UNIVERSITÄT BERN

Foundations of Distributed Trust

Christian Cachin, University of Bern

#BDLT19, Vienna, September 2019

What is distributed trust?

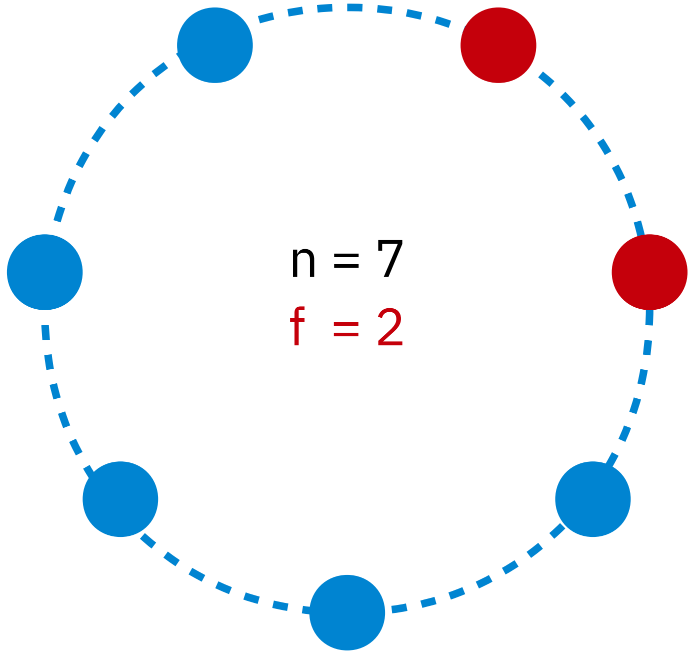
What is distributed trust?

- Do not trust any single node
- Tolerate corrupted nodes
- Everyone gets to have influence
- Majority is right
- System remains intact despite nodes that lie or misbehave

- Distributed protocols tolerate uncertainty, failures and attacks

Threshold trust assumption

- Trust by numbers
  - n nodes total
  - f faulty (Byzantine) nodes
- Typically requires $n > 3f$
- All nodes are equally trusted
- All nodes trust equally



b
UNIVERSITÄT
BERN

Symmetric trust

Why n > 3f ?

- Safety and liveness
- Reading from and writing to storage
- (Reliable broadcasts)
- (Consensus)

Safety & liveness

- Distributed protocols satisfy two orthogonal properties (Alpern & Schneider, 1987)

- Safety – Nothing "bad" will ever happen
  - Look at the past history if a "bad" event occurred
  - "Do nothing" is always safe

- Liveness – Something "good" will happen in the future
  - Look into the future of the execution if "good" event will occur
  - "Do something" is always live

- Only protocols that combine safety and liveness are useful

Blockchain safety

b
UNIVERSITÄT
BERN

- Participants reach the same decision in consensus

- All clients get the same view of the network's state
  - Ownership of "coins"
  - Assets of a smart contract

- Ledger does not fork

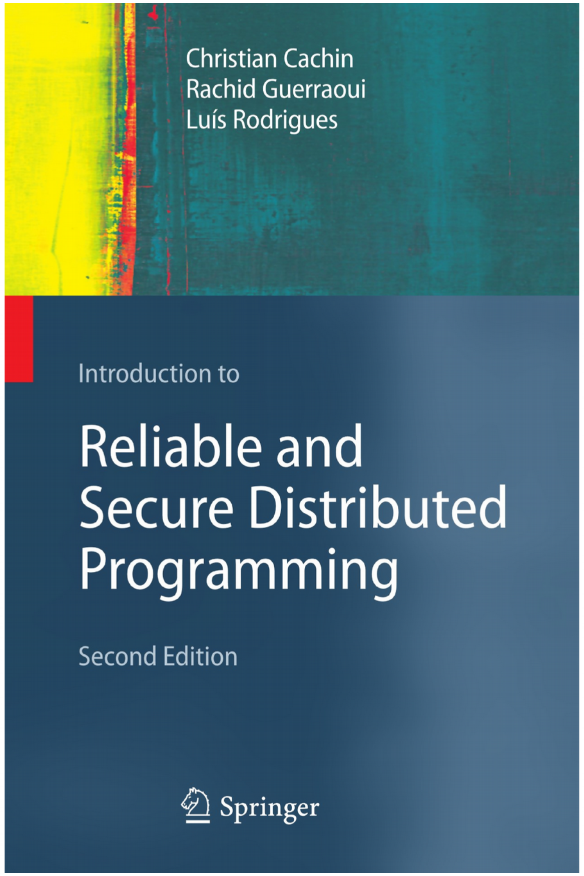
- Easy with a trusted centralized system, with no failures or attacks

Blockchain liveness

UNIVERSITÄT BERN

- Participants can execute transactions
- Network does not depend on manual intervention
- Market remains liquid due to continuous progress
- Ledger does not halt
- Easy with a trusted centralized system, with no failures or attacks

Background


Christian Cachin
Rachid Guerraoui
Luís Rodrigues

Introduction to

Reliable and Secure Distributed Programming

Second Edition

Springer

www.distributedprogramming.net

RW Register abstraction

- Two operations only
  - Write(x) → OK
  - Read() → x

- Every operation defined by two events
  - Invocation (IN)
  - Completion (OUT)

- Simplification
  - Only process w ("Whit") or p_w may write
  - Only process r ("Ron") or p_r may read

    ... single-reader, single-writer (SRSW) register

RW Register abstraction

- Convenient model for shared storage
  - Inspired by shared-memory multi-threaded or multi-processor systems ("processes" read and write)
  - Today also for cloud or blockchains (clients read and write)

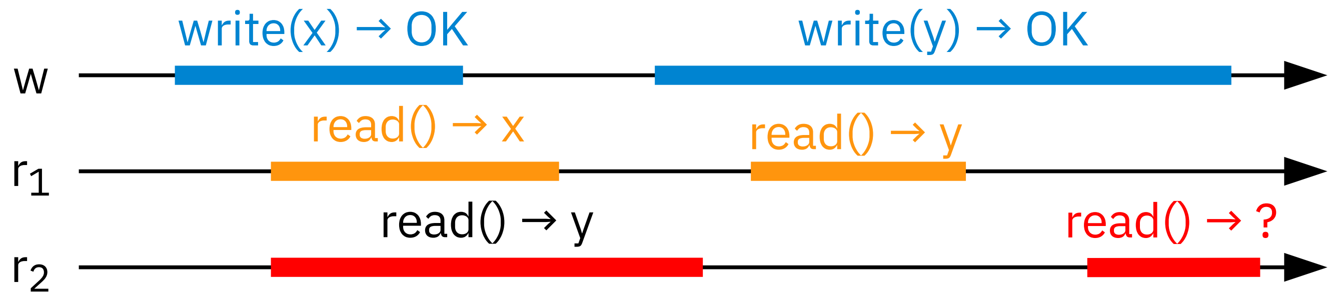
- Completely asynchronous
  - No common clocks
  - No bound on message delays of protocols
  - No bound on local processing time

- Many results on "Wait-free synchronization" (Herlihy & Wing, 1991)
  → CPU instructions: test-and-set or compare-and-swap

Concurrency

- Operation o precedes o' whenever completion of o occurs before invocation of o'
- Otherwise, o and o' are concurrent
- How should the RW register behave when accessed concurrently?

