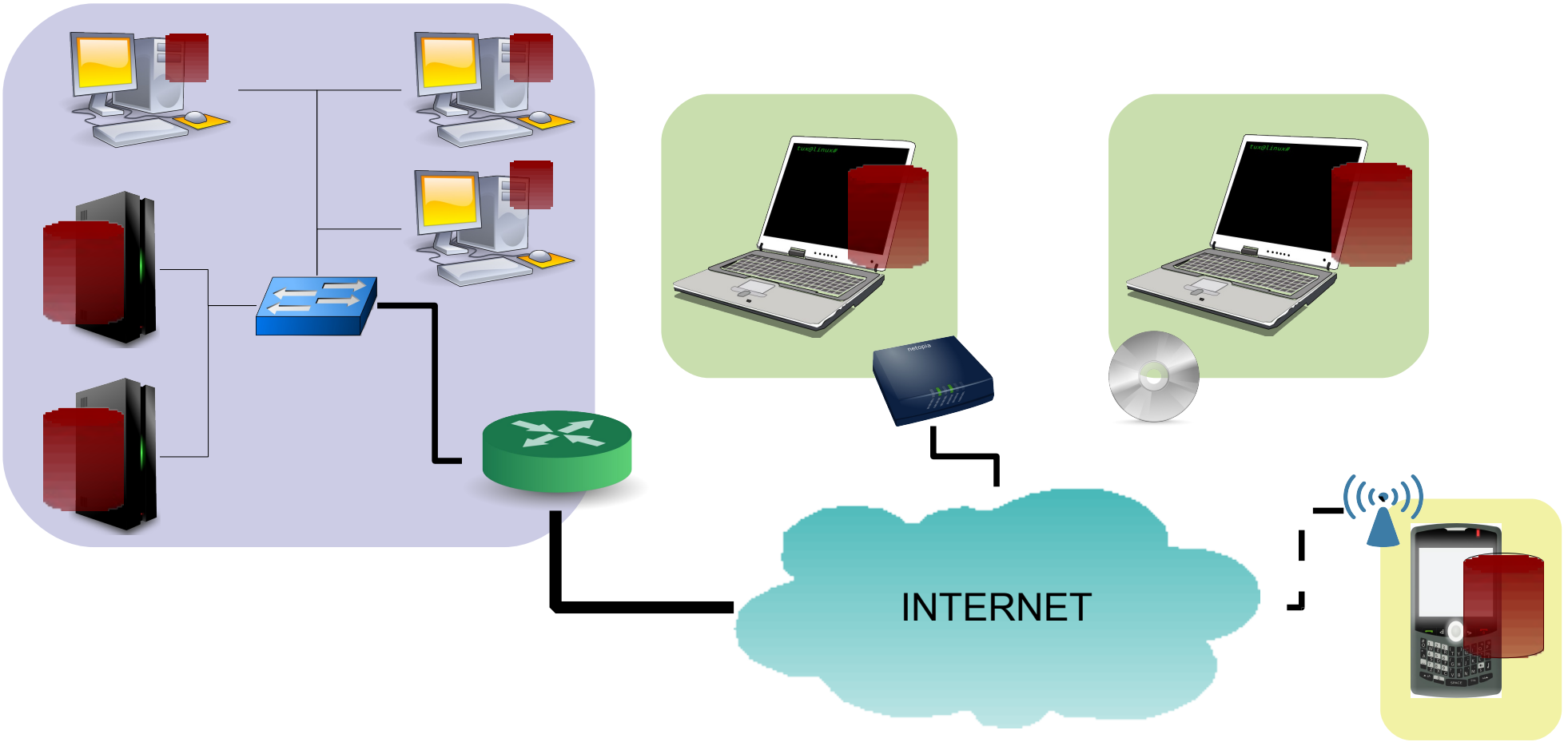
Bayou Anti-entropy Protocol

"Flexible Update Propagation for Weakly Consistent Replication"

K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer & A. J. Demers

(ACM Symposium on Operating Systems Principle – 97)

