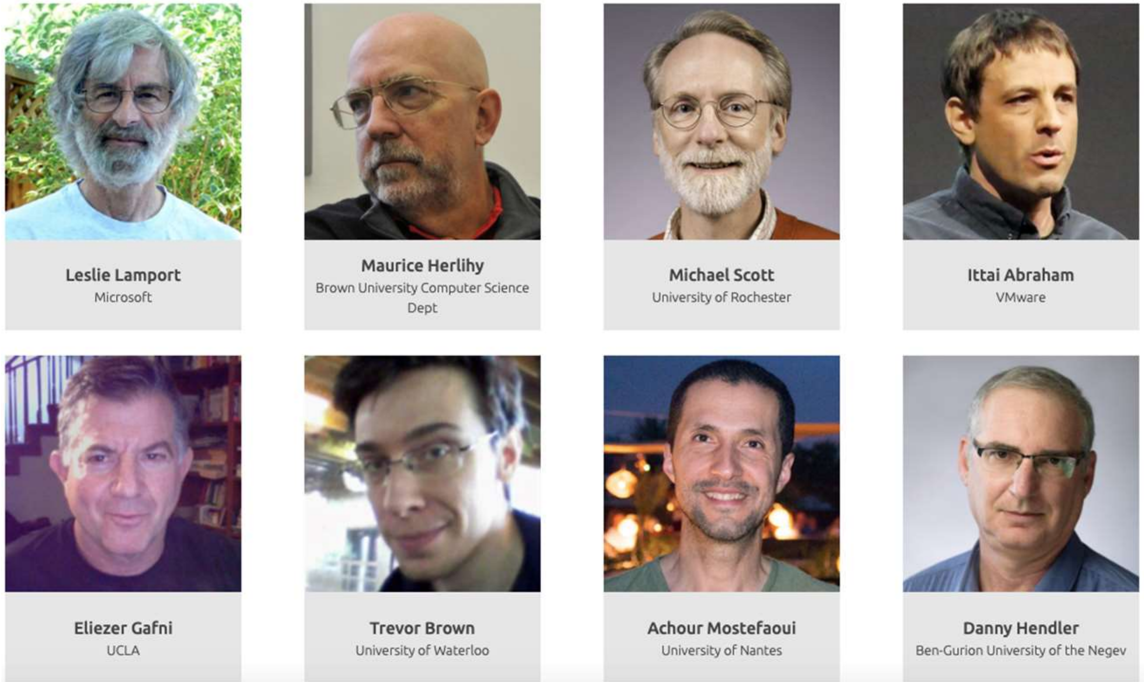
THE SECOND SUMMER SCHOOL ON PRACTICE AND THEORY OF DISTRIBUTED COMPUTING

July 8-12, 2019, St Petersburg, Russia



Algorithmic basics of blockchains

Concurrent data structures

Distributed computability

State-machine replication and Paxos

Byzantine fault-tolerance

https://sptdc.ru/en/

The Consensus Number of a Cryptocurrency

To appear at PODC 2019

Joint work with Rachid Guerraoui, Matteo Monti, Matej Pavlovic, and Adi Seredinschi

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network.

...

The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received.

Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing

Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford
EPFL

Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

Elli Androulaki
Artem Barger
Vita Bortnikov
IBM

Christian Cachin
Konstantinos Christidis
Angelo De Caro
David Enyeart
IBM

Christopher Ferris
Gennady Laventman
Yacov Manevich
IBM

Srinivasan Muralidharan*
State Street Corp.

Chet Murthy*

Binh Nguyen*
State Street Corp.

