Distributed Resilient Consensus

Nitin Vaidya
University of Illinois at Urbana-Champaign

Some research reported here is supported in part by
This is a draft version of the slides

Final version will be made available on the webpage below:

http://www.crhc.illinois.edu/wireless/tutorials.html
Distributed Resilient Consensus

Scope:

- Distribute consensus
- Resilient to
  - Node failures … with emphasis on Byzantine faults
  - Lossy communication links
- Coverage biased by my own research
  - There exists a much larger body of work by the community
Consensus ... Dictionary Definition

- General agreement
- Majority opinion
Many Faces of Consensus

- What time is it?

- Network of clocks …

agree on a common notion of time
Many Faces of Consensus

- Commit or abort?

- Network of databases …

agree on a common action
Many Faces of Consensus

- What is the temperature?

- Network of sensors …

agree on current temperature
Many Faces of Consensus

- Where to go for dinner?
- Network of friends…

agree on a location
Many Faces of Consensus

- Should we trust 😊?

- Web of trust …

agree whether 😊 is good or evil
Consensus

Correctness requirements:

- **Agreement** … participating nodes agree

- **Validity** … several variations
  - Output = input of a designated node
  - Output = in range of all (good) nodes
  - Output = average of all inputs
  - ...

- **Termination** … several variations
Many Faces of Consensus

- No failures / failures allowed (node/link)
- Synchronous/asynchronous
- Deterministically correct / probabilistically correct
- Exact agreement / approximate agreement
- Amount of state maintained
Failure Models

- Crash failure
  - Faulty node stops taking any steps at some point

- Byzantine failure
  - Faulty node may behave arbitrarily
  - Drop, tamper messages
  - Send inconsistent messages
Timing Models

- Synchronous
  - Bound on time required for all operations

- Asynchronous
  - No bound
Correctness

- Deterministic
  - Always results in correct outcome (agreement, validity)

- Probabilistic
  - Correctness only with high probability
  - Non-zero probability of incorrect outcome
Agreement

- **Exact**
  - All nodes agree on exactly identical value (output)

- **Approximate**
  - Nodes agree on approximately equal values
  - Identical in the limit as time $\to \infty$
State

How much state may a node maintain?

- No constraint
- “Minimal” state
Many Faces of Consensus

- No failures / failures allowed (node/link)
- Synchronous/asynchronous
- Deterministically correct / probabilistically correct
- Exact agreement / approximate agreement
- Amount of state maintained
Consensus in Practice

- Fault-tolerant servers
- Distributed control
Client-Server Model

Client

command A

state 0

Server
Client-Server Model

Client

command A

state A

Server

response (A)
Client-Server Model

Client

command A

Server

Crash failure
make service unavailable
Replicated Servers

- All (fault-free) servers must execute identical commands in an identical order

- Despite crash failures
Client-Server Model

Client

command A

%*$&#

Server

Byzantine failure can result in incorrect response
Replicated Servers

Client

command A

Servers

state 0

state 0
Replicated Servers

Client

Servers

state A

response (A)

%*$&#

response (A)

state A
Replicated Servers

- All (fault-free) servers must execute identical commands in an identical order
- Despite crash failures
Coordination

- Different nodes must take consistent action
- As a function of input from the various nodes
Binary Consensus
Consensus

Each node has a binary input

- Agreement: All outputs identical

- Validity: If all inputs 0 $\Rightarrow$ Output 0
  If all inputs 0 $\Rightarrow$ Output 1
  Else output may be 0 or 1

- Termination after bounded interval
Synchronous Fault-Free Systems

Trivial Algorithm

- Each node sends its input to all other nodes
- Choose majority of received values as output
Synchronous + Crash Failures

Previous algorithm does not work with a crash failure

- Suppose four nodes:
  \[ A : 0 , \quad B : 0 , \quad C : 1 , \quad D : 1 \]

- Node A fails after sending value to B, C without sending to D

- B, C : 0,0,1,1
  \[ D : 0,1,1 \]
Synchronous + up to f Crash Failures

Initialization: \( S = \text{input} \)

For \( f+1 \) rounds:

- Send any new value in \( S \) to all other nodes
- Receive values from other nodes

After \( f+1 \) rounds

Output = \( v \) if all values in \( S \) are \( v \)
else 0
Synchronous + Crash Failures

Lower bound:

at least $f+1$ rounds necessary to tolerate $f$ crash failures
Asynchronous + Crash Failure

- Consensus impossible in asynchronous systems in presence of even a single (crash) failure

**FLP** [Fischer, Lynch, Paterson]

- **Intuition** … asynchrony makes it difficult to distinguish between a faulty (crashed) node, and a slow node
How to achieve consensus with asynchrony?

