Gossip Algorithms
• In a gossip algorithm, each node in the network *periodically* exchanges information with a subset of nodes.

• This subset is usually *the set of neighbors of each node*.

Every node only has a local view of the network.

• **Objective**: each node receives some desired *global information*, through a certain number of periodically update of the nodes.
Rumor spreading

Problem

Design an algorithm so that all the nodes receive the rumor as fast as possible.

Solution 1  Initial node sends the rumor to one of its neighbours, and every informed node forwards it to all its neighbours.

• Downside 1: every node needs to interact with all its neighbours.
• Downside 2: every node receives its degree copies of the rumor.

Solution 2  Construct a spanning tree, and transfer the rumor only along the edges of the tree.

• Downside: Failure of links in the tree breaks rumor spreading process.

We need a simple, local, distributed, fast, and robust algorithm for information spreading.
Push protocol of rumor spreading

Protocol (Synchronous model)

- There is a rumor *initially* located at a node of a network;
- The protocol proceeds by *rounds*, in which each node only *contacts one of its neighbours*. 
Push protocol of rumor spreading

**PUSH**

Nodes with rumor sends to a random neighbour

**Algorithm Description**

1. \( t=0 \)
2. while \( t<T \) do
3. every informed node sends the rumor to its random neighbour.
4. \( t=t+1 \)

**Properties:**

- Nodes only contact with their neighbours; network’s global structure is unknown to each node.
- **Robust**: Failure of transmission among a few nodes won’t affect the algorithm’s performance.
- The algorithm **efficiently** sends a rumor to all nodes in the network.

*Randomisation is the key to ensure robustness and efficiency!*
Bad instance for the Push protocol

**Homework:** It takes $O(n \cdot \log n)$ rounds for all nodes to receive the rumor w.h.p.
Push-Pull Protocol

**PUSH**

Nodes with rumor sends to a random neighbour

**PULL**

Nodes without rumor asks a random neighbour

**Bad instance for PUSH**

**Bad instance for PULL**
**Push-Pull Protocol**

**Algorithm Description**

1. $t=0$
2. while $t<T$ do
3. 1. every informed node sends the rumor to its random neighbour.
3. 2. *every uninformed node calls a random neighbour, and gets the rumor if the neighbour has one.*
4. $t=t+1$
5. end
Analysis of the Push protocol

**Question**

How many rounds are needed before every node gets the rumor w.h.p.?

**Properties:**

- $\Omega(\text{Diam}(G))$ rounds are needed before every node gets the rumor.
- $\Omega(\log n)$ rounds are needed before every node gets the rumor.

*Since the number of informed vertices at most doubles after each round.*

**Theorem**

Let $G$ be a complete graph with $n$ nodes. Then, with high probability, every node gets the rumor after $\log n + \ln n + o(\log n)$ rounds.
Algorithm Description

1. Initial node $v$ sets $ID_v = 0$.
2. $t=0$
3. while $t<T$ do
   4-1. every node $v$ with ID sends $(ID_v, t)$ to its random neighbour.
   4-2. if a node $u$ without ID receives $(ID_v, t)$ from its neighbour, then
        $ID_u = 2^{t-1} + ID_v$

        Note: if node $u$ receives msg from multiple neighbours, $u$ chooses a random one to perform the operation above.

   4. $t=t+1$
5. end

Homework: Prove that every node receives a unique ID.
Leader election
Initial state (all not-elected)

Final state

leader
Why study rings?

- Simple starting point, easy to analyze

- Lower bounds and impossibility results for ring topology also apply to arbitrary topologies
LE algorithms in rings depend on...

Anonymous Rings
Non-anonymous Rings

Size of the network $n$ is known (non-unif.)
Size of the network $n$ is not known (unif.)

