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

Anja Kovačević, Boris Čuljak
Fakultet / Tehničkih Nauka
Email: boris@mail.com

Abstract—**This document presents the design, implementation, and simulation of linear consensus protocols within a multi-agent system (MAS). We explore the problem of reaching agreement among agents with imperfect measurements and limited communication. Specifically, we implement and compare a standard Linear Consensus protocol against a Max-Consensus protocol using the MESA framework in Python. The goal is to demonstrate how theoretical consensus algorithms can be translated into a working software simulation to analyze their convergence properties and behavior under different network topologies.**

I. Introduction

Multi-agent systems (MAS) consist of multiple interacting intelligent agents that cooperate to achieve a common objective. In this context, an *intelligent agent* refers to an autonomous entity capable of perceiving its local environment, exchanging information with neighboring agents, and making decisions based on predefined rules or algorithms.

MAS are particularly well-suited for solving problems that are difficult or impractical for centralized or monolithic systems. Examples include large-scale sensor networks, where no single node has access to global information, or distributed coordination tasks such as formation control and load balancing, where robustness and scalability are critical.

A fundamental problem in MAS is *consensus*, where a group of agents must agree on a specific value or state using only local measurements and interactions with neighbors. Consensus algorithms enable decentralized coordination without reliance on a central controller.

In this project, we analyze, implement, and simulate a linear consensus protocol in which agents aim to converge to the average of their initial measurements. To provide a comparative perspective, we also implement a Max-Consensus protocol, which allows agents to agree on the maximum initial value in the network.

The structure of this documentation is as follows:

- First, we introduce the **Theoretical Framework**, presenting relevant graph-theoretic concepts and the mathematical update rules for both linear and max-consensus protocols.
- Next, we describe the **Implementation**, explaining how the theoretical models are translated into simulations using the *Mesa* framework in Python, including agent behavior and simulation architecture.
- Finally, we present the **Simulation and Results**, where the behavior, convergence, and performance of the protocols are evaluated across different network topologies.

The theoretical foundation of this work is based on established results in consensus and cooperation in networked multi-agent systems, as discussed by Olfati-Saber *et al.* [1].

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

B. Linear Consensus Protocol

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

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

This task is commonly referred to as the *average consensus* problem.

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 initial state and information received from neighboring agents, and no agent possesses global knowledge of all states or the total number of agents. Furthermore, centralized computation 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 the global average in a fully distributed manner using only local interactions.

1) Mathematical Derivation and Convergence: The linear consensus protocol can be interpreted as a discrete-time approximation of a continuous-time dynamical system. In continuous time, the evolution of the agents' states is commonly modeled as

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

Here, $x_i(t) \in \mathbb{R}$ denotes the state of agent i at time t, which represents a scalar quantity held locally by the agent, such as a sensor measurement, estimate, or internal value. The vector