- Relax agreement or termination requirements
- We will discuss “approximate” agreement
Synchronous + Byzantine Faults
Synchronous + Byzantine Consensus

Each node has a binary input

- Agreement: All outputs identical

- Validity: If all fault-free inputs 0 ➔ Output 0
  If all fault-free inputs 0 ➔ Output 1
  Else output may be 0 or 1

- Termination after bounded interval
Byzantine Consensus via Byzantine Broadcast

Byzantine broadcast with source S

- Agreement: All outputs identical

- Validity: If source fault-free
  then output = source’s input
  Else output may be 0 or 1

- Termination after bounded interval
Byzantine Consensus via Byzantine Broadcast

- Each node uses Byzantine Broadcast to send its value to all others
- Each node computed output
  = majority of values thus received
Byzantine Broadcast

Example algorithm

- 4 nodes
- At most 1 faulty node

\[ n = 4 \]
\[ f = 1 \]

[Lamport, Shostak, Pease 1982]
Byzantine Broadcast

$n = 4$

![Diagram showing Byzantine Broadcast with $n = 4$ and a faulty node labeled "Faulty" connected to the source node S, which broadcasts input v to nodes 1, 2, and 3.]
n = 4

Byzantine Broadcast

Faulty

input v

S

1

2

3
Broadcast

input $v$

1 \rightarrow S 

S \rightarrow 1 \quad S \rightarrow 2 \quad S \rightarrow 3

1 \rightarrow 2 

2 \rightarrow 3
Broadcast

input $v$

1 \rightarrow S \rightarrow 2 \rightarrow 3

1 \rightarrow 2 \rightarrow ? \rightarrow ? \rightarrow ?
Broadcast

input v

S

1

2

3

V V V V V V V V

V V V V V V V V


47
Broadcast

input $v$

1 $\rightarrow$ 2 $\rightarrow$ 3

[?,$v$,?]

S $\rightarrow$ 1

$v$

$\rightarrow$ 2

$v$

$\rightarrow$ 3

[?,$v$,?]

$\rightarrow$ S

$\rightarrow$ 1

$v$

$\rightarrow$ 2

$v$

$\rightarrow$ 3

[?,$v$,?]

$\rightarrow$ S

$v$
Majority vote → Correct
Bad source may attempt to diverge state at good nodes
Broadcast
Broadcast
Vote identical at good nodes
Known Bounds on Byzantine Consensus

- \( n \geq 3f + 1 \) nodes to tolerate \( f \) failures
- Connectivity \( \geq 2f + 1 \)
- \( \Omega(n^2) \) messages in worst case
- \( f+1 \) rounds of communication
\( \Omega(n^2) \) Message Complexity

- Each message at least 1 bit

- \( \Omega(n^2) \) bits “communication complexity” to agree on just 1 bit value
Can we do better?
Lower Overhead

Two solutions:

- Probabilistically correct consensus (single instance)

- Amortizing cost over many instances of consensus
  - We will illustrate this approach for multi-valued consensus
LSP Algorithm

- Replication code
Multi-Valued Consensus

L-bit inputs

- Agreement: All nodes must agree

- Validity: If all fault-free nodes have same input, output is that input else output may be any value

- Termination
L-bit inputs

Potential solution: Agree on each bit separately

\[ \Omega(n^2 L) \] bits communication complexity

\[ \Omega(n^2) \] bits communication complexity per agreed bit
L-bit inputs

Potential solution: Agree on each bit separately

$\Omega(n^2 L)$ bits communication complexity

$\Omega(n^2)$ bits communication complexity per agreed bit

Possible to achieve $O(n)$ per agreed bit for large $L$
Intuition

- Make the common case fast
  - *Common case:* No failure

- Failure detection in “normal” operation, *not tolerance*

- Learn from past misbehavior, and adapt
Make Common Case Fast

Two-bit value a, b

S

a

1

b

2

a+b

3
Two-bit value $a, b$

Diagram:

1. Starting from state S, the transitions are:
   - $S \rightarrow 1$ with label $a$
   - $S \rightarrow 2$ with label $b$
   - $S \rightarrow 3$ with label $a+b$

2. From state 1:
   - $1 \rightarrow S$ with label $b$
   - $1 \rightarrow 2$ with label $a+b$

3. From state 2:
   - $2 \rightarrow S$ with label $a+b$
   - $2 \rightarrow 3$ with label $b$

4. From state 3:
   - $3 \rightarrow S$ with label $a+b$
   - $3 \rightarrow 1$ with label $a+b$

The states are labeled with the two-bit values $[a, b, a+b]$. The transitions are labeled with the single-bit values $a$, $b$, and $a+b$.
Make Common Case Fast

S

[a, b]

1 [a, b, a+b]

2 [a, b, a+b]

3 [a, b, a+b]

a

b

a+b

a

b

a+b

a+b

a+b
Make Common Case Fast

Two-bit value $a, b$

Parity check passes at all nodes $\Rightarrow$ Agree on $(a,b)$
Make Common Case Fast

Two-bit value a, b

Diagram:
- S
- 1
- 2
- 3
- Arrows: a, b, a+b, ?
Two-bit value $a, b$

Parity check fails at a node if 1 misbehaves to it
Make Common Case Fast

Check fails at a good node if S sends bad codeword (a,b,z)
Make Common Case Fast

The example

- \( n = 4 \) nodes, \( f = 1 \) (at most 1 failure)
- Uses \((3,2)\) code over binary symbols

The general algorithm for \( f \) faults

- Uses \((n, n-f)\) error detecting code, slightly differently
- Symbol size a function of \( n \) (number of nodes)
Algorithm Structure

- Fast round (as in the example)
Algorithm Structure

- Fast round  (as in the example)
Algorithm Structure

- Fast round  (as in the example)
- Fast round

... 

- Fast round in which failure is detected
Algorithm Structure

- Fast round (as in the example)
- Fast round

... 

- Fast round in which failure is detected
Algorithm Structure

- Fast round (as in the example)
- Fast round

... 

- Fast round in which failure is detected
- Expensive round to learn new info about failure
- Fast round
- Fast round

... 

- Expensive round to learn new info about failure.
Algorithm Structure

- Fast round (as in the example)
- Fast round

... 
- Fast round in which failure is detected
- Expensive round to learn new info about failure
- Fast round
- Fast round

...
- Expensive round to learn new info about failure.

After a small number of expensive rounds, failures completely identified
Algorithm Structure

- Fast round (as in the example)
- Fast round

... 

- Fast round in which failure is detected
- Expensive round to learn new info about failure
- Fast round
- Fast round

... 

- Expensive round to learn new info about failure.

After a small number of rounds failures identified

- Only fast rounds hereon
Algorithm “Analysis”

- Many fast rounds
- Few expensive rounds

- When averaged over all rounds, the cost of expensive rounds is negligible

- Average cost depends only on the fast round which is very efficient

- For large L, communication complexity is

\[ L \times \frac{n(n-1)}{(n-f)} = O(nL) \]
Algorithm Structure

- Fast round (as in the example)
- Fast round

... 

- Fast round in which failure is detected
- Expensive round to learn new info about failure
Life After Failure Detection

- Learn new information about identity of the faulty node

How?

- Ask everyone to broadcast what they did during the previous “faulty fast round”

- This is expensive
Life After Failure Detection

- “He-said-she-said”
- S claims sending $a+b$, C claims receiving $z \neq a+b$
Life After Failure Detection

- **Diagnosis graph** to track trust relationship

- Initial graph:

```
  S
  ↓↓↓
  1 -- 2 -- 3
```
Life After Failure Detection

- S and 3 disagree
- Remove edge S-3
The fast algorithm adapted to avoid using the link between $S$ and $3$

**Key:** Only use links between nodes that still trust each other

If more than $f$ nodes do not trust some node, that node must be faulty
Algorithm “Analysis”

