Distributed Algorithms
The Synchronous Model and Leader Election

Maruth Goyal

UT Austin

Spring 2021

Table of Contents

Synchronous Model

Model of distributed computing in which all participants are "synchronized".

1. Participants (referred to as processes) represented as nodes in a di-graph $G = (V, E)$

Synchronous Model

Model of distributed computing in which all participants are "synchronized".

1. Participants (referred to as processes) represented as nodes in a di-graph $G = (V, E)$
2. Directed edges represent channels of communication between processes

Synchronous Model

Model of distributed computing in which all participants are "synchronized".

1. Participants (referred to as processes) represented as nodes in a di-graph $G = (V, E)$
2. Directed edges represent channels of communication between processes
3. Execution is organized in **rounds**. At the end of every round processes may send messages to neighbors, and subsequently receive messages before beginning the next round.

Synchronous Model Formalism

Definition: Process in Synchronous Model

For each vertex $v \in V$, there is an associated process with the following:

1. Σ_v : set of possible states

Synchronous Model Formalism

Definition: Process in Synchronous Model

For each vertex $v \in V$, there is an associated process with the following:

1. Σ_v : set of possible states
2. $\mathcal{S}_v \subseteq \Sigma_v$: set of possible starting states

Synchronous Model Formalism

Definition: Process in Synchronous Model

For each vertex $v \in V$, there is an associated process with the following:

1. Σ_v : set of possible states
2. $\mathcal{S}_v \subseteq \Sigma_v$: set of possible starting states
3. $\mu_v : \Sigma_v \times \mathsf{out}(v) \to \mathcal{M} \cup \{\text{null}\}$: The message generating function

Synchronous Model Formalism

Definition: Process in Synchronous Model

For each vertex $v \in V$, there is an associated process with the following:

1. Σ_v : set of possible states
2. $\mathcal{S}_v \subseteq \Sigma_v$: set of possible starting states
3. $\mu_v : \Sigma_v \times \mathrm{out}(v) \to \mathcal{M} \cup \{\mathrm{null}\}$: The message generating function
4. $\tau_v : \Sigma_v \times (\mathcal{M} \cup \{\mathrm{null}\})^{\mathrm{in}(v)} \to \Sigma_v$: The state transition function

Synchronous Model Formalism

Definition: Process in Synchronous Model

For each vertex $v \in V$, there is an associated process with the following:

1. Σ_v : set of possible states
2. $\mathcal{S}_v \subseteq \Sigma_v$: set of possible starting states
3. $\mu_v : \Sigma_v \times \mathrm{out}(v) \rightarrow \mathcal{M} \cup \{\mathrm{null}\}$: The message generating function
4. $\tau_v : \Sigma_v \times (\mathcal{M} \cup \{\mathrm{null}\})^{\mathrm{in}(v)} \rightarrow \Sigma_v$: The state transition function

Similar in spirit to Turing Machines, with addition of communication.

Synchronous Model Formalism

Definition: Execution

Each vertex $v \in V$ starts in some state $\sigma_v^0 \in \mathcal{S}_v$. At each "round" r,

1. Apply μ_v to σ_v^r to generate outgoing messages for each neighbor

Synchronous Model Formalism

Definition: Execution

Each vertex $v \in V$ starts in some state $\sigma_v^0 \in \mathcal{S}_v$. At each "round" r,

1. Apply μ_v to σ_v^r to generate outgoing messages for each neighbor
2. Place each message in appropriate channels

Synchronous Model Formalism

Definition: Execution

Each vertex $v \in V$ starts in some state $\sigma_v^0 \in \mathcal{S}_v$. At each "round" r,

1. Apply μ_v to σ_v^r to generate outgoing messages for each neighbor
2. Place each message in appropriate channels
3. Apply τ_v to σ_v^r and incoming messages to generate the next state σ_v^{r+1}.