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

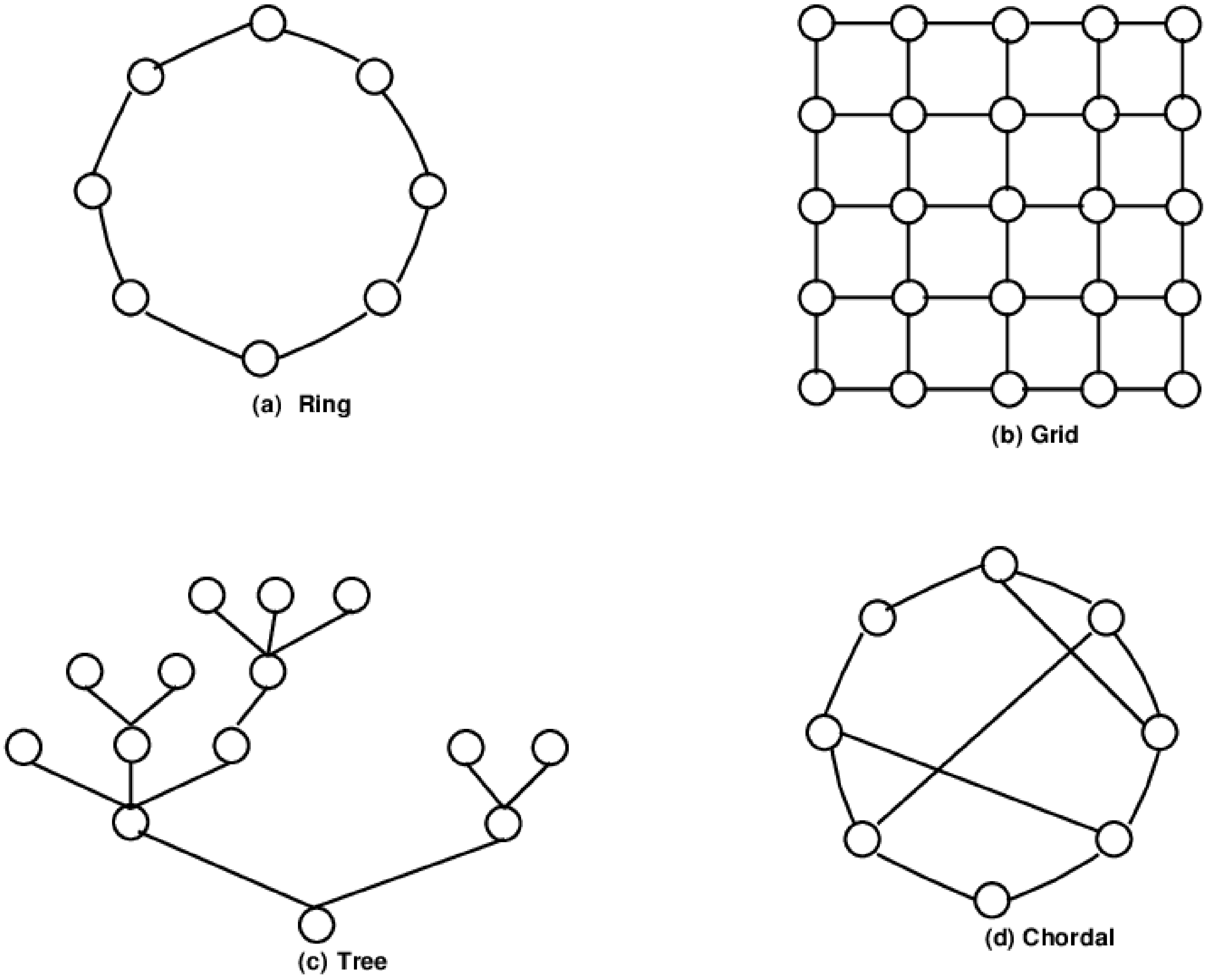


Fig. 1. Examples of common communication topologies used in multi-agent systems: (a) ring, (b) grid, (c) tree, and (d) chordal graph.

collects the states of all N agents at the same time instant into a single global state vector. Although this global 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.

The matrix L appearing in the system dynamics captures how agents exchange information over the communication network. To construct this matrix, the interaction topology is first described using standard graph-theoretic representations.

The communication network is modeled as a graph whose nodes correspond to agents and whose edges represent communication links. From this graph, the *adjacency matrix* $A \in \mathbb{R}^{N\times N}$ is defined to encode neighborhood relationships between agents. Its entries are given by

$$A_{ij} = \begin{cases} 1, & \text{if agent } j \text{ communicates with agent } i, \\ 0, & \text{otherwise.} \end{cases}$$

For example, in a network of three agents where agent 2 communicates with both agents 1 and 3, the adjacency matrix takes the form

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The *degree matrix* $D \in \mathbb{R}^{N\times N}$ is a diagonal matrix that summarizes how many neighbors each agent has. Its diagonal entries are defined as

$$D_{ii} = \deg(i) = \sum_{j=1}^{N} A_{ij},$$

with all off-diagonal entries equal to zero. For the example above, the degree matrix is

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The degree reflects only the number of communication links connected to an agent and does not represent any physical measurement.