- Many fast rounds
- Few expensive rounds

- When averaged over all rounds, the cost of expensive rounds is negligible

- Average cost depends only on the fast round which is very efficient

- For large L, communication complexity is

\[ L \times \frac{n(n-1)}{(n-f)} = O(nL) \]
Impact of network capacity region
Impact of Network

- Traditional metrics ignore network capacity region
  - Message complexity
  - Communication complexity
  - Round complexity

- How to quantify the impact of capacity?
Throughput

- Borrow notion of throughput from networking

- \( b(t) = \) number of bits agreed upon in \([0, t]\)

\[
Throughput = \lim_{t \to \infty} \frac{b(t)}{t}
\]
Impact of Network

- How does the network affect Byzantine broadcast/consensus?
Questions

- What is the maximum achievable throughput of Byzantine broadcast?

- How to achieve it?
Throughput of LSP algorithm = 1 bit/second

Optimal throughput = 11 bits/second
Multi-Valued Byzantine Broadcast

- A long sequence of input values
- Each input containing many bits

input 1  input 2  input 3  input 4 ...
Proposed Algorithm

- Multiple phases for each input value
- Trust relation
For each input ...

- **Phase 1**: Unreliable broadcast

- **Phase 2**: Failure detection
  - Agreement on whether failure is detected

- **Phase 3** (optional): Dispute control
  - Agreement on which nodes trust each other
For each input ... 

- Phase 1: Unreliable broadcast
- Phase 2: Failure detection
  - Agreement on whether failure is detected
- Phase 3 (optional): Dispute control
  - Agreement on which nodes trust each other

sender sends its input to all
For each input …

- Phase 1: Unreliable broadcast

- Phase 2: Failure detection
  - Agreement on whether failure is detected

- Phase 3 (optional): Dispute control
  - Agreement on which nodes trust each other

check if all good nodes got same value
For each input …

- Phase 1: Unreliable broadcast
- Phase 2: Failure detection
  - Agreement on whether failure is detected
- Phase 3 (optional): Dispute control
  - Agreement on which nodes trust each other
For each input …

- Phase 1: Unreliable broadcast

- Phase 2: Failure detection
  - Agreement on whether failure is detected

- Phase 3 (optional): Dispute control
  - Agreement on which nodes trust each other
For each input …

- **Phase 1:** Unreliable broadcast

- **Phase 2:** Failure detection
  - Agreement on whether failure is detected

- **Phase 3 (optional):** Dispute control
  - Agreement on which nodes trust each other

*maintain consistent knowledge*
Phase 1: Unreliable broadcast

Phase 2: Failure detection
  • Agreement on whether failure is detected

Phase 3 (optional): Dispute control
  • Agreement on which nodes trust each other
For each input ...

- Phase 1: Unreliable broadcast

- Phase 2: Failure detection
  - Agreement on whether failure is detected

- Phase 3 (optional): Dispute control
  - Agreement on which nodes trust each other

amortized cost negligible when input size large
For each input …

- Phase 1: Unreliable broadcast

- Phase 2: Failure detection
  - Agreement on whether failure is detected

- Phase 3 (optional): Dispute control
  - Agreement on which nodes trust each other
Phase 1: Unreliable Broadcast

Broadcast without fault tolerance

- Only links between trusting node pairs used

- Transmission along directed spanning trees rooted at source

- Rate for Phase 1 = number of directed unit-capacity spanning trees rooted at S
Phase 1: Unreliable Broadcast
Phase 1: Unreliable Broadcast

Unit capacity links
Phase 1: Unreliable Broadcast

Two directed spanning trees
Phase 1: Unreliable Broadcast

Two directed spanning trees

Each link in spanning tree carries 1 symbol
Phase 1: Unreliable Broadcast

Each link in spanning tree carries 1 symbol
Phase 1: Unreliable Broadcast
Phase 1: Unreliable Broadcast

Faulty nodes may tamper packets
For each input …

- Phase 1: Unreliable broadcast

- Phase 2: Failure detection
  - Agreement on whether failure is detected

- Phase 3 (optional): Dispute control
  - Agreement on which nodes trust each other
Phase 2: Failure Detection

- Equality checking for every subset of $n-f$ nodes

- Faulty nodes should not be able to make unequal values appear equal
Failure Detection (Equality Checking)
Failure Detection (Equality Checking)
Checking with **Undirected** Spanning Trees