Presented by:
Atif Khan
Jan 18th, 2010



Attributes

- Clients
- Data
- Network

Challenges

- High availability
- Performance
- Scalability

Bayou Anti-entropy

Replication of shared data

- Email
- Calendar
- Address Book
- Documents

Outline

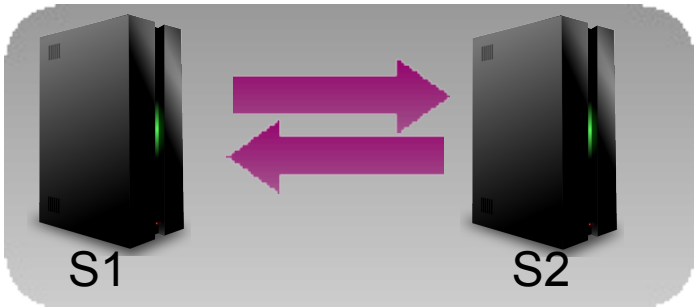
- **Introduction**
- **Anti-entropy**
- **Anti-entropy Extensions**
- **Anti-entropy Policies**
- **Conclusion**
- **Discussion / Q & A**

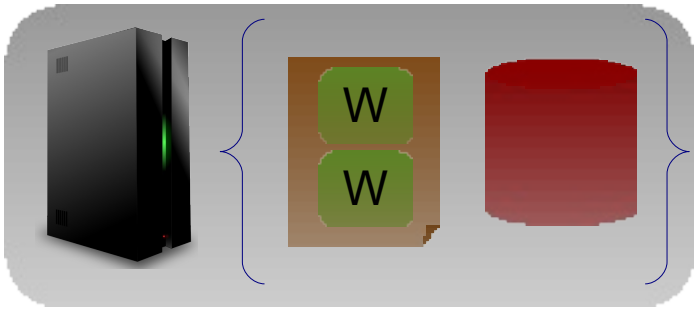
Introduction

- **Design goals**
  - Servers (replicas) state reconciliation
    - Pairwise (any two)
  - Via ***writes*** propagation
  - Relaxes data consistency
    - Provides flexibility (whom, when, what, how)
  - Arbitrary communication topologies
    - Low-bandwidth, off-line update
  - Incremental
  - Eventual consistency

Anti-entropy

- **Anti-entropy**
  - S1 & S2 update each other
- **Bayou primitives**
  - Server→{write-log, database}
    - Write-log: order log of writes/updates
    - Database: Result of in-order execution of writes
  - Write = {U, D, M}
    - Updates,
    - Dependency check, merge procedure

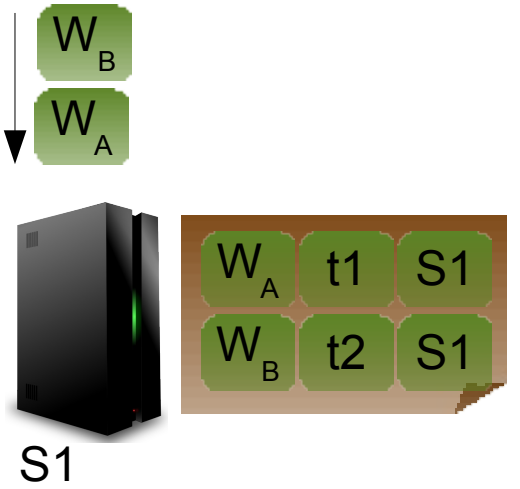




Conflict detection & resolution

Bayou Primitives (Anti-entropy)

- **Accept-order**
  - Accept time (*acceptTime, server id*)
    - *Total order* over all *server* writes
    - *Partial* (accept) order over *all system* writes
  - $W_A \rightarrow W_B$
    - W_A accepted *before* W_B by ***the same server***
  - Write-log *follows* accept-order



Bayou Primitives (Anti-entropy)

- **Prefix property**
  - Updates follow the rule over the set of writes

> If the write-log of R contains a write w from S,
> then write-log of R also contains
> all writes accepted by S prior to w.