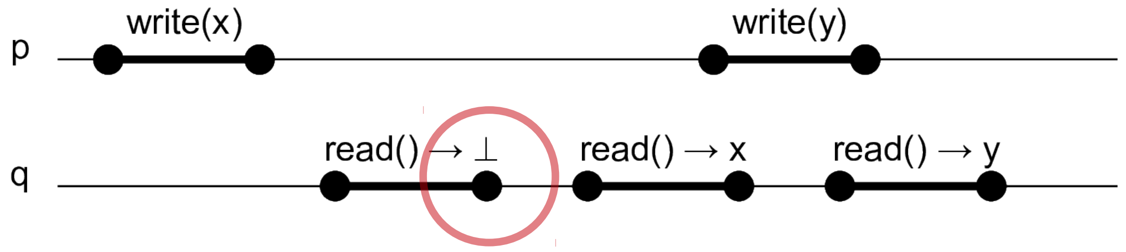
Semantics of MRSW register operations



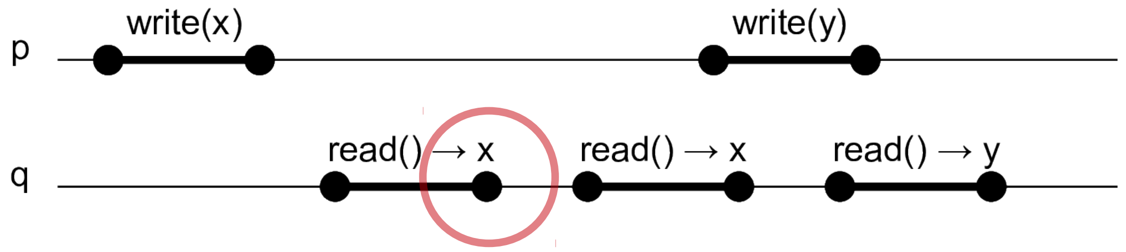
- Safe – Every read not concurrent with a write returns the most recently written value.
- Regular – Safe & any read concurrent with a write returns either the most recently written value or the concurrently written value: r_2 may read() → x or y
- Atomic – Regular & all read and write operations occur atomically (= linearizable): r_2 must read() → y

Example executions of SRSW register

- Not regular



- Regular



BFT implementation of SRSW reg. register

- Protocol with $n > 3f$ replicated processes, f may be Byzantine/faulty

- Writer p_W maintains a logical timestamp ts
  - Increments ts for each write() operation
  - Issues digital signature s on pair (ts, v)
  - Sends a timestamp/value/signature tuples (ts, v, s) to replicas
  - Waits for $> (n+f)/2$ replicas to acknowledge ("Byzantine quorum")

- Reader p_r asks replicas for their current (ts, v, s) tuples
  - Verifies that signature s from p_W is valid
  - Receives $> (n+f)/2$ such tuples ("Byzantine quorum")
  - Extracts value v with highest timestamp ts and returns v

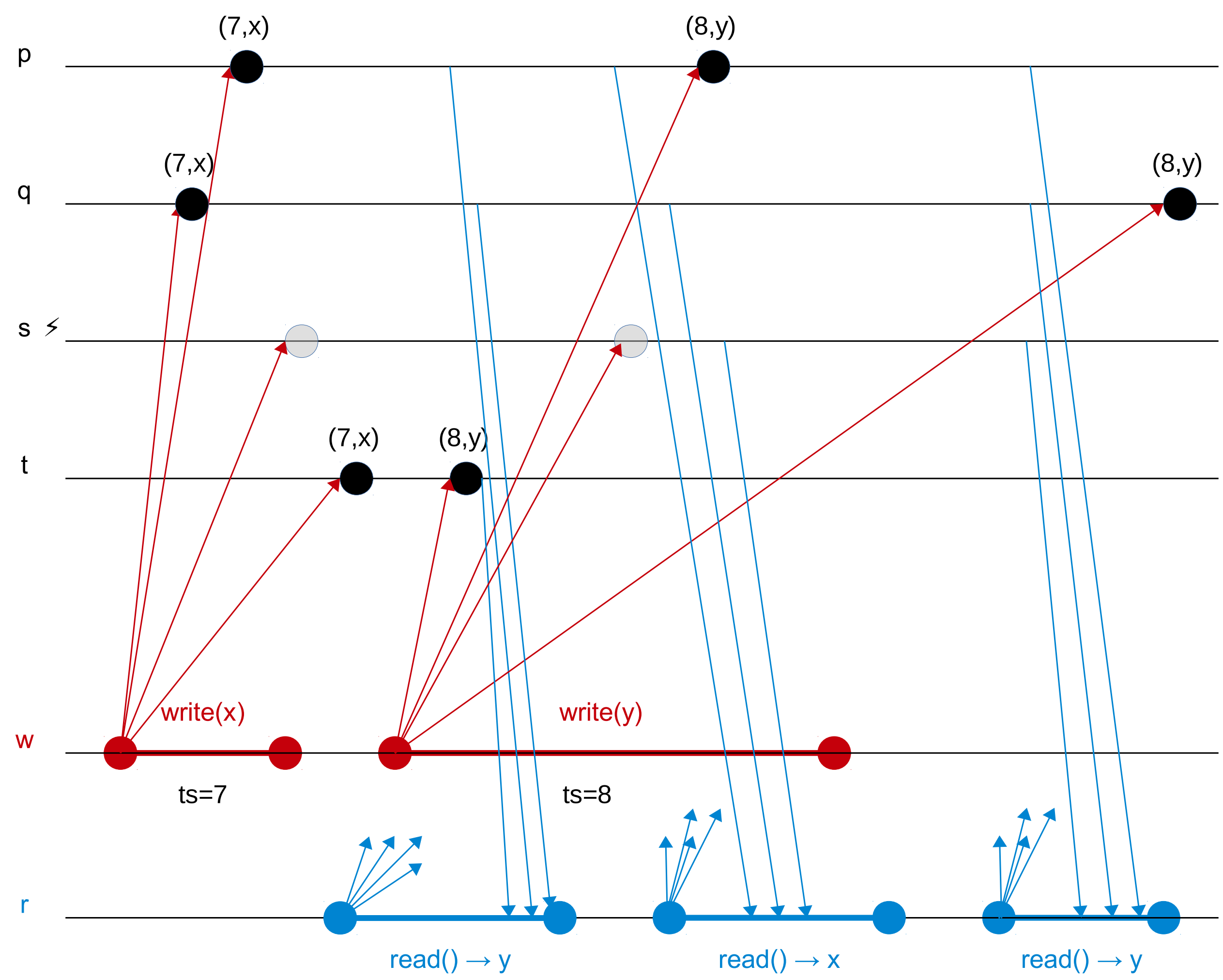
SWMR regular register protocol with Byzantine processes (process p_i).

```
State
    $wts$: sequence number of write operations, stored only by writer $p_w$
    $rid$: identifier of read operations, used only by reader
    $ts, v, \sigma$: current state stored by $p_i$: timestamp, value, signature

upon invocation $write(v)$ do                                    // only if $p_i$ is writer $p_w$
    $wts \leftarrow wts + 1$
    $\sigma \leftarrow_{\$} sign_w(\text{WRITE}\|w\|wts\|v)$
    send message $[\text{WRITE}, wts, v, \sigma]$ to all $p_j \in \mathcal{P}$
    wait for receiving a message $[\text{ACK}]$ from more than $\frac{n+f}{2}$ processes

upon invocation $read$ do                                        // only if $p_i$ is reader $p_r$
    $rid \leftarrow rid + 1$
    send message $[\text{READ}, rid]$ to all $p_j \in \mathcal{P}$
    wait for receiving messages $[\text{VALUE}, r_j, ts_j, v_j, \sigma_j]$ from more than $\frac{n+f}{2}$ processes such that
        $r_j = rid$ and $verify_w(\sigma_j, \text{WRITE}\|w\|ts\|v_j)$
    return $highestval(\{(ts_j, v_j)\})$

upon receiving a message $[\text{WRITE}, ts', v', \sigma']$ from $p_w$ do           // every process
    if $ts' > ts$ then
        $(ts, v, \sigma) \leftarrow (ts', v', \sigma')$
    send message $[\text{ACK}]$ to $p_w$

upon receiving a message $[\text{READ}, r]$ from $p_r$ do                  // every process
    send message $[\text{VALUE}, r, ts, v, \sigma]$ to $p_r$
```

