Distributed Algorithms

PART I
SYNCHRONOUS SYSTEMS

Distributed system = graph
- Node = processors (or processes)
- Edge = channel

Directed graph $G=(V,E)$
- $n = |V|$, number of nodes in the graph
- For $i \in V$
  - Out-nbrs, set of nodes $j$ s.t. $(i,j) \in E$
  - In-nbrs, set of nodes $j$ s.t. $(j,i) \in E$

Process
- For each $i \in V$, we have a process $i$
  - States, a set (not necessarily finite) of states
  - Start, a non-empty subset of states
  - Msgs, a message generation function
  - Trans, a state transition function

Message generation
- function : states x out-nbrs -> $M \cup \{\bot\}$
- $M$ is the set of all possible messages

Transition function
- function : states x Vector($M \cup \{\bot\}$) x In-nbrs -> states

Channel and executions
- For each $(i,j) \in E$, we have a channel$(i,j)$
  - It is just a location that, at any time, can hold at most one message from $M$ or can be empty

Execution
- All processes in arbitrary start states
- Rounds: all processes in lock-step repeatedly do
  - Apply the message generation function and put the outgoing messages in the appropriate channel
  - Apply the state transition function to the current state and the vector of incoming messages; remove messages from the channel

Halting
- When the execution halts?
  - There isn’t a specific mechanism

  - No messages are generated and the state transition is a self-loop

  For those accustomed with automata
  - Halting states are not accepting states
  - Halting states serve only to halt the process
  - What is computed must be determined in other ways (e.g., value of variable)

Failures
- Process failures
  - Stopping, at any time
    - Even somewhere in the middle of messages send step
      - Some messages are sent, some are not sent
  - Byzantine failures
    - Arbitrary behavior
    - Malicious

- Channel failures
  - Loss of messages
    - The channel does not record the messages
• State variables
  – Some variables are designated as “input”
  – Some are designated as “output”

• Different input states allow to have different inputs
  – This is why we have multiple start states

• Output variables are write-once variables
  – That is, it is not possible to change the output

• Overall states
  – System state = state of all processes
  – Channels state = state (content) of each channel

• Formally an execution is a sequence
  \[ C_0, M_1, N_1, C_1, M_2, N_2, C_2, \ldots \]
  where
  \[ C_r \text{ are system states} \]
  \[ M_r, N_r \text{ are messages sent and received} \]
  – might be different because of channel failures
  \[ r \text{ is the round} \]

• Indistinguishable executions
  – Two executions \( \alpha \) and \( \alpha' \) can be indistinguishable to process \( i \)
    • Process \( i \) has the same sequence of states, the same sequence of incoming and outgoing messages
    \[ \alpha \approx_i \alpha' \]

• If two executions \( \alpha \) and \( \alpha' \) are indistinguishable to \( p_i \)
  – Then \( p_i \) makes same sequence of steps
    • same decisions (output) in \( \alpha \) and \( \alpha' \)

• Invariant assertions
  – A property of the system state true in all executions
  – Round numbers can be used in the assertions
    • Can be proven by induction
      – True for round \( r=0 \)
      – Assume true for round \( r \), prove for round \( r+1 \)

• Simulations
  – Show that an algorithm A “implements” an algorithm B (same input/output relation)
    – Simulation relation

• Time complexity
  – Number of rounds

• Communication complexity
  – Number of messages sent
  – Sometimes, number of bits sent

**EADEAR ELECTION IN A RING**
Leader election in a ring

- Leader election
  - One processor eventually outputs “I am the leader”
  - Might also require other processors to output “I am not the leader”

- $G$ is a ring of $n$ processors
  - Numbered 1 through $n$: $p_1, p_2, \ldots, p_n$
  - Ring can be unidirectional or bidirectional
  - $n$ can be known or unknown to the processors
  - Processors can be either totally identical, or can be distinguished by a unique identifier

LCR algorithm

- $LCR$: LeLann, Chang, Roberts [1979]
- Assumes
  - Unidirectional communication
  - Processors have UIDs
  - Only leader performs an output
  - Size $n$ unknown

Messages $M = \text{UIDs}$

- states:
  - $uid \in \text{UIDs}$
  - $send \in \text{UIDs} \cup \{\bot\}$
  - $status \in \{\bot, \text{leader}\}$

Message generation function

- $msgs_i$: send value of $send$ to process $i+1$ (mod $n$)

Transition function $trans_i$:

\[
\text{if incoming message is } v, \text{ a UID, then} \quad \text{case}
\]

\[
\begin{align*}
  &v > uid: \quad \text{send } := v \\
  &v = uid: \quad status = \text{leader} \\
  &v < uid: \quad \text{do nothing}
\end{align*}
\]