Synchronous Model Formalism

Definition: Execution

Each vertex $v \in V$ starts in some state $\sigma_v^0 \in \mathcal{S}_v$. At each "round" r,

1. Apply μ_v to σ_v^r to generate outgoing messages for each neighbor
2. Place each message in appropriate channels
3. Apply τ_v to σ_v^r and incoming messages to generate the next state σ_v^{r+1}.
4. Remove all messages from channels

Failures

In a real distributed system, many sorts of failures can occur. Thus, imperative to add these failures to our model to make algorithms more practical.

Types of failures:

1. **Link failures**: channels can get corrupted, or simply shut down (processes cannot communicate)

Failures

In a real distributed system, many sorts of failures can occur. Thus, imperative to add these failures to our model to make algorithms more practical.

Types of failures:

1. **Link failures**: channels can get corrupted, or simply shut down (processes cannot communicate)
2. **Byzantine failures**: processes continue executing, but generating transitions and messages in fashion inconsistent with τ_v, μ_v.

Failures

In a real distributed system, many sorts of failures can occur. Thus, imperative to add these failures to our model to make algorithms more practical.

Types of failures:

1. **Link failures**: channels can get corrupted, or simply shut down (processes cannot communicate)
2. **Byzantine failures**: processes continue executing, but generating transitions and messages in fashion inconsistent with τ_v, μ_v.
3. **Process failures**: Process fails in any of the aforementioned steps of execution in each round.

Complexity Measures

Definition: Time Complexity

The time complexity of a synchronous algorithm is measured as the number of **rounds** it takes to execute.

Important to assess speed of algorithm.

Complexity Measures

Definition: Time Complexity

The time complexity of a synchronous algorithm is measured as the number of **rounds** it takes to execute.

Important to assess speed of algorithm.

Definition: Communication Complexity

The communication complexity of a synchronous algorithm is measured as the total bits of information that is sent across channels in the execution of the algorithm.

Motivated by ensuring minimal traffic on network channels, especially in situations where channels may have high congestion, latency, and/or forms of error and loss.

Table of Contents

Problem Statement

1. It is often useful in a distributed system to have a leader who can co-ordinate the actions of all the other nodes.

Problem Statement

1. It is often useful in a distributed system to have a leader who can co-ordinate the actions of all the other nodes.
2. However, if all nodes start in the same state they must talk to each other in order to elect a leader

Problem Statement

1. It is often useful in a distributed system to have a leader who can co-ordinate the actions of all the other nodes.
2. However, if all nodes start in the same state they must talk to each other in order to elect a leader
3. For now we consider the simpler case where the graph $G = (V, E)$ is a ring.

Problem Statement

Leader Election

Given a graph of processes $G = (V, E)$ such that G is a ring, at the end of some finite r rounds it must be the case that precisely one vertex $v \in V$ has updated its state to ``leader''}$ while **all** other nodes have their state as $\text{``notleader''.

Requirements to solve Leader Election

Do we need any assumptions on the processes to be able to solve leader election?

Requirements to solve Leader Election

Do we need any assumptions on the processes to be able to solve leader election? Let's find out

Requirements to solve Leader Election

Do we need any assumptions on the processes to be able to solve leader election? Let's find out

Suppose that all the processes are indistinguishable from each other. i.e., all have same starting state, μ and τ. Moreover, they keep track of identical state.

Requirements to solve Leader Election

Do we need any assumptions on the processes to be able to solve leader election? Let's find out

Suppose that all the processes are indistinguishable from each other. i.e., all have same starting state, μ and τ. Moreover, they keep track of identical state. **can we solve leader election here?**

Requirements to solve Leader Election

Do we need any assumptions on the processes to be able to solve leader election? Let's find out

Suppose that all the processes are indistinguishable from each other. i.e., all have same starting state, μ and τ. Moreover, they keep track of identical state. **can we solve leader election here?** No!

