

Secure Acknowledgement of Multicast Messages in Open Peer-to-Peer Networks

Antonio Nicolosi

David Mazières

{**nicolosi**, dm}@cs.nyu.edu

New York University



2nd IRIS Student Workshop

November 7th, 2004



Cambridge, MA, USA

Message Acknowledgment

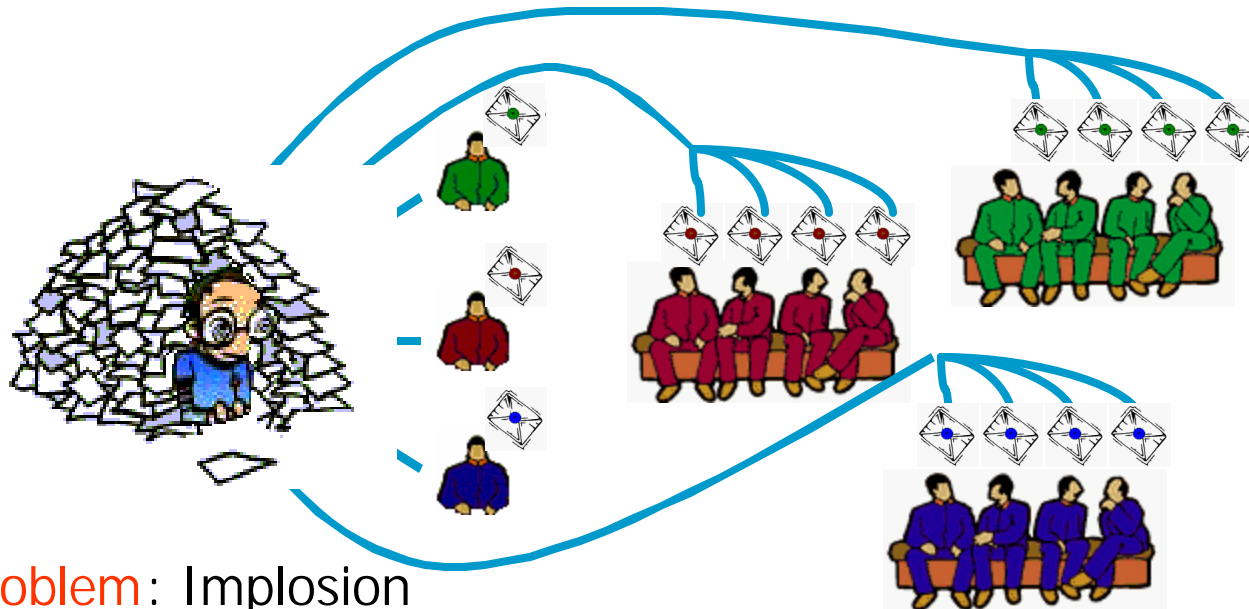
- A sender of messages, Bob
- A **large** set of recipients
 - So large that storing/downloading complete membership list requires too much resources
- Goal: **Verify** that all interested **nodes are receiving** messages
 - Or get **details of who didn't** receive

Applications

- Cache Invalidation
 - By registering with the publisher, **caches** are sure they **won't miss** any **update**
 - By collecting acks, **publisher** knows when **no stale info** is around anymore
- Security Update Center
 - OS maintainers can ensure security-aware users have got the latest patch
- *Secure* Reliable Multicast Trees
 - Ensure every node is receiving messages
 - Detect "**deadbeat**" internal nodes

Straw-Man Solution

- Bob is keeping a list of friends interested in his jokes
- To join the list, new friends have to give Bob a PK
- When Bob sends out a new message ...
- ... each friend signs an ack and sends it back to Bob



➤ **Problem:** Implosion

Acknowledgment Compression

- Goal: Reduce Bob's communication & state
- Solution: New cryptographic primitive
- At each node:
 - $O(k + \log_k n)$ storage
 - $O(k)$ bandwidth & computation for each ack
 - $O(k \log_k n)$ bandwidth & computation for (arbitrarily many) simultaneous joins

n: # users
k: max degree

The Model

- **Init:** sender picks SK x_s , and publishes PK y_s
 - x_s will be used to sign “Join Receipts”
- **Collect:** sender obtains acks from all users
 - Users gather their signatures in a **constant-size**, compressed proof, that the sender can validate
- **Join:** batch-oriented user addition protocol:
 - Sender signs a single **constant-size**, combined receipt, which is delivered to all newcomers
 - Yet each new user **is guaranteed** that he/she will get all future content, or the sender will notice
- **Leave:** batch-oriented user deletion protocol

