# Analysis and Implementation of Consensus Protocols in Multi-Agent Systems

Anja Kovačević, Boris Čuljak
Fakultet Tehničkih Nauka
{kovacevic.e275.2024, culjak.e265.2024} @uns.ac.rs

***Abstract*—This project investigates discrete-time consensus in networked multi-agent systems by combining theoretical analysis with a Python-based simulation in Mesa. We implement (i) a linear (average) consensus protocol and (ii) a max-consensus protocol, and we evaluate their convergence behavior under varying communication topologies (ring, random, fully connected) and additive noise. Results confirm that linear consensus converges asymptotically with topology-dependent convergence speed, while max-consensus reaches exact agreement in finite time bounded by the graph diameter but exhibits higher sensitivity to noise-induced extrema. The study demonstrates how Laplacian-based models translate into executable agent simulations and how network structure and uncertainty shape agreement dynamics.**

## I. Introduction

Multi-agent systems (MAS) consist of multiple interacting agents that cooperate to achieve a common objective without relying on a central coordinator. In this context, an agent is an autonomous entity capable of sensing its local environment, exchanging information with neighboring agents, and updating its internal state according to a prescribed rule. MAS are well-suited for problems where centralized solutions are impractical due to scalability, communication constraints, or robustness requirements, such as distributed sensing, formation control, and resource allocation.

A fundamental coordination task in MAS is *consensus*, where agents aim to reach agreement on a quantity of interest using only local interactions. In many applications, agents start from heterogeneous initial measurements and repeatedly combine information from their neighbors so that their states converge to a common value. Consensus protocols provide a principled mechanism for such decentralized agreement and are typically analyzed through graph-theoretic models of the underlying communication network [1].

In this project, we analyze, implement, and simulate two canonical distributed agreement mechanisms. The first is *linear (average) consensus*, a linear Laplacian-based protocol in which agents converge asymptotically to the average of their initial states. The second is *max-consensus*, a nonlinear protocol in which the maximum initial value propagates through the network and exact agreement is reached in finite time. Beyond comparing these protocols at an algorithmic level, we investigate how the *communication topology* and *measurement uncertainty* shape convergence behavior. Concretely, we evaluate ring, random, and fully connected graphs, and we study the impact of additive noise on the evolution of agent states. This provides a controlled simulation-based validation of theoretical expectations regarding (i) topology-dependent convergence speed for linear consensus and (ii) diameter-bounded convergence for max-consensus, as well as their respective robustness characteristics.

The simulation is implemented in Python using the Mesa framework, enabling an interactive and reproducible environment where the number of agents, network topology, consensus protocol, step size, and noise level can be configured. For consistency, unless stated otherwise, simulations assume synchronous discrete-time updates, connected communication graphs, and a fixed step size chosen to satisfy standard stability conditions for the linear protocol.

The remainder of this document is organized as follows:

- Section II presents the theoretical framework, including graph preliminaries and the update rules and convergence properties of linear consensus and max-consensus.
- Section III describes the Mesa-based implementation, covering agent behavior, protocol encapsulation, and simulation architecture.
- Section IV reports simulation results across different network topologies and noise levels and compares convergence behavior using quantitative indicators (e.g., iterations to near-consensus and dispersion of agent states).
- Section V discusses the observed trade-offs between convergence speed and robustness and summarizes the main findings.

## II. Theoretical Framework

### A. Graph Theory Preliminaries

The interaction topology of a network of $N$ agents is modeled as a graph $G = (V, E)$, where $V = \{1, \dots, N\}$ is the set of nodes (agents) and $E \subseteq V \times V$ is the set of edges (communication links). If $(i, j) \in E$, then agent $i$ can receive information from agent $j$. The set of neighbors of agent $i$ is denoted by

$$N_i = \{j \in V : (i, j) \in E\}.$$

Throughout this work, the communication graph is assumed to be fixed and connected. Unless stated otherwise, graphs are considered undirected, implying bidirectional information exchange between neighboring agents. Agent interactions evolve in discrete time, and state updates are performed synchronously: at each time step, all agents update their states simultaneously based on information available from the previous time step.

