Advanced Blockchain Engineering (ASCI a27)

Fault Tolerance and Consensus

1. Problem definitions
2. Stopping failures
3. Byzantine failures
4. Randomized solutions

Dick H.J. Epema
Distributed Systems Group

April 2018
Fault tolerance and consensus (1/4)

- Two persons (Alice and Bob) try to make an appointment
- Two propositions:
  - $P_A$: Alice wants to have the appointment
  - $P_B$: Bob wants to have the appointment
- Alice sends message $A_1$ that she wants the appointment
- Bob receives $A_1$, and knows $P_A$: $K_B(P_A)$
- Bob sends back a message $B_1$ that he wants to go too
- Alice receives $B_1$, and so $K_A(P_B)$ and $K_A(K_B(P_A))$ hold
- Alice sends confirmation $A_2$ back
- This continues **forever: no solution**
- Problem: messages may have **arbitrary delays** and may **get lost**
Fault tolerance and consensus (2/4)

- Everybody uses such knowledge concepts daily

**Not enough:**
everybody knows(right has priority)

**Required:**
everybody knows(everybody knows(right has priority))

**Common knowledge:** $K^n(P)$ for $n=1, 2, \ldots$
Fault tolerance and consensus (3/4)

- Processors may need to reach **consensus**
- Agreement modeled as **agreeing on the value of a single bit**
- Reaching consensus is a problem **in the face of failures**

**Applications:**
- **commit a transaction** in a distributed database:
  - all participating sites have to **agree** on committing the results
- in a **distributed database with replication**:
  - when a record is to be modified, the database servers holding the replicas have to **agree** on the modification
Fault tolerance and consensus (4/4)

- **Original motivation:**
  - in a **replicated computation**:
    - processors have to start with **the same input value** (e.g., from a sensor), so they have to **agree** on this value
    - but the sensor may be broken and give **very different readings**
Fault classification

- Possible processor failures:
  - **fail-stop** (crash) failures:
    - a processor just stops
  - **omission** failures:
    - a processor fails to send or receive a message
  - **performance** failures:
    - a processor does not meeting timing specifications
  - **Byzantine** failures:
    - a processor exhibits random (malicious) behavior
Model aspects

• **Synchronous versus asynchronous:**
  o reaching agreement is much more difficult in *asynchronous* systems: difference between long delay and processor/link failure cannot be detected

• **Authentication:**
  o **without:** messages *can be forged* or altered by a process before passing them along to others
  o **with:** messages *cannot be forged* or modified
  o agreement much more difficult to reach when messages are **non-authenticated**

• **Network connectivity:**
  o we assume a complete network
Agreement with stopping failures

• All processes start with an initial value from some set V

• Every process has to decide on a value in V with conditions:
  
  1. Agreement  no two processes decide on different values
  
  2. Validity   if all processes start with the same value v, then no process decides on a value different from v
  
  3. Termination all non-faulty processes decide within finite time
Byzantine generals

• City surrounded by armies
• Armies have to attack **simultaneously** in order to conquer the city
• Communication between generals by **means of messengers**
• Some generals of the armies are **traitors**
The Byzantine agreement problem

• **One process** (the *source* or *commander*) starts with a binary value \( C (0/1) \)
• Each of the remaining processes (the *lieutenants*) has to decide on a binary value such that:
  1. **Agreement** all non-faulty processes agree on the same value
  2. **Validity** if the source is non-faulty, then all non-faulty processes agree on the initial value of the source
  3. **Termination** all processes decide within finite time

• So if the source is faulty, the non-faulty processes can agree on any value
• It is irrelevant on what value a faulty process decides
Two variations

- **All generals** start with a value $v_i$
- **Variation 1:** $(v_1, v_2, \ldots, v_n)$
  - all non-faulty generals have to agree on a vector with a value for every general
  - solution: run a copy of an algorithm for the previous problem for every general
- **Variation 2:** \text{majority}(v_1, v_2, \ldots, v_n)
  - all non-faulty generals have to agree on a single value
  - solution: apply the same decision rule on the vector in every general (e.g., majority function)
A solution for stopping failures (1/3)

- Solution by repeatedly **flooding** decision values
- No more than $f$ failing processes
- Every process starts with a value $v$
- Every process maintains a set $W$ (with decision values seen so far)
- **Initially**: $W=\{v\}$
- **Then**, do $f+1$ rounds
  - broadcast($W$) to all other processes
  - receive($W_j$) from all processes $P_j$ and set $W = \bigcup_j W_j$
- **Finally**, if $W$ contains only a single element $v$, **decide($v$)**
  - else **decide**(default)
A solution for stopping failures (2/3)