u^b

^{b} UNIVERSITÄT BERN

Example SRSW register execution



Why regular?

- Read **without** concurrent write
  - Last write by p_w of (ts, x) has updated > (n+f)/2 processes to (ts, x)
  - Reader p_r obtains > (n+f)/2 value/timestamp pairs
  - Since any two sets of > (n+f)/2 overlap in > f, processes, at least one answer from honest
  - p_r receives one pair (ts, x) and outputs the most recently written value x

- Read **with** concurrent write
  - Either p_r either receives concurrently written value from (ts, x)
  - Or p_r outputs most recently written value, from argument above

BFT protocols in the threshold model

- Usually $n > 3f$ replicated processes
- f may be Byzantine/faulty

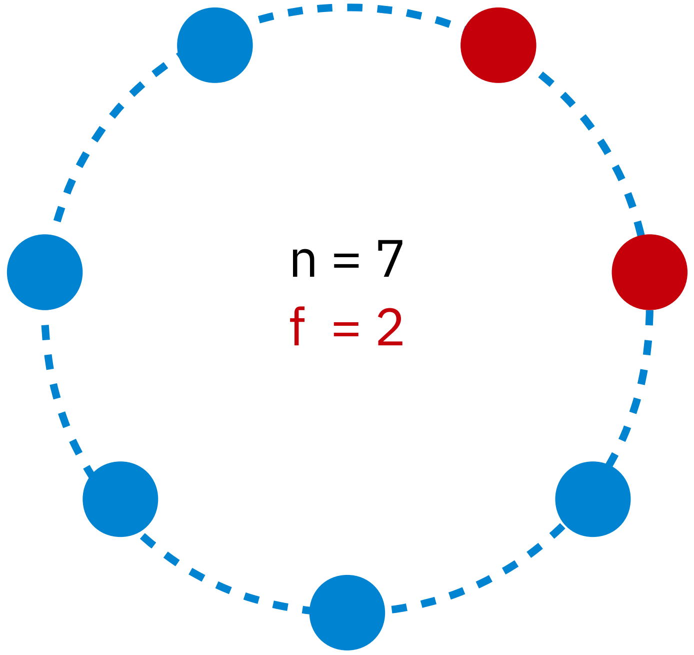
- Protocols implement many tasks
  - Consistent broadcast
  - Reliable broadcast
  - RW registers
  - Consensus
  - State-machine replication

UNIVERSITÄT
BERN

From symmetric to asymmetric trust

Recall – Threshold trust is symmetric

- Trust by numbers
  - n nodes total
  - f faulty (Byzantine) nodes
- Typically requires $n > 3f$
- All nodes are equally trusted
- All nodes trust equally



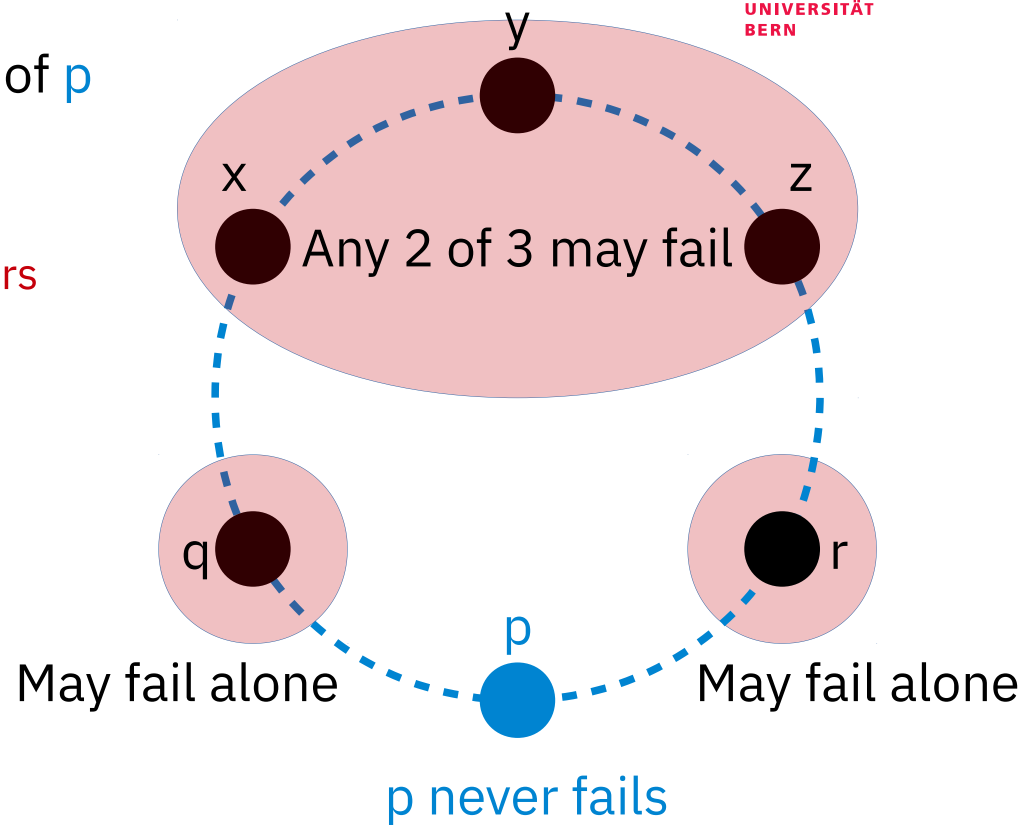
Asymmetric trust

- Subjective trust (= ¬failure) assumption of p
  - p itself never fails
  - Neighbor nodes q and r
    - May fail by themselves, not together with others
  - Remote nodes x, y, x
    - Any 2 of these 3 may fail together

- Fail-prone system of process p
  {{q}, {r}, {x,y}, {y,z}, {x,z}}

- What if each one of the 6 processes used its own subjective trust like this?