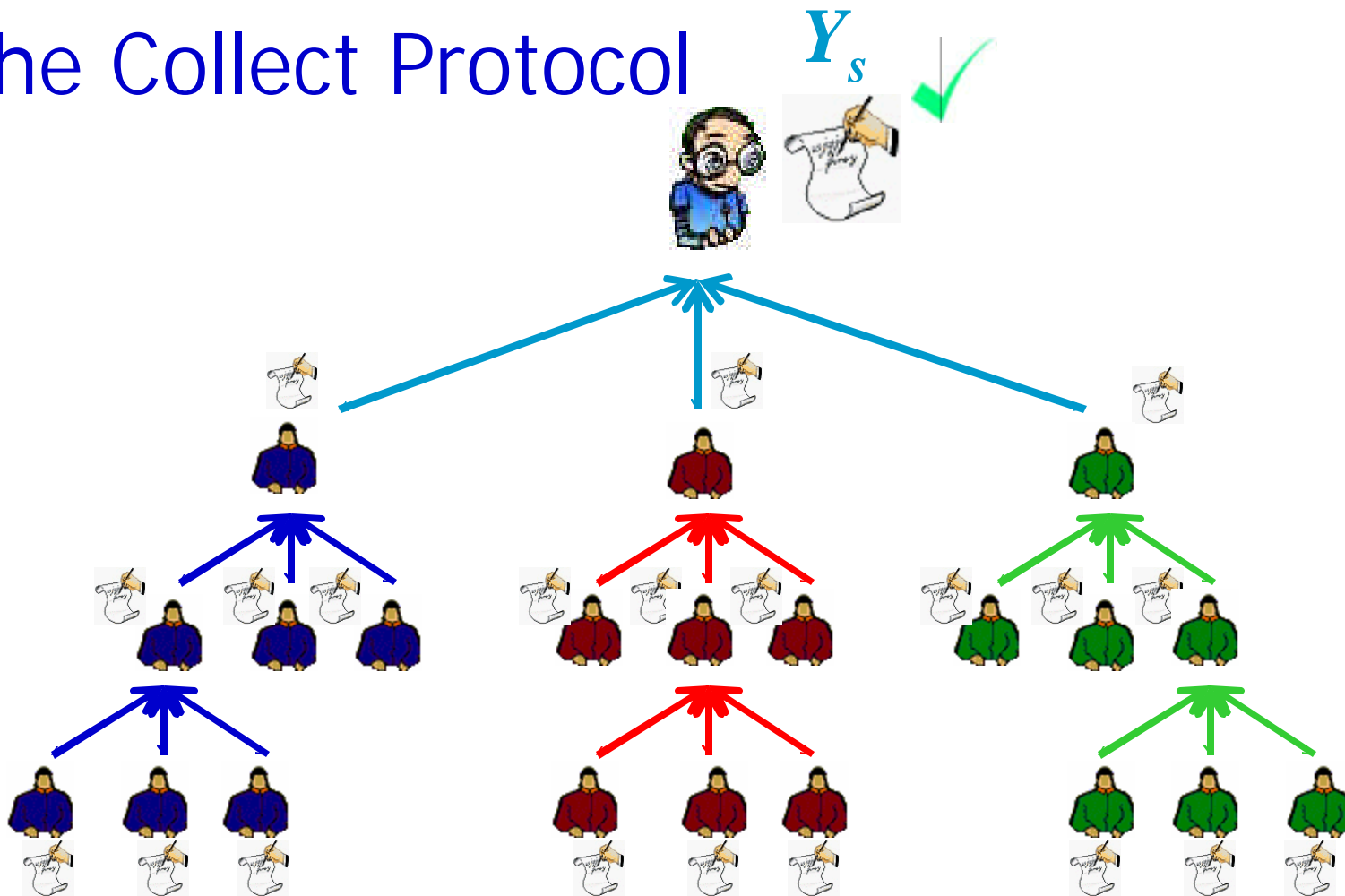
The Crypto Behind: Aggregate Sigs

- Signatures with nice “aggregation” property
 - Pick a message m and two PKs y_1, y_2
 - Let $s_1 := \{m\}_{y_1^{-1}}$ and $s_2 := \{m\}_{y_2^{-1}}$
 - Multiplying PKs yields aggregate PK: $Y := y_1 y_2$
 - Multiplying sigs yields aggregate sig: $S := s_1 s_2$
 - Notice that
 - Aggregates do not grow in size
 - Aggregate sig S is valid under aggregate PK Y
- Generalizes to many PKs and sigs

The Scheme

- Assume nodes form a multicast tree
- Each node i has a key pair x_i, y_i
- Node i maintains:
 - Aggregate PK $Y_i =$ product of PKs of all its descendents
 - IP address of each child j (signed under Y_j)
- Sender s holds $Y_s =$ product of all PKs