LCR algorithm: execution

<table>
<thead>
<tr>
<th>UID</th>
<th>Send</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>578</td>
<td>578</td>
<td>I</td>
</tr>
<tr>
<td>287</td>
<td>287</td>
<td>I</td>
</tr>
<tr>
<td>399</td>
<td>399</td>
<td>I</td>
</tr>
<tr>
<td>156</td>
<td>156</td>
<td>I</td>
</tr>
<tr>
<td>899</td>
<td>899</td>
<td>I</td>
</tr>
<tr>
<td>626</td>
<td>626</td>
<td>I</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UID</th>
<th>Send</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>578</td>
<td>578</td>
<td>I</td>
</tr>
<tr>
<td>287</td>
<td>578</td>
<td>I</td>
</tr>
<tr>
<td>399</td>
<td>399</td>
<td>I</td>
</tr>
<tr>
<td>156</td>
<td>899</td>
<td>I</td>
</tr>
<tr>
<td>899</td>
<td>899</td>
<td>I</td>
</tr>
<tr>
<td>626</td>
<td>626</td>
<td>I</td>
</tr>
</tbody>
</table>
\section*{LCR algorithm: execution}

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{lcra1}
\caption{LCR algorithm: execution}
\end{figure}

\section*{LCR algorithm: final state}

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{lcra2}
\caption{LCR algorithm: final state}
\end{figure}

\section*{Formal proof}

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{lcra3}
\caption{Formal proof}
\end{figure}

\section*{Formal proof}

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{lcra4}
\caption{Formal proof}
\end{figure}

\begin{Lemma}
Let $i_{\text{max}}$ be the index of leader
\begin{itemize}
\item $uid_{\text{max}}$ be UID of process $i_{\text{max}}$
\end{itemize}
\end{Lemma}

\begin{Invar}
for $0 \leq n-1$
\begin{itemize}
\item after $r$ rounds, $send_{i_{\text{max}}} = uid_{\text{max}}$
\end{itemize}
\end{Invar}

\begin{Proof}
base: $r=0$: $send_{i_{\text{max}}}=uid_{\text{max}}$ true (initial values)
- induction: assume true for $r$, prove for $r+1$:
  - true for $r$, means $send_{i_{\text{max}}}=uid_{\text{max}}$
  - this implies that process $i_{\text{max}}+r$ sends $uid_{\text{max}}$ in
    round $r$ and thus process $i_{\text{max}}+r+1$ receives $uid_{\text{max}}$
  - By the transition function process $i_{\text{max}}+r+1$ updates
    its $send$ variable to $uid_{\text{max}}$
\end{Proof}
Lemma 2: No process other than $i_{\text{max}}$ outputs “leader”.

- Define $[ij] = \{i,i+1,...,j-1\}$
  - start at $i$ and move along ring direction to $j$
- Invariant 2: For any $ij$ after any $r$ rounds, if $i \neq i_{\text{max}}$ and $j \in [i_{\text{max}}]$ then send $j \neq \text{uid}_i$.

Informally: no uid variable “can get through” the processor with max uid

Proof:

(base) $r=0$, send $i-1$ = uid, assertion true because UIDs are unique ($\text{uid}_i \neq \text{uid}_{i-1}$).

(Induction) Assume true for $r$. Prove for $r+1$.

- fix $ij$ (notice that in $i_{\text{max}}$)
  - true for $i$ means, send $i$ = uid; for all $k \leq i_{\text{max}}$ and thus also for $k+1$ (unless $j = i_{\text{max}}$ in which case we easily have send $j = \text{uid}_{i_{\text{max}}} = \text{uid}_{j}$)
  - So send $j \neq \text{uid}_i$.
  - Since send at end of round $r+1$ can be equal to uid, only if process $j$ receives uid, and any $\geq \text{uid}_i$ (and $\text{uid}_i > \text{uid}_j$). But this is impossible because at the end of round $r$ send $j \neq \text{uid}_i$.

Theorem: LCR solves the leader election problem

- The algorithm we have seen elects a leader
  - but only outputs “leader” for the leader
  - no output for other processes

- We can transform it into an algorithm that
  - outputs “leader” for the leader
  - outputs “non-leader” for other processes

Exercise
- design such a transformation
HS algorithm

- HS: Hirschberg, Sinclair [1980]

  - Same setting as before with bidirectional communication
    - better communication complexity: $O(n \log n)$

- Phases $k=0,1,2,...$
  - Each process sends out its UID (probe)
  - in both directions
  - the probe is intended to travel at a distance $2^k$ and then come back
  - other processors in the path let the probe pass only if their UID is smaller

**Phase 0: distance $2^0=1$**

**Phase 1: distance $2^1=2$**

**Phase 2: distance $2^2=4$**

**Phase 3: distance $2^3=8$**

**Message complexity analysis**

- Messages sent in first Phase ...
  - at most 4 messages sent per processors
  - $n$ processors send messages