  (Positive answer follows!)



b
UNIVERSITÄT
BERN

Towards blockchains with asymmetric trust

b
UNIVERSITÄT
BERN

Ripple

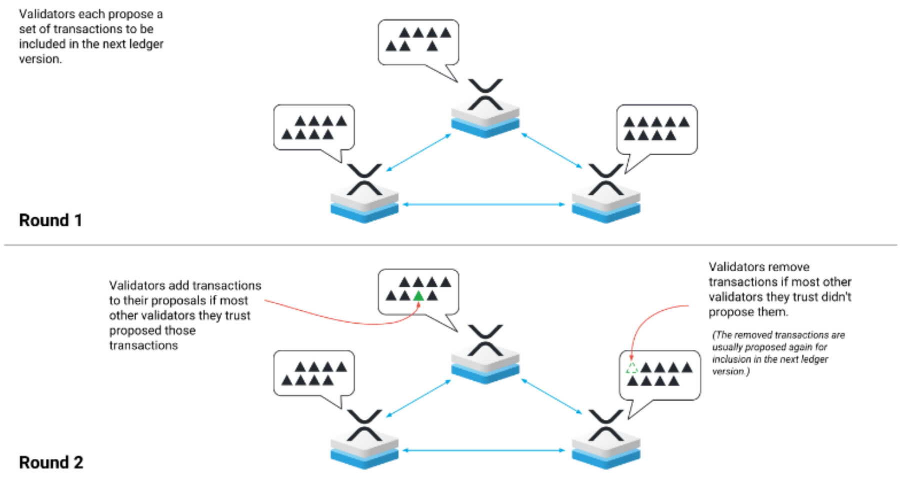
Consensus in Ripple

- Ripple started 2012
  - Today ranks 3rd by market cap
- Ripple protocol consensus algorithm
  - Schwartz, Youngs, Britto (2014)
- Each node declares which other nodes it trusts (Unique Node List)
- Intends to achieve Byzantine fault-tolerant consensus

Consensus

The servers on the network share information about candidate transactions. Through the consensus process, validators agree on a specific subset of the candidate transactions to be considered for the next ledger. Consensus is an iterative process in which servers relay proposals, or sets of candidate transactions. Servers communicate and update proposals until a supermajority[4] of chosen validators agree on the same set of candidate transactions.

During consensus, each server evaluates proposals from a specific set of servers, known as that server's trusted validators, or *Unique Node List (UNL)*.[5] Trusted validators represent a subset of the network which, when taken collectively, is "trusted" not to collude in an attempt to defraud the server evaluating the proposals. This definition of "trust" does not require that each individual chosen validator is trusted. Rather, validators are chosen based on the expectation they will not collude in a coordinated effort to falsify data relayed to the network[6].


Validators each propose a set of transactions to be included in the next ledger version.
Round 1
Validators add transactions to their proposals if most other validators they trust proposed those transactions
Validators remove transactions if most other validators they trust didn't propose them.
(The removed transactions are usually proposed again for inclusion in the next ledger version.)
Round 2

In this document:

https://developers.ripple.com/consensus.html

What does subjective trust mean?

- Each node declares its own list of trusted nodes (UNL)
- The UNLs of two nodes must overlap
- But…
  - If the UNLs overlap, by how much?
  - Which nodes may fail?
  - If some nodes that I trust fail, what consequence does this have for me?

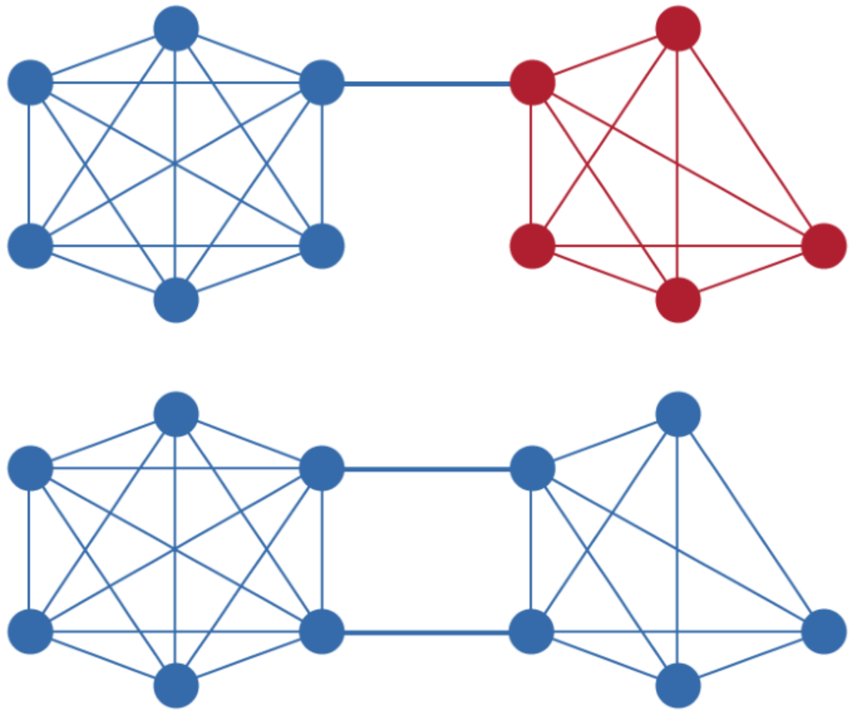


Figure 2. An example of the connectivity required to prevent a fork between two UNL cliques.

prove agreement is given by:

$$|UNL_i \cap UNL_j| \geq \frac{1}{5} \max(|UNL_i|, |UNL_j|) \forall i, j \quad (3)$$

Ripple white paper (2014)

Overlap of node lists?

- 20%
  - Ripple protocol consensus paper (2014)
- 40%
  - Armknecht et al. (TRUST 2015)
- "almost" 100% (!)
  - Chase & MacBrough (arxiv.org 2018)

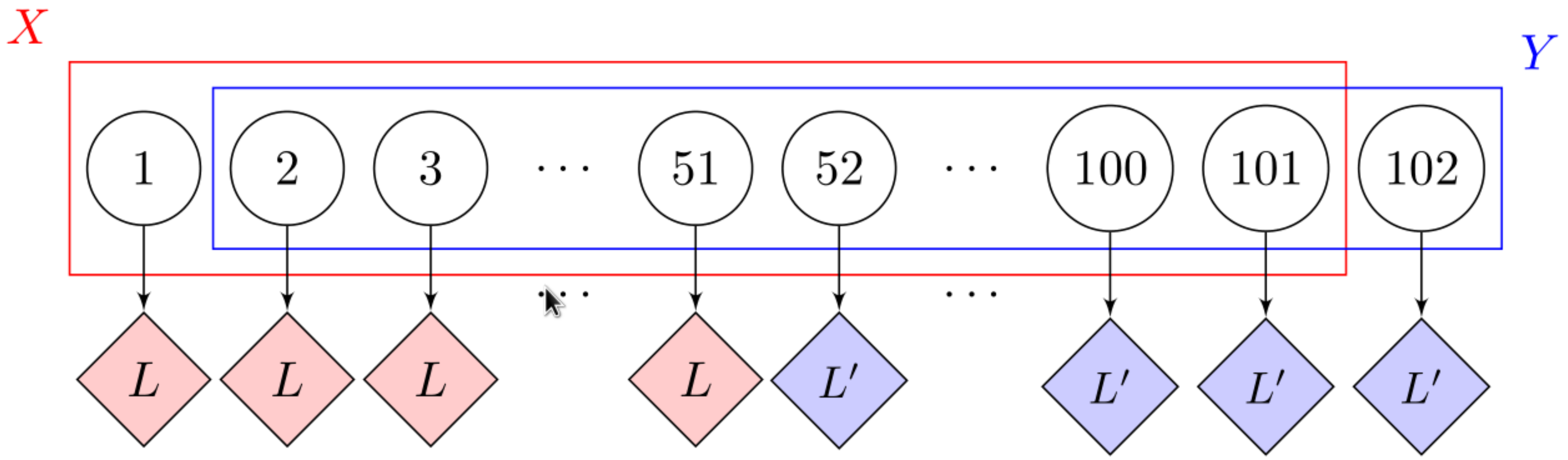


Figure 6: Example of stuck network with 99% UNL overlap and no Byzantine faults.

