Verifying Data Integrity in Peer-to-Peer Media Streaming

Presented by

[1]Ahsan Habib

Joint work with

[2]Dongyan Xu, [2]Mikhail Atallah,
[2]Bharat Bhargava, and [1]John Chuang

[1]University of California at Berkeley, {habib,chuang}@sims.berkeley.edu,

[2]Purdue University {dxu,mja,bb}@cs.purdue.edu

Multimedia Computing and Networking (MMCN '05), Jan 19 2005.

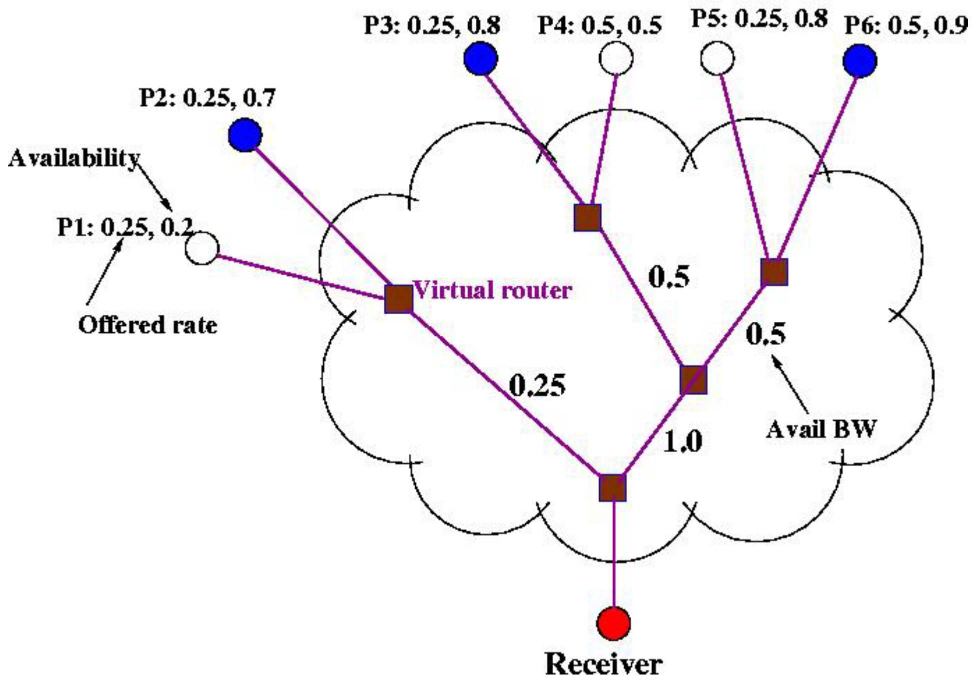
Problem Statement

When watching *The Matrix* over the Internet from several *untrustworthy peers*, how to ensure in *real time*

- The data are *not corrupted*
- The data belong to *The Matrix* not *Star Wars*

Setup

- Many-to-one (not one-to-many, i.e., multicast)
  - PROMISE [MM ’03]
  - Supplier selection is done by underlying P2P substrate
- The content is video data
  - Watched in real time
  - Bandwidth requirement is high, and
  - Session duration is long (hours)



PROMISE Peer Selection

Challenges

- Like multicast, there is no trusted authority to sign all packets
  - Peers are not trustworthy. Signing by peers is not acceptable to others
- Verify the integrity of the content in real time
- Validate the content

Contribution

- Propose two protocols to verify data integrity in P2P media streaming
- Provide a detailed analysis among existing and proposed protocols
- Compare protocols for communication and computation overheads
- Simulation and wide area Internet experimental study to show their performance

Outline

- Introduction
  - Setup, challenges, and contribution
- Existing tools and techniques
- Proposed Solution
  - BOPV
  - TFDP
- Analytical comparison
- Simulation and experimental results
- Conclusion