Manish Sethi
Gari Singh
Keith Smith
Alessandro Sorniotti
IBM

Chrysoula Stathakopoulou
Marko Vukolić
Sharon Weed Cocco
Jason Yellick
IBM

Thunderella: Blockchains with Optimistic Instant Confirmation

SBFT: a Scalable Decentralized Trust Infrastructure for Blockchains

Guy Golan Gueta (VMware Research)
Shelly Grossman (TAU)
Michael K. Reiter (UNC-Chapel Hill)
Dahlia Malkhi (VMware Research)
Orr Tamir (TAU)
Ittai Abraham (VMware Research)
Benny Pinkas (BIU)
Dragos-Adrian Seredinschi (EPFL)
Alin Tomescu (MIT)

Algorand: Scaling Byzantine Agreements for Cryptocurrencies

Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, Nickolai Zeldovich
MIT CSAIL

Hot-Stuff the Linear, Optimal-Resilience, One-Message BFT Devil

Ittai Abraham, Guy Gueta, Dahlia Malkhi
VMware Research

March 15, 2018

FruitChains: A Fair Blockchain

Rafael Pass
Cornell Tech
rafael@cs.cornell.edu

Elaine Shi
Cornell University
elaine@cs.cornell.edu

May 5, 2017

Hybrid Consensus: Efficient Consensus in the Permissionless Model

Rafael Pass and Elaine Shi

CornellTech, Cornell, Initiative for CryptoCurrency and Contracts (IC3)*


coindesk
Blockchain 101
Technology
Markets
Business
Data & Research
Consensus
Search
Announcing the CoinDesk Career Center. Find Your Next Job in Blockchain
consensus
2018
May 14 - 16
New York Hilton Midtown
ABOUT
AGENDA
SPEAKERS
REGISTER
SPONSORS
SCHOLARS
PoW PITCH
HACKATHON
COMMUNITY
HOTEL
FAQ
@COINDESK
BLOCKCHAIN WEEK
LIVE