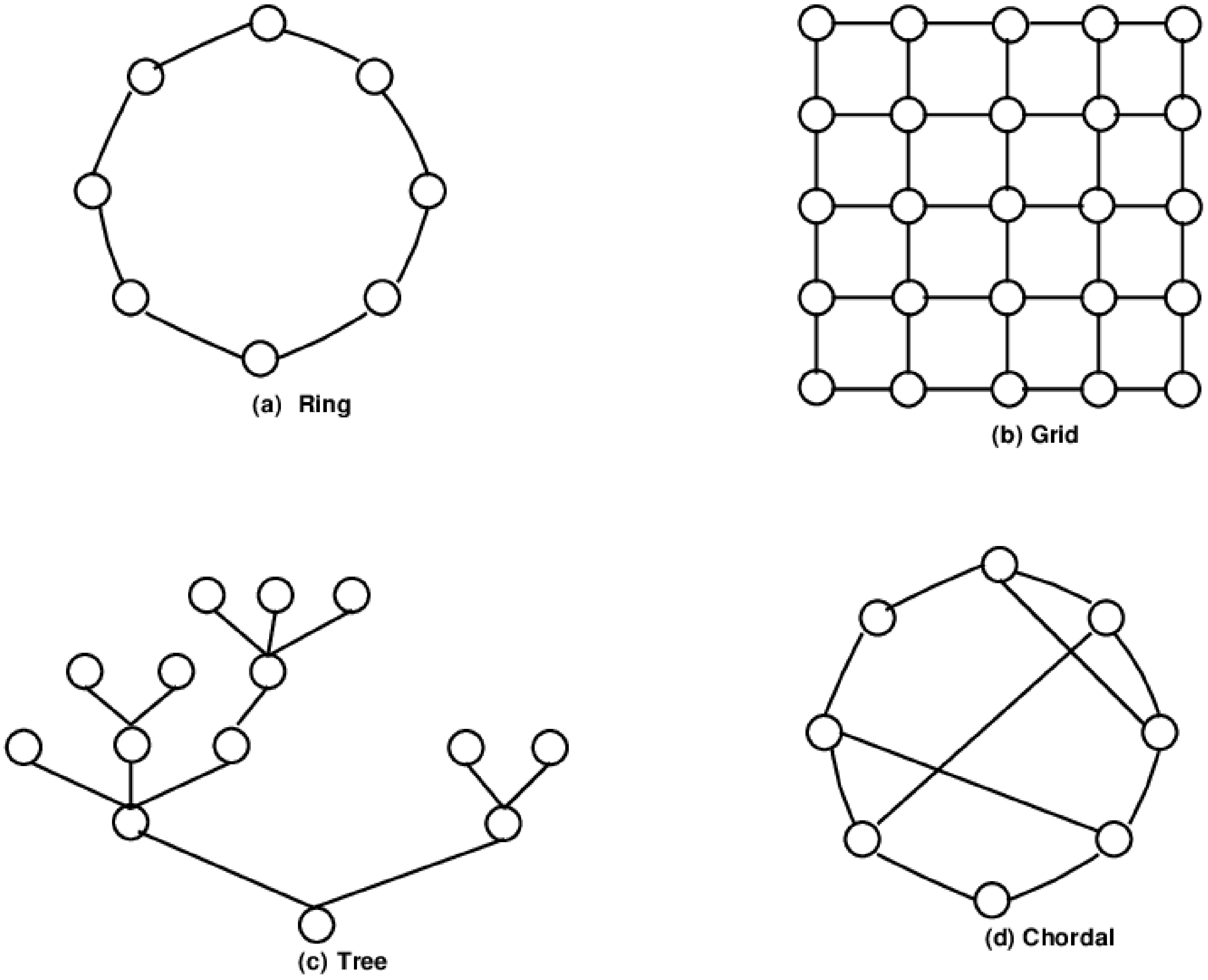


Fig. 1. Examples of common communication topologies used in multi-agent systems: (a) ring, (b) grid, (c) tree, and (d) chordal graph.

## B. Linear Consensus Protocol

The objective of the linear consensus protocol is for all agents to converge to the average of their initial states,

$$x_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} x_i(0),$$

a task commonly referred to as the *linear consensus* problem.

At each discrete time step, every agent performs exactly one state update. If an agent is connected to multiple neighbors, information from all neighbors is combined within a single update through aggregation. No pairwise or sequential updates are performed; instead, all agents update their states simultaneously using the same local update rule.

Although the average value can be computed directly in a centralized setting, such an approach is generally infeasible in multi-agent systems. Each agent has access only to its own state and to information received from its immediate neighbors, and no agent possesses global knowledge of all states or direct access to the entire network. Centralized computation also requires a global coordinator and reliable global communication, which may be impractical or undesirable in large-scale, dynamic, or failure-prone networks. Consensus protocols address these limitations by enabling agents to compute global agreement values in a fully distributed manner using only local interactions.

*1) Mathematical Derivation and Convergence:* The linear consensus protocol can be interpreted from both control-theoretic and game-theoretic perspectives. From a control viewpoint, it corresponds to a dynamical system that drives agent states toward agreement over time. From a game-theoretic viewpoint, it emerges as the outcome of local cost minimization, where each agent seeks to reduce disagreement with its neighbors.

*a) Continuous-time dynamics.:* In continuous time, the evolution of agent states is modeled as

$$\dot{x}(t) = -Lx(t),$$

where $\dot{x}(t) = \frac{d}{dt}x(t)$ denotes the time derivative of the state vector. For an individual agent, $\dot{x}_i(t)$ represents the rate of change of its state over time. For example, if an agent's state evolves linearly as $x_i(t) = \gamma t$, then $\dot{x}_i(t) = \gamma$ is constant.

The vector

$$x(t) = [x_1(t), x_2(t), \ldots, x_N(t)]^T$$

collects the states of all $N$ agents at time $t$. Although this global state vector is used for mathematical analysis, no individual agent has access to $x(t)$ in its entirety; each agent observes only its own state and those of its neighbors.

*b) Game-theoretic interpretation.:* The Laplacian-based dynamics arise naturally from a local disagreement minimization problem. Each agent $i$ is associated with a local cost function

$$J_i(x) = \frac{1}{2}\sum_{j\in N_i}(x_i - x_j)^2,$$

which penalizes deviations between the agent's state and the states of its neighbors. This formulation is common in applied game theory, where each agent acts as a player attempting to minimize its own cost using only local information.

Taking the gradient of $J_i$ with respect to $x_i$ yields

$$\frac{\partial J_i}{\partial x_i} = \sum_{j\in N_i}(x_i - x_j).$$

Choosing a gradient-descent control law,

$$\dot{x}_i(t) = -\frac{\partial J_i}{\partial x_i},$$

results in

$$\dot{x}_i(t) = -\sum_{j\in N_i}(x_i(t) - x_j(t)).$$

Stacking the dynamics of all agents yields the compact vector form $\dot{x}(t) = -Lx(t)$. Thus, the linear consensus protocol can be interpreted as a distributed gradient flow that drives the system toward a Nash equilibrium[1] in which all agents hold identical states.

*c) Graph Laplacian structure.:* The matrix $L$ is the graph Laplacian, defined as $L = D - A$, where $A$ is the adjacency matrix and $D$ is the degree matrix of the communication graph. For undirected graphs, the Laplacian has the entry-wise form

$$L_{ij} = \begin{cases} \deg(i), & i = j, \\ -1, & j \in N_i, \\ 0, & \text{otherwise.} \end{cases}$$

[1]In this setting, each agent minimizes a local cost function $J_i(x)$. A Nash equilibrium is a state in which no agent can unilaterally reduce its cost by changing its own state. For a connected graph, this condition is satisfied if and only if $x_i = x_j$ for all agents, meaning that all agents hold identical states.

