Technische Universität Braunschweig

Gefördert durch DFG Deutsche Forschungsgemeinschaft

Institute of Operating Systems and Computer Networks

Hybrid Fault-Tolerant Consensus in Asynchronous and Wireless Embedded Systems

22nd International Conference on Principles of Distributed Systems

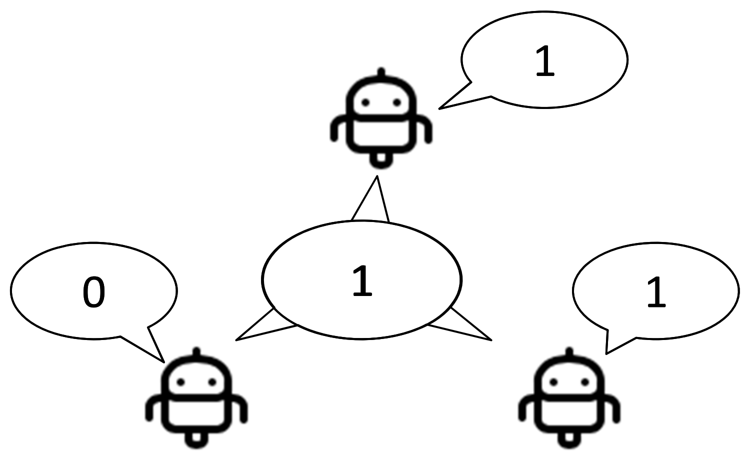
Wenbo Xu, Signe Rüsch, Bijun Li, Rüdiger Kapitza
TU Braunschweig

Controlling Concurrent Change

18.12.2018, Hong Kong

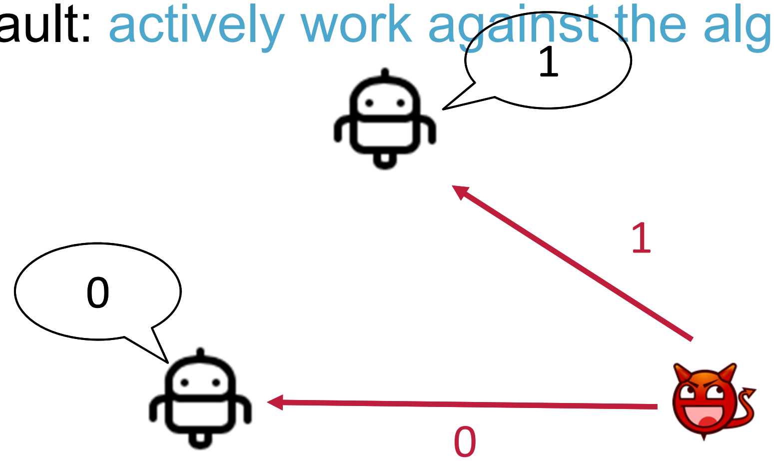
Background: (Binary) Byzantine-Fault Tolerant Consensus

- Fundamental problem in distributed systems
- Totally *n* node in the group, each proposes a value , 0 or 1
- In the end all nodes should decide the same value → consensus



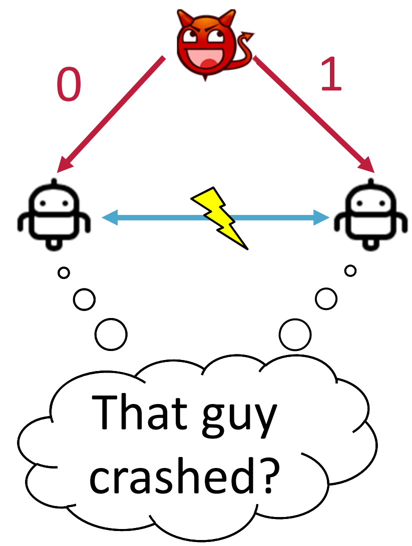
Background: (Binary) Byzantine-Fault Tolerant Consensus

- Fundamental problem in distributed systems
- Totally *n* node in the group, each proposes a value , 0 or 1
- In the end all nodes should decide the same value → consensus
- At most *f* faulty nodes
  - Crash
  - Byzantine fault: actively work against the algorithm