Synchronous Algorithms
Asynchronous Algorithms
Impossibility for Anonymous Rings

**Theorem**

There is no leader election algorithm for anonymous rings, even if
- the algorithm knows the ring size (non-uniform)
- in the synchronous model

**Proof Sketch (for non-unif and sync rings):**

- Every processor begins in same state (*not-elected*) with same outgoing msgs (since anonymous)
- Every processor receives same msgs, does same state transition, and sends same msgs in round 1
- And so on and so forth for rounds 2, 3, ...
- Eventually some processor is supposed to enter an elected state.
  But then they all would.
Since the theorem was proven for non-uniform and synchronous rings, the same result holds for *weaker* models:

- uniform
- asynchronous
Chang-Roberts algorithm: High-level Ideas

- Suppose the network is a ring
  - We assume that each node has 2 points to nodes it knows about
    - Next
    - Previous
    - (like a circular doubly linked list)
  - The actual network may not be a ring

- Every node send \( \max(\text{own ID}, \text{received ID}) \) to the next node
- If a processor receives its own ID, it is the leader
- It is \textit{uniform}: number of processors does not need to be known to the algorithm
Chang-Roberts algorithm: discussion

- Works in an asynchronous system
- Correctness: Elects processor with largest ID
  - *msg containing that ID passes through every processor.*

- Message complexity $O(n^2)$
  - When does it occur?
  - Worst case to arrange the IDs is in the decreasing order:
    - 2$^{\text{nd}}$ largest ID causes $n - 1$ messages
    - 3$^{\text{rd}}$ largest ID causes $n - 2$ messages
    - Etc.
    - Total messages $= n + (n - 1) + (n - 2) + \ldots + 1 = O(n^2)$
Hirschberg-Sinclair algorithm

• Assume all nodes want to know the leader

• $k$-neighborhood of node $p$

• How does $p$ send a message to distance $k$?
  • Message has a “time to live variable”
  • Each node decrements $m$.TTL on receiving
  • If $m$.TTL=0, don’t forward any more
Hirschberg-Sinclair algorithm: Message complexity

Question

What is the message complexity?

- In phase $i$
  - At most one node initiates message in any sequence of $2^{i-1}$ nodes
  - So, $n/2^{i-1}$ candidates
    - Each sends 2 messages, going at most $2^i$ distance, and transfers
      
      $2 \times 2 \times 2^i$ messages in total
    - $O(n)$ messages in phase $i$, and there are $O(\log n)$ phases
    - Total of $O(n \log n)$ messages.
Distributed Consensus
$G = (V, E)$, undirected graph (bidirected edges)

- Synchronous model, $n$ processes
- Each process has input 1 (attack) or 0 (don’t attack).
- Any subset of the messages can be lost.
- All should eventually set decision output variables to 0 or 1.

Correctness conditions:

- **Agreement**: No two processes decide differently.

- **Validity**:
  - If all start with 0, then 0 is the only allowed decision.
  - If all start with 1 and all messages are successfully delivered, then 1 is the only allowed decision.
• **Stronger validity condition:**
  
  – *If anyone starts with 0 then 0 is the only allowed decision.*
  
  – *If all start with 1 and all messages are successfully delivered, then 1 is the only allowed decision.*

• **Guidelines:**
  
  – For designing algorithms, try to use stronger correctness conditions (*better algorithm*).
  
  – For impossibility results, use weaker conditions (*better impossibility result*).
• Stopping failures (crashes) and Byzantine failures (arbitrary processor malfunction, possibly malicious)

• Agreement problem:
  – $n$-node connected, undirected graph, known to all processes.
  – Input $v$ from a set $V$, in some state variable.
  – Output $v$ from $V$, by setting decision $:= v$.
  – Bounded number $\leq f$ of processors may fail.

• Bounded number of failures:
  – A typical way of describing limited amounts of failure.
  – Alternatives: Bounded rate of failure; probabilistic.
Assume process may stop working at any point:
- Between rounds.
- While sending messages at a round; any subset of intended messages may be delivered.

Correctness conditions:
- **Agreement**: No two processes decide on different values.
  - “Uniform agreement”
- **Validity**: If all processes start with the same $v$, then $v$ is the only allowable decision.
- **Termination**: All nonfaulty processes eventually decide.

Alternatively:
- **Stronger validity condition**: Every decision value must be some process’ initial value.
• “Byzantine Generals Problem”
  – Originally “Albanian Generals”
• Faulty processes may exhibit “arbitrary behavior”:
  – Can start in arbitrary states, send arbitrary messages, perform arbitrary transitions.
  – But can’t affect anyone else’s state or outgoing messages.
  – Often called “malicious” (but they aren’t necessarily).
• Correctness conditions:
  – **Agreement**: No two nonfaulty processes decide on different values.
  – **Validity**: If all nonfaulty processes start with the same $v$, then $v$ is the only allowable decision for nonfaulty processes.
  – **Termination**: All nonfaulty processes eventually decide.
A Byzantine agreement algorithm doesn’t necessarily solve stopping agreement.

