Practical uses of synchronized clocks in distributed systems

Liskov, Barbara. "Practical uses of synchronized clocks in distributed systems."
Distributed Computing 6.4 (1993): 211-219.

Presented by Fernanda Mora
April, 2016

“Memory is the diary that we all carry about with us”.

~Oscar Wilde, The Importance of Being Earnest

Contents

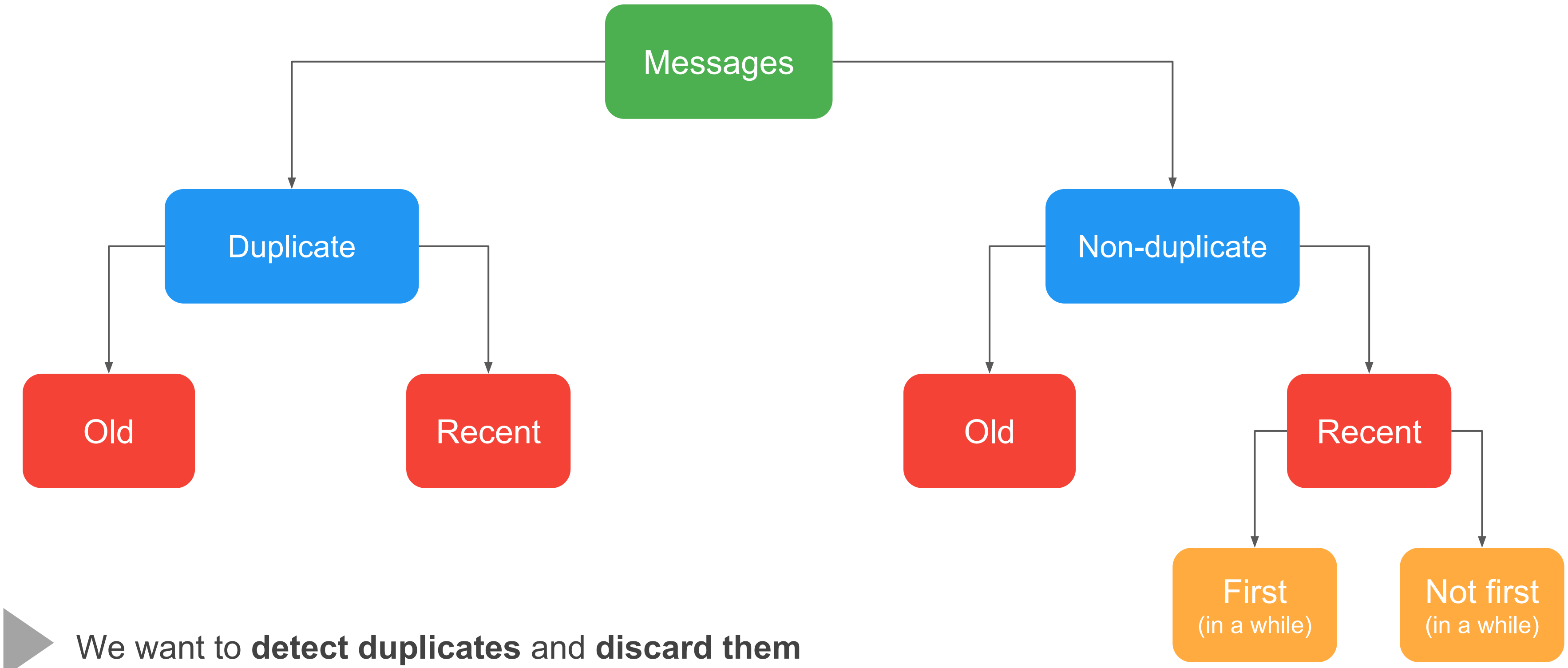
Introduction

Motivation

- SCs can improve the performance of a distributed algorithm replacing communication with local computation.
- This paper describes the practical role of SC in distributed algorithms of current/future use.
- We assume a CS algorithm already exists: $\forall$ clocks c_1, c_2 $P(|Tc_1 - Tc_2| < \varepsilon) \approx 1$