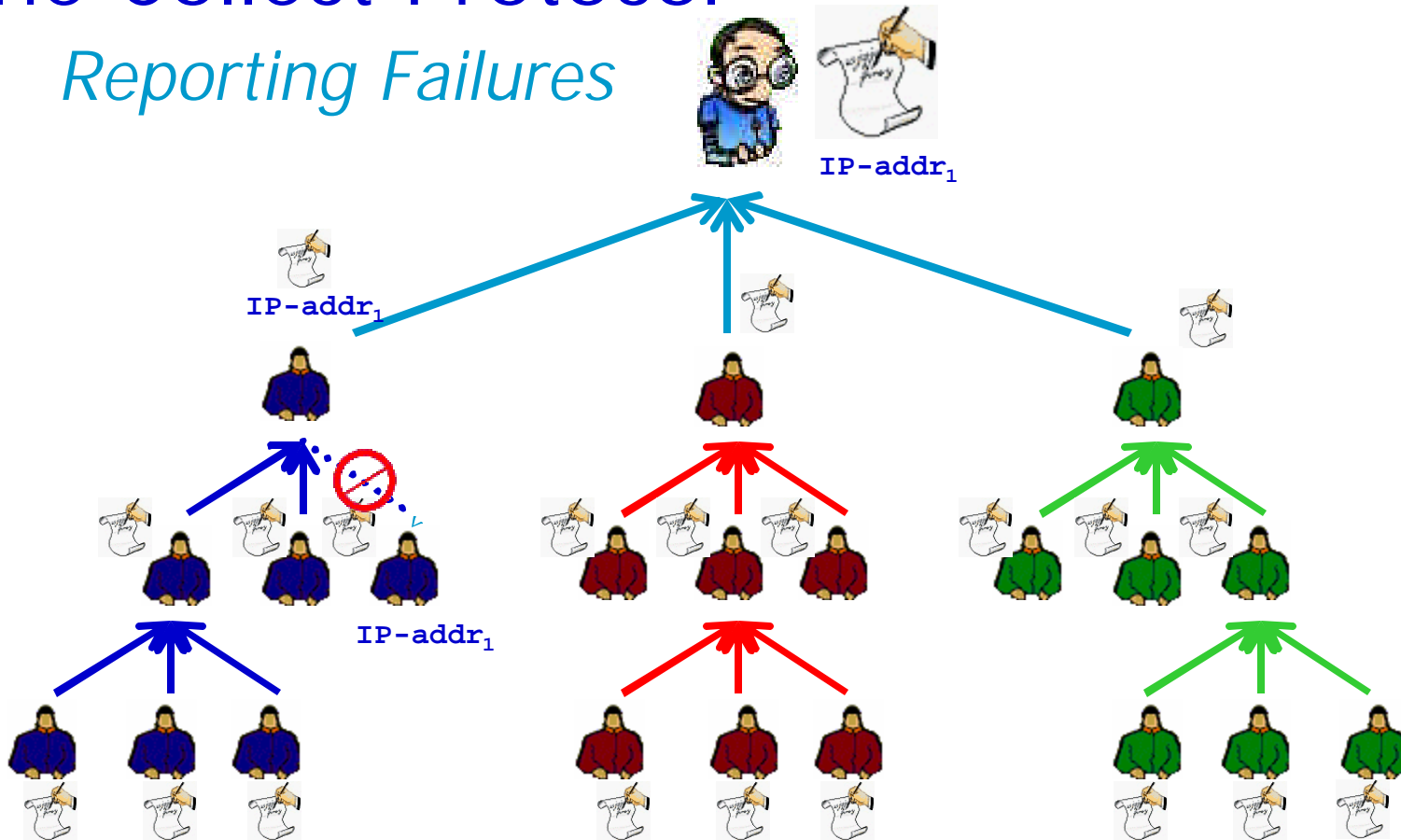
The Collect Protocol



- Each node i creates signed ack s_i ...
- ... and computes aggregate sig $S_i := s_i S_{j_1} \dots S_{j_k}$

The Collect Protocol

Reporting Failures



- Include signed IP address for missing components

The Join Protocol

- As new users join, combined PK Y_s evolves
- Need a way to guarantee that:
 - Each new user's PK gets included
 - No PK of pre-existing users is "factored out"
 - No fake PK is "multiplied in"
- Join achieves this incrementally
 - Starting from the leaves, each user i asks its descendents to sign new value of Y_i
 - At the end, everybody has signed new value of Y_s
- See the paper for details

Thank you!



Any questions?



?



Overhead at Each Node

Computational cost

n : # users
 k : max degree

- Cost of GDH-ops (estimated, [BKLS02]):
 - Signing : *4 ms*
 - $\leq \log_k n$ msgs per run (any operation)
 - Verification : *53 ms*
 - *1* for Collect, $\log_k n$ for Join/Leave
 - Aggregation : *< 1 ms*
 - k for Collect, $k \notin \log_k n$ for Join/Leave