Chase & MacBrough, Analysis of the XRP Ledger Consensus Protocol (2018)

Ripple – A consensus protocol?

- No liveness if UNLs differ

- https://developers.ripple.com/consensus-protections.html
  - For all participants in the XRP Ledger to agree on what they consider validated, they must start by choosing a set of trusted validators that are fairly similar to the sets chosen by everyone else. In the worst case, less than about 90% overlap could cause some participants to diverge from each other. For that reason, Ripple publishes a signed list of recommended validators, including trustworthy and well-maintained servers run by the company, industry, and community.

  - In mid 2017 – 5 validators of Ripple that trust each other and no other node
  - In mid 2019 – 31 validators (7 Ripple; 24 non-Ripple)

b
UNIVERSITÄT
BERN

Stellar

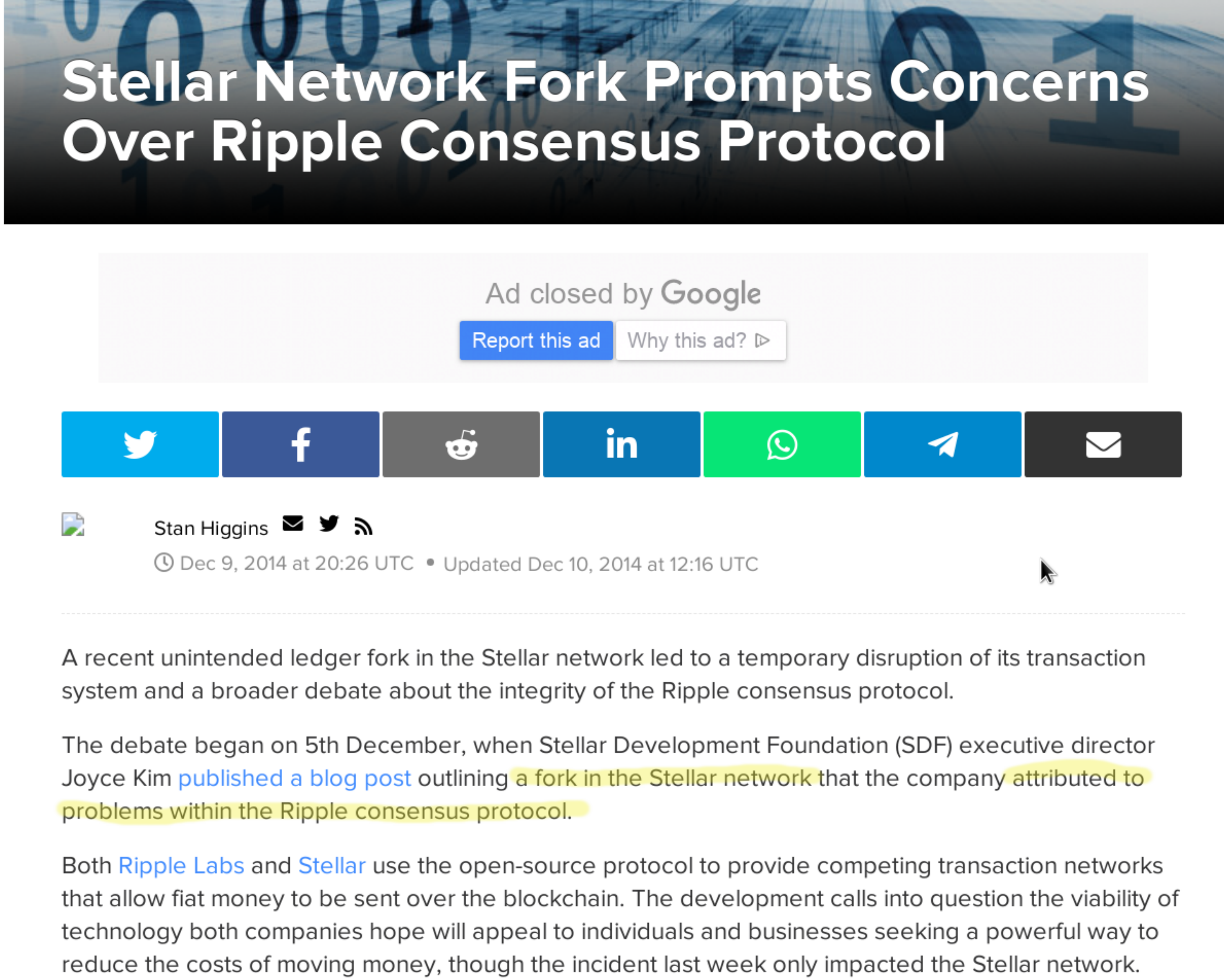
Consensus in Stellar

- Stellar forked from Ripple in 2013
  - Originally used Ripple's protocol and code
  - Today number 10 in market cap

- Stellar consensus failed and ledger forked in 2014

- Protocol was redesigned from scratch
  - Mazières, "The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus" (white paper, 2015)

- Aims to achieve federated Byzantine fault-tolerant consensus



Stellar Network Fork Prompts Concerns Over Ripple Consensus Protocol

Ad closed by Google

Report this ad | Why this ad?

Stan Higgins

Dec 9, 2014 at 20:26 UTC • Updated Dec 10, 2014 at 12:16 UTC

A recent unintended ledger fork in the Stellar network led to a temporary disruption of its transaction system and a broader debate about the integrity of the Ripple consensus protocol.

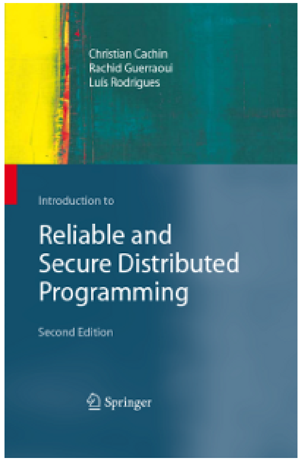
The debate began on 5th December, when Stellar Development Foundation (SDF) executive director Joyce Kim published a blog post outlining a fork in the Stellar network that the company attributed to problems within the Ripple consensus protocol.