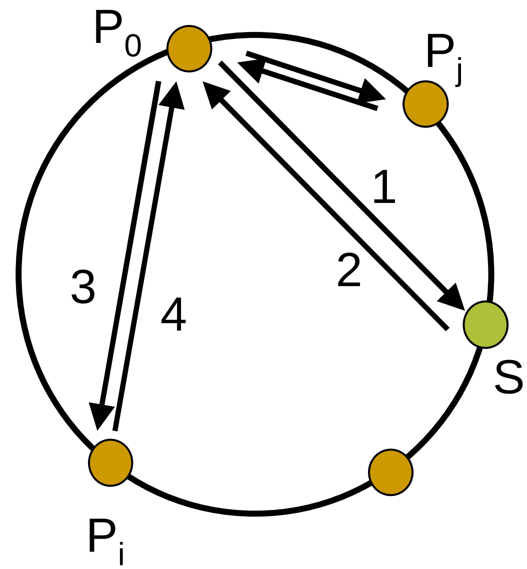
Existing Tools/Techniques

- Digital signature
  - RSA signature scheme [Comm of ACM '78]
  - One time signature [CCS '01], k-time signature [CCS '99]
- Signature chain
  - TESLA, EMSS [S&P '00, NDSS '01]
- Signature tree
  - SAIDA [S&P '02]
  - Tree chaining [TON '99] uses Merkle tree [Crypto '89]
- File sharing
  - Key escrow [EC '01]
  - Rate-less Erasure-code with homomorphic hash function [S&P '04]

Our solution (Preliminaries)

- Streaming model
  - Suppliers set, $\mathbf{P}=\{P_1, P_2, P_3, \ldots, P_m\}$
  - Media file consists of blocks $\mathbf{B}=\{B_1, B_2, \ldots, B_M\}$
  - Block consists of packets $\mathbf{B_i}=\{p_{i1}, p_{i2}, \ldots, p_{il}\}$
  - A series of N blocks makes a group
- Adversary model
  - Insert garbage data during streaming. A peer can pretend to have a file without actually having it.
- A point of reference (S)
  - S is Hollywood in legal content distribution model *or*
  - S is stored in a distributed fashion

Block Oriented Probabilistic Verification (BOPV) Protocol



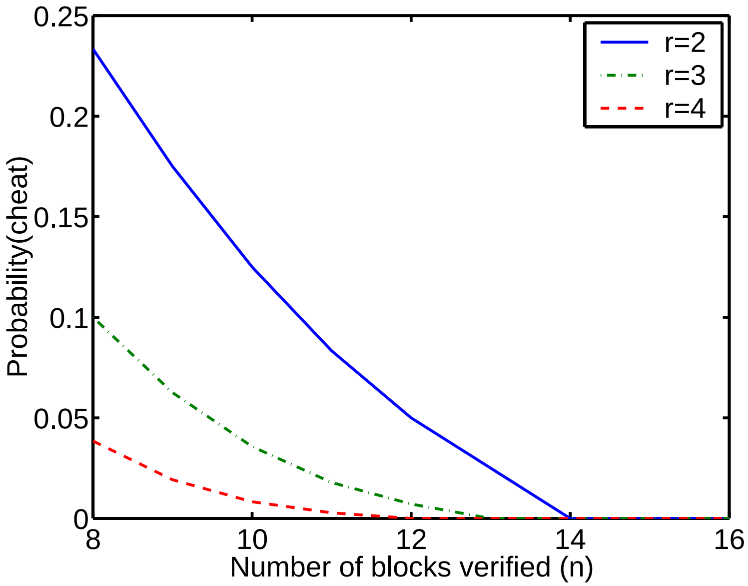
1. P_0 authenticates itself to S
2. S generates secret key $K_{i=1...M}$ for each block B_i, computes n (N > n) digests $\sigma_{j=1...n}$ for each group and sends them to P_0
3. P_0 gives key(s) to each supplier peer
4. Each peer supplies B_i and its digest.