- **Validity** and **termination** are trivially satisfied
- For **agreement**:
  - enough to show that all processes that are still active at the end of round $f+1$ then have the same set $W$
A solution for stopping failures (3/3)

• **Optimization:**
  - processes only need to know whether at the end $|W|=1$ or $|W|>1$
  - so let processes only broadcast at most two values:
    - their initial value
    - the first different value they receive
Conditions for a solution for Byzantine Agreement

- Number of processes: \( n \)
- Maximum number of possibly failing processes: \( f \)
- **Necessary and sufficient condition** for a solution to Byzantine agreement:
  - Minimal number of rounds in a deterministic solution:
  - There exist **randomized solutions** with a lower expected number of rounds
Example: three generals (1/3)

**Scenario 1**: Lieutenant $L_2$ is a traitor

Note all messages sent and received by $L_1$.
Example: three generals (2/3)

Scenario 2: Commander C is a traitor:

same messages sent and received by L₁
Example: three generals (3/3)

- \( L_1 \) has to decide 0 in scenario 1, because both \( L_1 \) and \( C \) are loyal and \( C \) starts with a 0
- Lieutenant \( L_1 \) cannot distinguish between the two scenarios

Contradiction: \( L_1 \) and \( L_2 \) are both loyal in scenario 2, but decide on different values!

This is an example of an impossibility result.
A solution for Byzantine agreement (1/9)

- Algorithm is recursive with \( f+1 \) levels
- Without authentication, modeled with Oral Messages (OM)
- When a message is supposed to be sent according to the algorithm, but a process does not send it, this is detected, and a default value (e.g., 0) is assumed
- Bottom case of the recursion: OM(0)
  - the commander broadcasts its initial value
  - every other process decides on the value it receives
    (or on the default value if it does not receive anything)
A solution for Byzantine agreement (2/9)

- \( \text{OM}(f), f>0 \) (resilient to \( f \) failures):
  - the commander broadcasts its initial value
  - process numbering: commander=0, lieutenants 1,2,...,n-1
  - let \( v_i \) be the value received from the commander by lieutenant \( L_i \), or the default if no value is received
  - recursive step:
    - \( L_i \) executes \( \text{OM}(f-1) \), acting as the commander for the other lieutenants \( (L_1, ..., L_{i-1}, L_{i+1}, ..., L_{n-1}) \)
    - let \( v_j \) be the value on which \( L_i \) decides in the recursive step with \( L_j \) as the commander (for \( j=1,2,...,n-1, \ i \neq j \))
  - \( L_i \) decides on majority\((v_1,...,v_i,...,v_{n-1})\)
A solution for Byzantine agreement (3/9)

here $L_i$ decides on its own $v_1$ as a lieutenant of $L_1$

$L_i$ receives $v_i$ immediately from the commander
A solution for Byzantine agreement (4/9)

n=7
f=2

C

L₁
L₂
L₃
L₆

L₆ receives a message from C

L₆ receives a message along every path of length 2

L₆ receives a message along every path of length 3
A solution for Byzantine agreement (5/9)

- So a lieutenant *does not decide on the majority* of all values it receives!!!
- But $L_i$ decides on:

\[
\text{majority( majority(), majority(), ..., } v_i, ..., \text{ majority(), ..., majority())}
\]

computed as the decision when acting as a lieutenant in OM(f-1)

obtained directly from the commander
A solution for Byzantine agreement (6/9)

• Number of executions:
  
  o $OM(f)$: 1 time
  
  o $OM(f-1)$: $(n-1)$ times
  
  o $OM(k)$: $(n-1)(n-2) \ldots (n-f+k)$ times for $k=0,1,\ldots,f-1$

• Total number of messages is of order $n^{f+1}$:
  
  o $OM(f)$: $n-1$
  
  o $OM(f-1)$: $(n-1)(n-2)$
  
  o $OM(k)$: $(n-1)(n-2) \ldots (n-(f-k))(n-(f-k+1))$
  
  o $OM(0)$: $(n-1)(n-2) \ldots (n-(f+1))$ ($f+1$ factors, this dominates)
A solution for Byzantine Agreement (7/9)