Impossibility Result

Claim

Given a ring $G = (V, E)$ such that all vertices $v \in V$ are indistinguishable in operation, and have the same starting state and tracked state, the leader election problem cannot be solved

Impossibility Result

Claim

Given a ring $G = (V, E)$ such that all vertices $v \in V$ are indistinguishable in operation, and have the same starting state and tracked state, the leader election problem cannot be solved

Proof (informal)

Suppose for contradiction the problem can be solved after $r + 1$ rounds.

Impossibility Result

Claim

Given a ring $G = (V, E)$ such that all vertices $v \in V$ are indistinguishable in operation, and have the same starting state and tracked state, the leader election problem cannot be solved

Proof (informal)

Suppose for contradiction the problem can be solved after $r + 1$ rounds. Then, by assumption, since each node behaves identically and has the same starting state it can be shown by induction that they have the same state at round r.

Impossibility Result

Claim

Given a ring $G = (V, E)$ such that all vertices $v \in V$ are indistinguishable in operation, and have the same starting state and tracked state, the leader election problem cannot be solved

Proof (informal)

Suppose for contradiction the problem can be solved after $r + 1$ rounds. Then, by assumption, since each node behaves identically and has the same starting state it can be shown by induction that they have the same state at round r. But then, they must proceed identically at round $r + 1$ as well. i.e., if any one vertex elects itself leader, then so do all the others. Contradiction.

Breaking Symmetry

The key issue appears to be the complete symmetry of vertices. How do we break it?

Breaking Symmetry

The key issue appears to be the complete symmetry of vertices. How do we break it?

- What if we assign each vertex a unique ID (UID)?

Breaking Symmetry

The key issue appears to be the complete symmetry of vertices. How do we break it?

- What if we assign each vertex a unique ID (UID)?

Claim

Given a ring $G = (V, E)$ such that each vertex $v \in V$ has a unique ID $\mathrm{UID}(v) \in \mathbb{Z}_+$, then the leader election problem can be solved.

Utilizing Broken Symmetry

The unique identifiers break symmetry, but how do we utilize that to solve the problem?

Utilizing Broken Symmetry

The unique identifiers break symmetry, but how do we utilize that to solve the problem?

- Observe that by assuming the IDs lie in $\mathbb{Z}_+$ we have additional structure:

Utilizing Broken Symmetry

The unique identifiers break symmetry, but how do we utilize that to solve the problem?

- Observe that by assuming the IDs lie in $\mathbb{Z}_+$ we have additional structure: order
- Every finite, totally ordered set has a unique maximum/minimum.
- **Idea:** compute a distributed maximum over the UIDs, elect the maximum as leader.

LCR Algorithm

- Vertices compute maximum by just talking to each other and "spreading the word"

LCR Algorithm

- Vertices compute maximum by just talking to each other and "spreading the word"
- Each vertex tracks the maximum UID it has seen so far, beginning with its own.

LCR Algorithm

- Vertices compute maximum by just talking to each other and "spreading the word"
- Each vertex tracks the maximum UID it has seen so far, beginning with its own.
- At every round vertex tells it's (WLOG) left neighbor the maximum UID it has seen so far.

LCR Algorithm

- Vertices compute maximum by just talking to each other and "spreading the word"
- Each vertex tracks the maximum UID it has seen so far, beginning with its own.
- At every round vertex tells it's (WLOG) left neighbor the maximum UID it has seen so far.
- Termination? :

LCR Algorithm

- Vertices compute maximum by just talking to each other and "spreading the word"
- Each vertex tracks the maximum UID it has seen so far, beginning with its own.
- At every round vertex tells it's (WLOG) left neighbor the maximum UID it has seen so far.
- Termination? : when a vertex receives its own UID, elects itself as leader
- Optionally send a message declaring itself as leader

LCR Example

LCR Analysis

Claim

After $O(n)$ rounds, the LCR algorithm terminates with the maximum UID vertex as leader, and no one else.

Proof Sketch

In order for a vertex to receive its own UID, the ID must have travelled across $n - 1$ vertices, and be strictly greater than each of them. Thus, this happens iff the vertex's ID is the maximum. Moreover, this takes n rounds to travel, as desired.