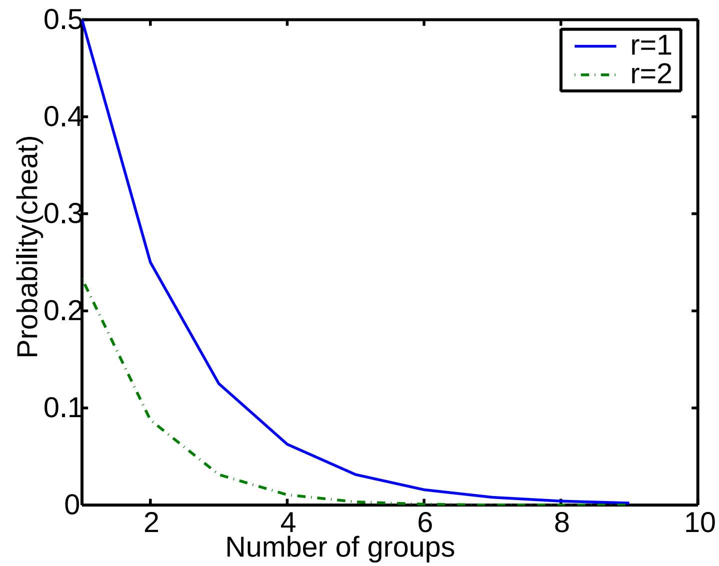
P_0 matches digests from step 2 and 4.

BOPV (Cont'd)

- Probabilistic verification
  - S provides n digests for N blocks (N > n). P_0 does not verify all blocks
  - Probability to cheat in r blocks by a peer, $\Pr[\text{cheat}(N,n,r)] = \frac{\binom{N-r}{n}}{\binom{N}{n}}$
- An example: The Matrix
  - File size 1.3 GBytes
  - 1 digest for 1 packet ≈ 26 MB digests to download from S
  - One block contains 32 pkts, digests ≈ 0.79 MB
  - Verifying 8 out of 16 blocks, digests ≈ 406 KB
  - Having 128 pkts per block, digests ≈ 107 KB

Probabilistic verification





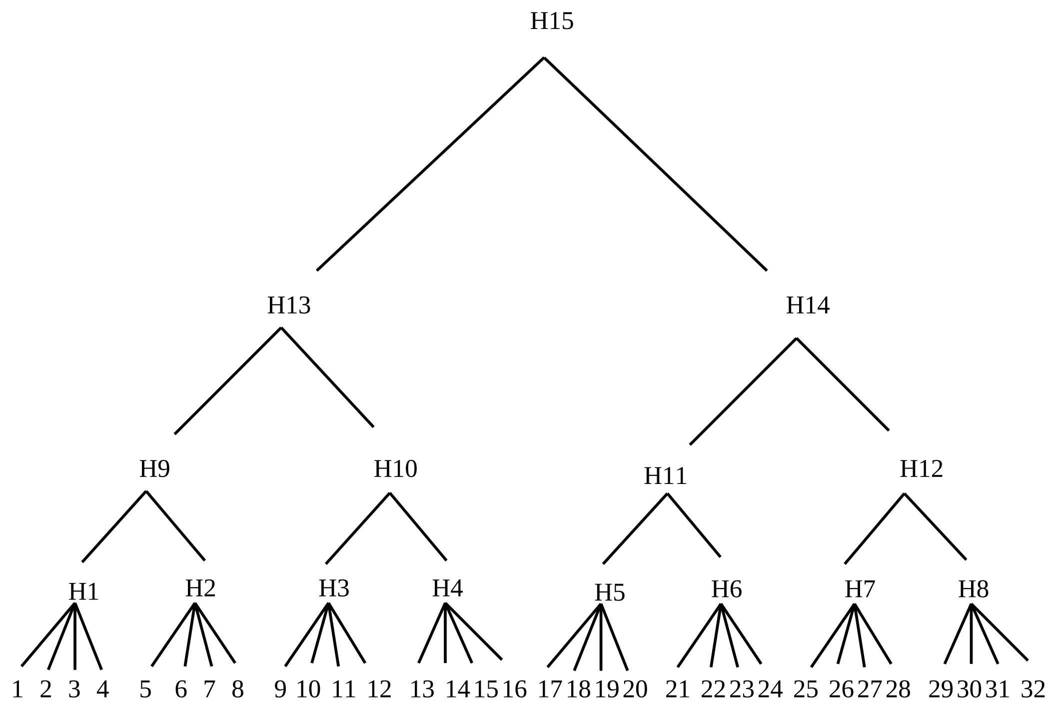
- N=16, n=8, r=1, Pr[cheat] = 50%
- N=16, n=9, r=4, Pr[cheat] = 1%
- 1 block corrupt in 10 groups,   Pr[cheat] = 0.002
- 2 blocks corrupt in 6 groups,   Pr[cheat] = 0.0008