Using the adjacency and degree matrices, the *graph Laplacian*[1] is defined as

$$L = D - A.$$

For example, in a network of three agents, the Laplacian matrix is

$$L = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

For an undirected graph, the Laplacian has the general entry-wise form

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

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

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

which is a discrete analogue of a spatial second-order derivative. This operation quantifies how different the state of agent i is from the average state of its neighbors.

In this sense, the graph Laplacian does not represent a simple difference between two scalar values, but rather a structured operator that captures local disagreement across the network. Consensus is achieved when these local differences vanish, i.e., when $Lx(t) = 0$, meaning that all neighboring agents hold identical states.

Discretizing the continuous-time system using a forward Euler method with step size $\epsilon > 0$ yields the discrete-time 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 i denotes the updating agent and j indexes its neighboring agents. The summation term represents a local correction based on state differences between agent i and its neighbors.

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

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

The matrix P arises directly from the Laplacian-based update rule and describes how each agent forms a weighted average of its own state and those of its neighbors at each time step.

For sufficiently small step sizes $\epsilon < 1/\Delta_{\max}$, where $\Delta_{\max}$ is the maximum node degree, the matrix P is *doubly stochastic*, meaning that the sum of each row and each column equals one. This property has important consequences for convergence.

[1]The term *graph Laplacian* is conceptually distinct from the Laplace transform used in control theory. It originates from the classical Laplacian operator ∇^2, which measures local variation of a quantity in continuous space and appears in diffusion and heat equations. The graph Laplacian is a discrete analogue of this operator defined on a network, quantifying how the state of each agent differs from those of its neighbors.

First, invariance of the sum of states follows directly from double stochasticity:

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

This implies that the total mass of the system is conserved over time, and therefore any common agreement value must equal the average of the initial states.

Second, repeated multiplication by P reduces disagreements among agents. In particular, the variance of the agent states, which measures how far individual states deviate from their mean, decreases monotonically over time. As $k \to \infty$, the matrix power P^k converges to a rank-one matrix of the form

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

where $\mathbf{1}$ is the vector of all ones. A rank-one matrix maps any initial state vector to a vector with identical entries, implying that all agents converge to the same value.

Together, these properties guarantee that the linear consensus protocol converges asymptotically to the average of the initial agent states.

C. Max-Consensus Protocol

While the linear consensus protocol aims to compute the average of the agents' initial states through iterative local averaging, the Max-Consensus protocol addresses a different coordination objective. Instead of aggregating 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

$$x_{\max} = \max_i x_i(0).$$

This protocol is particularly useful in applications such as leader election, alarm triggering, or detection of extreme measurements, for example identifying the highest temperature or load in a distributed sensor network.

1) Mathematical Derivation and Convergence: Unlike linear consensus, Max-Consensus is governed by a nonlinear update rule that relies on comparison rather than averaging. Each agent updates its state by selecting the largest value among itself and its neighbors:

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

This update rule ensures that information flows monotonically from agents holding larger values toward the rest of the network.

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

1) **Monotonicity:** The state of each agent is non-decreasing over time. An agent updates its value only when it observes a strictly larger value from one of its neighbors.

2) **Propagation of the maximum:** The global maximum initial value acts as a source that propagates through the network. At each iteration, agents adjacent to the current holders of the maximum adopt this value.
3) **Finite-time convergence:** For a connected network, the maximum value reaches all agents in at most D iterations, where D is the diameter of the graph. In contrast to linear consensus, which converges asymptotically, Max-Consensus achieves exact agreement in finite time.

b) Comparison with linear consensus.: The choice between linear and Max-Consensus depends on the desired coordination objective. Linear consensus is appropriate when agents must compute an aggregate quantity, such as an average, and when smooth convergence is acceptable. Max-Consensus, on the other hand, is preferable when the goal is to identify or propagate an extreme value rapidly and exactly, such as in leader selection or threshold-based decision making.

D. Illustrative Example

To make these concepts concrete, consider a simple network of 3 agents connected in a line: $1-2-3$. **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 $\frac{10+20+5}{3} = 11.67$. The global maximum is 20.