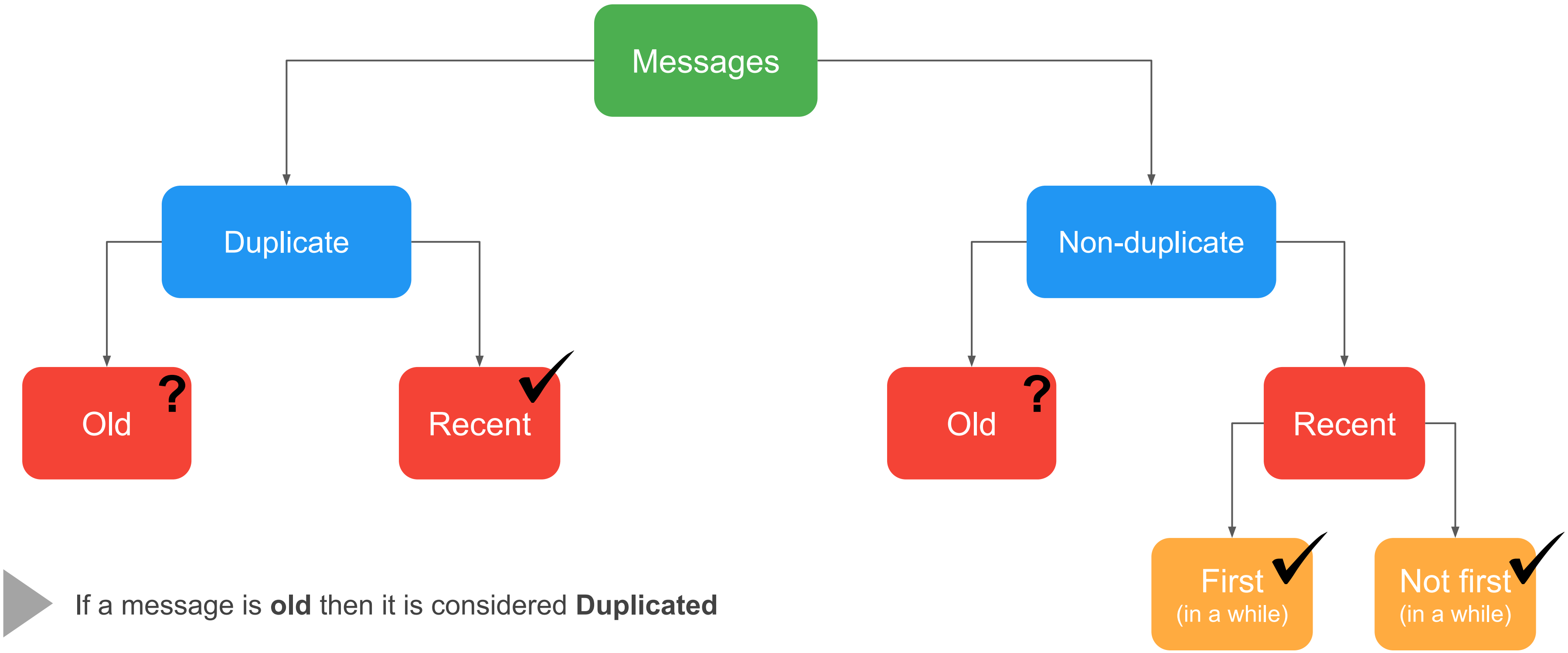
At-most-once messages

A possible classification of messages



We want to **detect duplicates** and **discard them**

This algorithm identifies correctly



- If a message is **old** then it is considered **Duplicated**
- **Error** when the old message **was not duplicate** -> but this error is maintained **low**

1st algorithm using SC: SCMP

- Guarantees at-most-once delivery at low cost.
- Provides an absolute guarantee that all duplicate messages will be detected.
- Building block for protocols with higher performance.

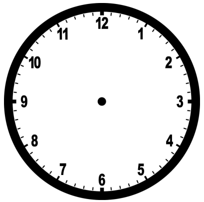
1st algorithm using SC: SCMP

- Receivers keep a table of active senders to determine duplicated messages -> how large?

- If no info about the sender then accept/reject? reject

  1st time in-a-while messages? Handshakes (channel): not always cost-effective.