LCR Analysis

Claim

After $O(n)$ rounds, the LCR algorithm terminates with the maximum UID vertex as leader, and no one else.

Proof Sketch

In order for a vertex to receive its own UID, the ID must have travelled across $n-1$ vertices, and be strictly greater than each of them. Thus, this happens iff the vertex's ID is the maximum. Moreover, this takes n rounds to travel, as desired.

Claim

The LCR algorithm requires $O(n^2)$ communication to elect the leader/

Proof

At each round, every vertex sends one message, thus n messages per rounds and n rounds in total, giving us $O(n^2)$.

Can We do Better?

LCR algorithm works, but has $O(n^2)$ communication complexity. Can we do better?

Can We do Better?

LCR algorithm works, but has $O(n^2)$ communication complexity. Can we do better? Hirschberg-Sinclair algorithm $O(n \log n)$

HS Algorithm

- In LCR algorithm, search space shrinks very slowly. Intuitively, to reach $O(n \log n)$ need to be more aggressive.
- Key Idea: Send your value to 2^l neighbors in **both directions** for each "phase" l. Value comes back to you iff you're larger than the $\sim 2^l$ neighbors in at least one direction. Elect self if value comes back in $< 2^{l+1}$ rounds.
- Eliminating all but one in each set of $2^l + 1$ neighbors in each phase. Thus, at most $\frac{n}{2^l+1}$ processes take part in phase $l+1$. Each process accounts for $2 \cdot 2 \cdot \cdot 2^l$ messages, therefore the messages per phase is bounded as
$$4 \cdot 2^{l+1} \cdot \frac{n}{2^l + 1} \leq 8n$$
- Clearly at most $\log n$ phases, thus $O(n \log n)$

Table of Contents

Lower Bounds

Theorem

Let A be a comparison-based algorithm for electing a leader in a ring of size n. Then there is an execution of A which requires $\Omega(n \log n)$ messages to elect a leader.

Proof Sketch: Lower Bounds

Theorem

Let A be a comparison-based algorithm for electing a leader in a ring of size n. Then there is an execution of A which requires $\Omega(n \log n)$ messages to elect a leader.

Sketch of Proof Sketch

1. Design pessimistic/adversarial example of ring
2. Lower bound number of rounds required for this ring
3. Lower bound number of messages sent in rounds
4. Complete argument

c-symmetric Rings

Definition: order equivalence

Two sets of UIDs $U = (u_1, \ldots, u_k)$ and $V = (v_1, \ldots, v_k)$ are said to be order equivalent if for all $1 \leq i, j \leq k$ we have $u_i \leq u_j$ iff $v_i \leq v_j$.

Definition: c-symmetric ring

For $0 \leq c \leq 1$, a ring R is said to be c-symmetric if for every l such that $\sqrt{n} \leq l \leq n$, and every segment S or R of length l there are at least $\lfloor \frac{cn}{l} \rfloor$ segments in R that are order equivalent to S

c-symmetric Rings

Definition: order equivalence

Two sets of UIDs $U = (u_1, \ldots, u_k)$ and $V = (v_1, \ldots, v_k)$ are said to be order equivalent if for all $1 \leq i, j \leq k$ we have $u_i \leq u_j$ iff $v_i \leq v_j$.

Definition: c-symmetric ring

For $0 \leq c \leq 1$, a ring R is said to be c-symmetric if for every l such that $\sqrt{n} \leq l \leq n$, and every segment S or R of length l there are at least $\lfloor \frac{cn}{l} \rfloor$ segments in R that are order equivalent to S