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

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

  State: $[12, 15, 8]$. Sum: 35 (Invariant).
- **k=2**:
  - Agent 1: $12 + 0.2(15-12) = 12.6$
  - Agent 2: $15 + 0.2((12-15) + (8-15)) = 15 + 0.2(-3-7) = 13$
  - Agent 3: $8 + 0.2(15-8) = 9.4$

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

Notice how the values are pulling closer together towards the average of 11.67.

2) Max-Consensus Walkthrough:

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

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

In this specific topology and initialization, consensus is reached in just 1 step because the center node held the maximum. If Agent 1 held the max, it would take 2 steps to reach Agent 3.

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, separating domain logic, models, and simulation control.

A. Agent Communication and Model

In our simulation, agents do not send messages over a network socket but rather "communicate" by inspecting the state of their neighbors via the shared environment (the Model). This abstraction is handled by the `ConsensusAgent` class.

At each step, an agent queries the model's grid to identify its neighbors. This corresponds to the set N_i defined in the Theoretical Framework.

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)
```

By passing `neighbor_states` to the protocol, we simulate the transmission of $x_j(k)$ values from neighbors $j \in N_i$ to agent i.

B. Protocols

The consensus logic is encapsulated in strategy classes implementing the `AbstractProtocol` interface. This allows us to switch between different algorithms easily.

1) Linear Consensus Implementation: The `LinearConsensus` class implements the update rule described in Equation (1). As defined in the theory section, the new state is the current state plus a weighted sum of differences.

In the code, this is directly translated as follows:

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
```

This snippet shows the direct application of the theoretical formula $x_i(k) + \epsilon \sum (x_j(k) - x_i(k))$.

2) Max-Consensus Implementation: Similarly, the `MaxConsensus` class implements the update rule from Equation (2). The theory states that an agent should update its value to the maximum of its own value and its neighbors' values.

The implementation is concise:

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)
```

This effectively realizes the discrete-time update $x_i(k+1) = \max(x_i(k), \max_{j \in N_i} x_j(k))$.

C. Simulation Model (`ConsensusModel`)

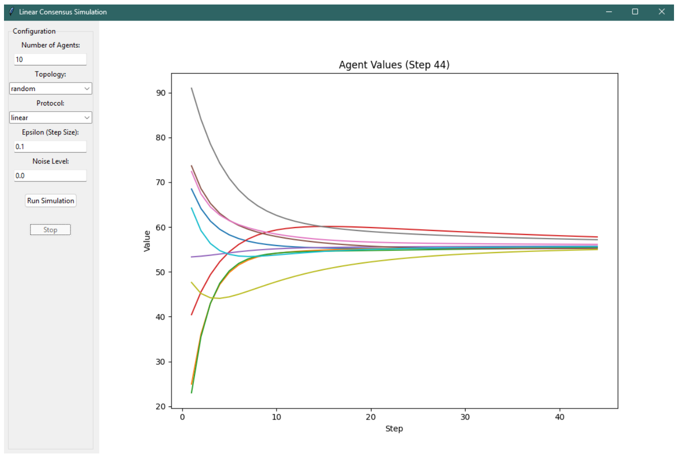
The `ConsensusModel` class manages the global state of the simulation. It is responsible for:

- **Network Initialization**: Creating the interaction graph using `networkx`. Supported topologies include Random (Erdos-Renyi), Ring, and Fully Connected graphs.
- **Agent Creation**: Instantiating agents with random initial values and placing them on the network grid.
- **Scheduling**: Executing the `step` and `advance` methods for all agents in each simulation cycle.

IV. Simulation and Results

The proposed consensus protocols were implemented and evaluated using an interactive simulation environment based on the Mesa framework. The simulator allows the user to configure the number of agents, communication topology, consensus protocol, step size, and noise level through a graphical interface.

Unless stated otherwise, all simulations were performed with $N = 10$ agents, zero measurement noise, and a fixed step size $\epsilon = 0.1$. 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.


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 44 iterations.

A. Simulation Setup

The control panel on the left-hand side of the interface allows selection of the network topology (random, ring, or fully connected), the consensus protocol (linear or max-consensus), and the numerical parameters governing the update rules. The main plotting area visualizes the evolution of agent states over discrete time steps.

B. Protocol Comparison

Figure 2 shows the behavior of the linear consensus protocol on a random communication topology. Agents are initialized with heterogeneous values drawn uniformly from the interval $[0, 100]$. Despite large initial discrepancies, all agent states gradually converge toward a common agreement value corresponding to the global average.

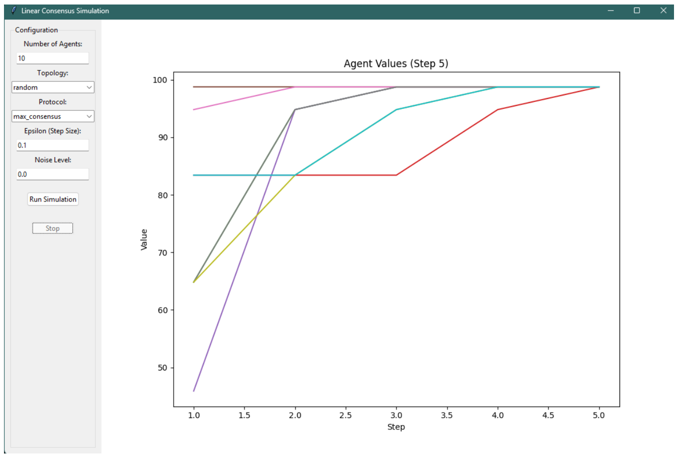
As expected from the theoretical analysis, convergence is asymptotic: the disagreement between agents decreases monotonically over time, but exact equality is achieved only in the limit. The relatively slow convergence observed here is characteristic of random and sparse topologies, where information diffusion depends on the algebraic connectivity of the graph.

In contrast, Figure 3 illustrates the max-consensus protocol under the same conditions. Instead of gradual averaging, the maximum initial value propagates through the network. Agents update their states only when they observe a higher value from a neighbor, resulting in a monotonic increase of individual states.