- SCMP (synchronized clock message passing) uses SC to remember recent communications.



- What is a recent message?

1st algorithm using SC: SCMP

PROS: never accept a duplicate.

CONS: Non-duplicated messages are rejected sometimes.

Protocol ingredients

- Every node reads G.time from local clock
- Every message m has a unique message id with:
  - Time-stamp m.ts (same ts for copies of m)
  - Connection identifier m.conn (different for each sender and type of message)
- Every receiver has:
  - Connection table G.CT: has timestamp of the last message accepted on the connection
  - If G.CT[C].ts≤G.time-ρ-ε then C is old
    - Remove C from G.CT
    - Update bound: G.upper=max(G.upper, G.CT[C].ts). G.upper is a **representative** of old messages.

ρ=message lifetime interval.
ρ>>ε
Minimizes probability of false positives

Algorithm

- Idea: determine a per-connection bound that distinguishes recent from old:
  - if m.conn ∈ G.CT then its bound is the most recent timestamp and recent
    - Check if is duplicated
  - else (it is first or old) its bound is G.upper:
    - if $t_{last} \leq G.upper \leq G.time - \rho - \varepsilon$ then old and discard
    - G.upper<m.ts, then first and store in G.CT
- How to determine if a message that arrives after crash is a duplicate of a message that arrived before a crash?
  - Keep on stable storage a timestamp G.latest=G.time+b>m.ts for all m
  - If m.ts > G.latest then discard or delay.
  - After a crash G.upper=G.latest

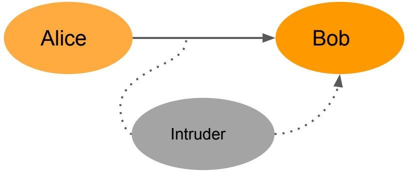
Algorithm

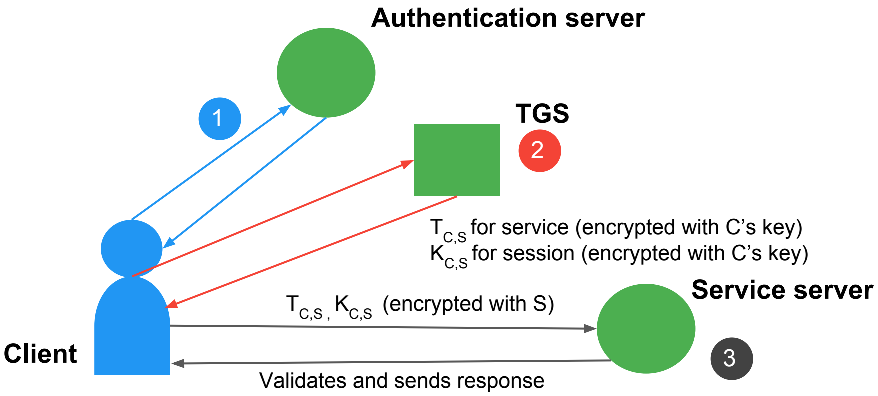
- We established a quantitative notion of recent and old.
- A reasonable way to detect duplicates.
- We avoided frequent communications.
- We saved storage at receivers by only keeping recent messages
- If clocks get out of synch there is no risk of accept duplicates but:
  - Clock is slower: its messages are more prone to be considered duplicates
  - Clock is faster: it considers messages as duplicates more easily

Authentication tickets in Kerberos

Kerberos

- Allows secure and authenticated communication between client-server.
- Uses private keys encrypted with DES
- Uses synchronized clocks to limit use of tickets and to help detect replay attacks.