- In generic phase $k$
  - At most $4 \times 2^k$ messages per processor
  - How many processors send messages?
    - In phase $k$, a processor send messages if it is not “defeated” in phase $k-1$
    - Within any group of $2^{k-1}+1$ only one goes to the next phase
    - Hence at most $\lfloor n/(2^{k-1}+1) \rfloor$
  - $4 \times 2^k \times \lfloor n/(2^{k-1}+1) \rfloor \leq 8n$
• How many phases?
  – 1+[log n]

• Total complexity: $8n(1+\log n) = O(n \log n)$

• Time complexity
  – phase $k$ takes $2^k = 2^{k+1}$ rounds
  – last phase takes $n$ rounds
  – $2^{k+1}+2^k + \ldots + 2^k + n = O(n)$

• Exercise: write the algorithm
  – states, start, msg, trans,

Complexity analysis

Breaking symmetry

• Leader election is a form of “symmetry breaking”
  – UIDs allow us to distinguish processors
  – Without UIDs?

**Theorem:** Let $A$ be a bidirectional ring of size $n$. If all processors of $A$ are identical then $A$ cannot solve the leader election problem

**Proof (informal):** all processors are identical, they all do the same (either all leaders, or no leaders).

Comparison based

• Comparison of UIDs allow us to break symmetry

• Comparison-based algorithms
  – only perform comparison of UIDs: $\leq, >, =$
  – clearly, UIDs can also be stored, copied, sent

• Using comparison-based algorithms we still have some “symmetry”
  – Key idea: $3 < 7$ is the same as $5 < 7$ ... in this situation $5$ and $3$ are “equivalent”

• Can use to prove a lower bound on messages

Comparison based

• Order equivalence
  – $U = (u_1, u_2, \ldots, u_k)$ and $V = (v_1, v_2, \ldots, v_k)$
  – are order equivalent if $\forall 1 \leq i \leq k$: $u_i \leq v_j$ iff $v_i \leq v_j$

• Examples: $(5,3,7,0)$, $(4,2,6,1)$ and $(5,3,6,1)$

• Some definition
  – active round: at least one message is sent
  – $k$-neighborhood of $i$: $2k+1$ processors around $i$ (with $i$ in the middle), $k < n/2$

Comparison based

**Lemma 1:** Let $A$ be a comparison-based algorithm. Size of ring is $n$, let $k = 0 \leq k < n/2$. Let $i/j$ be two processes with order-equivalent sequences of UIDs in their $k$-neighborhoods.

Then at any time after at most $k$ active rounds, $i$ and $j$ are in corresponding states with respect to UID sequences in their $k$-neighborhoods.

Example: $3$-neighborhood of $i$: $(1,6,3,8,4,10,7)$
$3$-neighborhood of $j$: $(4,10,7,12,9,13,11)$. The lemma says that $i$ and $j$ are in corresponding states as long as no more than $3$ active rounds have occurred.
Distributed Algorithms

Theorem 1

Case when

We omit the proof.

Inductive step: Assume true for \( r'-1 \). Fix \( r \) for which \( i \) and \( j \) have order-equivalent sequences of UIDs in their \( k \)-neighborhood, and suppose that the first \( r \) rounds include at most \( k \) active rounds.

Assume \( i \) or \( j \) receive a msg at round \( r \). Round \( r \) is active. Hence first \( r-1 \) rounds include at least \( k-1 \) active rounds.

Proof (ctnd):

- \( i-1 \) sends msg to \( i \), \( i+1 \) does not.
- \( i-1 \) and \( j-1 \) are in corresponding states after \( r-1 \) rounds. Hence \( j-1 \) also sends a msg to \( j \) at round \( r \).
- Similarly, \( j+1 \) does not send a msg to \( j \) at round \( r \).
- The msg sent by \( j+1 \) corresponds to the msg sent by \( i+1 \) with respect to the \( (k-1) \)-neighborhoods of \( i-1 \) and \( j-1 \), and consequently also with respect to the \( k \)-neighborhoods of \( i \) and \( j \).
- Since \( i \) and \( j \) are in corresponding states after \( r-1 \) rounds and receive corresponding messages, they remain in corresponding states.
- Other cases are similar.

What the lemma says?

- High order-equivalence of UIDs require many active rounds to break symmetry.

To prove the lower bound we need to find rings with many order-equivalent neighborhoods:

- Fix \( c, 0 \leq c \leq 1 \)
- A ring \( R \) is \( c \)-symmetric if for every \( \lambda, n^\lambda \leq \lambda \leq n \), and for every segment \( S \) of length \( \lambda \) there are at least \( \lceil cn/\lambda \rceil \) segments in \( R \) that are order equivalent to \( S \), counting \( S \) itself.