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

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,


Linear Consensus Simulation

Configuration

Number of Agents: 10

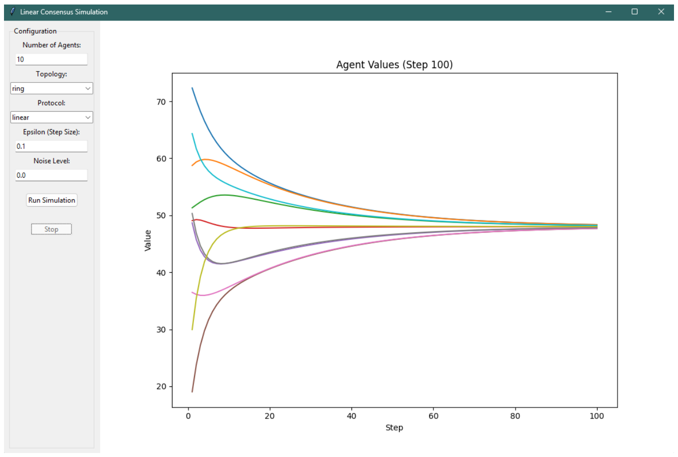
Topology: random

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 (5 steps).


Linear Consensus Simulation

Configuration

Number of Agents: 10

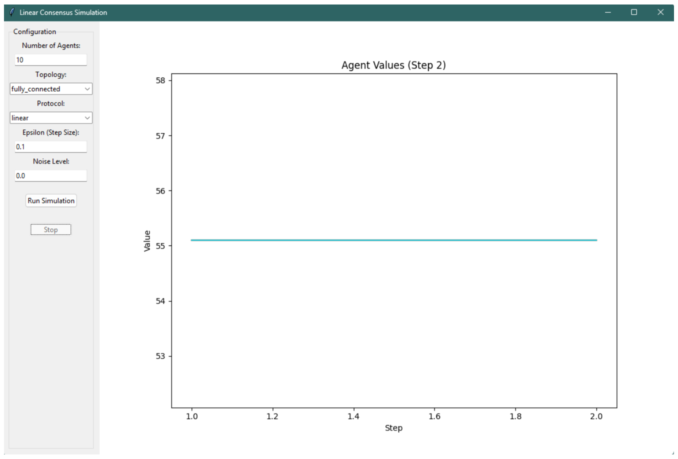
Topology: ring

Protocol: linear

Epsilon (Step Size): 0.1

Noise Level: 0.0

Fig. 4. Linear consensus on a ring topology with $N = 10$ agents and $\epsilon = 0.1$. Convergence is slow due to sparse local connectivity, requiring approximately 100 iterations to approach consensus.


Linear Consensus Simulation

Configuration

Number of Agents: 10

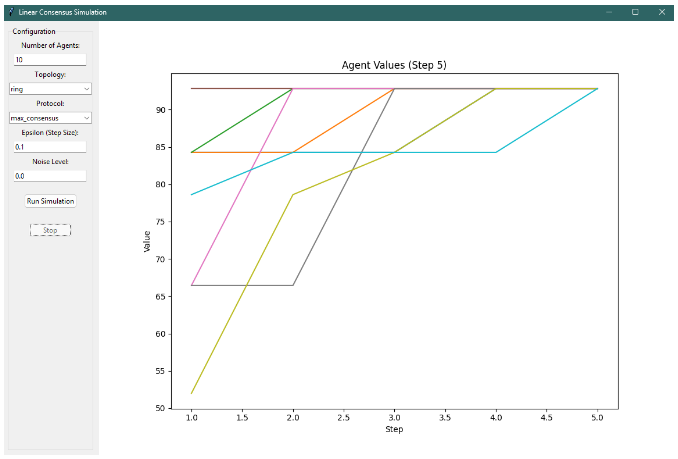
Topology: fully_connected

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 $\epsilon = 0.1$. Direct communication between all agents leads to immediate convergence in one iteration.


Linear Consensus Simulation

Configuration

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 (5 iterations).

both linear and max-consensus protocols were evaluated under different network structures, namely ring and fully connected topologies, while keeping all other parameters fixed.

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

In contrast, Figure 5 demonstrates linear consensus on a fully connected topology. In this case, each agent directly exchanges information with all others, effectively accessing the global state distribution at each iteration. This results in immediate convergence, with all agents reaching the consensus value within a single update step.

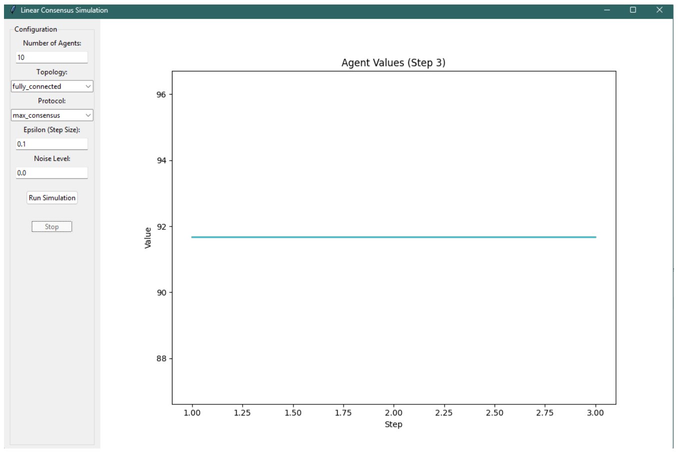
These results highlight the strong dependence of linear consensus convergence speed on the algebraic connectivity of the communication graph.

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

Figure 7 presents the max-consensus protocol on a fully connected topology. Since each agent directly observes the global maximum at the first update, consensus is achieved immediately, requiring only a single iteration.

D. Effect of Measurement Noise

To evaluate the robustness of the consensus protocols to uncertainty, additional simulations were performed on a random


Linear Consensus Simulation

Configuration

Number of Agents: 10

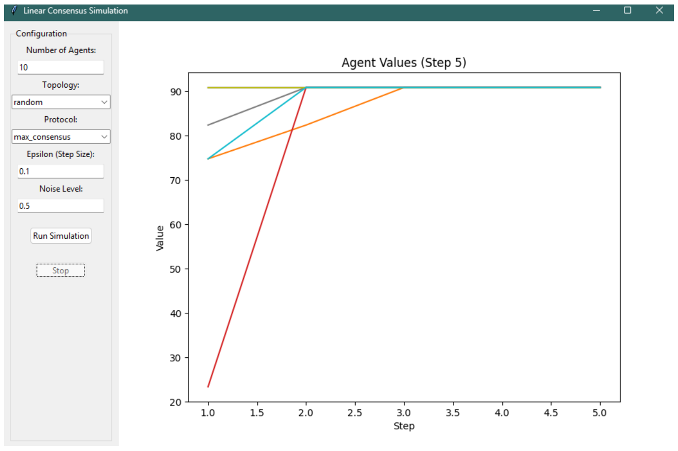
Topology: fully_connected

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.


Linear Consensus Simulation

Configuration

Number of Agents: 10

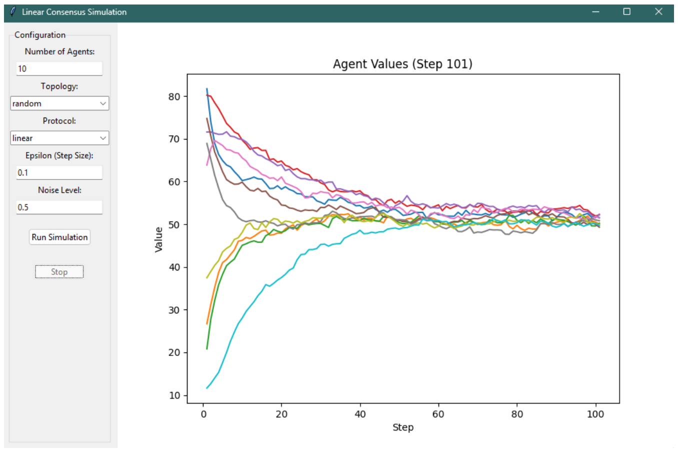
Topology: random

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 converges in finite time (5 iterations), but the final value reflects sensitivity to noise-induced extrema.