In lieutenant $L_6$

- In order to decide, every lieutenant $L_i$ creates a labelled tree with $f+1$ levels:
  - **level 0**: the root with **label 0** (the commander)
  - **level 1**: $n-2$ children with all labels except 0 and $i$
  - at every **subsequent level**: all ids that have not yet occurred on the path from the root and are different from $i$
  - the **degree** decreases by 1 at every level

<table>
<thead>
<tr>
<th>n</th>
<th>f</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
A solution for Byzantine Agreement (8/9)

Label the nodes of the tree in $L_i$ with additional labels:

- **level 0**: $v_i$ (value received from the commander)
- **level 1**: the value that $L_j$ told $L_i$ that the commander told him
- label of any node: the value that was passed to $L_i$ from the commander **through the chain of lieutenants on the path** from the root to the node
A solution for Byzantine Agreement (9/9)

- Decide by propagating the result up with the majority function:
  - at the leaf level: decide on the value received (OM(0))
  - at every next higher level: take the majority of the local value and the decisions at child nodes
  - the final value at the root is the final decision
Example: four generals (1/2)

Every loyal lieutenant receives: v,v,?
Example: four generals (2/2)

Every loyal lieutenant receives: x,y,z
Byzantine agreement with authentication (1/2)

- Every message **carries a signature**
- The signature of a loyal general **cannot be forged**
- Alteration of the contents of a signed message can be detected
- Every (loyal) general can **verify the signature** of any other (loyal) general
- **Any number f of traitors** can be allowed
- Commander is process 0
- **Structure of message** from (and signed by) the commander, and subsequently signed and sent by lieutenants $L_{i_1}, L_{i_2}, \ldots$:
  - $\langle v : s_0 : s_{i_1} : \ldots : s_{i_k} \rangle$
- Every lieutenant maintains a **set of orders** $V$
- Some choice function on $V$ for deciding (e.g., majority, minimum)
Byzantine agreement with authentication (2/2)

- Algorithm in commander:
  
  \[ \text{send}(v : s_0) \text{ to every lieutenant} \]

- Algorithm in every lieutenant \( L_i \):
  
  \[
  \text{upon receipt of } (v : s_0 : s_{i_1} : \ldots : s_{i_k}) \text{ do }
  \]
  
  \[
  \begin{align*}
  \text{if } (v \text{ not in } V) \text{ then } \\
  V := V \cup \{v\} \\
  \text{if } (k < f) \text{ then } \\
  \text{for } (j \in \{1, 2, \ldots, n-1\} \setminus \{i, i_1, \ldots, i_k\}) \text{ do } \\
  \text{send}(v : s_0 : s_{i_1} : \ldots : s_{i_k} : s_j) \text{ to } L_j \\
  \text{if } (L_i \text{ will not receive any more messages}) \text{ then } \\
  \text{decide}(\text{choice}(V))
  \end{align*}
  \]

sign and propagate messages “long enough”
Example: three generals

Commander C is traitor:

Format: 
\[ \text{value}:\text{signature(s)} \]

\[ V = \{0,1\} \]

\[ L_1 \]

\[ 1:0 \]

\[ 0:0:2 \]

\[ 0:0 \]

\[ 1:0:1 \]

\[ V = \{0,1\} \]

\[ L_2 \]
Efficient solutions to Byzantine Agreement

- In the OM algorithm, there are $n^{f+1}$ messages
- There are algorithms with polynomial message complexity
- Algorithms with authentication are “pretty easy”
Authenticated broadcast (1/7): simulation

• Make the communication “look” authenticated by means of a new communication primitive called authenticated broadcast or consistent broadcast

• In a sense a simulation:

  authenticated Byzantine generals algorithm

  synchronous system with authenticated broadcast

  basic synchronous system with Byzantine faults
Authenticated broadcast (2/7): properties

- Broadcast message from process $P_i$ in round $r$: $(i,m,r)$
- A process that receives $(k,m,r)$ and can verify the signature of $P_k$, accepts the message (possibly much later if $P_k$ is faulty)
- A process that accepts a message, immediately relays it to everybody else
- Properties of authenticated broadcast:
  1. Correctness: if correct process $P_i$ broadcasts $(i,m,r)$ in round $r$, every correct process accepts $(i,m,r)$ in the same round
  2. Unforgeability: if process $P_i$ is correct and does not broadcast $(i,m,r)$, then no correct process accepts $(i,m,r)$
  3. Relay: if a correct process accepts $(k,m,r)$ in round $r' \geq r$, then every other correct process accepts $(k,m,r)$ in round $r' + 1$ or earlier
Authenticated Broadcast (3/7): algorithm