Limitation (BOPV)

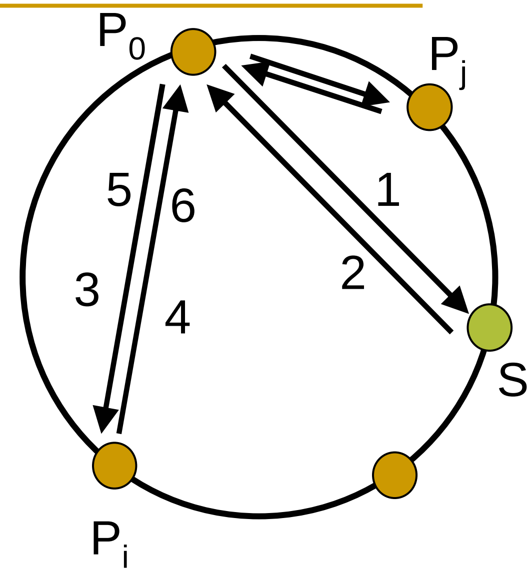
- If a packet is lost, the whole block is useless
  - Multiple hashes (BOPV + MH) [S&P '00, IBM TR '97]
    - Each packet contains digests of other packets
    - If a packet is lost, its digest can be found in other packets
  - FEC (BOPV + FEC)
    - FEC is used to encode digests
    - `t` packets (instead of `k<t`) are sent by the senders and k out of `t` packets are required to recover all packets
    - FEC overhead, α = `t/k`
- Heavily depends on S. Initial digest download is also high.

Tree-based Forward Digest Protocol (TFDP)

- Build Merkle tree for a media file
- Besides data, peers cache digests to compute the root
- Peers forward digests first before data
- N_{min} blocks are verified at a time. Number of extra digest $=(d-1)\log_d(M/N_{\min})$



TFDP



1. P_0 authenticates itself to S
2. S provides P_0 the digest of the root of the tree
3. P_0 tells the suppliers to send the digests to verify N_{min} blocks.
4. The assigned peers send P_0 the digests of the leaves and other digests to verify the root digest
5. P_0 computes the root digest with the digests at step 4 and verifies it with the digest at step 2.
6. If there is a match, P_0 signals all suppliers to send data

P_0 verifies each block individually during streaming

The process is repeated for the whole file.

Analytical Comparison

- Compute communication and computation overheads for each protocol
  - Communication overhead: extra bytes downloaded by the receiver for integrity verification
  - Computation overhead: time to compute digest, decode, and verify signature. Use openSSL crypto library and Reed-Solomon code for FEC.
- Symbols
  - Total blocks = M, total packets in a block = l, FEC overhead = α, probability to verify a packet = v, extra digest to send with each packet (BOPV+MH) = β