Theorem

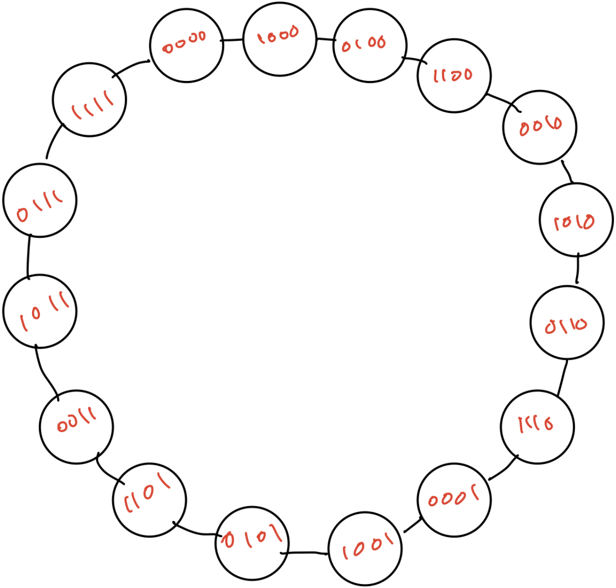
There's a constant c such that for every n there is a c-symmetric ring of size n.

c-symmetric Ring Example

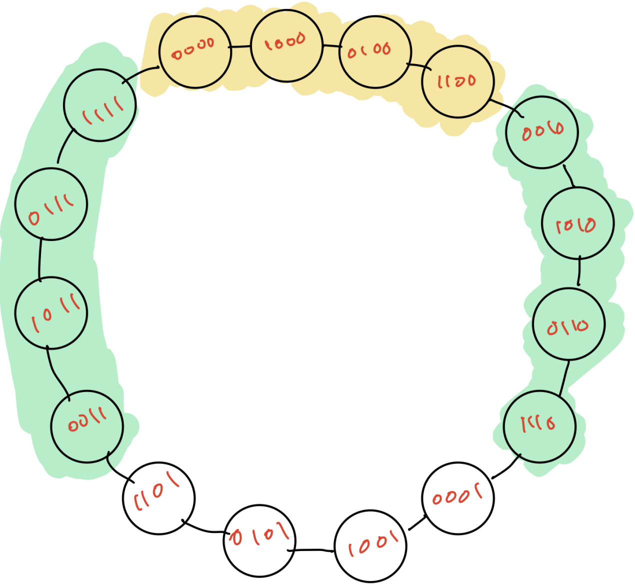
Examples

A bit-reversal ring R is a ring of size $n = 2^k$ where the UID of node i is $\mathrm{rev}(b_i)$ where $b_i \in \{0,1\}^k$ is the bit-string representation of i. Then, R is $\frac{1}{2}$-symmetric.

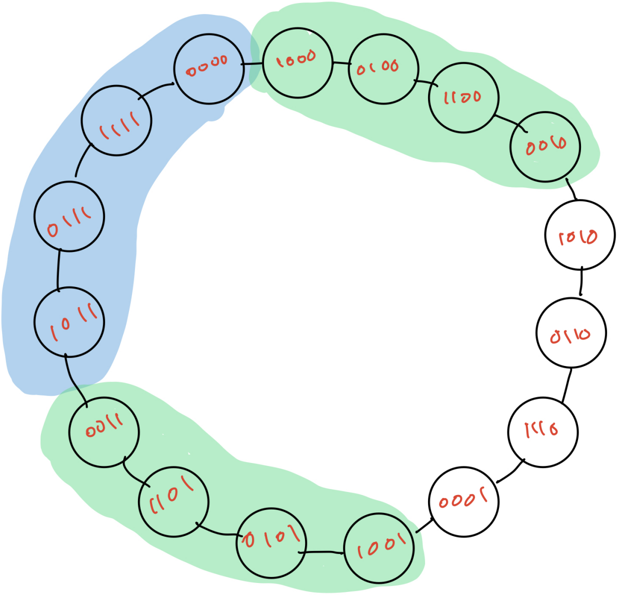
Bit Reversal Ring Examples



Bit Reversal Ring Examples



Bit Reversal Ring Examples



Bit Reversal Ring Examples

Theorem

A bit-reversal ring of size n is $\frac{1}{2}$-symmetric

Bit Reversal Ring Examples

Theorem

A bit-reversal ring of size n is $\frac{1}{2}$-symmetric

Proof.