When the Laplacian multiplies the state vector, the $i$-th component of $Lx(t)$ is given by

$$(Lx(t))_i = \sum_{j\in N_i}\big(x_i(t) - x_j(t)\big).$$

This expression quantifies the local disagreement between agent $i$ and its neighbors. Although sometimes described as a "second-order" operator, this refers to a second-order *spatial* effect on the graph, analogous to diffusion, and not to a second derivative with respect to time. The system dynamics remain first order in time.

Consensus corresponds to the condition $Lx(t) = 0$, which holds if and only if all agents have identical states on a connected graph.

*d) Discrete-time implementation.:* To obtain a discrete-time update suitable for digital simulation, the continuous-time dynamics are discretized using a forward Euler method with step size $\epsilon > 0$. Approximating the time derivative as

$$\dot{x}(t) \approx \frac{x(k+1) - x(k)}{\epsilon},$$

and substituting $\dot{x}(t) = -Lx(t)$ yields

$$x(k+1) = x(k) - \epsilon Lx(k).$$

In component form, this results in the local update rule

$$x_i(k+1) = x_i(k) + \epsilon\sum_{j\in N_i}\big(x_j(k) - x_i(k)\big), \qquad (1)$$

where each agent aggregates information from all neighbors within a single synchronous update.

*e) Convergence properties.:* The collective system dynamics can be written in matrix form as

$$x(k+1) = Px(k), \quad \text{where} \quad P = I - \epsilon L.$$

For an undirected and connected graph, and for sufficiently small step sizes $\epsilon < 1/\Delta_{\max}$, where $\Delta_{\max}$ denotes the maximum node degree, the matrix $P$ is nonnegative and doubly stochastic.

Double stochasticity implies invariance of the sum of agent states, since multiplying a state vector by a matrix whose rows sum to one preserves the total sum of its entries:

$$\sum_{i=1}^{N} x_i(k+1) = \sum_{i=1}^{N} x_i(k).$$

Consequently, the average of the agent states is conserved over time.

In addition, repeated multiplication by $P$ reduces disagreement among agents. As $k \to \infty$, the matrix power $P^k$ converges to

$$\lim_{k\to\infty} P^k = \frac{1}{N}\mathbf{1}\mathbf{1}^T,$$

where $\mathbf{1}$ denotes the vector of all ones. This implies that all agents converge asymptotically to a common value equal to the average of their initial states.

### C. *Max-Consensus Protocol*

While the linear consensus protocol aims to compute an aggregate quantity through iterative averaging, the Max-Consensus protocol addresses a different coordination objective. Instead of combining information, it propagates an extreme value across the network, enabling all agents to agree on the maximum initial state.

Formally, the goal of Max-Consensus is for all agents to converge to

$$x_{\max} = \max_{i \in V} x_i(0).$$

This protocol is particularly useful in applications where detecting or propagating extreme values is required, such as leader election, alarm triggering, or identifying the highest measurement in distributed sensor networks.

*1) Mathematical Derivation and Convergence:* Unlike linear consensus, Max-Consensus is governed by a nonlinear update rule based on comparison rather than averaging. At each discrete time step, every agent updates its state by selecting the largest value among itself and its neighbors:

$$x_i(k+1) = \max\big(x_i(k), \max_{j \in N_i} x_j(k)\big). \tag{2}$$

As in the linear case, updates are performed synchronously, and each agent aggregates information from all neighbors within a single update.

*a) Game-theoretic interpretation.:* The Max-Consensus protocol can be interpreted from a game-theoretic perspective as a best-response dynamic. Each agent aims to maximize its own state value based on locally observed information. At each iteration, selecting the maximum among neighboring values constitutes a best response, since adopting a smaller value cannot improve the agent's outcome under this objective.

States corresponding to the global maximum form absorbing states of the dynamics: once an agent attains the maximum value, it cannot be reduced in subsequent updates. Consequently, the network evolves toward a configuration in which all agents adopt the same maximal value, which constitutes a Nash equilibrium of the induced game, as no agent can unilaterally improve its state beyond the global maximum.

*b) Convergence properties.:* The convergence of the Max-Consensus protocol follows from three key structural properties:

1) **Monotonicity:** The state of each agent is non-decreasing over time. An agent updates its state only when it observes a strictly larger value from one of its neighbors.
2) **Propagation of the maximum:** The agent holding the global maximum acts as a source node. At each iteration, the maximum value propagates to all neighboring agents, expanding its influence across the network.
3) **Finite-time convergence:** For a connected communication graph, the maximum value reaches all agents in a finite number of steps. Specifically, convergence occurs in at most $D$ iterations, where $D$ denotes the diameter of the graph [2], [5]. This follows from the fact that the maximum value advances by at least one graph hop per iteration along shortest paths.

In contrast to linear consensus, which converges asymptotically, Max-Consensus achieves exact agreement in finite time. Once all agents have adopted the maximum value, the system remains in this consensus state indefinitely.

### D. *Illustrative Example*

To make the preceding concepts concrete, consider a simple network of three agents connected in a line topology: $1-2-3$. All updates are performed synchronously: at each iteration $k$, all agents compute their new states using values from the previous step and apply the update simultaneously.

**Scenario:** Three temperature sensors measure the following initial values:

- Agent 1: $x_1(0) = 10$
- Agent 2: $x_2(0) = 20$
- Agent 3: $x_3(0) = 5$

The global average is $\frac{10+20+5}{3} = 11.67$, and the global maximum is 20.