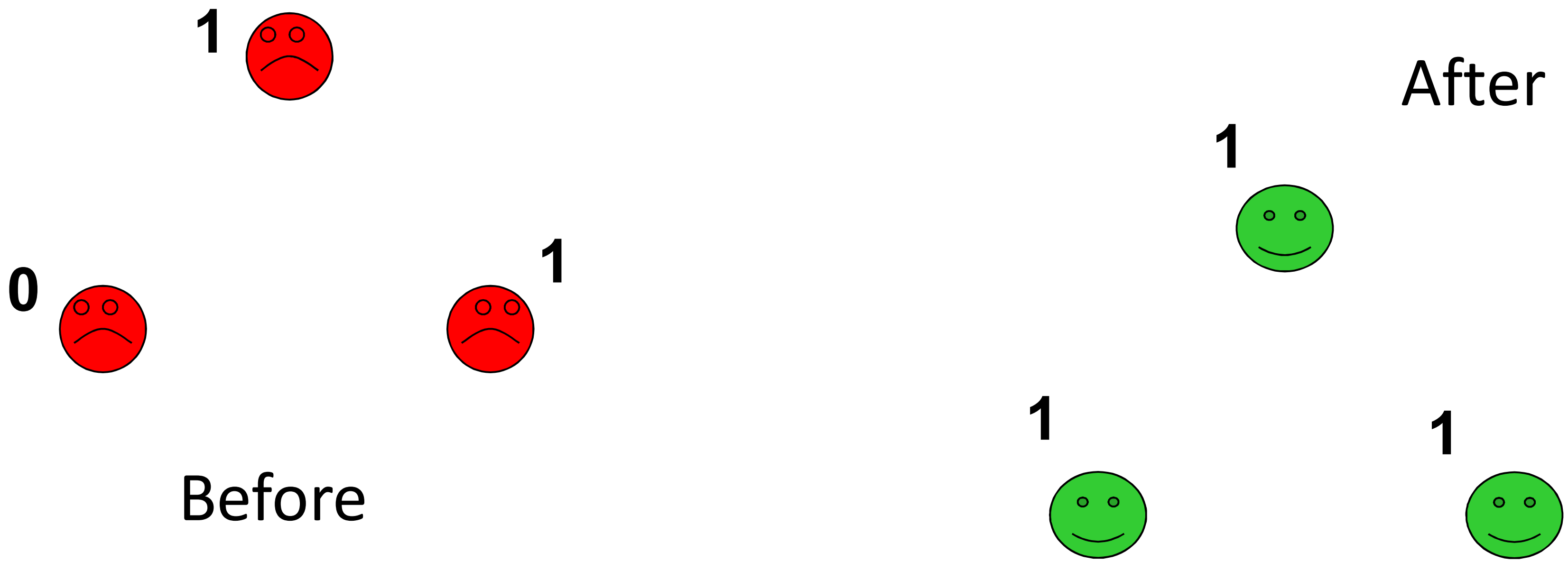
This talk

Cryptocurrency does not require consensus

- Consensus number of the asset transfer data type:
  - ✓k-owned (smart contracts with k parties) – k
- Asynchronous asset transfer algorithm
  - ✓1-owned: secure broadcast
  - ✓k-owned: k-consensus + secure broadcast

Consensus

Processes *propose* values and must *agree* on a common decision value so that the decided value is a proposed value of some process



But why consensus is interesting?

Because it is universal!

- If we can solve consensus among N processes, then we can *implement* any object shared by N processes
- A key to implement a generic fault-tolerant service (replicated state machine or blockchain)

Is consensus necessary for a cryptocurrency?

What is a “cryptocurrency”?

State:

- *P* - set of processes
- *A* - set of *accounts*
- μ: $A \rightarrow 2^P$ - *ownership map* (single owner if $A \rightarrow P$)
- β: $A \rightarrow \mathbf{N}$ – balance map (β_0 – initial balances)

Interface:

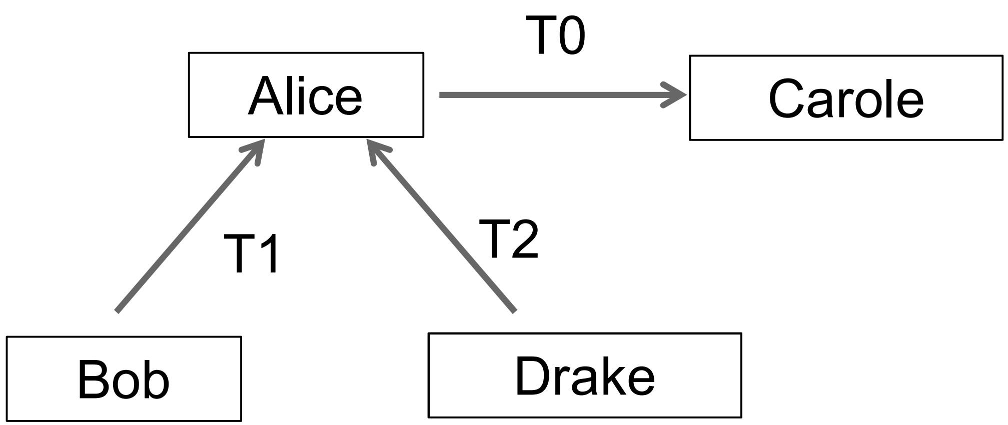
- *transfer*(a,b,x) – called by an owner of a, returns a boolean (success or failure)
- *read*(a) – returns the balance

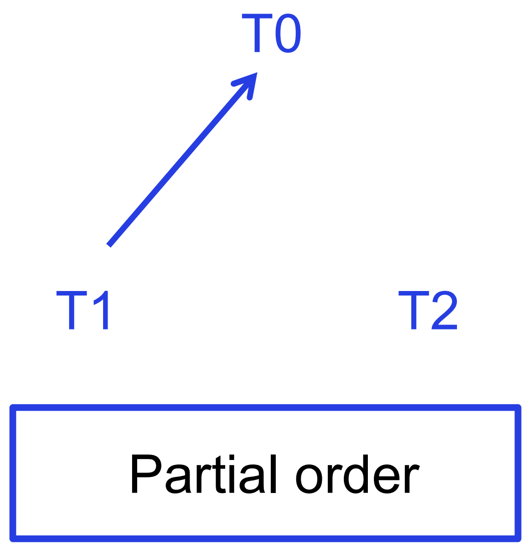
We call it asset transfer data type

Commutativity and causality

- T0: $100 from Alice to Carole
- T1: $100 from Bob to Alice
- T2: $100 from Drake to Alice