Observe that in a bit reversal ring, if we consider vertices i and $i+1$, then if $i \equiv 0 \pmod 2$ then $i \leq_O i+1$, else $i+1 \leq_O i$ where $\leq_O$ is the ordering induced under bit-reversal.

Bit Reversal Ring Examples

Theorem

A bit-reversal ring of size n is $\frac{1}{2}$-symmetric

Proof.

Observe that in a bit reversal ring, if we consider vertices i and $i+1$, then if $i \equiv 0 \pmod 2$ then $i \leq_O i+1$, else $i+1 \leq_O i$ where $\leq_O$ is the ordering induced under bit-reversal. In particular, under this ordering neighboring vertices switch order, and hence we make the following key observation: **two sequences S, R are order-equivalent iff their first elements are either both odd, or both even**.

Bit Reversal Ring Examples

Theorem

A bit-reversal ring of size n is $\frac{1}{2}$-symmetric

Proof.

Observe that in a bit reversal ring, if we consider vertices i and $i+1$, then if $i \equiv 0 \pmod 2$ then $i \leq_O i+1$, else $i+1 \leq_O i$ where $\leq_O$ is the ordering induced under bit-reversal. In particular, under this ordering neighboring vertices switch order, and hence we make the following key observation: **two sequences S, R are order-equivalent iff their first elements are either both odd, or both even**. Thus for any fixed length l, since there are $\frac{n}{l}$ sequences, for any given sequence S, only $\frac{1}{2}$-fraction of these sequences will have a first element with the same odd/even-ness. Thus, the ring is $\frac{1}{2}$-symmetric. $\square$

Key Lemma

Definition: k-neighborhood

The k-neighborhood of a vertex i is the set of vertices at distance at most k from it (including itself).

Key Lemma

Definition: k-neighborhood

The k-neighborhood of a vertex i is the set of vertices at distance at most k from it (including itself).

Definition: Corresponding States

Two processes p_1, p_2 are said to be in corresponding states with respect to UID sequences U, V iff their states are identical except u_i is replaced with v_i (and vice-versa) in their respective states.

Key Lemma

Definition: k-neighborhood

The k-neighborhood of a vertex i is the set of vertices at distance at most k from it (including itself).

Definition: Corresponding States

Two processes p_1, p_2 are said to be in corresponding states with respect to UID sequences U, V iff their states are identical except u_i is replaced with v_i (and vice-versa) in their respective states.

Key Lemma

Let A be a comparison-based algorithm executing in a ring R of sized n, and k be an integer $0 \leq k < \lfloor n/2 \rfloor$. Let i, j be two processes that have order-equivalent sequences of UIDs in their k-neighborhoods. Then, at any point after at most k active rounds, i, j are in corresponding states, with respect to UID sequences in their k-neighborhoods.

Key Lemma

Key Lemma

Let A be a comparison-based algorithm executing in a ring R of sized n, and k be an integer $0 \leq k < \lfloor n/2 \rfloor$. Let i, j be two processes that have order-equivalent sequences of UIDs in their k-neighborhoods. Then, at any point after at most k active rounds, i, j are in corresponding states, with respect to UID sequences in their k-neighborhoods.

- Intuitively, if the neighborhood of two vertices have similar ordering, then since the algorithm only depends on relative order these two vertices should behave similarly.
- Proof is by induction on number of rounds, and case-work on processes receiving messages from their neighbors.

Lower bounding number of rounds

Theorem

Suppose A is executing on a c-symmetric ring of size n, and A elects a leader. Further suppose that k such that $\sqrt{n} \leq 2k+1$ and $\lfloor \frac{cn}{2k+1} \rfloor \geq 2$. Then A has **more** than k rounds.