Theorem 1: There exists a constant \( c \) such that for all \( n \), there is a \( c \)-symmetric ring of size \( n \).

- We omit the proof.

However we show \( \frac{1}{2} \)-symmetric rings for the case when \( n \) is a power of 2

Bit-reversal rings

- UID of \( p \) is the inverse of binary representation of \( i \)

- \( i = 0, 1, 2, \ldots, n-1 \)
Lemma 2: Let A be a comparison-based leader election algorithm, for a c-symmetric ring of size n. Let k integer s.t. \( n^2 \leq 2k+1 \leq n \) and \( \lfloor cn/(2k+1) \rfloor \geq 2 \). Then A has more than \( k \) active rounds.

**Proof:** By contradiction. Assume that A elects leader, say i, in at most \( k \) active rounds. Let S be the \( k \)-neighborhood of i. S has length \( 2k+1 \). Choose \( \lambda = 2k+1 \).

Since ring is \( c \)-symmetric, there are at least \( \lfloor cn/(2k+1) \rfloor \geq 2 \) segments order equivalent to S. Thus, at least 1 other segment order equivalent to S. Let \( i \) be the midpoint of such a segment. Process \( j \) and \( j \), by Lemma 1, remain in corresponding states. Since \( i \) becomes leader, also \( j \) becomes leader.

**Exercise:** prove the above

**Theorem 2:** For any comparison-based algorithm there is an execution in which \( \Omega(n \log n) \) messages are sent to elect a leader.

**Proof:** Fix \( c \) satisfying Theorem 1 and construct a \( c \)-symmetric ring \( R \) of size \( n \).

Define \( k = \lceil (cn-2)/4 \rceil \). Then \( n^2 \leq 2k+1 \leq n \) and \( \lfloor cn/(2k+1) \rfloor \geq 2 \) (for \( n \) large enough).

By Lemma 2 there are at least \( k+1 \) active rounds. Consider \( n^k \) active round, \( n^2 \leq r \leq k+1 \). Round is active, some process i sends a message. Let S be the \( (r-1) \)-neighborhood of i. R is \( c \)-symmetric, at least \( \lfloor cn/(2r-1) \rfloor \) segments of R are order equivalent to S. By Lemma 1, just before round \( r \), the midpoints of those segments are in corresponding states. Thus they all send messages.

**Proof (cntd):**
Let \( r_1 = \lfloor n^2 \rfloor + 1 \) and \( r_2 = k+1 = \lfloor cn-2)/4 \rfloor + 1 \).

Total number of messages:
\[
\sum_{p_i \in R} [cn/(2r-1)] \geq \sum_{r_1 \leq r \leq r_2} [cn/(2r-1)] = \Omega(n)\sum_{r_1 \leq r \leq r_2} 1/r - \Omega(n) \]

\( \Omega(n) = \Omega(n \log(n)) \)

**Other algorithms**
- UIDs
- Comparisons of UIDs with round numbers

**Assumes** n known

**Use absence of msgs to convey information**

**Phases:** 1,2,3...
- Each of n rounds
- In phase k only UID=k can travel
- Phase k consists of rounds k=1...n
- If a process has not yet received any msg by phase \( (k-1)n+1 \) and its own UID is k, sends a msg ("k leader") around the ring
TimeSlice algorithm

- Need comparisons between UIDs and round numbers
  - phases of \( n \) rounds
  - in phase \( k \) only the UID = \( k \) can travel around the ring
- \( O(n) \) msgs
  - actually, exactly \( n \) msgs
- \( O(m \times n) \) rounds
  - where \( m = \text{min UIDs} \)
- Exercise: design an algorithm for \( n \) unknown
  - Hint: Each process sends its own UID. Messages travel at different speeds; exploit overtaking.

Leader election in general graphs

- We assume
  - unique identifiers (from totally ordered space)
  - a connected directed graph \( G = (V, E) \)
- FloodMax algorithm (informal)
  - Process keep track of max UID seen
  - At each round, every process \( p_i \)
    - sends \( \text{max-uid} \),
    - updates \( \text{max-uid} \) according to received messages
  - After \( \text{diam} \) rounds the leader is elected
    - the leader is \( \text{max-uid} \) (which is the same for all \( i \))

Synchronous algorithms

BFS, Shortest Path (,MST)

- Directed graph \( G = (V, E) \) and a source node \( s \)
  - processes have UIDs
  - network topology unknown
  - \( n \) unknown
- Visit the graph breadth-first
  - breadth-first directed spanning tree of \( G \)
    - spanning tree rooted at source node \( s \)
    - nodes at distance \( d \) from \( s \), appear at level \( d \) of the tree
- Broadcast
  - convenient structure to convey broadcast messages