Background: Asynchronous System

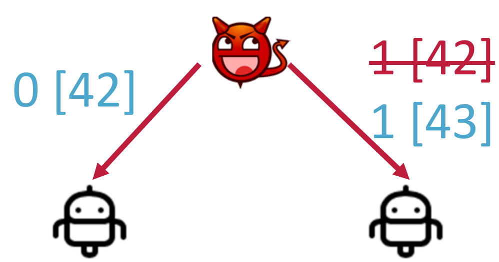
- Nodes communicate via messages
- Asynchronous network
  - No message omissions
  - But messages can take arbitrarily long time
  - → Too slow? Or he didn't send? Cannot wait forever!



- Strong adversary: the worst case
  - The adversary can inspect the status of every message and node
  - … then reorder arrivals of messages, and adjust faulty nodes' behavior
  - Cannot break cryptography and a trusted subsystem

Background: Hybrid Fault Model

- Trusted subsystem, tamperproof
- A strict monotonic counter to prevent “two-faced cheating”
- Faulty nodes cannot send contradictory messages in one broadcast



Related Work and Motivation

- Randomization to bypass FLP impossibility of asynchrony
  - Crash fault tolerance with $n \geq 2f+1$: Ben-Or's algorithm [1]
  - Byzantine fault tolerance requires $n \geq 3f+1$
- Limit the Byzantine behavior with a trusted subsystem
  - Only requires $n \geq 2f+1$
  - Built upon complex algorithm stacks, e.g. reliable broadcast primitive
  - Not resilient against strong adversary → not terminate in worst

2f+1 consensus , but less complex and suitable in wireless embedded systems

Correctness proof under all cases, even strong adversary

[1] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, 1983.

Outline

- Trusted-Ben-Or Algorithm
- A Common Issue in the Proof of Termination
- Experiment

Original Ben-Or's Algorithm

Round based, 2 phases per round

PR: Propose Phase

Propose a value 0 or 1

Round		Node 1	Node 2	Node 3
1	PR	0	1	0
	VO			
2	PR			
	VO			

Ben-Or's Algorithm

Round based, 2 phases per round

PR: Propose Phase

VO: Vote Phase

Wait for *(n-f)* proposals

If >n/2 propose the same *v*

→ Vote for *v*

Else

→ Vote for ⊥ (default)

Round		Node 1	Node 2	Node 3
1	PR	0	1	0
	VO	0	⊥	⊥
2	PR			
	VO			

Ben-Or’s Algorithm

Round based, 2 phases per round

PR: Propose Phase

VO: Vote Phase

PR: Propose Phase

Wait for *(n-f)* votes

If all vote for ⊥

→ Propose (*$*, *R*),

$ is a random value

If someone votes for *v*

→ Propose (*v*, *D*)

Round		Node 1	Node 2	Node 3
1	PR	0	1	0
	VO	0	⊥	⊥
2	PR	0, D	0, D	0, R
	VO			

R = Randomly get the value

D = Deterministically get the value

Ben-Or's Algorithm

Round based, 2 phases per round

PR: Propose Phase

VO: Vote Phase

PR: Propose Phase

VO: Vote Phase

...

If >n/2 vote for the same *v*

→ Decide *v*

Round		Node 1	Node 2	Node 3
1	PR	0	1	0
	VO	0	⊥	⊥
2	PR	0, D	0, D	0, R
	VO	0	0	0
		decide 0	decide 0	decide 0

Ben-Or’s Algorithm

Round based, 2 phases per round

PR: Propose Phase

VO: Vote Phase

PR: Propose Phase

VO: Vote Phase

…

Round		Node 1	Node 2	Node 3
1	PR	0	1	0
	VO	0	⊥	⊥
2	PR	0, D	0, D	0, R
	VO	0	0	0
		decide 0	decide 0	decide 0

Only tolerate crash fault, no Byzantine fault!