- **Version vector**
  - R.V[S] = largest accept stamp accepted by S known to R

Basic Protocol (Anti-entropy)

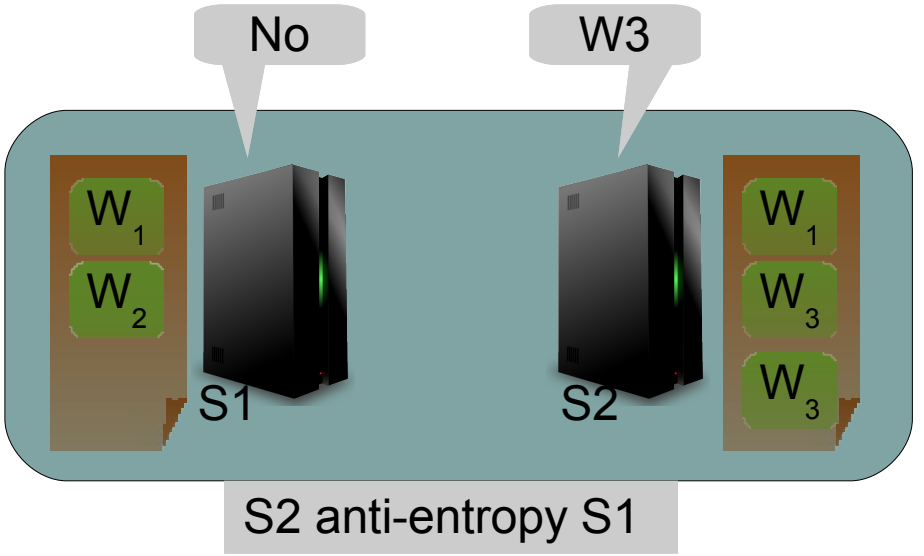
One-way operation between pairs of servers

Occurs through the propagation of write operations

Write propagation is constrained by accept-order

Basic Protocol (Anti-entropy)

```
anti-entropy(S,R) {
  Get R.V from R
  #now send all the writes
    unknown to R
  w = first write in S.write-log
  WHILE(w) DO
    IF w is new for R
      SendWrite(R,w)
    w = next write in S.write-log
  END
}
```



Multiple topologies

Pick your own partner

Incremental update

Eventual consistency

All That Glitters Is Not Gold (Anti-entropy)

- **Ordered delivery of writes**
  - To preserve the prefix property
  - Easy to fix
- **Write-log remembers all writes**
  - Writes can be purged
    - Received by all servers
    - Become *stable* (Bayou)

Stable (Committed) Writes (Anti-entropy)

- **A write is stable if**
  - *Never* has to be *reapplied*
  - *Stable* (position) in *write-log*
  - Commit sequence number (CSN)
    - Assigned when a write is committed
- **New partial order**
  - CSN + TS (time-stamp) + Server id
  - A→B if ($CSN_A > CSN_B$) OR ($TS_A > TS_B$) by same server
  - Committed writes are always totally ordered

Committed Write Protocol(Anti-entropy)

❖ Committed write protocol

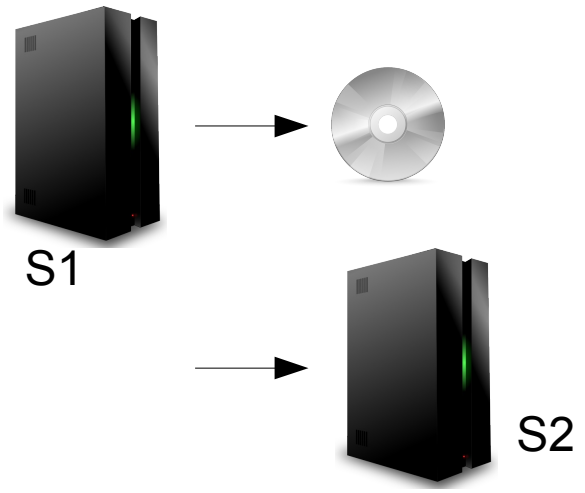
```
anti-entropy(S,R) {
  Get R.V and R.CSN from R
  #first send all the committed writes unknown to R
  IF R.CSN <S.CNS THEN
    w = first committed write unknown to R
    Send all writes or commit notifications
  END

  #now send all the tentative writes
  w = first tentative write
  WHILE(w) DO
    IF w is new for R
      SendWrite(R,w)
    w = next write in S.write-log
  END
}
```