T0 causally depends on T1 (not enough funds otherwise)
T1 and T2 commute (T0 succeeds regardless of the order)





Consensus number

An object O has consensus number k if

- k-process consensus can be solved using registers and any number of copies of O but (k+1)-consensus cannot

(k is the maximal number of processes that can be synchronized using copies of O and registers)

Consensus hierarchy:

- cons(read-write register)=1
- cons(T&S)=cons(queue)=2
- …
- cons(CAS)=∞

Consensus number of asset transfer

Upon $transfer(a, b, x)$

$S := AS.snapshot()$

if $p \notin \mu(a) \vee balance(a, S) < x$ **then**

  return $false$

$ops_p := ops_p \cup \{(a, b, x)\}$

$AS.update(ops_p)$

return $true$

Upon $read(a)$

$S := AS.snapshot()$

return $balance(a, S)$

Single-owner: 1

- use atomic-snapshot memory to exchange the account balance

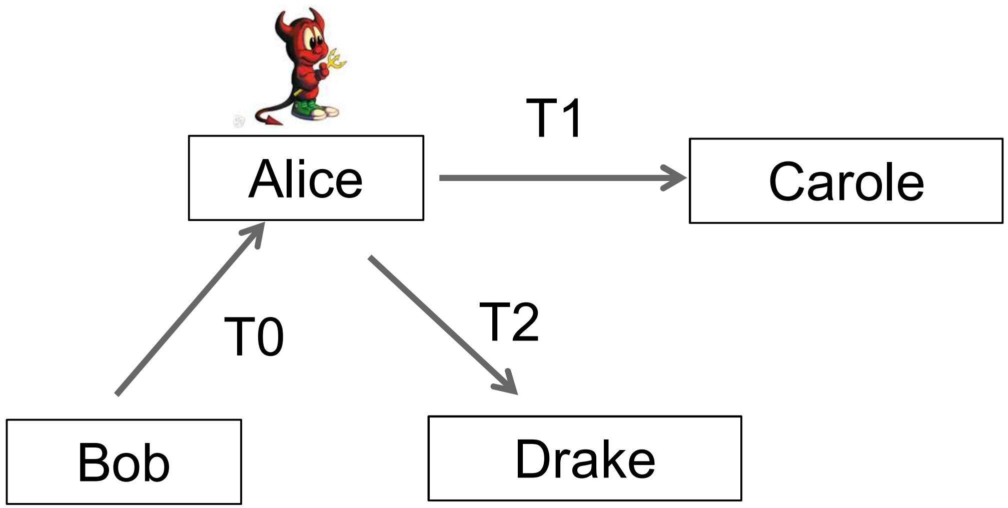
k-owner (up to k can debit an account): k

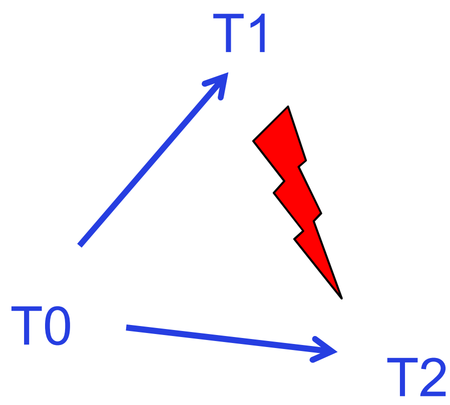
- account owners use k-consensus to agree on the per-account order of debit operations
- A single k-owned account solves k-consensus

What about double-pending?