Each tree checks 1 data symbol
Each tree checks 1 data symbol
Shortcoming

- Each link used by *multiple* sets of $n-f$ nodes
Failure Detection (Equality Checking)
Shortcoming

- Each link used by *multiple* sets of $n-f$ nodes
  - Inefficient to check separately

- Reuse symbols for
  - solving all equality checks *simultaneously*
Phase 2: Failure Detection (Equality Checking)

- Using local coding
Local Coding

Each node sends linear combinations of its data symbols.

\[ X_1, Y_1, Z_1 \]

\[ X_3 + 3Y_3 + 9Z_3 \]

\[ X_3, Y_3, Z_3 \]

\[ X_3 + 8Y_3 + 64Z_3 \]

\[ X_2, Y_2, Z_2 \]

\[ X_S, Y_S, Z_S \]
Each node checks consistency of received packets with own data

\[ X_2 + 3Y_2 + 9Z_2 = X_S + 3Y_S + 9Z_S \]
All equality checks can be solved simultaneously if

\[ \# \text{ data symbols} \leq \min \# \text{ undirected spanning trees over all sets of } n-f \text{ nodes} \]
Throughput Analysis
Throughput Analysis

Overhead of Phases 1 and 2 dominant

\[ B = \text{minimum rate of Phase 1} \]
\[ U = \text{minimum rate of Phase 2} \]

\[
\text{Throughput} \geq \frac{1}{1/B + 1/U} = \frac{BU}{B + U}
\]
Throughput Analysis

Upper bound on optimal throughput (proof in paper)

\[ \leq \min(B, 2U) \]

Our algorithm \[ \geq \frac{BU}{B + U} \]

\[ \geq \frac{\text{optimal}}{3}, \text{ in general} \]
\[ \geq \frac{\text{optimal}}{2}, \text{ if } B \leq U \]
Proof Sketch (1)

Consider any N-f nodes: 1, 2, …, N-f

Each has value $X_i = [X_{i,1} \ X_{i,2} \ ... \ X_{i,r}]$

Define difference: $D_i = X_i - X_{N-f}$

$D_1 = D_2 = ... = D_{N-f-1} = 0 \iff X_1 = X_2 = ... = X_{N-f}$
Proof Sketch (2)

Checking in Phase 2 is equivalent to checking

$$[D_1 \ D_2 \ \ldots \ D_{N-f-1}] \ C = 0$$

$C$ -- Matrix filled with coefficients

$(N-f-1)r$ rows

Each column corresponds to one unit-capacity link and the corresponding linear combination
Proof Sketch (3)

If $r \leq \# \text{undirected spanning trees among these nodes}$

$\Rightarrow \quad \#\text{columns} \geq \#\text{rows} = (N-f-1)r$

$M \quad \text{(N-f-1)r-by-(N-f-1)r submatrix of C}$

$\text{corresponds to the links along r spanning trees}$

\begin{align*}
\text{We can make M invertible. In this case} \\
[D_1 \ D_2 \ \ldots \ D_{N-f-1}] \ C = 0 & \Rightarrow & [D_1 \ D_2 \ \ldots \ D_{N-f-1}] \ M = 0 \\
& \Rightarrow & [D_1 \ D_2 \ \ldots \ D_{N-f-1}] = 0 \\
& \Rightarrow & X_1 = X_2 = \ldots = X_{N-f}
\end{align*}

\textbf{Solves equality checking for one set of N-f nodes}
Proof Sketch (4)

If $r \leq \min \# \text{undirected spanning trees}$ over all sets of $N-f$ nodes

We can make all $M$ matrices invertible simultaneously!

Solves equality checking for all sets of $N-f$ nodes
So far ...
Summary

Exact consensus

- Byzantine consensus with crash failures
- Byzantine consensus with Byzantine failures
- Impact of network capacity
Iterative Algorithms for Approximate Consensus
Outline

- Iterative average consensus (no failures)
  - Impact of lossy links

- Iterative Byzantine consensus
  - With and without synchrony
Average Consensus

- want to decide which way to fly
- Every proposes a direction

Consensus $\approx$ average of inputs
Centralized Solution

- A leader collects **inputs** & disseminates **decision**
Distributed Iterative Solution