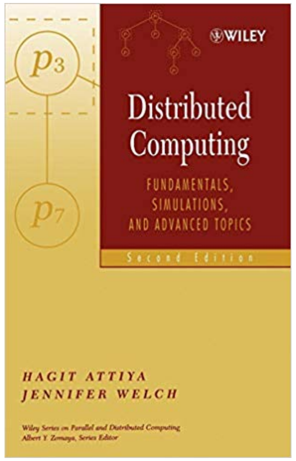
Both Ripple Labs and Stellar use the open-source protocol to provide competing transaction networks that allow fiat money to be sent over the blockchain. The development calls into question the viability of technology both companies hope will appeal to individuals and businesses seeking a powerful way to reduce the costs of moving money, though the incident last week only impacted the Stellar network.

https://coindesk.com

Quorum "slices" in Stellar consensus

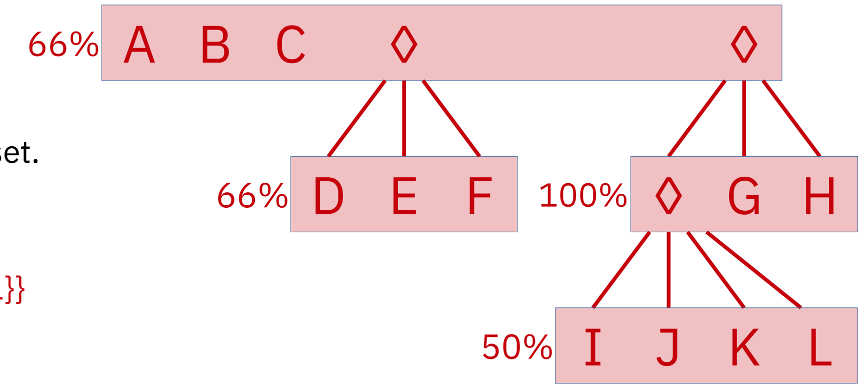
- When a node hears a "slice" assert a statement, the node adopts that
- Each node p_i declares its own set of slices S_i
- A set of nodes T such that $\forall\ p_i \in T : \exists\ S_i \subseteq T$ is called a "quorum"
- Unclear relation to consensus literature





Stellar's QUORUM_SET example

```
# QUORUM_SET is a required field
# This is how you specify this server's quorum set.
#
# It can be nested up to 2 levels: {A,B,C,{D,E,F},{G,H,{I,J,K,L}}}
# THRESHOLD_PERCENT is how many have to agree (1-100%) within a given set.
# Each set is treated as one vote.
# So for example in the above there are 5 things that can vote:
# individual validators: A,B,C, and the sets {D,E,F} and {G,H with subset {I,J,K,L}}
# the sets each have their own threshold.
# For example with {100% G,H with subset (50% I,J,K,L}}
#   means that quorum will be met with G, H and any 2 (50%) of {I, J, K, L}
#
# a [QUORUM_SET.path] section is constructed as
# THRESHOLD_PERCENT: how many have to agree, defaults to 67 (rounds up).
# VALIDATORS: array of node IDs
# additional subsets [QUORUM_SET.path.item_number]
# a QUORUM_SET  must not contain duplicate entries {{A,B},{A,C}} is invalid for example
#  The key for "self" is implicitely added at the top level, so the effective
#   quorum set is [t:2, self, QUORUM_SET].
```

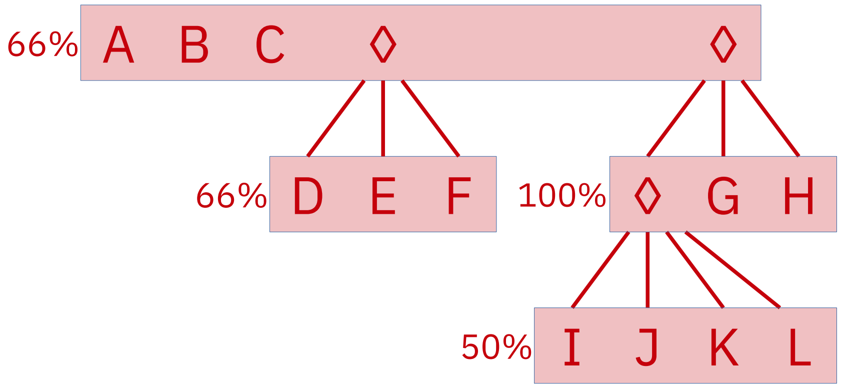


https://github.com/stellar/stellar-core/blob/master/docs/stellar-core_example.cfg

Stellar's QUORUM_SET example

UNIVERSITÄT BERN

```
[QUORUM_SET]
THRESHOLD_PERCENT=66
VALIDATORS=[
  "GDQWITFJLZ5HT6JCOXYEVV5VFD6FTLAKJAUDKHAV3HKYGVJWA2DPYSQV  A_from_above",
  "GANLKVE4WOTE75MJS6FQ73CL65TSPYYMFZKC4VDEZ45LGQRCATGAIGIA  B_from_above",
  "GDV46EIEF57TDL4W27UFDAUVPDDCKJNVBYB3WIV2WYUYUG753FCFU6EJ  C_from_above"
]

[QUORUM_SET.1]
THRESHOLD_PERCENT=67
VALIDATORS=[
  "$self", # 'D' from above is this node
  "GDXJAZZJ3H5MJGR6PDQX3JHRREAVYNCVM7FJYGLZJKEHQV2ZXEUO5SX2 E_from_above",
  "GB6GK3WWTZYY2JXWM6C5LRKLQ2X7INQ7IYTSECCG3SMZFYOZNEZR4SO5 F_from_above"
]

[QUORUM_SET.2]
THRESHOLD_PERCENT=100
VALIDATORS=[
  "GCTAIXWDDBM3HBDHGSAOLY223QZHPS2EDROF7YUBB3GNYXLOCPV5PXUK G_from_above",
  "GCJ6UBAOXNQFN3HGLCVQBWGEZO6IABSMNE2OCQC4FJAZXJA5AIE7WSPW H_from_above"
]

[QUORUM_SET.2.1]
THRESHOLD_PERCENT=50
VALIDATORS=[
  "GC4X65TQJVI3OWAS4DTA2EN2VNZ5ZRJD646H5WKEJHO5ZHURDRAX2OTH I_from_above",
  "GAXSWUO4RBELRQT5WMDLIKTRIKC722GGXX2GIGEYQZDQDLOTINQ4DX6F J_from_above",
  "GAWOEMG7DQDWHCFDTPJEBYWRKUUZTX2M2HLMNABM42G7C7IAPU54GL6X K_from_above",
  "GDZAJNUUDJFKTZX3YWZSOAS4S4NGCJ5RQAY7JPYBG5CUFL3JZ5C3ECOH L_from_above"
]
```



https://github.com/stellar/stellar-core/blob/master/docs/stellar-core_example.cfg

Recent developments ...

- "Is Stellar As Secure As You Think?" (Kim et al., 2019)
  - Exploration of the Stellar trust graph shows high centralization

- In Apr./May 2019, Stellar started to make its trust graph less centralized ...