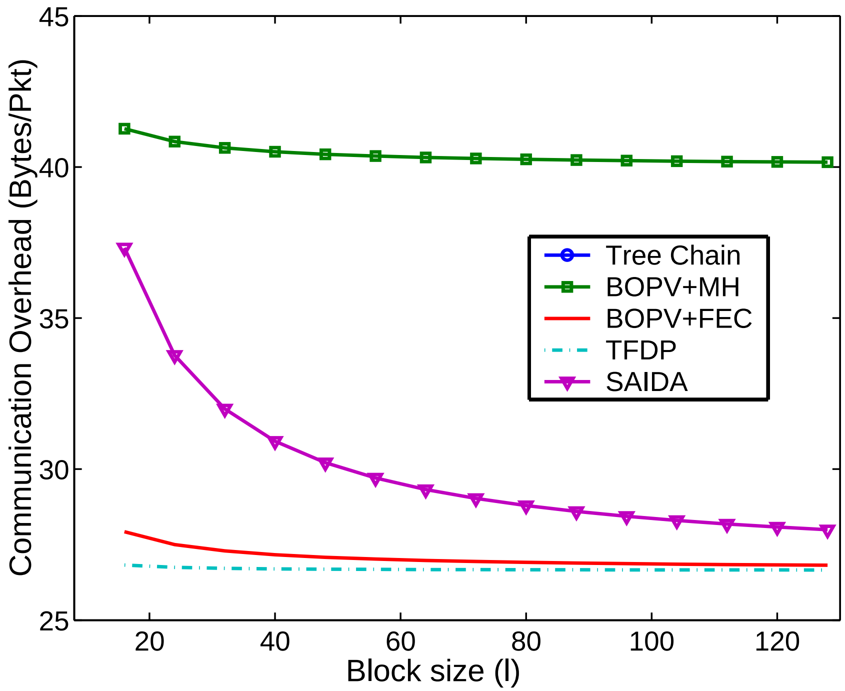
Communication Overhead

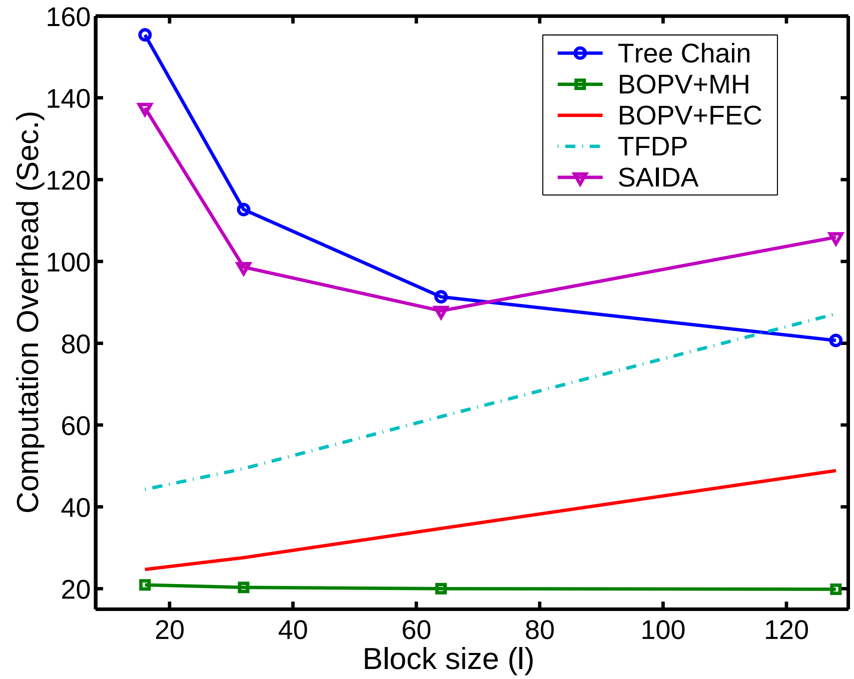
	Download from suppliers	Download from S
BOPV+MH	$20Ml\beta$	$+ (20+K)Mv$
BOPV + FEC	$20Ml\alpha$	$+ (20+K)Mv$
SAIDA	$(128+20l)M\alpha$	
Tree Chaining	$(128+20\log l)Ml$	
TFDP	$20[Ml+M+M/N_{min}\log(M/N_{min})]\alpha + 20$	

Computation Overhead

	Digest computation	Digest decode	Sign verify
BOPV+MH	M(l+1)v		
BOPV + FEC	M(l+1)v	M	
SAIDA	M(l+1)	M	M
Tree Chaining	M(2l-1)		M
TFDP	$Ml+M/N_{min}[(N_{min}-1)\log(M/N_{min})]$	M	