Write-log Management (Anti-entropy)

- **Each server can**
  - Purge *stable writes only*
  - *Free to choose* how many to remove
  - Worst case: Transfer full database
- **Write-log truncation**
  - Omitted sequence number (OSN)
    - Detects missing stable writes
    - $OSN_A > CSN_B$ ► A has truncated stable writes missing from B
    - Protocol has support for OSN
      - Please refer to the paper

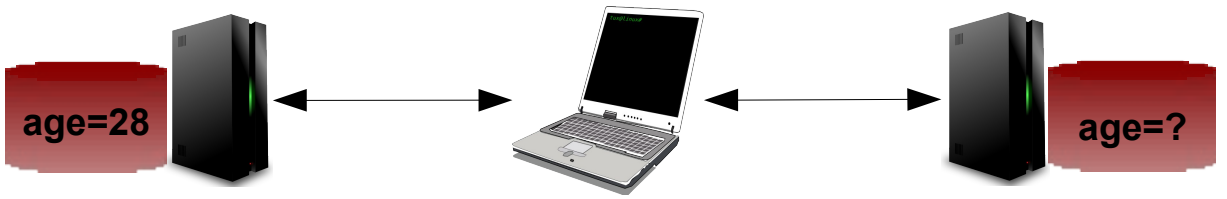
Anti-entropy Extensions

- **Transportable media support**
  - Writes are written to file
  - CSN + Version Vector
    - Minimum requirement for anti-entropy
  - CSN+Version Vector of S1
    - Optimal update check
  - Incremental backup sessions
    - Sequential file fragments



Anti-entropy Extensions

- **Session guarantees**
  - Question
    - Is data consistent for client?



  - Definition
    - *Client* observed *inconsistencies* when *accessing data* from different *replicas* (servers)

Anti-entropy Extensions

- **Session guarantees**
  - Session guarantees require a casual order
  - Casual-accept-order
    - Refinement on accept-order
      - $W_A \rightarrow W_B$ ***iff*** W_A is known before W_B is accepted
    - Each server has a logical clock
      - Clock advances when
        - A write is received from a client
        - A write with *higher accept-stamp* is received through anti-entropy
    - Casual-accept-order > accept-order

Anti-entropy Extensions

- **Eventual consistency**
  - Total order is *required* for *eventual* database consistency
  - Similar to stable-order
  - For all system writes
  - To total order
    - Accept-order by using serverId
    - Stable-order by using CSN, acceptTime, serverId

Anti-entropy Extensions

- **Server management**
  - Lightweight
  - Supported via anti-entropy
    - Version vector to include/exclude servers
  - Mechanism required to
    - Assign unique identifier to new servers
    - Determine the state (new/retired)

Anti-entropy Extensions

- **Server creation**
  - By sending *creation write* to another server R
  - R acknowledges by accepting the write
    - (∞, $acceptTime_R$, serverId_R) added to R.*writeLog*
  - $serverId_S = (acceptTime_R, serverId_R)$
  - $acceptTime_S = acceptTime_R + 1$
    - Preserves casual-accept-order for future writes

Anti-entropy Extensions

- **Server retirement**
  - By issuing a *retirement write* to *itself*
  - No new writes are accepted by S
  - Must *remain alive* until it performs a *successful* anti-entropy (with R)
  - Prefix property requires R to
    - Process all *accepted writes* from S before removing S from version vector

Anti-entropy Extensions

- **Missing VV entry**
  - The Problem
    - S starts an anti-entropy session with R
    - S has a write from X, but R does not know about X
  - Two scenarios
    - R does not know about new X
      - S sends all writes (X accepted) to R
    - R knows X has retired
      - S should not send (X accepted) writes to R

Anti-entropy Policies

- **Four policies for reconciliation**
  - When
    - Periodic, manual, event based
  - Choosing receiver
    - Connectivity, receiver state
  - Write-log truncation
    - Trade off between storage and network resources
  - Server selection for new server creation
    - Mostly all of the above

Conclusion

- **Basic Design**
  - Pair-wise reconciliation, exchange of writes, writes governed by an order
- **Incremental**
- **Eventual reconciliation**
- **Session guarantees**
- **Lightweight Server management**
  - Based on anti-entropy