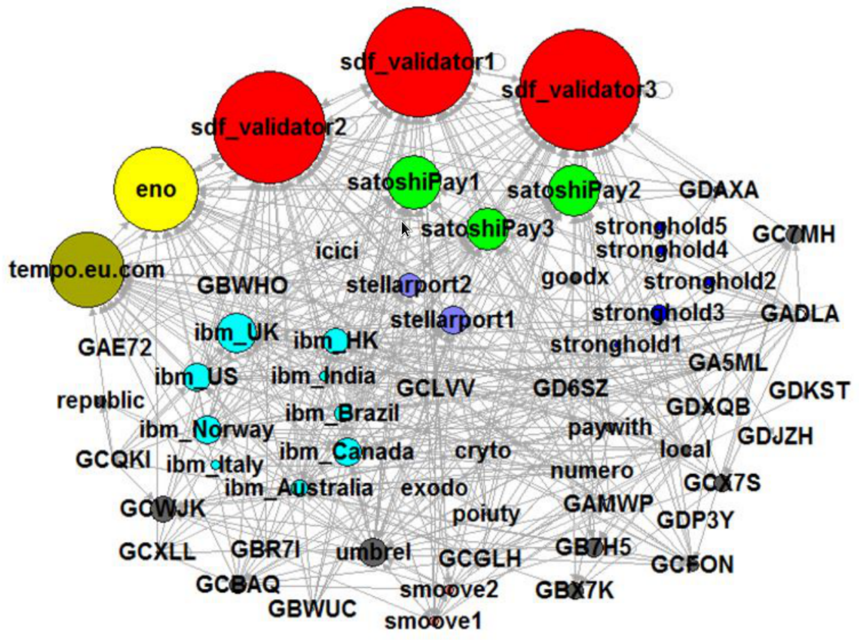
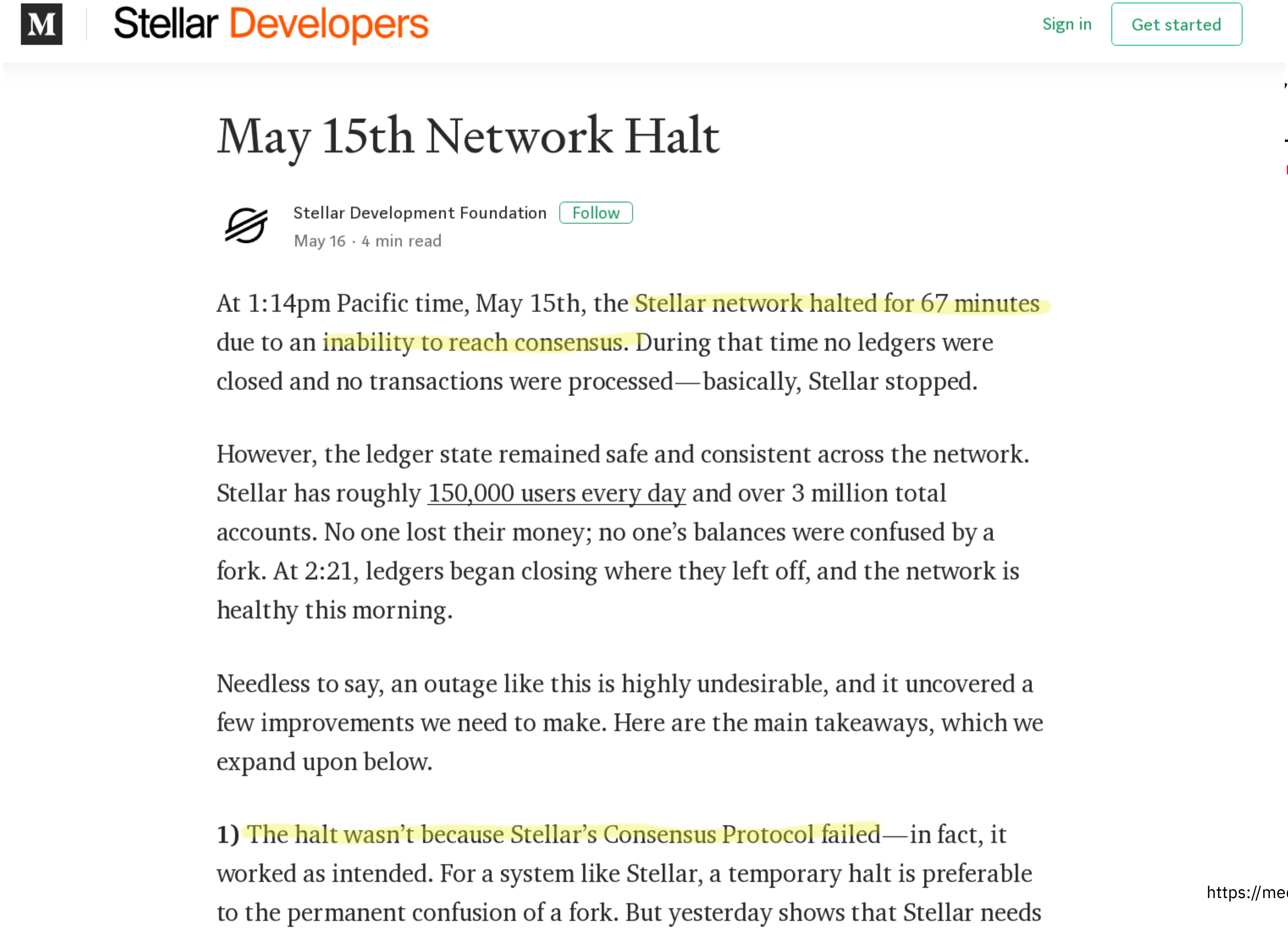


Figure 2. Directed graph of quorum slices in Stellar

b

RSITÄT



M | Stellar Developers Sign in Get started

May 15th Network Halt

Stellar Development Foundation Follow
May 16 · 4 min read

At 1:14pm Pacific time, May 15th, the Stellar network halted for 67 minutes due to an inability to reach consensus. During that time no ledgers were closed and no transactions were processed—basically, Stellar stopped.

However, the ledger state remained safe and consistent across the network. Stellar has roughly 150,000 users every day and over 3 million total accounts. No one lost their money; no one's balances were confused by a fork. At 2:21, ledgers began closing where they left off, and the network is healthy this morning.

Needless to say, an outage like this is highly undesirable, and it uncovered a few improvements we need to make. Here are the main takeaways, which we expand upon below.

1) The halt wasn't because Stellar's Consensus Protocol failed—in fact, it worked as intended. For a system like Stellar, a temporary halt is preferable to the permanent confusion of a fork. But yesterday shows that Stellar needs

https://medium.com

Stellar – A consensus protocol?