Trusted-Ben-Or: Tackle Byzantine faults

- Message uniqueness per phase

→ Trusted monotonic counter for message authentication

- Unbiased random number

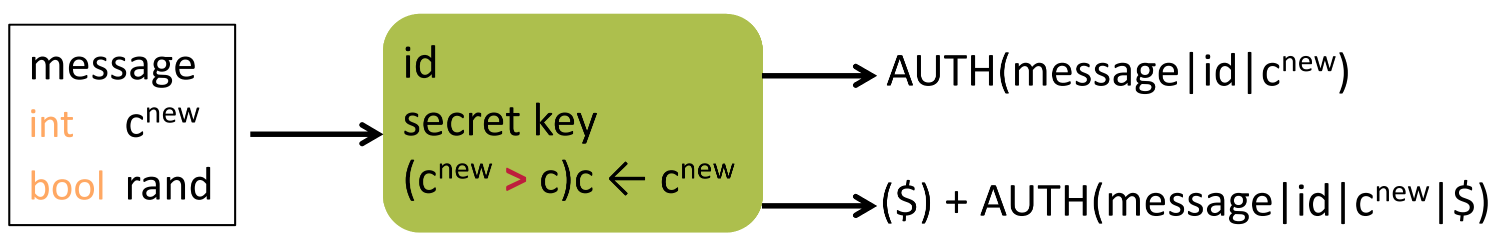
→ Trusted random number generator (combined with the counter)

- Semantic correctness

→ Message certificate

Message Uniqueness | Unbiased Random | Semantic Correctness

- In round *k,* each node only sends 2 messages
- Trusted monotonic counter authentication:
  - <PR, k, *, *> with counter value [k|0]
  - <VO, k, *> with counter value [k|1]
- Trusted random number generator
- Protected by hardware, can only crash but not Byzantine



Message Uniqueness | Unbiased Random | **Semantic Correctness**

- Piggyback received, authenticated messages to proof the correctness
- No recursive certificates
  - Limited message size (≤ $n+2$ messages in one certificate)
  - Faulty node can include invalid into a certificate

Adaption to Embedded Wireless Systems

- Local broadcast instead of peer-to-peer communication
- Tackle (limited) omission faults:
  - Stubborn re-transmission of last message
  - Round jumping when received a valid message of future round

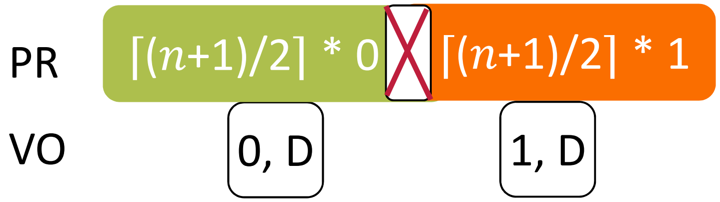
  → No specific network protocols / primitives required for reliable communication
- HMAC in trusted subsystem instead of digital signature

Outline

- Trusted-Ben-Or Algorithm
- A Common Issue in the Proof of Termination
- Experiment

Proof of Termination

- No valid proposals of (0, D) and (1, D) at the same time



- In a lucky round:
  - All trusted coins of each node toss the same random value *v*
  - … which is the same as the valid deterministic value

→ Terminate in this round

Proof of Termination

A corner case of flaw

- Firstly let a node R-get v

Round	Node 1	Node 2	Node 3
PR	0,D	1,D	
VO	⊥	⊥	
PR	0,R		
VO			
PR			

Proof of Termination

A corner case of possible flaw

- Firstly let a node R-get v
- Then let another node D-get (1-

→ Turn the lucky value into unluck

Round	Node 1	Node 2	Node 3
PR	0,D	1,D	1,D
VO	⊥	⊥	1
PR	0,R		1,D
VO			
PR			

Proof of Termination

A corner case of possible flaw

- Firstly let a node R-get v
- Then let another node D-get (1-

→ Turn the lucky value into unluck

Round	Node 1	Node 2	Node 3
PR	0,D	1,D	1,D
VO	⊥	⊥	1
PR	0,R		1,D
VO	⊥		⊥
PR	1,R		

Is 0 still the lucky value here?