*1) Linear Consensus Walkthrough ($\epsilon = 0.2$):*

- $k = 0$**:** $x = [10, 20, 5]$
- $k = 1$**:**
  - Agent 1 (neighbor: 2): $x_1(1) = 10 + 0.2(20 - 10) = 12$
  - Agent 2 (neighbors: 1, 3): $x_2(1) = 20 + 0.2\big((10 - 20) + (5 - 20)\big) = 15$
  - Agent 3 (neighbor: 2): $x_3(1) = 5 + 0.2(20 - 5) = 8$

  State: $[12, 15, 8]$. Sum: 35 (invariant).
- $k = 2$**:**
  - Agent 1: $x_1(2) = 12 + 0.2(15 - 12) = 12.6$
  - Agent 2: $x_2(2) = 15 + 0.2\big((12 - 15) + (8 - 15)\big) = 13$
  - Agent 3: $x_3(2) = 8 + 0.2(15 - 8) = 9.4$

  State: $[12.6, 13, 9.4]$. Sum: 35.

Notice how the agent states move closer together at each iteration while preserving the total sum, gradually converging toward the average value of 11.67.

*2) Max-Consensus Walkthrough:*

- $k = 0$**:** $x = [10, 20, 5]$
- $k = 1$**:**
  - Agent 1 observes agent 2: $\max(10, 20) = 20$
  - Agent 2 observes agents 1 and 3: $\max(20, 10, 5) = 20$
  - Agent 3 observes agent 2: $\max(5, 20) = 20$

  State: $[20, 20, 20]$.

In this specific topology and initialization, consensus is reached in a single iteration because the center node initially holds the maximum value. Since the diameter of the line graph is two, at most two iterations would be required in the worst case for the maximum value to propagate to all agents.

## III. Implementation

The simulation is implemented in Python using the Mesa framework, which provides a modular environment for agent-based modeling. The codebase follows a clean architecture pattern that separates domain logic, agent behavior, and simulation control. This design choice facilitates a clear mapping between the theoretical consensus models and their software realization.

### A. Agent Communication and Model

In our simulation, agents do not exchange explicit messages over a communication channel. Instead, interaction is abstracted through shared access to the simulation environment (the Model), where each agent can observe the current states of its neighbors. This abstraction corresponds directly to the neighbor-based information structure assumed in the theoretical framework, in which each agent has access only to the states of agents in its neighborhood.

Each agent is represented by an instance of the `ConsensusAgent` class. At every simulation step, an agent queries the model's grid structure to identify its neighbors. This operation yields the set of agents corresponding to the neighborhood $N_i$ defined in Section II.

Listing 1. Agent step method used for neighbor-based updates.

```
# src/simulation/consensus_agent.py

def step(self):
    # Get neighbors from the model's grid/network
    neighbors =
    self.model.grid.get_neighbors(self.pos,
    include_center=False)

    # "Receive" information by reading neighbor
    states
    neighbor_states = [n.state for n in neighbors
    if isinstance(n, ConsensusAgent)]

    # Calculate next value using the protocol
    self.next_value =
    self.protocol.calculate_next_value(self.state,
    neighbor_states)
```

Each invocation of the `step()` method corresponds to a single discrete-time iteration $k$ in the theoretical model. During this phase, the agent computes its next state $x_i(k+1)$ using only state information from the previous iteration, namely the set $\{x_j(k) : j \in N_i\}$. This design is consistent with the synchronous update assumption used in the consensus analysis.

The intermediate variable `next_value` plays a crucial role in enforcing synchronous updates. All agents first compute their next states based on neighbor information from time step $k$, and only afterward are these values committed simultaneously by the scheduler. This two-phase update mechanism prevents partial or sequential updates and preserves the discrete-time dynamics assumed in the theoretical derivation.

By passing `neighbor_states` to the protocol, the implementation provides each agent with the exact information required by the consensus update rules. The protocol object then applies either the Laplacian-based aggregation for linear consensus or the maximization operator for max-consensus, thereby directly realizing the theoretical update equations derived in the previous section.

### B. Protocols

The consensus logic is encapsulated in strategy classes that implement the `AbstractProtocol` interface. This design follows the strategy pattern and allows different consensus algorithms to be interchanged without modifying agent or model code. From a theoretical perspective, this structure mirrors the abstraction used in the analysis, where the interaction topology is fixed while the local update rule may vary.

Each protocol implements a function that computes the next agent state $x_i(k+1)$ based on the current state $x_i(k)$ and the set of neighbor states $\{x_j(k) : j \in N_i\}$. In this way, the protocol classes correspond directly to the discrete-time update maps derived in the theoretical framework.

*1) Linear Consensus Implementation:* The `LinearConsensus` class implements the discrete-time linear consensus update rule derived from the forward Euler discretization of the continuous-time Laplacian dynamics. As established in Section II, the update rule for agent $i$ is given by

$$x_i(k+1) = x_i(k) + \epsilon \sum_{j \in N_i} \big(x_j(k) - x_i(k)\big),$$

where $\epsilon > 0$ is the step size.

In the implementation, the aggregation of neighbor information is realized by explicitly computing the sum of pairwise state differences between the agent and its neighbors. This corresponds exactly to the Laplacian-based correction term in the theoretical model.

Listing 2. Linear consensus update rule implementation.

```
# src/protocols/linear_consensus.py

def calculate_next_value(self, current_state:
    AgentState,
                          neighbor_states:
    List[AgentState]) -> float:
    # Calculate the sum of differences: sum(x_j -
    x_i)
    diff_sum = sum(neighbor.value -
    current_state.value
                   for neighbor in neighbor_states)

    # Update rule: x_i(k+1) = x_i(k) + epsilon *
    diff_sum
    next_val = current_state.value +
    self.config.epsilon * diff_sum + noise
    return next_val
```

Here, `current_state.value` represents the agent's state $x_i(k)$, while the list `neighbor_states` contains the values $\{x_j(k) : j \in N_i\}$. The variable `diff_sum` therefore implements the local disagreement term $\sum_{j \in N_i}(x_j(k) - x_i(k))$. The step size $\epsilon$ corresponds to the discretization parameter introduced in the theoretical analysis.