- T0: $100 from Bob to Alice
- T1: $100 from Alice to Carole
- T2: $100 from Alice to Drake

Alice’s initial balance is 0, but it claims to both beneficiaries to have received money from Bob



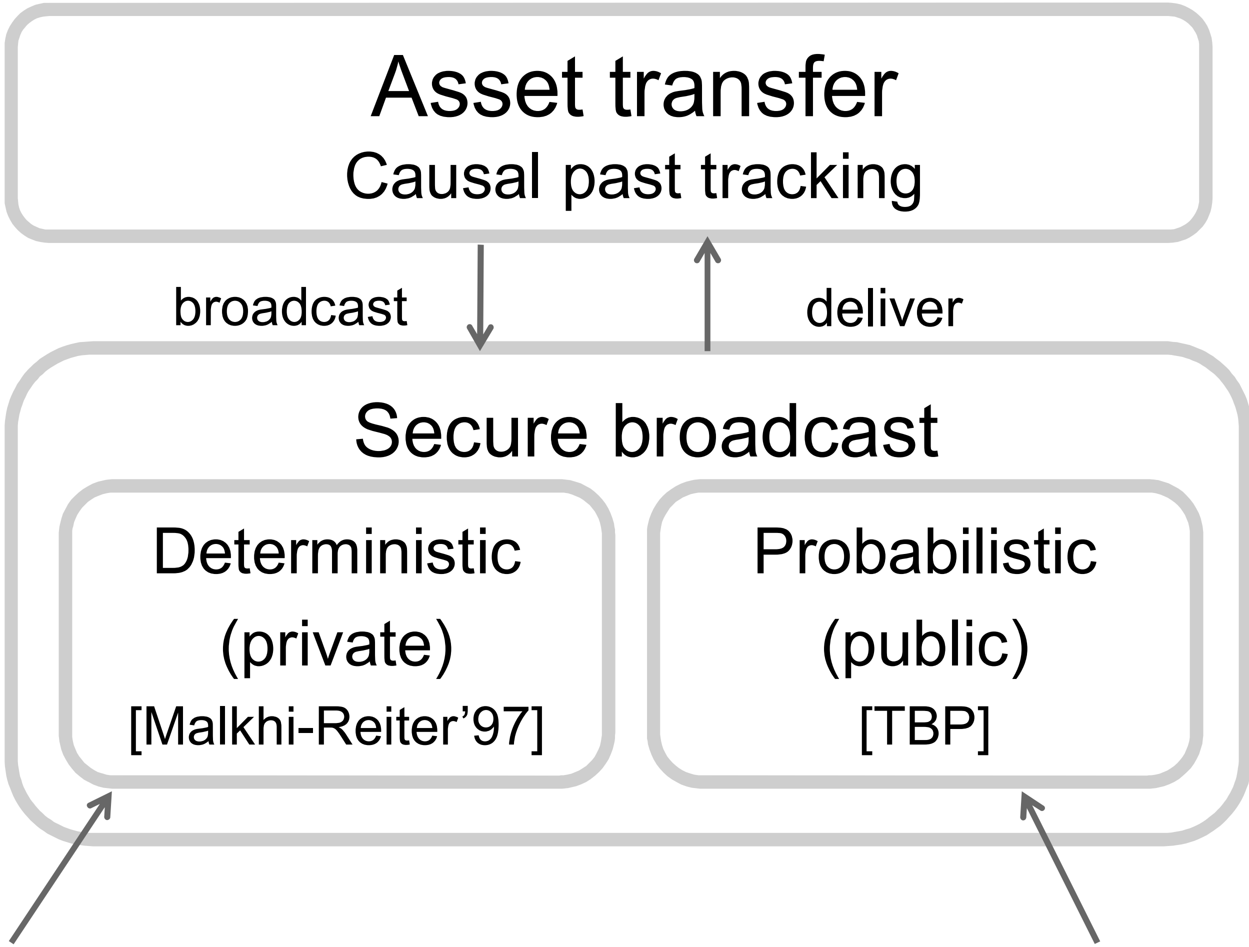


Asset transfer implementation

Message-passing, Byzantine failures

- Each transfer is equipped with it causal past (a set of incoming transactions)
- Make sure that a faulty account holder cannot lie about its causal past
- Secure broadcast [Bracha, 1987, Malkhi-Reiter, 1997]
  - ✓Source-order: messages by the same source are delivered in the same order