BFS

- The algorithm produces a BFS tree
- Formally we could write an invariant
  - after every round \( r \), every node at distance \( d \) from \( s \), \( 1 \leq d \leq r \), has its parent pointer defined; moreover such a pointer points to a node at distance \( d-1 \) from \( s \).
- Complexity
  - \( O(\text{diam}) \) time
  - \( O(|E|) \) messages
  - exact number can be reduced by not sending search msgs to nodes from which such a message has arrived - still \( O(|E|) \)

BFS
BFS and broadcast

• Can broadcast a message during construction
  – message sent by the source
  – piggybacked on search messages

• Broadcast after construction
  – need to know “child” pointers
  – the algorithm only constructs “parent” pointers

• If G is bidirectional
  – easy to let the parent know its children
    • enough that the children send a message

BFS and child pointers

• What if G is directed?
  – harder

• Once a node u knows its parent
  – uses a new instance of the algorithm
  – source is u, destination is sender of search message
  – response to search message
    • “parent” or “not parent”
    – the sender of search message gets the message (it’s a broadcast)

• Many instances of the algorithm in parallel

Complexity analysis

• How processes know that the tree has been built?

• Convergecast
  – once a node has received
    • response to all its search messages
    • notification of completion from its children
    – sends a notification of completion to its parent

• Analysis:
  – bidirectional: O(diam) time, O(|E|) msgs
  – unidirectional: O(diam^2) time, O(diam^2|E|) msgs

BFS applications

• Broadcast
  – send messages on the tree
    • O(diam) time, O(n) messages

• Global computation
  – convergecast
    • each node sends its own value to the root

• Electing a leader
  – all processes initiate BFS in parallel and broadcast their UID

• Computing the diameter
  – each process i initiates BFS, and with a convergecast computes
    the depth of its tree, depth_i
    – With an extra broadcast on the tree, depth_i is sent to all processes
    – diameter is max of depth_i

Distributed Shortest Path

• Input: directed graph G=(V,E)
  – edge weights: w(i,j), for each (i,j)∈E
  – source node s

• Output: in each node
  – pointer to parent node in shortest path from s to d

• Assume
  – n = |G| is known (or at least an upper bound)
  – both i and j know w(i,j)
Bellman-Ford

- Each node \( u \) keeps track of
  - \( \text{dist}_u \)
    - shortest distance from \( s \) to \( u \), so far
    - initially, \( \text{dist}_s = 0 \) and \( \text{dist}_u = \infty \) for \( u \neq s \)
    - parent
    - pointer to parent on path with overall weight = \( \text{dist}_u \)

- At each round
  - each process sends its \( \text{dist} \) to all its neighbors
  - When \( i \) receives \( \text{dist}_j \) from \( j \)
    - "relaxation" step
    - \( \text{dist}_i = \min (\text{dist}_i, \text{dist}_j + w(i,j)) \)

- After \( n-1 \) rounds shortest paths are found

---

Minimum Spanning Tree

... will see the asynchronous version!

- To explain the asynchronous version we will first see a synchronous version
- Better to do one after another

---

Consensus

- The problem
  - processes have to agree on a value
- Easy to solve if no failures
- So, will consider
  - link failures
  - node failures, stop and byzantine
- Practical applications
  - commit or abort a transaction
  - agree on the estimated of sensors’ readings
  - agree on a failed component

---

Synchronous algorithms

**CONSENSUS**

- Analysis
  - \( O(n) \) time, exactly \( n-1 \) rounds
  - \( O(n \times |E|) \) messages, exactly \( (n-1) \times |E| \)

- Exercise
  - write an invariant assertion that can be used to prove formally the correctness of the algorithm
    - hint use round number \( r \)
    - up to round \( r \) shortest paths with at most \( r \) edges are known

---

Coordinated attack problem

- Several generals plan a coordinated attack
  - Each has its own opinion about attack/not attack
- Generals (and their army) located in different places
  - communication only via messengers
    - messengers can be captured, so messages can be lost
- Each general has to decide: attack/not attack
  - Success: only if all decide to attack
Commit problem

• A distributed transaction is being performed, each process involved has to decide
  — commit/abort the transaction
• Each process wishes to
  — commit: if all local computations succeeded
  — abort: otherwise
• The system will keep a “sound” state only if all processes make the same decision
• Communication: messages can be lost

Consensus problem

• Each process starts with an input value in \{0,1\}
• Eventually each process outputs 0 or 1
• Communication: messages can be lost
• Agreement: All outputs are equal
• Validity:
  — if all processes start with 0, then 0 is the only possible output
  — if all processes start with 1 and all messages are delivered, then 1 is the only possible output
• Termination: All process eventually decide

Consensus with link failures

**Theorem:** There is no algorithm that solves consensus if messages can get lost.

**Proof:** Let \( G \) be a graph with 2 nodes connected by an edge. Assume by contradiction that exists A that solves problem in \( G \).
Let \( \alpha \) be the execution where both start with 1 and all messages are delivered.
By the validity condition, both decide 1. Let \( r \) be the round by which both decisions are made.
W.l.o.g., assume in \( \alpha \) each process sends a msg in each round (can add dummymsgs).