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.

communication topology with additive noise applied to agent updates. The noise level was set to 0.5, representing moderate perturbations in local measurements or communication.

1) Linear Consensus with Noise: Figure 8 shows the behavior of the linear consensus protocol under noisy conditions. Although agent trajectories exhibit noticeable fluctuations, the collective behavior still converges toward a common neighborhood around the average value. The presence of noise prevents exact convergence, resulting instead in bounded oscillations around the consensus point.

This behavior reflects the averaging nature of the linear consensus protocol, which attenuates random disturbances through repeated local aggregation.

2) Max-Consensus with Noise: Figure 9 presents the max-consensus protocol under the same noisy conditions. As in the noise-free case, the maximum value propagates rapidly through the network, and consensus is reached in finite time. However, noise influences which value is identified as the maximum, potentially amplifying transient fluctuations or spurious peaks.

V. Discussion

The conducted simulations highlight the interplay between consensus protocol design, network topology, and robustness to noise. Across all experiments, the observed behaviors closely align with the theoretical convergence properties derived earlier, providing empirical validation of the underlying models.

Linear consensus exhibits asymptotic convergence driven by diffusive averaging. Its convergence speed is strongly influenced by the communication topology: sparse structures such as ring or random graphs lead to slow information propagation and long convergence times, while fully connected networks enable near-instantaneous agreement. Importantly, linear consensus demonstrates inherent robustness to noise. Even under moderate perturbations, agent states remain clustered around the global average, indicating bounded-error convergence and effective attenuation of random disturbances.

Max-consensus, in contrast, achieves exact agreement in finite time, with convergence speed governed primarily by the diameter of the communication graph. Increased connectivity directly reduces convergence time, as observed in the immediate agreement achieved under fully connected topologies. However, this speed comes at the cost of robustness: max-consensus is highly sensitive to noise and outliers, as transient fluctuations can dominate the final agreed value. As a result, while max-consensus is efficient for extremum detection and leader election, it is less suitable for noisy measurement fusion tasks.

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.