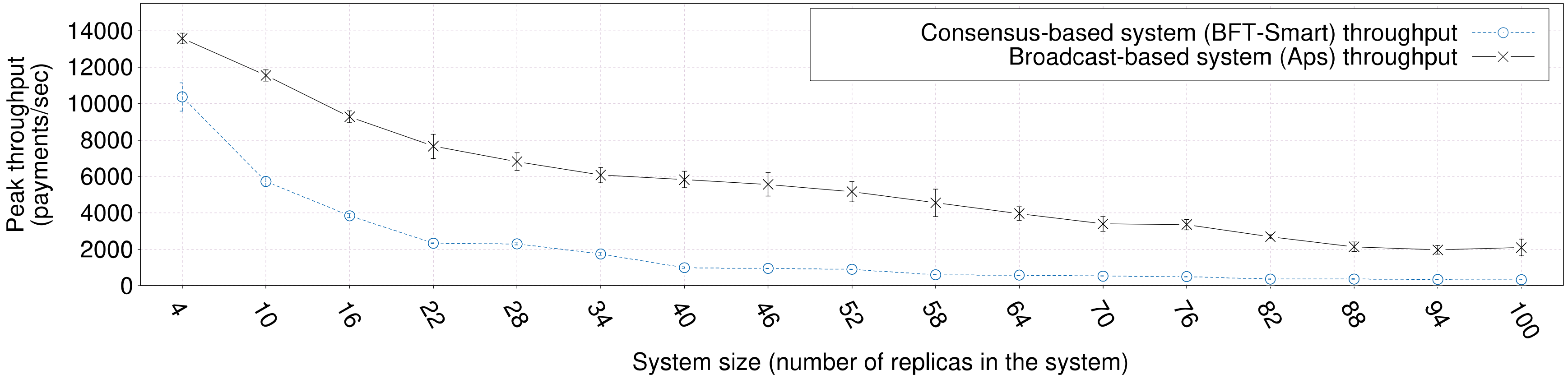
Modular approach: private and public

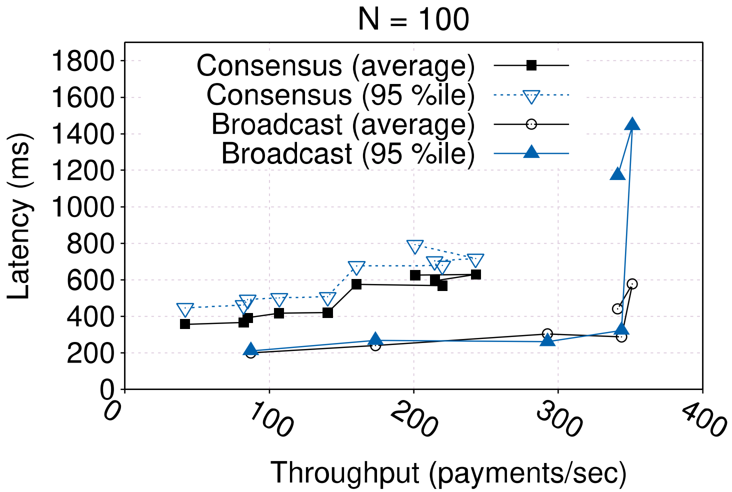


Intuition: deliver only if accepted by a Byzantine quorum (of 2f+1)

Intuition: deliver only if enough sample members are "ready"

Performance





Performance wrt BFT-Smart:

- Throughput 1,5-6x higher
- Latency 2x lower

Take-aways

- Asset transfers do not always require total order
  - ✓Source order is sufficient for consistency
  - ✓(Asynchronous) secure broadcast
- Can be generalized to (limited-scope) ”smart contracts”
  - ✓only account owners need consensus, but still no global total order
- Coming: probabilistic and Sybill-tolerant secure broadcast can be implemented (coming)
  - ✓Permissionless asset transfer

Thank you!