Proof (ctnd):

ContradicMon. \( \square \)
• What does it mean?
  – there is little we can do in the face of unreliable communication

• Real systems
  – not all messages get lost!

• We can relax the problem requirements or strengthen the model
  – have an upper bound on the number of failures
  – use randomization

• For any particular adversary
  – any fixed set of initial values
  – any fixed pattern of failures

• Any particular set of random choices made by the processes
  – determines a unique execution

• Random choices \(\Rightarrow\) Probability distribution over set of all executions

• Given a \(\gamma\), we define the information level
  – \(\text{level}(i,k)\), for any process \(i\) and round \(k\):
    • \(k=0\): \(\text{level}(i,k)=0\)
    • \(k=0\) and \(\exists i' s.t. \{i\} \not\subseteq \{i'\} \Rightarrow \text{level}(i,k)=0\)
    • \(k=0\) and \(\forall i', j, (i,j) \not\subseteq (i',j)\):
      \(\forall i', j, \lambda \equiv \max\{\text{level}(i',k): (i',k) \subseteq (i,k)\}\) and
      \(\text{level}(i,k) = 1+ \min\{\lambda, \gamma\}\)

• Information level (informal)
  – what processes known about other processes
    • 0 at the beginning
      – 1 when it hears from all other processes
    • increases to \(\lambda+1\) when process knows that all other processes have reached level \(\lambda\)

• Agreement: For every adversary \(B\),
  \(\Pr[(\text{different outputs})] \leq \epsilon\)

• Adversary \(B\) chooses input values and failures

• Communication pattern
  – \(\gamma \subseteq (i,j,k): (i,j) \in E\) and \(k\) integer, \(k \geq 1\)

• A communication pattern is good if
  – \(\forall (i,j,k) \in \gamma, k \leq r\)

• Adversary can choose
  – assignment of input values
  – any good communication pattern
\begin{itemize}
  \item RandomAttack algorithm
    \begin{itemize}
      \item Each process \( i \) keeps track of its own information level, \( \text{level}(i,k) \).
      \item Process 1, at the very beginning, chooses a random integer value \( k \in [1,r] \).
      \item This value is piggybacked on all msgs.
      \item The initial values and the levels known by process \( i \) are also piggybacked on all msgs sent by \( i \).
      \item That's all the information known by \( i \).
      \item Processes send msgs at every round.
      \item At round \( r \) a decision is made.
      \begin{itemize}
        \item \( 1 \): if \( \text{level}(i,k) \geq k \) and all initial values are 1.
        \item 0: otherwise.
      \end{itemize}
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item Lemma 1: For any good communication pattern \( \gamma \), any \( k, 0 \leq k \leq r \), and any \( i \) and \( j \):
    \[ \text{level}(i,k) - \text{level}(j,k) \leq 1. \]
  \item Proof: Left as exercise.
\end{itemize}

\begin{itemize}
  \item Lemma 2: If \( \gamma \) is the "complete" communication pattern, then for all \( i \) and all \( k \), \( \text{level}(i,k) = k \).
  \item Proof: Left as exercise.
\end{itemize}
Distributed Algorithms

Problem definition for stop failures:

- **Agreement**: No two different outputs.
- **Validity**: if all processes start with the same initial value \( v \in V \), then \( v \) is the only possible output.
- **Termination**: All nonfaulty processes decide.

for Byzantine failures:

- **Validity**: If all nonfaulty processes start with same \( v \in V \), then \( v \) is the only possible output for a nonfaulty process.

**Proof (ctd):** Termination is obvious.

**Validity**: If all start with 0, 0 is the only possible decision. If all start with 1 and all msgs delivered. By Lemma 2 we have \( \text{level}(i,r) = r \), and thus, by the code, in round \( r \), level(i)_r = r.

Since level(i)_r \( \geq 1 \), it follows that key = \( \bot \) and \( \text{val}(i, \bot) \). Since key is always \( \leq r \), 1 is the only possible decision.

**Agreement**: \( W \), let key = level(i)_r at round \( r \). By Lemma 1, all \( \lambda \), are within 1 of each other.

If (chosen key > max(\( \lambda \))) or (some input 0) then all processes decide 0.

if (key \( \leq \min(\lambda) \)) and all inputs 1) all processes decide 1.