- Initial state  \( a, b, c = \text{input} \)
Distributed Iterative Solution

- State update (iteration)

\[
\begin{align*}
\text{a} &= \frac{3}{4}a + \frac{1}{4}c + \frac{1}{4}c \\
\text{b} &= \frac{3}{4}b + \frac{1}{4}c + \frac{1}{4}b \\
\text{c} &= \frac{1}{4}a + \frac{1}{4}b + \frac{1}{2}c
\end{align*}
\]
\[
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
:=
\begin{pmatrix}
3/4 & 0 & 1/4 \\
0 & 3/4 & 1/4 \\
1/4 & 1/4 & 1/2
\end{pmatrix}
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
= \mathbf{M}
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
\]

\[
b = 3b/4 + c/4
\]
\[
c = a/4 + b/4 + c/2
\]
\[
a = 3a/4 + c/4
\]
after 2 iterations

\[
\begin{pmatrix}
a \\ b \\ c
\end{pmatrix} := M \begin{bmatrix}
a \\ b \\ c
\end{bmatrix} = M^2 \begin{pmatrix}
a \\ b \\ c
\end{pmatrix}
\]

after 1 iteration

\[
b = 3b/4 + c/4
\]

\[
c = a/4 + b/4 + c/2
\]

\[
a = 3a/4 + c/4
\]
after $k$ iterations

$$
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
:=
M^k
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
$$

$$
c = \frac{a}{4} + \frac{b}{4} + \frac{c}{2}
$$

$$
b = 3\frac{b}{4} + \frac{c}{4}
$$

$$
a = 3\frac{a}{4} + \frac{c}{4}
$$
Reliable Nodes & Links

- Consensus achievable iff at least one node can reach all other nodes

- Average consensus achievable iff strongly connected graph

with suitably chosen transition matrix $M$
Reliable Nodes & Links

- Consensus achievable iff at least one node can reach all other nodes

- Average consensus achievable iff strongly connected graph with suitably chosen transition matrix $M$

Row stochastic $M$

Doubly stochastic $M$

with suitably chosen transition matrix $M$
\[
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix} := M^k \begin{pmatrix}
a \\
b \\
c
\end{pmatrix} \Rightarrow \begin{pmatrix}
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3}
\end{pmatrix} \begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
\]

Doubly stochastic $M$

\[
a = \frac{3a}{4} + \frac{c}{4}
\]

\[
b = \frac{3b}{4} + \frac{c}{4}
\]

\[
c = \frac{a}{4} + \frac{b}{4} + \frac{c}{2}
\]
An Implementation: Mass Transfer + Accumulation

- Each node “transfers mass” to neighbors via messages
- Next state = Total received mass

\[ c = \frac{a}{4} + \frac{b}{4} + \frac{c}{2} \]

\[ a = \frac{3a}{4} + \frac{c}{4} \]

\[ b = \frac{3b}{4} + \frac{c}{4} \]
An Implementation: Mass Transfer + Accumulation

- Each node “transfers mass” to neighbors via messages
- Next state = Total received mass

\[ \begin{align*}
  c &= a/4 + b/4 + c/2 \\
  b &= 3b/4 + c/4 \\
  a &= 3a/4 + c/4
\end{align*} \]
Conservation of Mass

- \( a + b + c \) constant after each iteration

\[
c = \frac{a}{4} + \frac{b}{4} + \frac{c}{2}
\]

\[
a = \frac{3a}{4} + \frac{c}{4}
\]

\[
b = \frac{3b}{4} + \frac{c}{4}
\]
Wireless Transmissions Unreliable

\[ c = \frac{a}{4} + \frac{b}{4} + \frac{c}{2} \]

\[ a = \frac{3a}{4} + \frac{c}{4} \]

\[ b = \frac{3b}{4} + \frac{c}{4} \]
Impact of Unreliability

\[
\begin{pmatrix}
a \\ b \\ c
\end{pmatrix} = 
\begin{pmatrix}
3/4 & 0 & 1/4 \\
0 & 3/4 & 0 \\
1/4 & 1/4 & 1/2
\end{pmatrix}
\begin{pmatrix}
a \\ b \\ c
\end{pmatrix}
\]

\[
b = \frac{3b}{4} + \frac{c}{4}
\]

\[
c = \frac{a}{4} + \frac{b}{4} + \frac{c}{2}
\]

