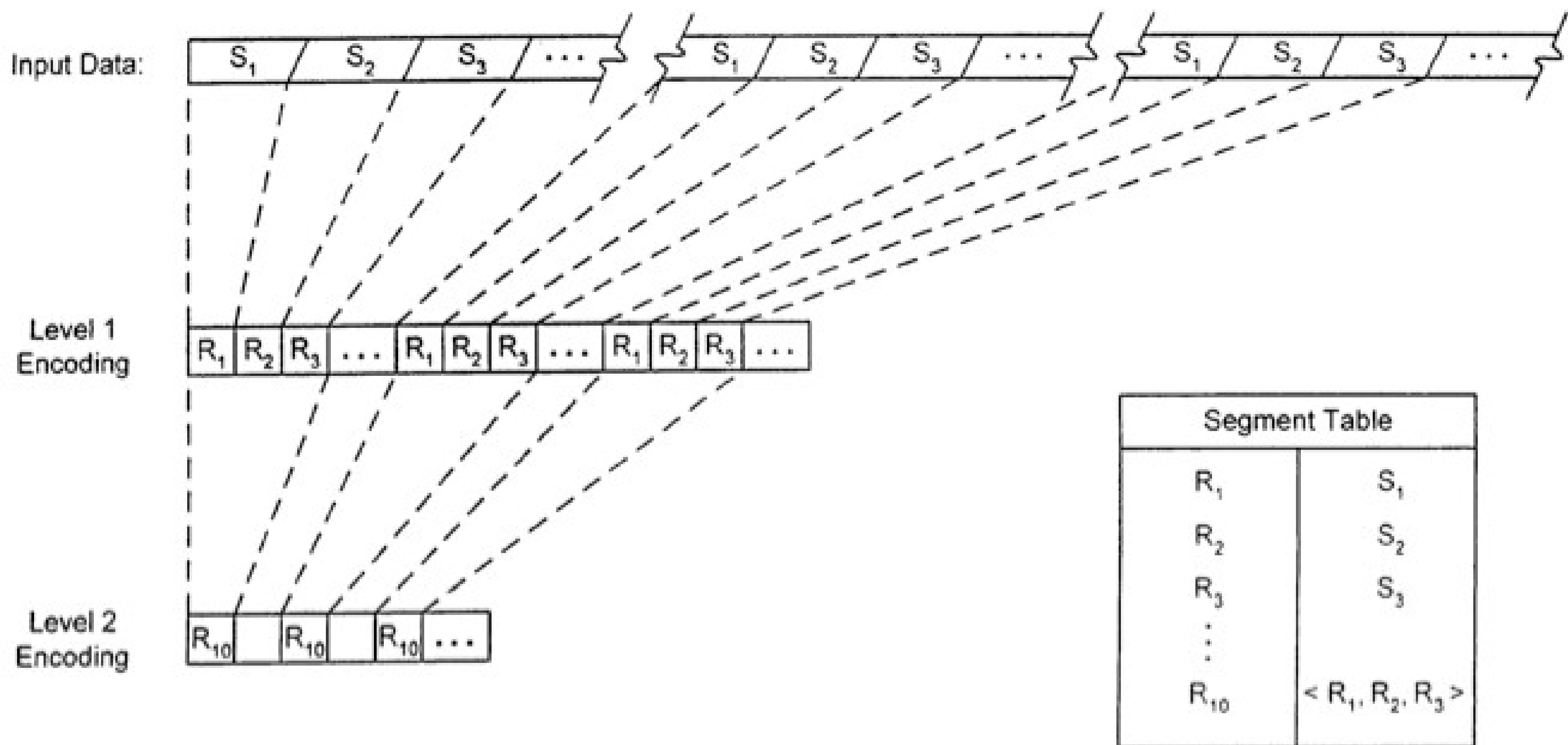
Our Thoughts (6)

- Observation: Fingerprint matches remove a lot of possible windows and lots of redundancy available => long-term performance may be acceptable for sender.
- What about cache synchronization?
  - Receiver needs to know received packet's fingerprints
  - Option 1: Send fingerprints with packet – defeats the purpose of having the scheme!
  - Option 2: Receiver calculates fingerprints – doubles the workload!

Patent: More features

- Synchronization
- Send fingerprints along with the packet
- Claim: Bandwidth overhead for fingerprints amortized over savings from same in the future
- Hierarchical Content-Induced Segmentation (HCS)
- Recursively run the same algorithm on input
- Adaptive # levels w/ level info in packets sent

Hierarchical Redundancy Check



Applications : Client - Server

- Client (CTA) → server (STA) request
- Client Proxy → C.TT → S.TT^{-1} → Server Proxy
- Synchronization:
  - S.PSS → S.RR → C.RR → C.PSS
- Server crafts decoded output string.
- Reverse for Server → Client msgs.

Applications: File systems

- Near Line File System:
  - Scheme effective only at file level, not block level. i.e. blocks need to be large enough
  - Useful for back-up systems
  - Operations on whole files (store, restore, delete)
- Caching
  - Regular file cache obtains whole files from NLFS.
  - Explodes special inode to regular FS inodes.
  - Inode operations.
  - Backup into system and save inode.
  - Tradeoff: Computation/storage complexity.