"Luckiness" should not depend on future events!

Marcos K Aguilera and Sam Toueg. The correctness proof of ben-or's randomized consensus algorithm. *Distributed Computing*, 25(5), 2012.

Proof of Termination

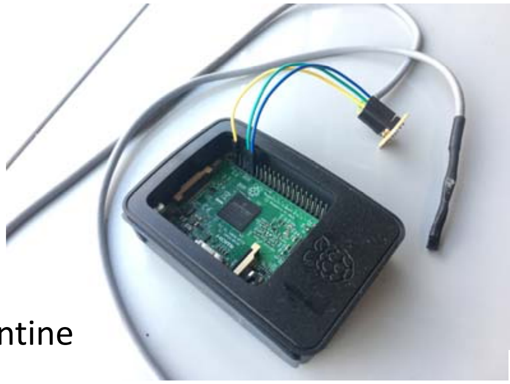
- In our work, termination is ensured by:
  - Counter authentication
  - Trusted random number generator
  - Semantic certificate
  - “Luckiness”
- Luckiness depends only on the current system state and past events!
- For more details please refer to our paper

Outline

- Trusted-Ben-Or Algorithm
- A Common Issue in the Proof of Termination
- Experiment

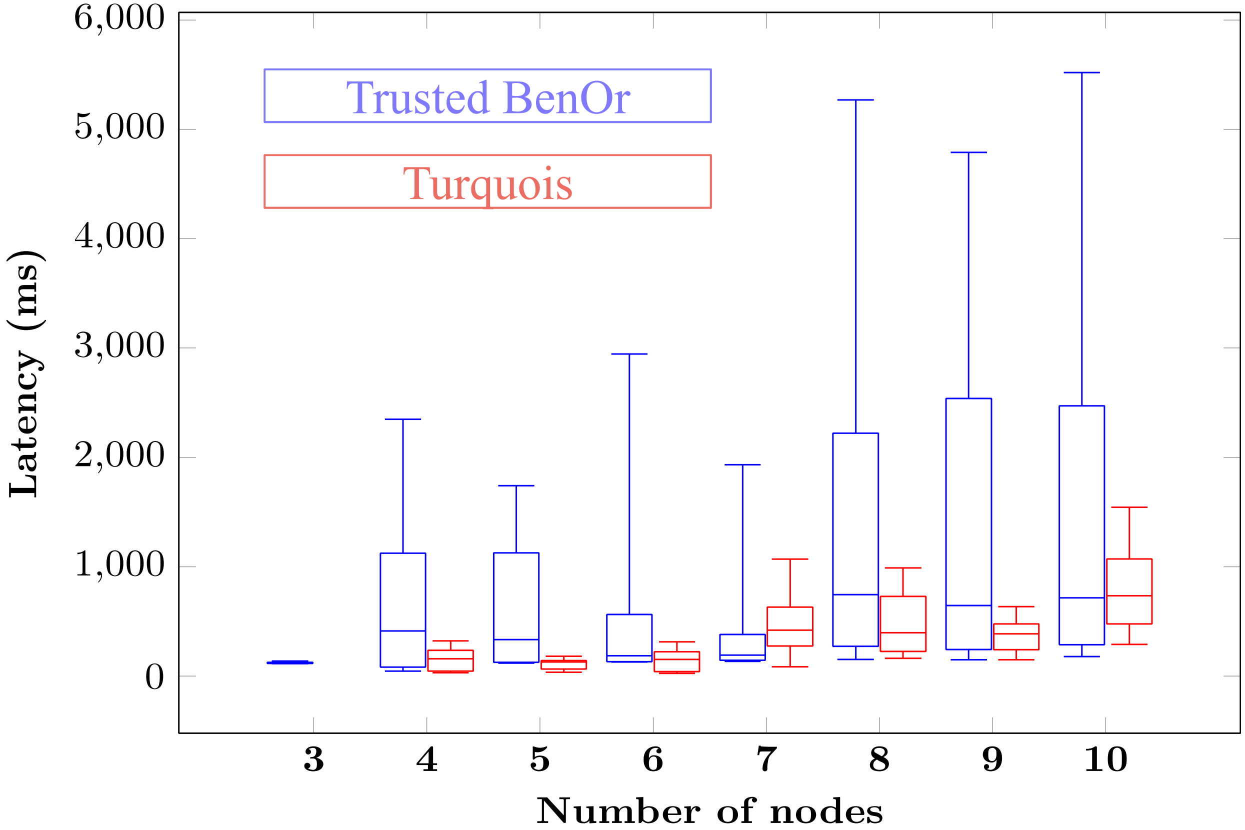
Experiments: Settings

- 3-10 RaspberryPi 3: ARM processor with TrustZone interface, distributed in different rooms in office building
- Wireless ad-hoc mode, UDP multicast
  - ICMP ping delay: (min, average, max) = (5.6 ms, 12.5 ms, >1000 ms)