Disagreement only when key = max(\( \lambda \)). Probability of this event is \( 1/r \). Indeed max(\( \lambda \)) is chosen by the adversary, while key is chose uniformly at random in \([1,r] \).

**Consensus with process failures**

- Now we assume
  - links do not fail
  - processes may fail
  - stop failures
  - byzantine failures

- Number of failures upper bounded
  - \( f \) is maximum number of failures

- Input values are taken from a fixed set \( V \)

**FloodSet**

- Assume a complete graph

- **FloodSet**
  - Each process \( i \) maintains a variable \( W_i \) that contains a subset of \( V \).
  - Initially \( W_i = \{ i \} \)
  - For each of \( f+1 \) rounds
    - each process \( i \) broadcasts \( W_i \)
    - adds all the elements of the received sets to \( W_i \)
  - After \( f+1 \) rounds processes decides
    - if \( W_i = \{ v \} \), a singleton set, decide \( v \)
    - otherwise decide on a default value \( v_d \)

**Lemma 1**: If no process fails during a particular round \( r \), \( 1 \leq r \leq f+1 \), then \( W_i(r) = W_j(r) \forall i, j \) active after \( r \) rounds.

**Theorem**: FloodSet solves consensus

**Proof**: Termination. All decide at round \( f+1 \).

Validity. If all initial values are \( v \), then \( v \) is the only value sent in msgs. Each \( W_i(f+1) \) is non-empty (initially contains \( v \)). Hence \( W_i(f+1) = \{ v \} \).

Agreement. By Lemma 3, \( W_i(f+1) = W_j(f+1) \).

processes \( i \) and \( j \), decide the same value.
- Complexity analysis
  - time: \( f+1 \) rounds
  - msgs: \( \Omega((f+1)n^3) \)
    - actually since msgs are of size \( O(n) \), we should consider this size. Let \( k \) bits needed for one value. each message is \( O(nk) \).
    - msgs: \( O((f+1)n^2) \)

- Exercise: think about how to reduce msgs
  - Hint: Think about what the processes need to know about \( W \) to make a decision ...

- EIG algorithms
  - processes relay initial values for several rounds
  - record values and the "route" followed by values
    - each value is received along multiple paths
  - At the end, a commonly agreed-upon rule is used do decide a value as a function of the recorded values
  - EIG algorithms are costly
    - not worth for stopping failures
    - will be useful for Byzantine failures
      - stopping case is a simple introduction

- Exponential Information Gathering

- EIG stop algorithm
  - Each process maintains a copy of the EIG tree
  - Computation proceeds for \( f+1 \) rounds
    - processes relay initial values on all possible paths
  - Processes store values in nodes of the EIG tree
    - The root of process \( i \), gets \( v \)
    - Node \( j \), at level 1, gets the value that \( j \) tells \( i \)
    - If at level \( k \), node with label \( i_1,i_2,...,i_k \), gets \( v \) means
      - \( i_1 \) has told \( i_2 \) at round \( k \) that \( i_1 \) has told \( i_2 \) at round \( k-1 \) that \( i_2 \) has told \( i_3 \) at round \( k-2 \) ... that \( i_k \) has told \( i_{k-1} \) at round 1 that \( i_1 \) initial value is \( v \).
    - A node can be empty
      - the communication chain has been broken by a fault

- EIG tree

- FloodSet

- EIG tree
**EIG stop algorithm**

- Let $W_i$ be the set of all values in the EIG tree for process $i$.

**Lemma:** If process $i$ and $j$ are both non-faulty then $W_i = W_j$.

- Decision: if $W = \{v\}$ is a singleton, decide $v$ otherwise decide $v_0$, a default value.

**Theorem:** EIG Stop solves consensus.

---

**Byzantine failures**

- A faulty node has unrestricted behavior
  - can be malicious
- We still consider complete graphs of $n$ nodes
- Byzantine consensus is more difficult
  - stop failures, can tolerate any number $f$ of faults
  - Byzantine failures: we need $n > 3f$

---

**Execution $\alpha_1$**

- $p_1$ and $p_3$ send their initial value
- $p_2$ (faulty process) tells truth

**Execution $\alpha_2$**

- $p_1$ (faulty process) tells truth
- $p_2$ and $p_3$ send their initial value

---

What is the decision?
Distributed Algorithms

Lemma 2: After f+1 rounds, if x is a label ending with the index of a non-faulty process then there is a value v s.t. $\text{val}(x) = \text{newval}(x) = v$ for all non-faulty processes i.

Proof: If $k \notin \{i,j\}$ then since k is non-faulty it sends the same message to i and j. The same holds if $k \in \{i,j\}$ (processes send msg's to themselves).