- No clear liveness guarantees
- Does not generalize existing consensus protocols
- No simple condition to evaluate if the chosen quorum slices ensure consensus
- Relation of Stellar consensus to existing BFT consensus remains open

UNIVERSITÄT BERN

Asymmetric quorum systems

Quorum systems

UNIVERSITÄT BERN

- Set of nodes $\mathbf{P} = \{p_1, \ldots, p_n\}$
- Fail-prone system $\mathbf{F} \subseteq 2^{\mathbf{P}}$: All $F \in \mathbf{F}$ may fail together
- Quorum system $\mathbf{Q} \subseteq 2^{\mathbf{P}}$, where any $Q \in \mathbf{Q}$ is a "quorum"
  - Consistency:

$$\forall\ Q_1, Q_2 \in \mathbf{Q}, \forall F \in \mathbf{F} : Q_1 \cap Q_2 \not\subseteq F.$$

  - Availability:

$$\forall F \in \mathbf{F} : \exists Q \in \mathbf{Q} : F \cap Q = \emptyset .$$

(Malkhi & Reiter, 1998)

Asymmetric quorum systems

- Asymmetric fail-prone system $\mathcal{F} = [\mathbf{F_1}, \ldots, \mathbf{F_n}]$,
  where $\mathbf{F_i} \subseteq 2^{\mathbf{P}}$ is fail-prone set for p_i; all $F \in \mathbf{F_i}$ may fail together (... acc. to p_i)

- Asymmetric quorum system $\mathcal{Q} = [\mathbf{Q_1}, \ldots, \mathbf{Q_n}]$,
  where $\mathbf{Q_i} \subseteq 2^{\mathbf{P}}$ is a quorum system for p_i and any $Q_i \in \mathbf{Q_i}$ is a "quorum for p_i"

  – Consistency:

$$\forall\, p_i, p_j, \forall\, Q_i \in \mathbf{Q_i}, \forall\, Q_j \in \mathbf{Q_j}, \forall\, F \in \mathbf{F_i}^* \cap \mathbf{F_j}^* : Q_i \cap Q_j \not\subseteq F.$$

  – Availability:

$$\forall\, p_i, \forall\, F \in \mathbf{F_i} : \exists\, Q \in \mathbf{Q_i} : F \cap Q = \emptyset.$$

(Based on Damgård, Desmedt, Fitzi, Nielsen, Asiacrypt 2007)

When do quorum systems exist?

UNIVERSITÄT BERN

- Q3 property for fail-prone system **F** (MR98, HM00)
  - No three elements of **F** cover **P** (e.g., for threshold quorums: $3*f < n$)

- B3 property for asymmetric $\mathcal{F}$ (and asymmetric quorum system $\mathcal{Q}$)
  - For all $F_i \in \mathbf{F_i}$, $F_j \in \mathbf{F_j}$, $F_{ij} \in \mathbf{F_i}^* \cap \mathbf{F_j}^*$: $\mathbf{P} \not\subseteq F_i \cup F_j \cup F_{ij}$

- Thm. (MR98): A quorum system for **F** exists ⇔ Q3(**F**)

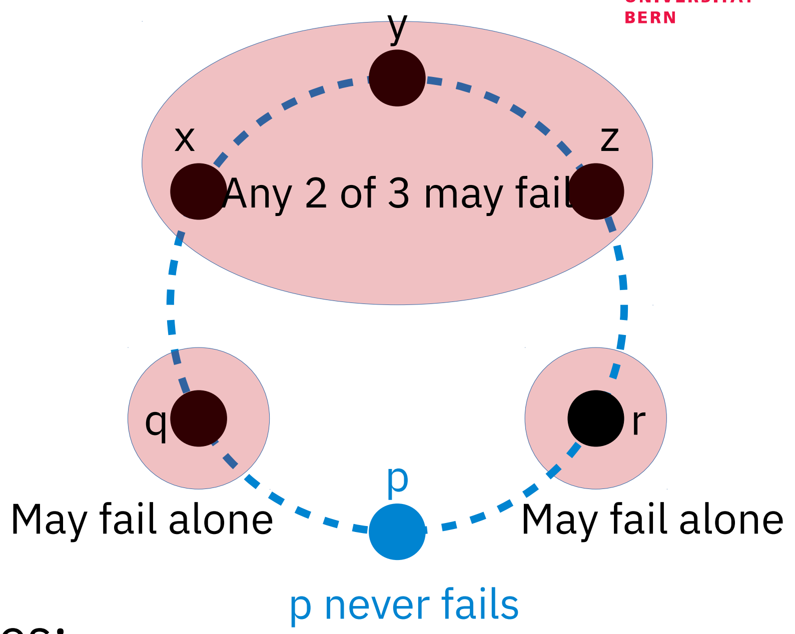
- Asymmetric Thm: An asymmetric quorum system for $\mathcal{F}$ exists ⇔ B3($\mathcal{F}$)

Example asymmetric quorum system

- Six nodes, arranged in a ring
- Failure assumptions of node p as shown
- All others (rotation) symmetric to p

- Satisfies B3 property
  ↔
  Asymmetric quorum system

- Each node mistrusts some 2-set of other nodes: impossible with threshold Byzantine quorums!



Execution model

- An execution defines the actually faulty nodes F

- Any node p_i is one of
  - Faulty – $p_i \in F$
  - Naive p_i – $F \notin \mathbf{F}_i^*$
  - Wise p_i – $F \in \mathbf{F}_i^*$

- Guarantees hold only for wise nodes
  - Naive nodes may be cheated
    (cf. ordinary, symmetric BFT system with $f \geq n/3$: all nodes are naive!)

Protocols with asymmetric trust

- Standard tasks in distributed computing
  - **Emulation of a shared (SWMR) register**
  - **Byzantine consistent broadcast**
  - **Byzantine reliable broadcast ("Bracha" broadcast)**
  - **Randomized Byzantine consensus (Cachin & Zanolini*)**
- Strict generalizations of standard protocols with symmetric trust

Asymmetric SWMR regular register protocol (process p_i).

```
State
    $wts$: sequence number of write operations, stored only by writer $p_w$
    $rid$: identifier of read operations, used only by reader
    $ts, v, \sigma$: current state stored by $p_i$: timestamp, value, signature

upon invocation $write(v)$ do                                        // only if $p_i$ is writer $p_w$
    $wts \leftarrow wts + 1$
    $\sigma \leftarrow_{\$} sign_w(\text{WRITE}\|w\|wts\|v)$
    send message $[\text{WRITE}, wts, v, \sigma]$ to all $p_j \in \mathcal{P}$
    wait for receiving a message $[\text{ACK}]$ from all processes in some quorum $Q_w \in \mathcal{Q}_w$

upon invocation $read$ do                                            // only if $p_i$ is reader $p_r$
    $rid \leftarrow rid + 1$
    send message $[\text{READ}, rid]$ to all $p_j \in \mathcal{P}$
    wait for receiving messages $[\text{VALUE}, r_j, ts_j, v_j, \sigma_j]$ from all processes in some $Q_r \in \mathcal{Q}_r$ such that
        $r_j = rid$ and $verify_w(\sigma_j, \text{WRITE}\|w\|ts\|v_j)$
    return $highestval(\{(ts_j, v_j) | j \in Q_r\})$

upon receiving a message $[\text{WRITE}, ts', v', \sigma']$ from $p_w$ do            // every process
    if $ts' > ts$ then
        $(ts, v, \sigma) \leftarrow (ts', v', \sigma')$
    send message $[\text{ACK}]$ to $p_w$

upon receiving a message $[\text{READ}, r]$ from $p_r$ do                           // every process
    send message $[\text{VALUE}, r, ts, v, \sigma]$ to $p_r$
```

Conclusion

- Quorum systems at the core of distributed BFT protocols
- Ripple and Stellar aim at asymmetric trust, but fail to achieve it
- Asymmetric quorums are a sound model for subjective distributed trust

- See blog and paper with Björn Tackmann
  - cryptobern.github.io/asymmetric
  - arxiv.org/abs/1906.09314

De gustibus non est disputandum