For stopping, all processes that decide, even ones that later fail, must agree (uniformity condition).

Too strong for Byzantine setting.
• **Time**: Number of rounds until all nonfaulty processes decide.

• **Communication**: Number of messages, or number of bits.
  
  – For Byzantine case, just count those sent by nonfaulty processes.
Assume complete $n$-node graph.

Idea:
- Processes keep sending all $V$ values they’ve ever seen.
- Use simple decision rule at the end.

In more detail:
- Process $i$ maintains $W \subseteq V$, initially containing just $i$’s initial value.
- Repeatedly: Broadcast $W$, and add received elements to $W$.
- After $k$ rounds:
  - If $|W| = 1$ then decide on the unique value.
  - Else decide on default value $v_0 \in V$.

**Question:** How many rounds?
Complexity Bounds

- Time: \( f + 1 \) rounds
- Communication:
  - Messages: \( \leq (f + 1) n^2 \)
  - Message bits: Multiply by \( n b \)

- Can improve communication:
  - Messages: \( \leq 2 n^2 \)
  - Message bits: Multiply by \( b \)

Number of values sent in a message

A fixed bound on number of bits to represent a value in \( V \).
• Each process broadcasts its own value in round 1.

• May broadcast at one other round, just after it first hears of some value different from its own.

• In that case, it chooses just one such value to rebroadcast.

• After $f + 1$ rounds:
  – If $|\mathcal{W}| = 1$ then decide on the unique value.
  – Else decide on default value $v_0$. 

Improved algorithm
A strategy for consensus algorithms, which works for Byzantine agreement as well as stopping agreement.

Based on EIG tree data structure.

EIG tree $T_{n,f}$, for $n$ processes, $f$ failures:
- $f + 2$ levels
- Paths from root to leaf correspond to strings of $f + 1$ distinct process names.

Example: $T_{4,2}$
Each process $i$ uses the same EIG tree, $T_{n,f}$.

Decorates nodes of the tree with values in $V$, level by level.

Initially: Decorate root with $i$’s input value.

Round $r \geq 1$:

- Send all level $r - 1$ decorations for nodes to everyone.
  - Including yourself---simulate locally.
- Use received messages to decorate level $r$ nodes---to determine label, append sender’s id at the end.
- If no message received, use $\bot$.

The decoration for node $(i_1, i_2, i_3, \ldots, i_k)$ in $i$’s tree is the value $v$ such that $(i_k$ told $i)$ that $(i_{k-1}$ told $i_k)$ that ...that $(i_1$ told $i_2)$ that $i_1$’s initial value was $v$.

Decision rule for stopping case:

- Trivial
- Let $W$ = set of all values decorating the local EIG tree.
- If $|W| = 1$ decide that value, else default $v_0$. 

EIG Stopping Agreement Algorithm
Byzantine Agreement
EIG Algorithm for Byzantine Agreement

- Use EIG tree.
- Relay messages for $f + 1$ rounds.
- Decorate the EIG tree with values from $V$, replacing any garbage messages with default value $v_0$.
- Call the decorations $\text{val}(x)$, where $x$ is any node label.
- Decision rule:
  - Redecorate the tree, defining $\text{newval}(x)$.
    - Proceed bottom-up.
    - Leaf: $\text{newval}(x) = \text{val}(x)$
    - Non-leaf: $\text{newval}(x)$ =
      - $\text{newval}$ of strict majority of children in the tree, if majority exists,
      - $v_0$ otherwise.
  - Final decision: $\text{newval}(\lambda)$ (newval at root)
\begin{itemize}
  \item $n > 3f$ is necessary!
    \begin{itemize}
      \item Holds for any $n$-node (undirected) graph.
      \item For graphs with low connectivity, may need even more processors.
      \item Number of failures that can be tolerated for Byzantine agreement in an undirected graph $G$ has been completely characterized, in terms of number of nodes and connectivity.
    \end{itemize}
  \item Theorem 1: 3 processes cannot solve BA with 1 possible failure.
  \item Theorem 2: $n$ processes can’t solve BA, if $n \leq 3f$.
\end{itemize}