\[
a = \frac{3a}{4} + \frac{c}{4}
\]
Conservation of Mass

\[
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
= \begin{pmatrix}
3/4 & 0 & 1/4 \\
0 & 3/4 & 0 \\
1/4 & 1/4 & 1/2
\end{pmatrix}
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
\]

\[c = a/4 + b/4 + c/2\]

\[b = 3b/4 + c/4\]

\[a = 3a/4 + c/4\]
Average consensus over lossy links ?
Solution 1

Assume that

transmitter & receiver **both know**

whether a message is delivered or not
Solution 2

When mass not transferred to neighbor,

keep it to yourself
Convergence ... if nodes intermittently connected

\[
\begin{pmatrix}
  a \\
  b \\
  c
\end{pmatrix}
= \begin{pmatrix}
  3/4 & 0 & 1/4 \\
  0 & 3/4 & 0 \\
  1/4 & 1/4 & 3/4
\end{pmatrix}
\begin{pmatrix}
  a \\
  b \\
  c
\end{pmatrix}
\]

\[c = \frac{a}{4} + \frac{b}{4} + \frac{c}{2} + \frac{c}{4}\]

\[a = \frac{3a}{4} + \frac{c}{4}\]

\[b = \frac{3b}{4} + \frac{c}{4}\]
Common knowledge on whether a message is delivered
Common knowledge on whether a message is delivered

S knows
R knows that S knows
S knows that R knows that S knows
R knows that S knows that R knows that ...

Solution 1
Reality in Wireless Networks

Common knowledge on whether a message is delivered

Two scenarios:

- A’s message to B lost … B does not send Ack
- A’s message received by B … Ack lost
Reality in Wireless Networks

- Common knowledge on whether a message is delivered

- Need solutions that tolerate lack of common knowledge

- Theoretical models need to be more realistic
Solution 2

- Average consensus **without** common knowledge

  ... using additional per-neighbor state
Solution Sketch

- $S =$ mass C wanted to transfer to node A in total so far
- $R =$ mass A has received from node C in total so far
Solution Sketch

- Node C transmits quantity S
  .... message may be lost

- When it is received,
  node A accumulates (S-R)
What Does That Do?
What Does That Do?

- Implements virtual buffers
Dynamic Topology

- When $C \rightarrow B$ transmission unreliable, mass transferred to buffer $(d)$

- $d = d + \frac{c}{4}$
**Dynamic Topology**

- When C→B transmission **unreliable**, mass transferred to buffer (d)

- \( d = d + \frac{c}{4} \)

No loss of mass even with message loss
Dynamic Topology

- When C → B transmission reliable, mass transferred to b

- \( b = \frac{3b}{4} + \frac{c}{4} + d \)

No loss of mass even with message loss
Does This Work?
Does This Work?
Why Doesn’t it Work?

- After k iterations, state of each node has the form
  \[ z(k) \times \text{sum of inputs} \]
  where \( z(k) \) changes each iteration (k)

- Does not converge to average
Solution

- Run two iterations in parallel

  - First: original inputs
  - Second: $\text{input} = 1$
Solution

- Run two iterations in parallel
  - First: original inputs
  - Second: \textit{input} = 1

- After \( k \) iterations …
  
  \begin{align*}
  \text{first algorithm:} & \quad z(k) \times \text{sum of inputs} \\
  \text{second algorithm:} & \quad z(k) \times \text{number of nodes}
  \end{align*}
Solution

- Run two iterations in parallel
  - First: original inputs
  - Second: input = 1

- After k iterations ...

  first algorithm: \( z(k) \times \text{sum of inputs} \)
  second algorithm: \( z(k) \times \text{number of nodes} \)

\[ \text{ratio} = \text{average} \]
Byzantine Faults

- Faulty nodes may misbehave arbitrarily
Iterative Consensus with Byzantine Failures
Byzantine node

\[ a = \frac{3a}{4} + \frac{c}{4} \]

\[ b = \frac{3b}{4} + \frac{c}{4} \]

No consensus!
Iterative Consensus

Dolev and Lynch, 1983

- Tolerate up to $f$ faults in complete graphs
Iteration at each node

- Receive current state from all other nodes
- Drop largest f and smallest f received values
- New state = average of remaining values & old state
Iteration at each node