- Commander $P_0$ starts with some value $v$
- All other processes start with $v=0$
- Only values of 1 are sent

- for $r=1$ to $f+1$ do
  - if ( $(v=1)$ and (process has not done a broadcast before) ) then
    - broadcast($i, 1, r$)
    - relay $r-1$ messages accepted in previous rounds that caused $v=1$
    - if ( (in rounds $r' \leq r$) accepted($j, 1, r'$) from $P_0$ and $r-1$ other processes $P_j$) then
      - $v:=1$
  - decide($v$)

processes do at most one broadcast

sufficient support for value 1
Authenticated broadcast (4/7): implementation

- Implementation can be done with digital signatures
- Here: \textit{without signatures}
- **Basic idea:** a correct process only accepts a broadcast if there are sufficient \textit{witnesses} for it
- Required: \(n > 3f\)
- Message primitives to be implemented: \textit{broadcast} and \textit{accept}
- Implemented with lower-level messages: \textit{init} and \textit{echo}
- **One round** of broadcast implemented with \textit{two phases} at the lower level

\[ P_i \xrightarrow{\text{“init”}} P_j \xrightarrow{\text{“echo”}} \]
Authenticated broadcast (5/7): implementation

- Broadcasting \((i,m,k)\) by \(P_i\), round \(k\)
  - Phase 2k-1: \(P_i\) send\((init,i,m,k)\) to all
  - Phase 2k:
    if \(P_j\) receives \((init,i,m,k)\) from \(P_i\) in phase 2k-1 then send\((echo,i,m,k)\) to all
    if \(P_j\) receives \((echo,i,m,k)\) from at least \(n-f\) processes then
      accept\((i,m,k)\)

![Diagram of authenticated broadcast](image-url)
Authenticated broadcast (6/7): implementation

- Broadcasting $\langle i,m,k \rangle$ by $P_i$, round $r \geq k+1$
  - Phases $2r-1$, $r$:
    - if ($P_j$ received $\langle \text{echo},i,m,k \rangle$ from at least $n-2f$ processes in previous phases and not sent $\langle \text{echo},l,m,k \rangle$) then
      - send $\langle \text{echo},i,m,k \rangle$ to all
    - if ($P_j$ received $\langle \text{echo},i,m,k \rangle$ from at least $n-f$ processes in this and previous phases) then
      - accept $\langle i,m,k \rangle$

\[ P_i \quad \text{"init"} \quad P_j \quad \text{"echo"} \]
Authenticated broadcast (7/7): Complexity

- Limit to correct processes
- Every correct processes does at most one broadcast and up to f relays
- A single broadcast by a correct process takes $n^2$ messages (1-to-n init and all-to-all echo)
- No need for explicit relays
- So the message complexity is of order $n^3$
Randomized Byzantine agreement (1/6)

• Solution for synchronous and asynchronous systems!!
• n processes, of which at most f fail, n>5f
• Every process has an initial value v
• The algorithm proceeds in rounds consisting of three phases:
  1. a notification phase (messages have message type N)
  2. a proposal phase (messages have message type P)
  3. a decision phase
• When a process expects messages from all other processes, it is no use waiting for more than n-f messages
• When not enough processors support a possible decision, a process starts the next round with a new random input value v
Randomized Byzantine agreement (2/6)

\[
\begin{align*}
\text{r=1; decided:=false} \\
\text{do forever} \\
\quad \text{broadcast}(N,r,v) \\
\quad \text{await } (n-f) \text{ messages of the form } (N,r,*) \\
\quad \text{if }>(n+f)/2 \text{ messages } (N,r,w), w=0,1 \text{ then} \\
\quad \quad \text{broadcast}(P,r,w) \\
\quad \text{else broadcast}(P,r,?) \\
\quad \text{if decided then STOP} \\
\quad \text{else await } (n-f) \text{ messages of the form } (P,r,*) \\
\quad \text{if }>f \text{ messages } (P,r,w), w=0,1 \text{ then} \\
\quad \quad v:=w \\
\quad \quad \text{if }>3f \text{ messages } (P,r,w) \text{ then} \\
\quad \quad \quad \text{decide}(w) \\
\quad \quad \text{decided:=true} \\
\quad \text{else } v:=\text{random}(0,1) \\
\quad r:=r+1 \\
\end{align*}
\]
Randomized Byzantine agreement (3/6)