The optional noise term models measurement or update uncertainty and is set to zero in noise-free experiments. When enabled, it allows the robustness properties of the linear consensus protocol to be evaluated under perturbations, as discussed in the simulation results.

*2) Max-Consensus Implementation:* The `MaxConsensus` class implements the nonlinear update rule of the Max-Consensus protocol. As derived in the theoretical framework, each agent updates its state by selecting the maximum value among its own state and those of its neighbors:

$$x_i(k+1) = \max\big(x_i(k), \max_{j \in N_i} x_j(k)\big).$$

Unlike linear consensus, this update rule does not involve averaging or summation. Instead, it relies on an order-preserving operation that propagates the global maximum through the network.

Listing 3. Max-consensus update rule implementation.

```
# src/protocols/max_consensus.py

def calculate_next_value(self, current_state:
    AgentState,
                           neighbor_states:
    List[AgentState]) -> float:
    # Find the maximum value among neighbors:
    max(x_j)
    max_neighbor_val = max(neighbor.value
                           for neighbor in
    neighbor_states)

    # Update rule: max(x_i, max(x_j))
    return max(current_state.value,
    max_neighbor_val)
```

In this implementation, the function computes the maximum of the neighbor states and compares it with the agent's current state. This directly realizes the theoretical max-consensus update and preserves the monotonicity property discussed earlier: once an agent attains the maximum value, it cannot decrease in subsequent iterations.

### *C. Simulation Model (`ConsensusModel`)*

The `ConsensusModel` class manages the global state and execution flow of the consensus simulation. Its primary role is to instantiate the communication topology, initialize agents, and enforce the synchronous update semantics assumed in the theoretical framework.

Specifically, the model is responsible for the following tasks:

- **Network Initialization**: Constructing the communication graph using the `networkx` library. Supported topologies include random (Erdős–Rényi), ring, and fully connected graphs. These topologies correspond directly to the graph structures analyzed in the theoretical and simulation sections and allow systematic evaluation of topology-dependent convergence behavior.
- **Agent Creation**: Instantiating agents with random initial states and assigning each agent an instance of the selected consensus protocol.
- **Scheduling and Execution**: Enforcing synchronous discrete-time updates by executing the `step` and `advance` methods for all agents in each simulation cycle.

The communication topology is represented internally as a graph $G = (V, E)$ and embedded into the simulation environment using Mesa's `NetworkGrid`. This structure provides each agent with access to its neighborhood $N_i$, directly realizing the graph-based interaction model used in the theoretical analysis.

The model executes the simulation in discrete time. During each iteration, all agents first compute their next states based on neighbor information from the previous step, and only afterward are these states applied simultaneously. This two-phase update mechanism preserves the synchronous dynamics required for the convergence guarantees derived earlier.

Simulation execution continues until a predefined maximum number of iterations is reached, at which point the model terminates automatically. The model also supports resetting the simulation state, allowing multiple runs with different initial conditions or network configurations while maintaining the same theoretical assumptions.

## IV. Simulation and Results

The proposed consensus protocols were implemented and evaluated using an interactive simulation environment built on the Mesa framework. The simulator provides a controlled setting in which key parameters such as the number of agents, communication topology, consensus protocol, step size, and noise level can be varied.

Unless stated otherwise, all simulations were performed with $N = 10$ agents, synchronous discrete-time updates, zero measurement noise, and a fixed step size $\epsilon = 0.1$. This value of $\epsilon$ satisfies the stability conditions derived for linear consensus and is used consistently across experiments to enable fair comparison. Each agent was initialized with a random scalar value uniformly drawn from the interval $[0, 100]$, representing heterogeneous local measurements at the start of the process.

The reported simulations are representative runs that illustrate the typical behavior of the consensus dynamics under the specified conditions. The focus is on qualitative and structural properties such as convergence behavior, agreement value, and sensitivity to topology and noise rather than on statistical averaging over multiple trials.

### *A. Simulation Setup*

The simulation environment allows configuration of the communication topology (random, ring, or fully connected), the consensus protocol (linear or max-consensus), and the numerical parameters governing the update rules.

During each simulation run, agent states are updated synchronously in discrete time, and the evolution of all agent values is recorded over successive iterations. The main visualization displays agent states as a function of time, enabling direct observation of convergence behavior, transient dynamics, and the influence of network structure.

By holding all parameters fixed except for the variable under investigation (e.g., topology or noise level), the simulation setup enables controlled comparison of protocol behavior under different conditions, consistent with the analytical results derived earlier.

## B. Protocol Comparison

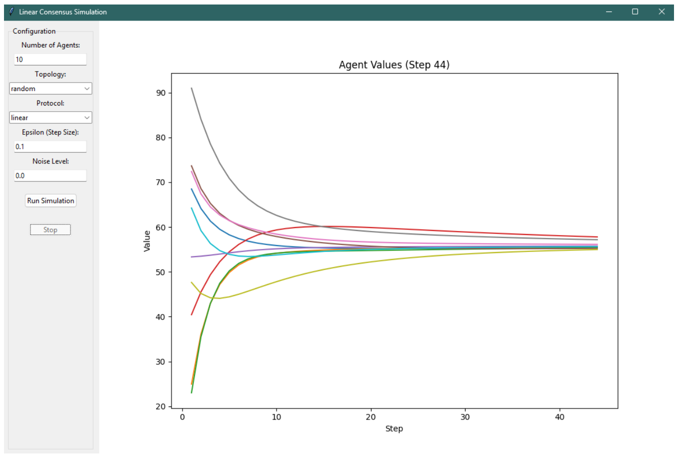
Figure 2 illustrates the behavior of the linear consensus protocol on a random communication topology. Despite large initial discrepancies, all agent states gradually converge toward a common agreement value corresponding to the global average.