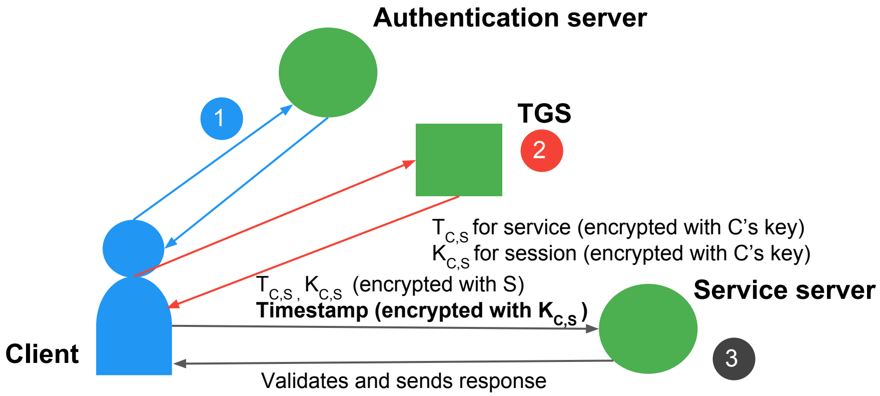
CONTROL USE OF TICKETS-KEYS!

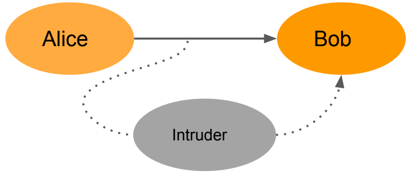
Limit use of keys: Avoid tickets to be stolen in absence of logout.

- Tickets have expiration time *E*: ***E<Cs-ε* ticket is valid**
- SC avoids communication between server and TGS

Kerberos

- Kerberos protects from malicious users trying to steal keys or trick the system about keys.
- Using CS in Kerberos requires that the CS algorithm is protected against similar threats like turning clocks very slow.





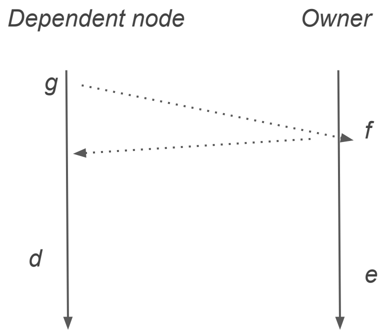
Help detect replay attacks:

- Uses authenticators: encrypted timestamps
- Very difficult to recreate: replayer can only reuse them
- Intercepted messages arrive **later**
- Messages with old authenticators are discarded
- Replays are still possible but rare if done **quickly** or if server is **slow**

Synchronized rates

How to use Synchronized rates

- In abstract:
  - Event of interest e (expiration of a privilege)
  - Owner of the event *e*, say O_e
  - T_e is the real time at which e occurs
  - Other nodes D depend on e and approximate to it using d occurring at T_d
  - We want to guarantee $T_d \leq T_e$



$$T_d = T_g + \lambda - \varepsilon$$

λ is the expiration interval, ε skew on how rates vary over λ

$$T_e = T_f + \lambda$$

$$T_g \leq T_f \quad \text{then } T_d \leq T_e$$

When to use them?

- When there is a communication already that allows the dependent node to estimate when the event of interest happens.
  - "At-most-once messages": Owner sends message to dependent without notification -> can't use rates
  - Kerberos: we can use rates
    - Event of interest: expiration of ticket.
    - Server is dependent, TGS is owner
    - If TGS communicate with the server before granting the ticket then the response of server is **g**
    - **f** is TGS's granting of the ticket.
    - But server doesn't communicate with TGS

Discussion

Remarks

- What if clocks get out of synchronization?
  - No effect in correctness in both algorithms (in Kerberos server should keep timestamps)
  - Other failures matter more: overuse tickets in Kerberos vs stolen tickets.
- We need to approximate “real rates”:
  - At-most-once messages: ρ should be approximate real delay
  - Kerberos: a ticket that is intended to last for one hour should last that hour
- How to incorporate SC in a distributed algorithm?
  - Find a way to avoid communication using timestamps (retain state for example)
  - Analyze consequences under worst case

“Memory is the diary that we all carry about with us”.

~Oscar Wilde, The Importance of Being Earnest

References

- Krzyzanowski, Paul. "Clock Synchronization." Lectures on distributed systems 2002 (2000).
- Liskov, Barbara, Liuba Shrira, and John Wroclawski. "Efficient at-most-once messages based on synchronized clocks." ACM Transactions on Computer Systems (TOCS) 9.2 (1991): 125-142.
- Liskov, Barbara. "Practical uses of synchronized clocks in distributed systems." Distributed Computing 6.4 (1993): 211-219.

Thanks