Lower bounding number of rounds

Theorem

Suppose A is executing on a c-symmetric ring of size n, and A elects a leader. Further suppose that k such that $\sqrt{n} \leq 2k + 1$ and $\lfloor \frac{cn}{2k+1} \rfloor \geq 2$. Then A has **more** than k rounds.

Proof Sketch

By contradiction. If it takes at most k rounds, we can pick midpoints of any 2 order-equivalent segments (which exist since c-symmetric). These midpoints must have corresponding states (Key Lemma) and thus will both get elected as leaders in the next round.

Bringing it all together

The main results so far:

Theorem 1 (worst-case example)

There's a constant c such that for every n there is a c-symmetric ring of size n.

Theorem 2 (lower-bound on rounds)

Suppose A is executing on a c-symmetric ring of size n, and A elects a leader. Further suppose that k such that $\sqrt{n} \leq 2k+1$ and $\lfloor \frac{cn}{2k+1} \rfloor \geq 2$. Then A has **more** than k rounds.

Key Lemma

Let A be a comparison-based algorithm executing in a ring R of sized n, and k be an integer $0 \leq k < \lfloor n/2 \rfloor$. Let i, j be two processes that have order-equivalent sequences of UIDs in their k-neighborhoods. Then, at any point after at most k active rounds, i, j are in corresponding states, with respect to UID sequences in their k-neighborhoods.

Bringing it all together

Main Theorem

Let A be a comparison-based algorithm for electing a leader in a ring of size n. Then there is an execution of A which requires $\Omega(n \log n)$ messages to elect a leader.

Proof Sketch

- Using Theorem 1 pick a c-symmetric ring R of size n.
- Define $k = \lfloor \frac{cn-2}{4} \rfloor$.
- Using Theorem 2, there must be at least $k+1$ active rounds.
- Pick active round r satisfying $\sqrt{n} + 1 \leq r \leq k + 1$.

Bringing it all together

Main Theorem

Let A be a comparison-based algorithm for electing a leader in a ring of size n. Then there is an execution of A which requires $\Omega(n \log n)$ messages to elect a leader.

Proof Sketch

- Using Theorem 1 pick a c-symmetric ring R of size n.
- Define $k = \lfloor \frac{cn-2}{4} \rfloor$.
- Using Theorem 2, there must be at least $k+1$ active rounds.
- Pick active round r satisfying $\sqrt{n}+1 \leq r \leq k+1$.
- Since active, some process i sends a message. Since R is c-symmetric, by definition at least $\lfloor \frac{cn}{2r-1} \rfloor$ order-equivalent segments R to the $(r-1)$-neighborhood S of i.

Brining it all together

Proof Sketch cont'd

- Since active, some process i sends a message. Since R is c-symmetric, by definition at least $\lfloor \frac{cn}{2r-1} \rfloor$ order-equivalent segments R to the $(r-1)$-neighborhood S of i.
- By the Key Lemma, the midpoint of each of these segments must have corresponding states at the end of round $r-1$. Thus, they will all also send a message along with i.
- Set $r_1 = \lceil \sqrt{n} \rceil + 1$ and $r_2 = k + 1$. Then, total messages is **at least**

$$\sum_{r=r_1}^{r_2} \lfloor \frac{cn}{2r-1} \rfloor \geq \sum_{r=r_1}^{r_2} \frac{cn}{2r-1} - r_2$$

Final Steps

Proof Sketch cont'd

- Set $r_1 = \lceil \sqrt{n} \rceil + 1$ and $r_2 = k + 1$. Then, total messages is **at least**

$$\sum_{r=r_1}^{r_2} \lfloor \frac{cn}{2r-1} \rfloor \geq \sum_{r=r_1}^{r_2} \frac{cn}{2r-1} - r_2$$

- Second term is $O(n)$, but first term is $O(n \log n)$ as:

$$\sum_{r=r_1}^{r_2} \frac{cn}{2r-1} = \Omega\left(n \sum_{r=r_1}^{r_2} \frac{1}{r} \right)$$
$$= \Omega(n(\ln r_2 - \ln r_1)) = \Omega(n \log n) \qquad \left(\int_{r_1}^{r_2} \frac{1}{r} = \ln r_2 - \ln r_1\right)$$

Proof of Key Lemma

Table of Contents

Non-comparison based algorithms

- We showed lower-bound for **comparison-based** algorithms of $\Omega(n \log n)$. Can we do better in a different paradigm?