  - iperf3 test: up to 24% data loss
- Trusted counter implemented on Linaro OPTEE
  - SHA-256 HMAC provided by OPTEE
- Compare with Turquois [1]



[1] Henrique Moniz, Nuno Ferreira Neves, and Miguel Correia. Turquois: Byzantine consensus in wireless ad hoc networks. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 537–546. IEEE, 2010.

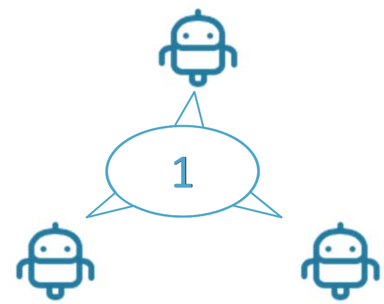
Experiment: Result with Byzantine Faults Injected



- Comparable median
- Higher variance
- Can tolerate more faults

Conclusion

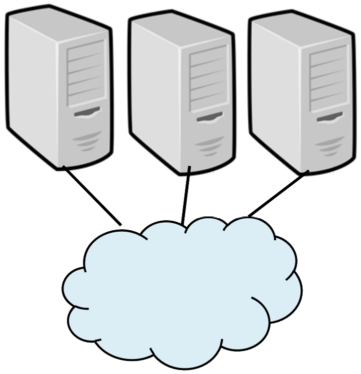
- Randomized binary consensus in asynchronous system
- Trusted monotonic counter for message authentication
- Resilient against strong adversary
- Tailored for embedded wireless systems
- Tolerate more faults with limited overhead (in most cases)

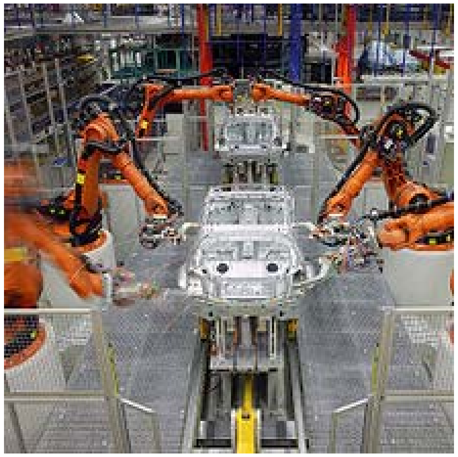


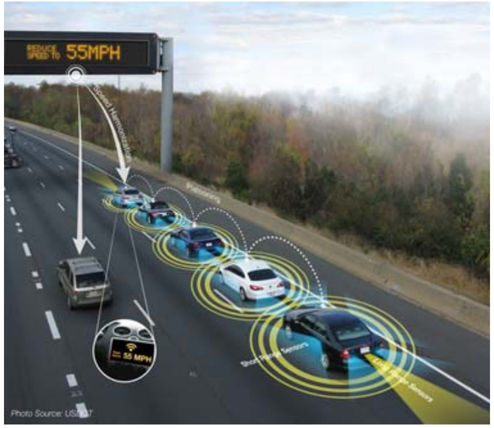
Thank you for your attention!