Bayou Anti-entropy Protocol

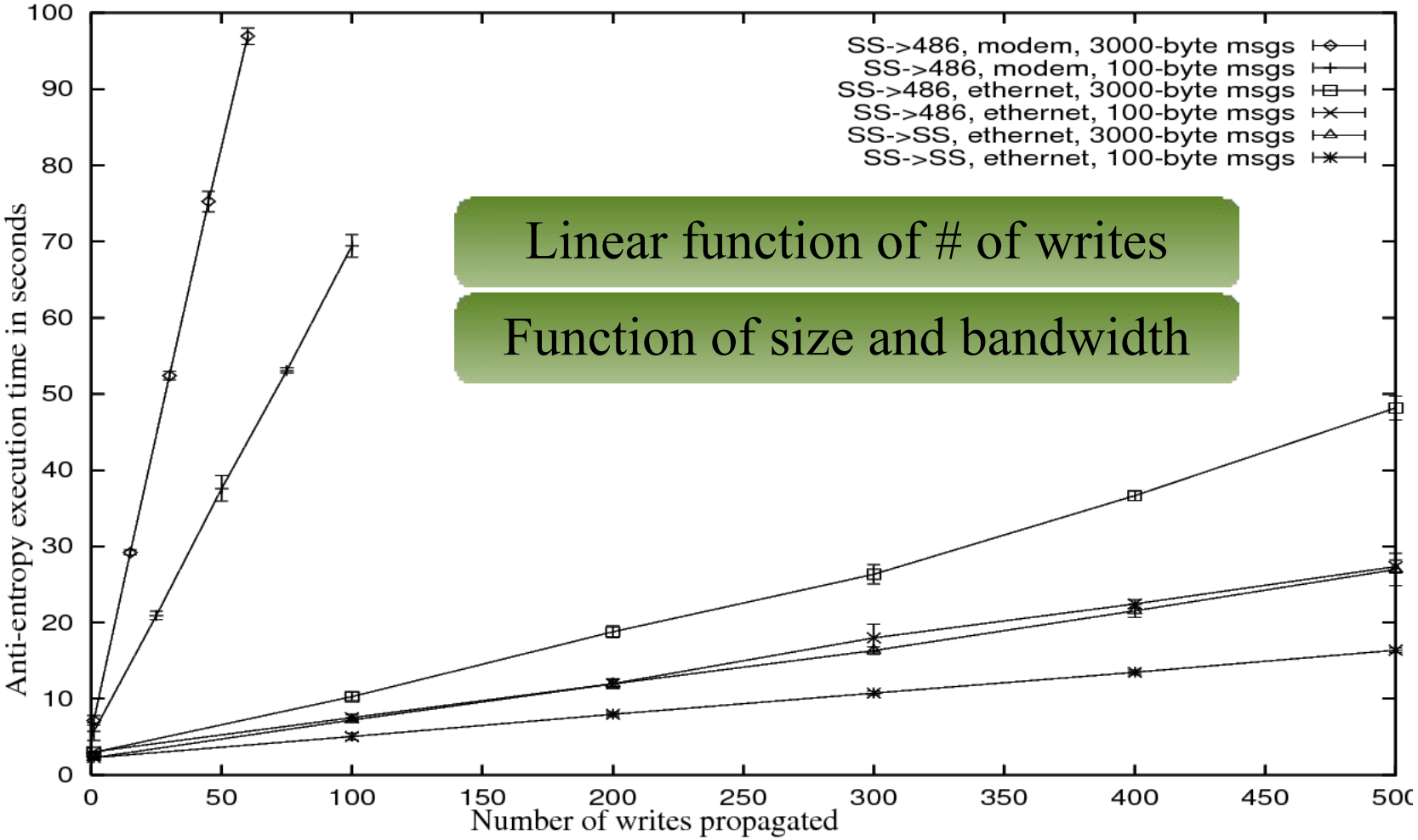
THANK YOU!