Consistent with the theoretical analysis, convergence is asymptotic: the disagreement between agents decreases monotonically over time, while exact equality is achieved only in the limit. The relatively slow convergence observed in this example is characteristic of random and sparse topologies, where information diffusion is governed by the algebraic connectivity of the communication graph [3], [4] rather than by direct global coupling.

In contrast, Figure 3 shows the behavior of the max-consensus protocol under the same initial conditions and network structure. Instead of gradual averaging, the maximum initial value propagates through the network as agents update their states only when a higher value is observed in their neighborhood. This results in monotonic, non-decreasing state trajectories for all agents.

As predicted by the theoretical framework, max-consensus achieves exact agreement in finite time, with the convergence time bounded by the diameter of the underlying graph. In this representative run, all agents reach the maximum value within five iterations, significantly faster than the asymptotic convergence observed for linear consensus.

Together, these results highlight the fundamental trade-off between the two protocols: linear consensus provides smooth, averaging-based convergence toward a global aggregate, while max-consensus enables rapid, finite-time propagation of extreme values.


Linear Consensus Simulation

Configuration

Number of Agents: 10

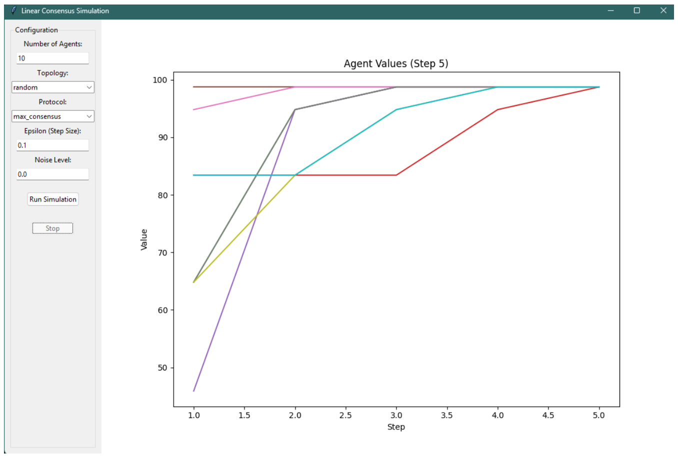
Topology: random

Protocol: linear

Epsilon (Step Size): 0.1

Noise Level: 0.0

Fig. 2. Linear consensus on a random topology with $N = 10$ agents, $\epsilon = 0.1$, and zero noise. Agent values asymptotically converge toward the global average, with near-consensus reached after approximately 40 iterations.


Linear Consensus Simulation

Configuration

Number of Agents: 10

Topology: random

Protocol: max_consensus

Epsilon (Step Size): 0.1

Noise Level: 0.0

Fig. 3. Max-consensus on a random topology with $N = 10$ agents, $\epsilon = 0.1$, and zero noise. The maximum value propagates through the network and exact consensus is reached in finite time (five iterations in this example).

## C. Effect of Network Topology

Beyond the choice of consensus protocol, the communication topology plays a critical role in determining convergence speed and overall system behavior. To illustrate this effect, both linear and max-consensus protocols were evaluated under different network structures, specifically ring and fully connected topologies, while keeping all other parameters fixed.

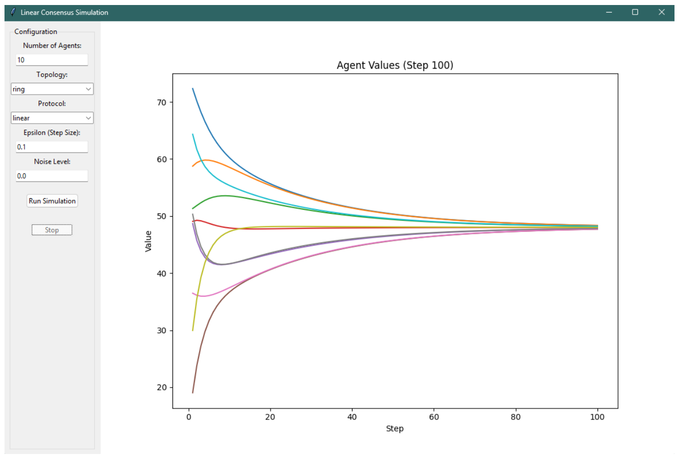
*1) Linear Consensus under Different Topologies:* Figure 4 shows the evolution of agent states for linear consensus on a ring topology. Due to the limited local connectivity, information propagates only between immediate neighbors, resulting in slow diffusion of state information across the network. As a consequence, convergence toward the global average is significantly delayed, with near-consensus reached only after approximately 100 iterations in this representative run.

In contrast, Figure 5 illustrates linear consensus on a fully connected topology. In this configuration, each agent exchanges information with all others at every iteration, effectively accessing the global state distribution. Under synchronous updates, this results in convergence within a single iteration for the chosen step size, as all agents compute the same weighted average simultaneously.

These results highlight the strong dependence of linear consensus convergence speed on the algebraic connectivity of the communication graph: sparse topologies slow down diffusive averaging, while dense connectivity accelerates agreement.