- “No simultaneous contradicting proposals by correct processes”
- **Lemma 1:**
  - If a **correct process proposes** \( v \) in round \( r \), then
    - **no other correct process proposes** 1-\( v \) in round \( r \)
- **Proof:**
  - a process that does a proposal has received more than \( (n+f)/2 \) messages \( (N,r,v) \)
  - of these, more than \( (n-f)/2 \) are from correct processes,
    which is a **majority of the correct processes**
  - so there can only be one proposed value
Randomized Byzantine agreement (4/6)

- “When all correct processes have the same value, immediate decision”
- Lemma 2:
  - If at the start of round $r$ all correct processes have the same value $v$, then they all decide $v$ in round $r$
- Proof:
  - Each correct process receives at least $n-f$ notification messages, at least $n-2f$ of which are from correct processes, and so of the form $(N,r,v)$
  - Because $n>5f$, we have $n-2f = n/2+n/2-2f > n/2+5f/2-2f = (n+f)/2$
  - This is exactly the condition for every correct process to propose $v$
  - So, each correct process receives at least $n-2f$ messages of the form $(P,r,v)$
  - Because $n>5f$, we have $n-2f>3f$, which is exactly the condition for every correct process to decide $v$
Randomized Byzantine agreement (5/6)

• “Decision of any correct process immediately followed by others”

• Lemma 3:
  o If a correct process decides $v$ in round $r$, all correct processes decide $v$ in round $r+1$

• Proof:
  o enough: all correct processes propose $v$ in round $r+1$
  o if a process decides $v$ in round $r$, it must have received more than $3f$ proposals for $v$, $m$ of which are from correct processes for some $m>2f$
  o so every other correct processor receives at least $m-f>f$ proposals for $v$, so it starts the next round with this value
  o now use Lemma 2
Randomized Byzantine agreement (6/6)

• **Theorem:**
  - If $n > 5f$, the algorithm guarantees *agreement, validity, and terminates with probability 1*

• **Proof:**
  - with probability 1, enough processors will pick the same value $v$ to have at least one correct process decide

• **Expected number of rounds** is of order $2^n$ (in fact, slightly better)

• **Remark:** randomization is used only if there is not enough initial support for any decision anyway
Randomized Coordinated Attack (1/12)

- Synchronous system, complete graph
- System runs for a fixed number $r$ of rounds
- Messages may get lost (all links may exhibit failures)
- Processes do not exhibit failures
- Validity:
  - if all processes start with 0, they all decide 0
  - if all processes start with 1 and all messages are received, they all decide 1
- Agreement with some probability:
  - $P[\text{some process decides 0 and some process decides 1}] \leq \epsilon$, for some $0 \leq \epsilon \leq 1$ (probability of disagreement)
- Termination trivial
RCA (2/12): Adversaries

- Faults modeled with an adversary who can on purpose try to deceive the system/processors
- Here, the adversary can choose:
  - the input values of the processors
  - the communication pattern (can omit arbitrary messages)
- In the algorithm, we get $\varepsilon=\frac{1}{r}$
RCA (3/12): Communication patterns (1)

- **Communication pattern**: a subset of the set 
  \{(i,j,k): (i,j) an edge in the processor graph, k a round number\}

- **We will define an ordering** \(\leq_\gamma\) for pairs \((i,k)\) for a communication pattern \(\gamma\)

- **Interpretation**: \((i,k) \leq_\gamma (j,l)\) means:
  - \(j\) has at least the same knowledge in round \(l\) as \(i\) had in round \(k\)
RCA (4/12): Communication patterns (2)

- **Ordering**: \( \leq_{\gamma} \) for pairs \((i,k)\) for a communication pattern \(\gamma\):
  1. **Knowledge is monotonic**: 
     \[(i,k) \leq_{\gamma} (i,l) \text{ if } k \leq l\]
  2. **All knowledge is transferred in messages**: 
     \[\text{if } (i,j,k) \text{ in } \gamma, \text{ then } (i,k-1) \leq_{\gamma} (j,k)\]
  3. **Transitive closure**: 
     \[\text{if } (i,k) \leq_{\gamma} (i',k') \text{ and } (i',k') \leq_{\gamma} (i'',k''), \text{ then } (i,k) \leq_{\gamma} (i'',k'')\]
RCA (5/12): Information level (1)