Comparing Protocols

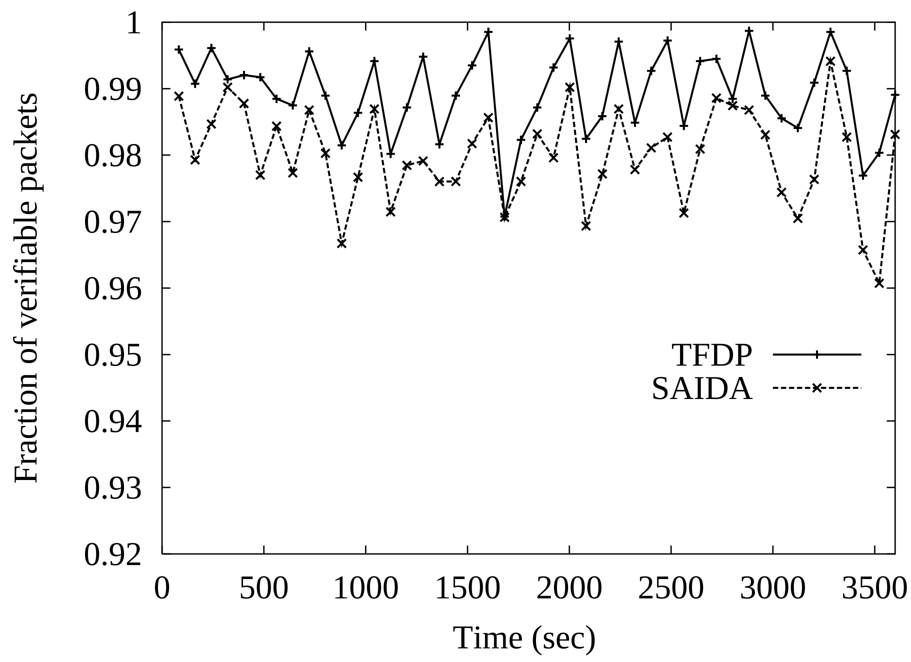




- Communication and computation overhead for *The Matrix.*
- Tree chaining has very high comm overhead (208 Bytes per pkt)
- TFDP outperforms others especially when `l` is small.

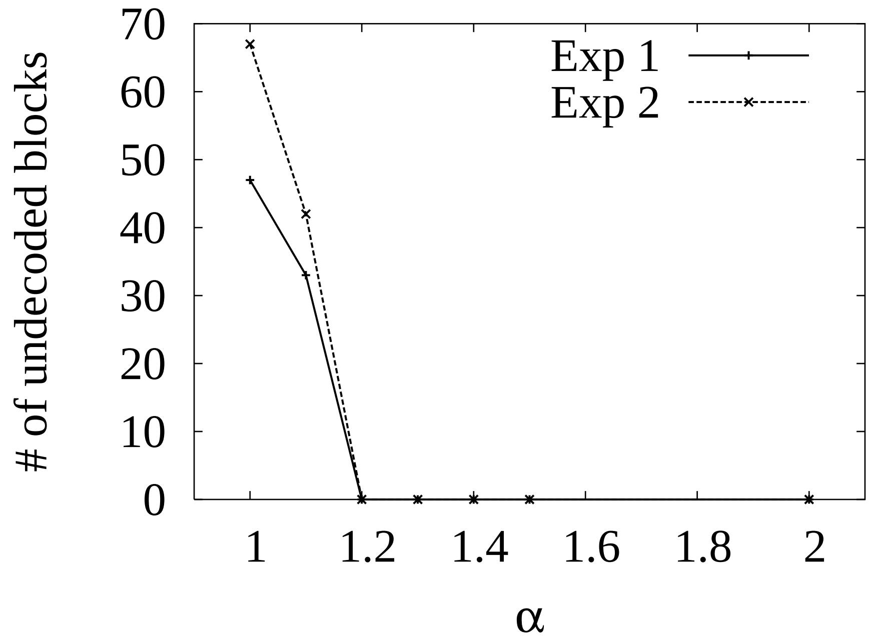
Experimental evaluation (Simulation)

- Use Gilbert model for bursty packet loss
- Compute fraction of verifiable packets during streaming
- SAIDA shows it's better than EMSS, we show we are better than SAIDA
- More than 97% of packets are verifiable all the time



Experimental evaluation (Planet-Lab)

- Use PROMISE implementation to conduct experiments in Planet-lab test-bed
- In our experiments
  - The stream can tolerate 20% packet loss due to FEC
  - Fraction of verifiable packets is ≥ 0.95 except a few instances when it goes to 0.90.
  - Use video trace of *Star Wars IV*, and *From Dusk till Dawn*



Conclusion

- We address an important security issue for P2P media streaming
- Our protocols reduce communication and computation overhead
- Tolerate bursty packet losses using FEC for digests
- Packet verifying probability is 97% or higher even when the loss is 20%
- In TFDP, a peer can verify data block by block and thus can become a supplier immediately in BITTORRENT-style file sharing system.

THANK YOU