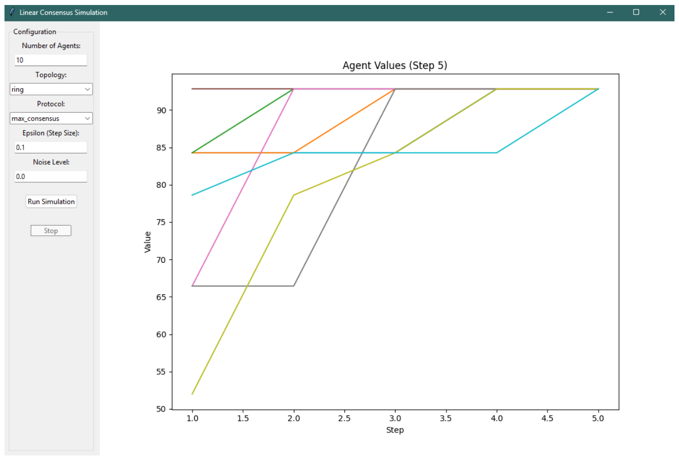
*2) Max-Consensus under Different Topologies:* A similar but more pronounced dependence on topology is observed for the max-consensus protocol. Figure 6 shows max-consensus on a ring topology. In this case, the maximum value propagates sequentially through neighboring agents, reaching all nodes within a number of iterations bounded by the diameter of the graph. In this configuration, exact consensus is achieved after approximately five iterations.

Figure 7 presents the behavior of max-consensus on a fully



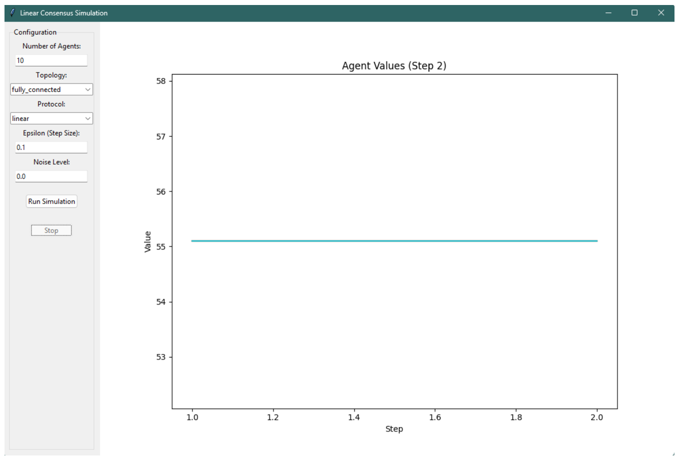
Number of Agents: 10
Topology: ring
Protocol: linear
Epsilon (Step Size): 0.1
Noise Level: 0.0

Fig. 4. Linear consensus on a ring topology with $N = 10$ agents and $\epsilon = 0.1$. Sparse local connectivity leads to slow convergence, with near-consensus reached after approximately 100 iterations.



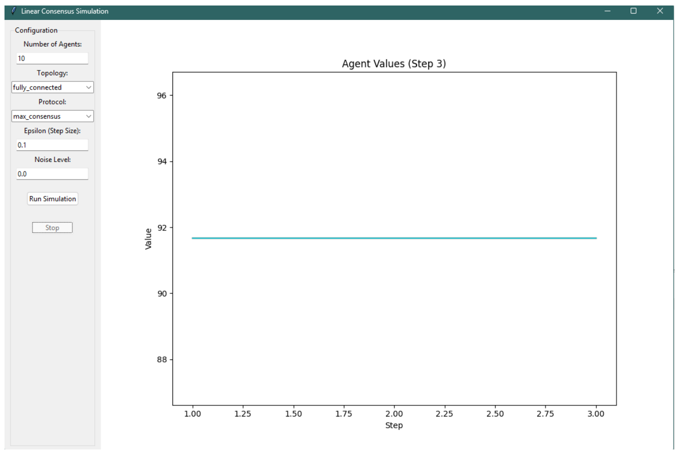
Number of Agents: 10
Topology: ring
Protocol: max_consensus
Epsilon (Step Size): 0.1
Noise Level: 0.0

Fig. 6. Max-consensus on a ring topology with $N = 10$ agents. The maximum value propagates locally and reaches all agents in finite time (five iterations in this example).



Number of Agents: 10
Topology: fully_connected
Protocol: linear
Epsilon (Step Size): 0.1
Noise Level: 0.0

Fig. 5. Linear consensus on a fully connected topology with $N = 10$ agents and $\epsilon = 0.1$. Full connectivity enables convergence within a single synchronous update.



Number of Agents: 10
Topology: fully_connected
Protocol: max_consensus
Epsilon (Step Size): 0.1
Noise Level: 0.0

Fig. 7. Max-consensus on a fully connected topology with $N = 10$ agents. Full connectivity enables immediate propagation of the maximum value and one-step convergence.

connected topology. Since each agent directly observes the global maximum during the first update, the network reaches consensus in a single iteration under synchronous updates.

Overall, these experiments confirm the theoretical predictions: while linear consensus exhibits topology-dependent asymptotic convergence driven by graph connectivity, max-consensus achieves exact agreement in finite time, with convergence speed determined primarily by the graph diameter.

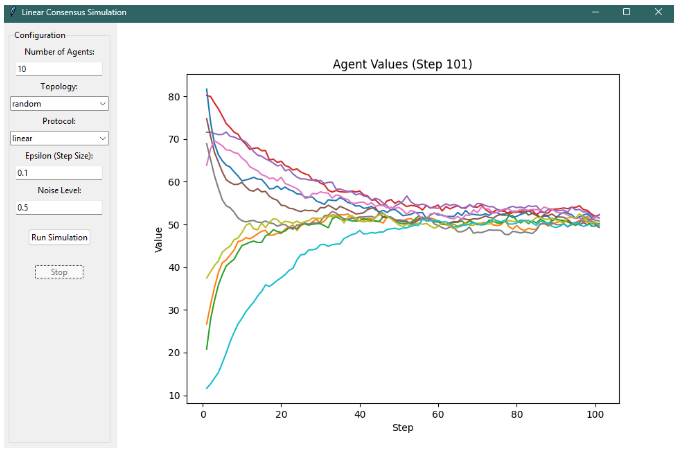
### D. Effect of Measurement Noise

To evaluate the robustness of the consensus protocols to uncertainty, additional simulations were performed on a random communication topology with additive noise applied to agent updates. The noise term is injected locally during state updates and models moderate perturbations in measurements or communication. Unless stated otherwise, the noise level was fixed at $0.5$, while all other parameters were kept identical to the noise-free experiments.