Implementing asset transfer

To perform a (successful) transfer:

- (securely) broadcast together with dependencies (the set of previously received and issued transfers) that justify it
- A delivered transfer is validated and accepted if:
  - ✓the transfer is issued by an owner
  - ✓the balance (based on declared dependencies) is sufficient
  - ✓ every its dependency transfer is validated (recursively)

Broadcasting and delivering

```
operation transfer(a, b, x) where a = p
  if balance(a, hist[p] ∪ deps) < x then
    return false
  broadcast([(a, b, x, seq[p] + 1), deps])
  deps := ∅

{ Secure broadcast callback }
upon deliver(q, m)          { Executed when p delivers message m from process q }
  let m be [(q, d, y, s), h]
  if s = rec[q] + 1 then
    rec[q] := rec[q] + 1
    toValidate := toValidate ∪ {(q, m)}
```

Validating transfers

upon $(q, \lfloor t, h \rfloor) \in \mathit{toValidate} \;\wedge\; \mathit{Valid}(q, t, h)$ $\quad$ { *Executed when a transfer delivered from q becomes v*
$\quad$ $\mathit{hist}[q] := \mathit{hist}[q] \cup h \cup \{t\}$ $\quad$ { *Update the history for the outgoing account* }
$\quad$ let t be (c, d, y, s)
$\quad\quad$ $\mathit{seq}[q] := s$
$\quad\quad$ **if** $d = p$ **then**
$\quad\quad\quad$ $\mathit{deps} := \mathit{deps} \cup \{(c, d, y, s)\}$ $\quad$ { *This transfer is incoming to account of local process p* }
$\quad\quad$ **if** $c = p$ **then**
$\quad\quad\quad$ **return** *true* $\quad$ { *This transfer is outgoing from account of local process p* }

function $\mathit{Valid}(q, t, h)$
$\quad$ let t be (c, d, y, s)
$\quad$ **return** $(q = c)$
$\quad\quad\quad$ **and** $(s = \mathit{seq}[q] + 1)$
$\quad\quad\quad$ **and** $(\forall (a, b, x, r) \in h : (a, b, x, r) \in \mathit{hist}[a])$
$\quad\quad\quad$ **and** $(\mathit{balance}(c, \mathit{hist}[q] \cup h) \geq y)$

Also, extended to the “k-owner” case

Quorums for samples

Needed: a Syblil-resistant sampling mechanism

- Every correct node maintains a sample of the system with a constant expected fraction of faulty nodes

"Proof-of-bandwidth": trust more to those who talk to you more

Brahms: Byzantine Resilient Random Membership Sampling [Bortnikov et al., 2010]

Gossip-based broadcast

- The idea: deliver once heard « enough » acks (e.g., a sample has confirmed)
- Analyze the probability
- Iterative construction:
  - ✓Probabilistic broadcast (validity+totality)=>
  - ✓Probabilistic consistent broadcast (consistency)=>
  - ✓Probabilistic secure broadcast (validity, totality, consistency)

ε-secure probabilistic broadcast

For any $\epsilon \in [0,1]$, we say that probabilistic broadcast is ϵ-secure if:

1. **No duplication**: No correct process delivers more than one message.
2. **Integrity**: If a correct process delivers a message m, and σ is correct, then m was previously broadcast by σ.
3. ϵ-**Validity**: If σ is correct, and σ broadcasts a message m, then σ eventually delivers m with probability at least $(1-\epsilon)$.
4. ϵ-**Totality**: If a correct process delivers a message, then every correct process eventually delivers a message with probability at least $(1-\epsilon)$.

- « Erdös-Rényi » gossip: to broadcast - send m to every member of the sample, once received m – deliver and send to the sample

ε-secure consistent broadcast

For any $\epsilon \in [0,1]$, we say that probabilistic consistent broadcast is ϵ-secure if:

1. **No duplication**: No correct process delivers more than one message.
2. **Integrity**: If a correct process delivers a message m, and σ is correct, then m was previously broadcast by σ.
3. ϵ-**Total validity**: If σ is correct, and σ broadcasts a message m, every correct process eventually delivers m with probability at least $(1-\epsilon)$.
4. ϵ-**Consistency**: Every correct process that delivers a message delivers the same message with probability at least $(1-\epsilon)$.

- Run ρ-secure probabilistic broadcast of m and wait until enough processes in « echo sample » deliver m