Motivation: Distributed Consensus

Replication system







Trusted BenOr Algorithm: Overview

//the initial round. Round number is omitted
send <PR, v, D-get>
wait for $\left\lceil \frac{n+1}{2} \right\rceil$ valid PR-messages
if $\left\lceil \frac{n+1}{2} \right\rceil$ <P, v> with the same v
    send <VO, v>
else
    send <VO, $\perp$>
wait for $\left\lceil \frac{n+1}{2} \right\rceil$ valid VO-messages
if $\left\lceil \frac{n+1}{2} \right\rceil$ <VO, v>
    decide v
//new round
if at least one <VO, v>
    send <PR, V, D-get>
else send <PR, \$, R-get>

Message Validity: Legality Certificate

Message type	When to send	Required certificate
<PR, k+1, v, R-get>	$\lceil(n+1)/2\rceil$<VO, k, ⊥>	$\lceil(n+1)/2\rceil$ <VO, k, ⊥>
<PR, k+1, v, D-get>	$\lceil(n+1)/2\rceil$ <VO, k, *> with at least one <VO, k, v>	$\lceil(n+1)/2\rceil$ <PR, k, v, *>
<VO, k+1, v>	$\lceil(n+1)/2\rceil$ <PR, k+1, v, *>	$\lceil(n+1)/2\rceil$ <PR, k+1, v, *> If there is a <PR, k+1, v, D-get>, then add $\lceil(n+1)/2\rceil$ <PR, k+1, v, *>
<VO, k+1, ⊥>	$\lceil(n+1)/2\rceil$ <PR, k+1, *, *> with different values	$\lceil(n+1)/2\rceil$ <PR, k+1, *, *> with different values Plus $\lceil(n+1)/2\rceil$ <VO, k, ⊥>

Proof of Agreement

A correct node decides *v* in round *k*

Exist $\lceil (n+1)/2 \rceil$ valid <VO, k, v>

Exist $\lceil (n+1)/2 \rceil$ <PR, k, v, *>

No valid <PR, k+1, 1-v, D-get>

No $\lceil (n+1)/2 \rceil$ < VO, k, ⊥>

No valid <PR, k+1, *, R-get>

Message type	Required certificate
<PR, k+1, v, D-get>	$\lceil (n+1)/2 \rceil$ <PR, k, v, *>
<VO, k+1, v>	$\lceil (n+1)/2 \rceil$ <PR, k+1, v, *> …

Message type	Required certificate
<PR, k+1, v, R-get>	$\lceil (n+1)/2 \rceil$ <VO, k, ⊥>

Proof of Agreement

A correct node decides *v* in round *k*

Exist $\lceil(n+1)/2\rceil$ valid <VO, k, v>

Exist $\lceil(n+1)/2\rceil$ <PR, k, v, *>

No valid <PR, k+1, 1-v, D-get>

No $\lceil(n+1)/2\rceil$ < VO, k, ⊥>

No valid <PR, k+1, *, R-get>

Only valid <PR, k+1, v, D-get>

Proof of Termination

Correct definition of "luckiness"

- 1 is lucky in round (3k+1)
- In round (3k-1) and (3k), depends on t0:
  - 0 is lucky before t0
  - Since t0

  Case A: a majority proposed 0 in (3k-1) → 0 is lucky

  Case B: no majority proposed 0 in (3k-1) → 1 is lucky

		Node 0	Node 1	Node 2
3k-1	PR			
	VO			
3k	PR	★		
	VO			
3k+1	PR	1,R	1,R	1,R
	VO			

t0 = the first time a correct node tosses a coin

"Luckiness" now only depends on current state and past events.

Trusted Subsystem: BiTrinc

- Restrict Byzantine nodes, not too faulty → Hybrid fault model
  - Most part can still be Byzantine
  - A small trusted part is never Byzantine (crash-fault-only)
- Can be protected by hardware. Example: ARM TrustZone, Intel SGX, other dedicated hardware security modules
- Minimal Trusted Computing Base
  - As simple and small as possible
  - Only critical functions and data