Threshold Logical Clocks

Manuel Vidigueira
Distributed and Decentralized Systems Lab (DEDIS)
École polytechnique fédérale de Lausanne (EPFL)

Supervised by Bryan Ford and Ceyhun Alp
Outline

- Motivation
- Threshold Logical Clocks (TLC)
- Experimental Results
- Using TLC
- Conclusion
Outline

- Motivation
- Threshold Logical Clocks (TLC)
- Experimental Results
- Using TLC
- Conclusion
Network models

**Synchronous**
- Synchronized clocks
- Bounded message transmission delay
- Bounded processing time

**Partially Synchronous**
- (Mostly) Asynchronous
- Eventually it behaves like a synchronous network

**Asynchronous**
- No assumptions

**Easier to prove/analyse**

**More robust**

Can we get the best of both worlds?
Measuring time in asynchronous systems

Node clocks can be out of sync!
Logical time: vector clocks

Alice

Meet tomorrow?  A: 1, B: 0

Bob

Meet today?  A: 2, B: 0

Yes!  A: 1, B: 1

Same order (and correct)

Nodes keep track of how many messages they saw from others
Adversarial models

Crash-stop
- Nodes only fail by crashing

Byzantine
- Nodes can do anything (behave arbitrarily)
No tolerance of byzantine failures!

Meet tomorrow? A: 1, B: 0, ...
Meet today? A: 2, B: 0, ...
Meet tomorrow? A: 2, B: 1, ...
Meet today?
Yes!
A: 2, B: 1, ...
Meet tomorrow?
Yes!
Meet today?

Hey Bob...
A: 2, B: 0, ...

Messages arrive out of order

Meet tomorrow?
Yes!
Meet today?
Nodes can advance arbitrarily forward in time. No enforced group synchronization!
Outline

● Motivation

● Threshold Logical Clocks (TLC)

● Experimental Results

● Using TLC

● Conclusion
Threshold Logical Clocks

Idea:

- Time is represented by a **round** number $R$
- Nodes must have received a threshold $T$ of messages to **advance** to the next round and send another message.

![Diagram showing the concept of Threshold Logical Clocks with Alice and Bob and round numbers R: 0, R: 1, R: 2, R: 3, and T = 2]
TLC - Design goals

Security goals

1. Fully Asynchronous
   No use of timeouts or synchronous assumptions.

2. Byzantine Fault Tolerant
   Can tolerate as many byzantine or malicious nodes as possible

Performance goals

3. Liveness
   Honest nodes must be able to make progress (go to next round)

4. Low latency
   Rounds should be fast and use few round trips.

5. Low bandwidth usage
   Should scale to at least 100s of nodes
TLC Interface

Every round:

- Provide a valid message $m$
- Receive a set $S$ of valid messages ($|S| \geq T$)

A validation function $f_{val}$ filters bad messages
What we want:

Round 0

TLC

Round 1

TLC

Round 2

TLC

Real time
TLC Interface

Two main parameters:

- message threshold $T$
- acknowledgement threshold $A$

**Certified** message:

- appears in the set $S$ of $A$ different nodes (same round)

Every set $S$ returned by TLC:

- contains at least $T$ different **certified** messages
Simple TLC

Every round has a logical time associated to it (0, 1, 2…)

Every round, each node:

1. Broadcasts its message, appending the round time
2. Broadcasts signed ACK for messages of that round
3. Waits for T messages where each has A different ACK
4. Delivers messages received and broadcast in that round
5. Increments round.
Communication pattern

Messages for one node

Broadcast
$O(N)$

Acknowledgements
$O(N^2)$

Simple TLC round split by trip time
Communication pattern

Messages for all nodes

Broadcast
$O(N^2)$

Acknowledgements
$O(N^3)$

~TLC round split by trip time
Threshold Witnessed TLC

Every round, each node:

1. Broadcasts its message, appending the round time
2. Sends signed ACK for messages of that round to their sender
3. Waits for A Acks for its message, aggregates signatures and sends certified message (message + signature).
4. Waits for T certified messages.
5. Delivers messages received and broadcast in that round
6. Increments round.
Communication pattern

Messages for one node

Broadcast
O(N)

Acknowledgements
O(N)

Rebroadcast
O(N)

Threshold Witnessed TLC round split by trip time
Communication pattern

Messages for all nodes

Broadcast \(O(N^2)\)

Acknowledgements \(O(N^2)\)

Rebroadcast \(O(N^2)\)

Threshold Witnessed TLC round split by trip time
Outline

● Motivation

● Threshold Logical Clocks (TLC)

● Experimental Results

● Using TLC

● Conclusion
Implementation & Experimental Setup

Implementation

- Go
  - Simple: ~420 lines
  - Threshold Witnessed: ~575 lines

- Libraries:
  - Kyber crypto library
  - Onet network library

- [https://github.com/dedis/student_19_tlc](https://github.com/dedis/student_19_tlc)

Deterlab setup

- 10 physical machines

- Network configuration:
  - 100 Mbps bandwidth
  - 200 ms round-trip latency
  - 1KB payloads
Evaluation: Bandwidth

\[ T = A = \frac{n+1}{2} \]

\[ T = A = \frac{2n+1}{3} \]
Evaluation: Round Time

T = A = (n+1)/2

T = A = (2n+1)/3
Outline

- Motivation
- Threshold Logical Clocks (TLC)
- Experimental Results
- Using TLC
- Conclusion
Potential Applications

- Threshold Cryptographic Signing
- Threshold Cryptographic Randomness
- Randomized Asynchronous Consensus
  - The communication logic is reduced to TLC time-steps.
  - Can be used for Byzantine consensus as well.
  - Details are currently in the works.
Outline

- Motivation
- Threshold Logical Clocks (TLC)
- Experimental Results
- Using TLC
- Conclusion
Conclusion

- Threshold Logical Clocks:
  - robust round based communication
  - group based notion of time
  - implementation with reduced bandwidth and latency
  - scales to 100s of nodes
  - many potential applications

Thanks!