- The **information level** on pairs \((i,k)\) is defined as:
  1. \(k=0\): \(\text{level}_v(i,0)=0\)
  2. \(k>0\): if there is a \(j \neq i\) such that \((j,0) \preceq_v (i,k)\), then \(\text{level}_v(i,k)=0\)
  3. \(k>0\): let \(l_j = \max\{\text{level}_v(j,k') : (j,k') \preceq_v (i,k)\}\);
     then \(\text{level}_v(i,k) = 1 + \min\{l_j : j \neq i\}\)

- \(l_j\) indicates the **maximum information level** of process \(P_j\) that \(P_i\) knows about
- the information level
  - starts at 0
  - indicates what a process **knows about other processes**
  - is incremented when a process has heard about the previous level of all other processes
RCA (6/12): Information level (2)

- It can be shown that
  - the information levels of different processes in the same round never differ by more than 1
  - if the communication pattern is complete (all triples \((i,j,k)\) appear), then \(\text{level}_\gamma(i,k)=k\) for all \(i\) and \(k\)
RCA (7/12): The algorithm (1)

• Ideas:
  1. Process 1 picks a uniformly random number \( k \) between 1 and \( r \)
  2. **Full information distribution** in every round (on correct links)
  3. Processes maintain information on the initial values \( v \) and the levels of all processes
  4. Messages are of the form \((L,V,k)\), with:
     - \( L \) a vector with the levels as far as known by the sending process
     - \( V \) a vector with the initial values of all processes
     - \( k \) the round number picked by process 1
  5. Initially, the levels and the initial values of other processes, and \( k \) are undefined
RCA (8/12): The algorithm (2)

I. Picking a round number in process 1:

\[
\text{if } ((i=1) \text{ and } (\text{round}=0)) \text{ then key:=random([1,r])}
\]

II. Sending a message in every round in every process

\[
\text{send(L,V, key) to all } j
\]

all locally known information
RCA (9/12): The algorithm (3)

III. Receiving all messages in a round in process $i$

    rounds := rounds+1

    for (j=1 to $i-1$, $i+1$ to $N$) do
        receive($L_j, V_j, k_j$) from j
        if ($k_j \neq$ undefined) then key:=k_j
        if (for all $l$, $V_j(l) \neq$ undefined) then $V_i(l) := V_j(l)$ /* copy initial values */
        if (for all $l$, $L_j(l) > L_i(l)$) then $L_i(l) := L_j(l)$ /* copy levels */
        $L_i(i):=1+\min\{L_i(j): j \neq i\}$ /* compute own level */
        if (rounds=r) then
            if (key $\neq$ undefined) and ($L_i(i) \geq$ key) and ($V_i(j)=1$ for all $j$) then
decision:=1
            else decision:=0

    own information level at least key
all processes started with 1
RCA (10/12): Use of levels and key

- In a sense, processes agree on their levels, i.e., on the actual round they have reached at the end of the algorithm.
- The key chosen by process 1 is a guess of this level.
RCA (11/12): Why do we get $\varepsilon=1/r$?

- **Sketch:**
  - Let $l_i$ be the final level reached by $P_i$ (value of $L_i(i)$) in round $r$.
  - The levels $l_i$ differ by at most 1.
  - If $\text{key} > \max\{l_i\} = a+1$ or at least one process has initial value 0, all processes decide 0 (agreement).
  - If $\text{key} \leq \min\{l_i\} = a$ and all processes have initial value 1, all processes decide 1 (again agreement).
  - So the only case where disagreement is possible, is when $\text{key} = \max\{l_i\} = a+1$.
  - This has probability $1/r$, since $\max\{l_i\}$ is determined by the adversary and $\text{key}$ is uniform on $[1,r]$. 

\[ l_i: a, \ldots, a, a+1, \ldots, a+1 \]
RCA (12/12): We can’t do much better

- It can be shown that:
  - Any r-round algorithm for the randomized coordinated attack problem has probability of disagreement at least equal to $1/(r+1)$
Summary of consensus algorithms

• Synchronous, only stopping failures
• Synchronous, byzantine failures, no authentication, recursive algorithm
• Synchronous, byzantine failures, with authentication, iterative formulation
• Synchronous, authenticated broadcast, polynomial message complexity
• Synchronous and asynchronous, randomized algorithm
• Synchronous, randomized coordinated attack, probability of disagreement