- Receive current state from all other nodes
- Drop largest $f$ and smallest $f$ received values
- New state = average of remaining values & old state

![Diagram with nodes A, B, C, D and edges labeled with numbers 0, 5, 3, 4, 1, f = 1]
Iteration at each node

- Receive current state from all other nodes
- Drop largest $f$ and smallest $f$ received values
- New state = average of remaining values & old state
Recall … consensus without faults

Consensus possible if a single node can influence all others

How to generalize to Byzantine faults?
Necessary Condition

\[
\begin{align*}
\text{for any } i \in L & \quad \geq f + 1 \\
\text{for any } j \in R & \quad \geq f + 1
\end{align*}
\]
Necessary Condition

Partition nodes into 4 sets

\[ F \quad L \quad C \quad R \]

\[ \geq f+1 \]

\[ \geq f+1 \]
Necessary Condition

L, R non-empty

\( F \geq f+1 \)

\( C \geq f+1 \)

Potential fault set
Necessary Condition

\[ \geq f+1 \]

For any \( i \) in \( L \) or \( j \) in \( R \) must exist
Necessary Condition

For any $i$ in $L$ and any $j$ in $R$, $i$ or $j$ must exist.
Intuition … proof by contradiction

\[ f = 2 \]
L1’s Perspective … if F1,F2 faulty

L1 must choose $\geq 0$
L1’s Perspective ... if R1, R2 faulty

L1 must choose $\leq 0$
L1’s Perspective ... if R1,R2 faulty

L1 must choose ≥ 0

L1 must choose ≤ 0
Equivalent Necessary Condition

Reduced graph

- Remove any potential fault set F
- Remove additional $f$ links from each remaining node
Equivalent Necessary Condition

Reduced graph

- Remove any potential fault set F
- Remove additional $f$ links from each remaining node

In each reduced graph, some node must influence all others
Sufficiency … previous algorithm works

Algorithm for each node

- Receive current state from all neighbors
- Drop largest $f$ and smallest $f$ received values
- New state = average of remaining values & old state
Sufficiency Proof

- State of the fault-free nodes converges to a value in the convex hull of their inputs
Proof Outline

- $V[t] = \text{state of fault-free nodes after } t\text{-th iteration}$
Proof Outline

- $V[t] = \text{state of fault-free nodes after } t\text{-th iteration}$
- Represent $t$-th iteration as

\[
V[t] = M[t] \cdot V[t-1]
\]
Proof Outline

- $V[t] = \text{state of fault-free nodes after } t\text{-th iteration}$
- Represent $t\text{-th iteration as}$

$$V[t] = M[t] \ V[t-1]$$

where $M[t]$ depends on
- $V[t-1]$
- Behavior of faulty nodes

but has nice properties
Proof Outline

- $V[t] = \text{state of fault-free nodes after } t\text{-th iteration}$
- Represent $t$-th iteration as

$$V[t] = M[t] \ V[t-1]$$

i-th row of $M[t]$ corresponds to update of node $i$’s state
Node $i$

$\text{f} = 1$
Node i

- Values used for state update in convex hull of state of good neighbors
Values used for state update in convex hull of state of good neighbors

\[(1/3) \times 1 + (1/3) \times 3 + (1/3) \times 4\]
Node i

- Values used for state update in convex hull of state of good neighbors

\[(1/3) \times 1 + (1/3) \times 3 + (1/3) \times 4 \]

\[1 = (4/5) \times 0 + (1/5) \times 5\]
Node i

\[ V[t] = M[t] \cdot V[t-1] \]

with contribution of faulty node D replaced by contribution from A,B
Node $i$

- $V[t] = M[t] V[t-1]$ with contribution of faulty node $D$ replaced by contribution from $A, B$

- With care is taken in replacement, $M[t]$ has following properties:
  - Row stochastic
  - Non-zero entries corresponding to links in a reduced graph
Reduced graph

- Remove any potential fault set F
- Remove additional $\leq f$ links from each remaining node

In each reduced graph, some node must influence all others
Iterative Byzantine Consensus

Results naturally extend to

- Asynchrony
- Time-varying topology
- Arbitrary fault sets
Summary

Consensus with

- Faulty nodes
- Lossy links