Discussion

- **Security**
  - Based on certificates
  - Servers are authenticated
  - Certificates are exchanged per write
- **Performance**
  - Number of writes
  - Bandwidth
  - Improvement could be made to
    - Security scheme
    - Header utilization

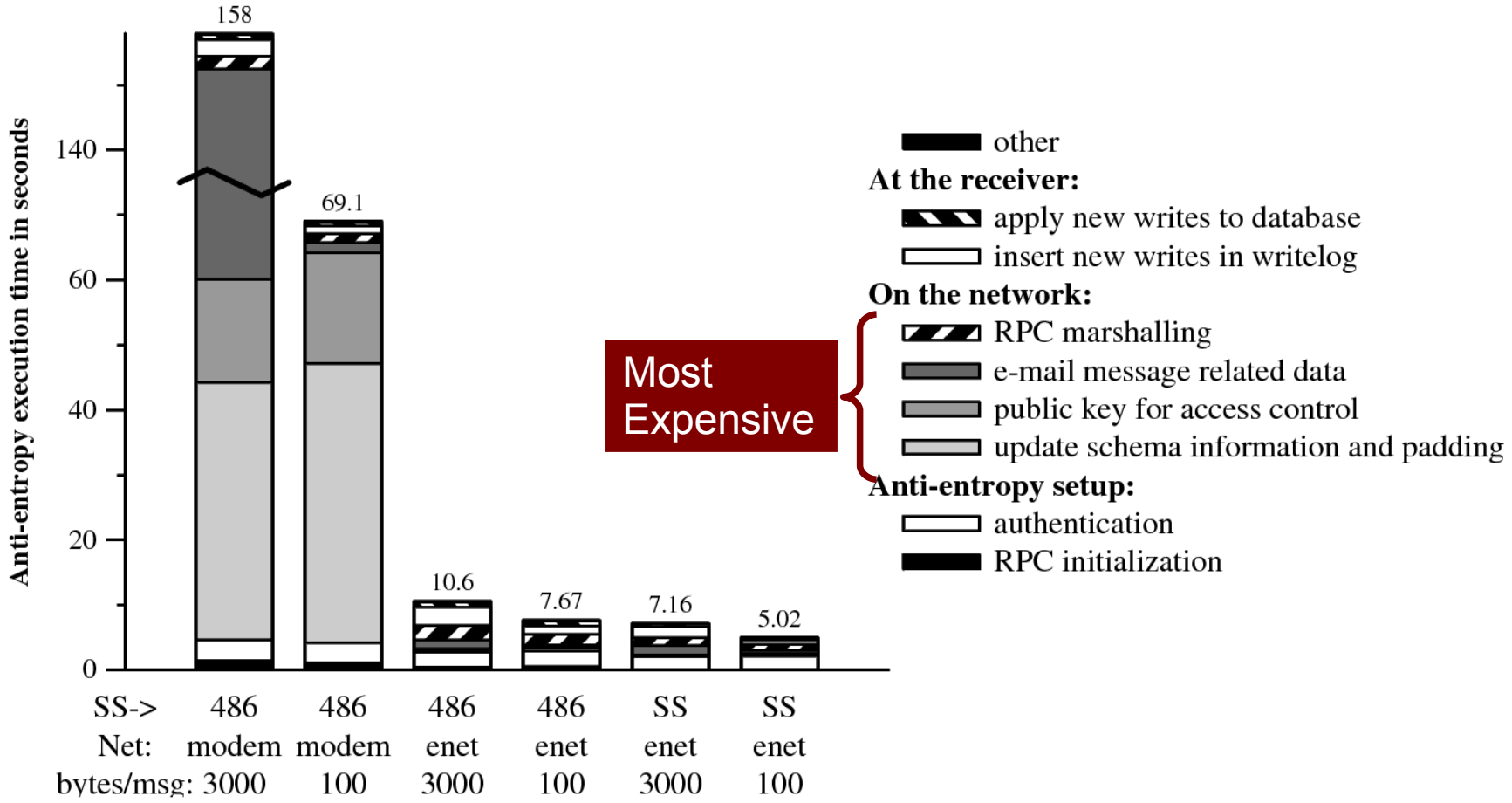
Discussion

- **Performance**



Discussion

❖ Performance



Discussion

- **Why not group-wise**