Non-comparison based algorithms

- We showed lower-bound for **comparison-based** algorithms of $\Omega(n \log n)$. Can we do better in a different paradigm? Yes! (and, no)
- We can achieve $O(n)$ communication complexity

Non-comparison based algorithms

- We showed lower-bound for **comparison-based** algorithms of $\Omega(n \log n)$. Can we do better in a different paradigm? Yes! (and, no)
- We can achieve $O(n)$ communication complexity
- However, we pay a big penalty in time complexity

Non-comparison based algorithms

We will broadly consider two settings:

1. The size of the ring, n, is known to all nodes ($\textsc{TimeSlice}$).
   $O(n)$ communication complexity, $O(n \cdot u_{\min})$ time complexity.

We will broadly consider two settings:

1. The size of the ring, n, is known to all nodes ($\textsc{TimeSlice}$).
   $O(n)$ communication complexity, $O(n \cdot u_{\min})$ time complexity.
2. The size of the ring is unknown ($\textsc{VariableSpeeds}$).
   $O(n)$ communication complexity, $O(n \cdot 2^{u_{\min}})$ time complexity.

TimeSlice

1. Every node knows n, the size of the ring.
2. Phases $1, \ldots$ each with n rounds.

TimeSlice

1. Every node knows n, the size of the ring.
2. Phases $1, \ldots$ each with n rounds.
3. In each phase p, only token with UID p is allowed to circulate.

TimeSlice

1. Every node knows n, the size of the ring.
2. Phases $1, \ldots$ each with n rounds.
3. In each phase p, only token with UID p is allowed to circulate.
4. If at the start of phase r some node u with UID r has not received any non-empty messages before, it elects itself as leader.

TimeSlice

1. Every node knows n, the size of the ring.
2. Phases $1, \ldots$ each with n rounds.
3. In each phase p, only token with UID p is allowed to circulate.
4. If at the start of phase r some node u with UID r has not received any non-empty messages before, it elects itself as leader.
5. If node with UID q receives non-null message at phase r where $q \neq r$, then q acknowledges r as leader, and passes along the message.
6. The n rounds to notify other nodes.

TimeSlice Analysis

1. Only n messages sent for notifying of leader status, thus communication complexity clearly $O(n)$
2. In each phase, every node still waits for messages at end of each round. First, and final phase in which messages are sent is $u_{\min}$. Thus, $O(n \cdot u_{\min})$ time complexity.

Nodes do not have knowledge of the ring size here.

1. Every node sends its UID around. First one to receive their own UID elects themselves as leader.

VariableSpeeds

Nodes do not have knowledge of the ring size here.

1. Every node sends its UID around. First one to receive their own UID elects themselves as leader.
2. Trick: a message for UID u is sent only one time every 2^u rounds.

VariableSpeeds

Nodes do not have knowledge of the ring size here.

1. Every node sends its UID around. First one to receive their own UID elects themselves as leader.
2. Trick: a message for UID u is sent only one time every 2^u rounds.
3. Smallest UID goes around fastest, while largest the slowest. Thus, min UID will get elected.

VariableSpeeds analysis

1. After $n \cdot 2^{u_{\min}}$ rounds, the node with least UID will get its UID back, and thus be elected leader.
2. The node with least UID takes n messages. However, second smallest will go at half speed and thus utilize only $n/2$ messages.
3. In general i^{th} smallest will use $n/2^{i+1}$ messages
4. Thus total messages is simply $n(\sum_{i=0}^{n} 2^{-i}) \leq 2n = O(n)$

Questions?

Thank You!