Lemma 1: After f+1 rounds, if $i,j,k$ are all non-faulty, with $i \neq j$, then $\text{val}(x) = \text{val}(x)$ for every label $x$ ending in k.

Proof: If $k \notin \{i,j\}$ then since k is non-faulty it sends the same message to i and j. The same holds if $k \in \{i,j\}$ (processes send msg's to themselves).
Lemma 3 (validity): If all non-faulty processes begin with same \( v \), then \( v \) is the decision.

**Proof:** All non-faulty processes send \( v \) in first round. Hence \( \text{val}(i) = v \) for non-faulty \( i,j \).

Lemma 2 implies that \( \text{newval}(i,j) = v \) for non-faulty \( i,j \).

The majority rule used for the decision implies that \( \text{newval}(\lambda,i) = v \) for non-faulty \( i \).

- *Path covering:* set of nodes \( C \) s.t. every path from root to leaf has at least one node in \( C \)
- *Common node:* in any execution \( \alpha \), after \( f+1 \) rounds, all non-faulty have same \( \text{newval}(x) \).

Lemma 4: After \( f+1 \) rounds, in any execution \( \alpha \), there exists a path covering with all nodes common in \( \alpha \).

**Proof:** Let \( C \) be the nodes with labels \( x_j \), where \( j \) is non-faulty.

By Lemma 2, all these nodes have \( \text{newval}(x) = v \) for non-faulty \( i \).

Hence nodes of \( C \) are common.
Moreover \( C \) is a path covering by construction of the EIG tree.

Lemma 5: The EIG root node \( \lambda \) is common.

**Proof:** Left as exercise.

Termination is obvious; Lemmas 3 and 5 imply:

**Theorem:** The EIG Byzantine algorithm solves the Byzantine consensus problem for \( n > 3f \).

Analysis:
- \( f+1 \) rounds
- \( O((f+1)n^2) \) msgs, \( O(n^{f+1}b) \) bits of communication.

To \( p_2 \) and \( p_3 \), \( \alpha - \alpha_1 \) in \( A \), where \( p_3 \) is faulty.

By validity in \( A \), \( p_2 \) and \( p_3 \) decide \( 0 \) in \( \alpha_1 \).

Hence we conclude that they decide \( 0 \) also in \( \alpha \).
To $p_i$ and $p'_i$: $\alpha = \alpha_2$ in $A$, where $p_i$ is faulty. By agreement in $A$, $p_i$ and $p'_i$ decide same value in $\alpha$. They decide same value also in $\alpha$.

**Theorem:** There is no algorithm for Byzantine consensus if $3 \leq n \leq 3f$.

**Proof:** By contradiction assume $A$ exists; we transform it into $B$ to solve problem for $n=3$, $f=1$.

Partition processes of $A$ into 3 subsets $A_1, A_2, A_3$ each of size at most $f$.

Algorithm $B$ for $p_1, p_2, p_3$:

- each $p_i, i=1, 2, 3$, keeps track of the states of all processes in $A_i$ and simulates those processes. Assigns its own initial value to all processes of $A_i$ any msgs sent/received by any process in $A_i$ are sent/received by $p_i$.

If any simulated process of $A_i$ decides a value, $p_i$ decides on that value (if more than 1, $p_i$ can choose any)

**Claim:** $B$ solves Byzantine consensus.

- Fix any execution $\alpha$ of $B$, with at most one failure, say $p'_1$.
- Let $\alpha'$ be execution of $A$ obtained with the simulation.

There are at most $f$ failures in $\alpha'$ [set $A_j$] and thus consensus is solved in $\alpha'$.

**Termination ($B$):** Consider $p_1 (p_3)$ in $\alpha$. Let $j$ be any process of $A_2$ ($A_3$) $j$ decides in $\alpha'$ and by the simulation $p_2$ decides in $\alpha$.

**Validity ($B$):** If all nonfaulty in $\alpha'$ start with $v$, then all nonfaulty in $\alpha'$ start with $v$. Hence they decide $v$ in $\alpha'$ and by the simulation decide $v$ in $\alpha$.

**Agreement ($B$):** By the simulation $p_1 (p_3)$ decide a value decide by any process in $A_2 (A_3)$. But in $\alpha'$ all processes in $A_1$ and $A_j$ decide same value.

**Theorem:** Byzantine consensus cannot be solved in less than $f+1$ rounds.

**Proof:** Omitted.

- So far
  - complete graph with $n$ node
    - can simulate global communication (at the expense of communication and time complexity)
    - but need to take into account network partitions in case of faults

**Theorem:** Byzantine consensus can be solved in general graph $G$ if and only if

1. $n > 3f$
2. $conn(G) > 2f$

**Theorem:** Stop-failure consensus can be solved in general graph $G$ if and only if

1. $conn(G) > f$