*1) Linear Consensus with Noise:* Figure 8 shows the behavior of the linear consensus protocol under noisy conditions. Although individual agent trajectories exhibit persistent fluctuations, the collective behavior remains stable and converges toward a bounded neighborhood around the average value. The presence of noise prevents exact convergence, resulting instead in sustained oscillations whose amplitude depends on the noise intensity.

This behavior reflects the averaging nature of the linear consensus protocol: repeated local aggregation attenuates random disturbances, yielding bounded-error agreement rather than exact consensus. Such robustness properties are characteristic of linear diffusive dynamics under stochastic perturbations.

*2) Max-Consensus with Noise:* Figure 9 presents the behavior of the max-consensus protocol under the same noisy conditions. As in the noise-free case, the protocol reaches agreement in finite time, with the maximum value propagating rapidly through the network. However, the presence of noise


Linear Consensus Simulation

Configuration

Number of Agents: 10

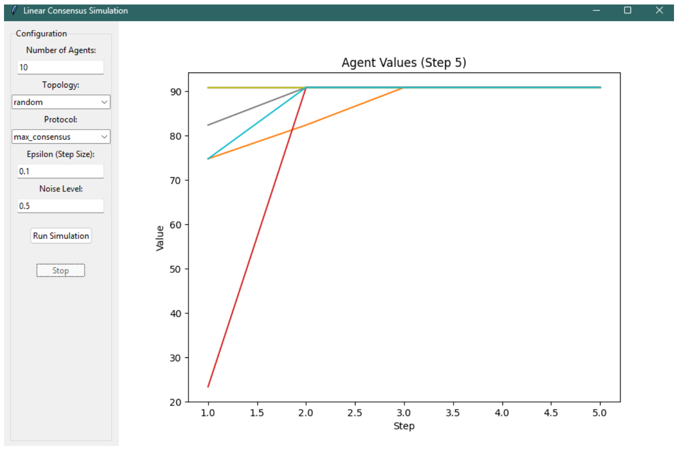
Topology: random

Protocol: linear

Epsilon (Step Size): 0.1

Noise Level: 0.5

Fig. 8. Linear consensus on a random topology with $N = 10$ agents, $\epsilon = 0.1$, and noise level $0.5$. Despite persistent fluctuations, agent values remain clustered around the average after approximately 100 iterations.


Linear Consensus Simulation

Configuration

Number of Agents: 10

Topology: random

Protocol: max_consensus

Epsilon (Step Size): 0.1

Noise Level: 0.5

Fig. 9. Max-consensus on a random topology with $N = 10$ agents, $\epsilon = 0.1$, and noise level $0.5$. The protocol reaches agreement in finite time (five iterations in this example), but the final value reflects sensitivity to noise-induced extrema.

significantly influences which value is ultimately selected as the maximum.

Because the max-consensus update rule amplifies extreme values, transient noise-induced peaks can dominate the final consensus value. While the protocol retains its finite-time convergence property, it is inherently sensitive to measurement noise and outliers, making it less suitable for noisy aggregation tasks without additional filtering or thresholding mechanisms.

## V. Discussion

The simulation results highlight the interplay between consensus protocol design, communication topology, and robustness to uncertainty. Across all experiments, the observed system behavior is consistent with the theoretical convergence properties derived in the preceding sections, providing empirical support for the analytical models.

Linear consensus exhibits asymptotic convergence driven by diffusive averaging of local information. Its convergence speed is strongly influenced by the underlying communication topology: sparse structures, such as ring or random graphs, lead to slow information propagation and extended convergence times, while dense topologies significantly accelerate agreement. Importantly, linear consensus demonstrates inherent robustness to measurement noise. Even under moderate perturbations, agent states remain confined to a bounded neighborhood around the global average, indicating effective attenuation of random disturbances and bounded-error convergence.

In contrast, max-consensus achieves exact agreement in finite time, with convergence speed primarily determined by the diameter of the communication graph. Increased connectivity directly reduces the number of iterations required for agreement, as illustrated by the one-step convergence observed under fully connected topologies. This rapid convergence, however, comes at the cost of robustness. Because the protocol amplifies extreme values, max-consensus is highly sensitive to noise and outliers, and transient perturbations can dominate the final agreed value.

These observations underline a fundamental trade-off between the two protocols. Linear consensus is well suited for distributed averaging and noisy measurement fusion, where robustness and smooth convergence are desirable. Max-consensus, on the other hand, is preferable in applications that require rapid extremum detection or leader selection in low-noise environments, where finite-time convergence outweighs sensitivity to disturbances.

## References

[1] R. Olfati-Saber, J. A. Fax and R. M. Murray, ”Consensus and Cooperation in Networked Multi-Agent Systems,” in *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215-233, Jan. 2007.

[2] W. Ren and R. W. Beard, “Consensus seeking in multiagent systems under dynamically changing interaction topologies,” *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 655–661, May 2005. :contentReference[oaicite:8]index=8

[3] M. Fiedler, “Algebraic connectivity of graphs,” *Czechoslovak Mathematical Journal*, vol. 23, no. 2, pp. 298–305, 1973. :contentReference[oaicite:9]index=9

[4] L. Xiao and S. Boyd, “Fast linear iterations for distributed averaging,” *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, Sep. 2004. :contentReference[oaicite:10]index=10

[5] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Princeton, NJ, USA: Princeton University Press, 2010. :contentReference[oaicite:12]index=12