Federated Learning Architectures: Comparative Analysis Report

Executive Summary

This report presents a comprehensive comparative analysis of multiple federated learning architectures implemented and evaluated on the MNIST dataset under Non-IID (label skew) data distribution.

The study primarily analyzed on following

1. **Algorithmic convergence**: How quickly each method reaches high accuracy – Global Model (average) Accuracy vs Number of rounds of communication
2. **System speed**: Total virtual wall-clock time to convergence – Global Model (average)Accuracy vs Time taken to converge
3. **Network bandwidth**: Total data volume transferred – Global Model (average) Accuracy vs Data volume transferred
4. **Control message overhead**: Number of messages exchanged – Number of messages per round across different architectures.

Please note that for centralized federated learning, we have a global model which is getting trained vs for a decentralized federated learning, we take average of accuracy across different models (in nodes).

A round of communication here can be defined as following

1. For centralized federated learning approaches -
   Step 1 (Downlink): A central server selects a subset of clients and broadcasts the Global Model weights/bias to them.
   Step 2 (Local Execution): Clients perform SGD on their own data using global model weights as the starting point for E epochs.
   Step 3 (Uplink): Clients send their updated weights (or gradients) back to the server. The server averages them to create new weights.

In our case all of nodes/servers are processes

2. For Decentralized learning -
   Step 1 (Peer & Layer selection): A node selects some i neighbour node, the node selection here can be smart or some random logic. We have implemented multiple algorithms for same.
   Step 2 (Pulling model) - A specific node shall pull set of layers or complete model from other neighbour node.
   Step 3 (Local training and aggregation) - Node averages models received from peers and does one round of SGD to arrive at updated model.

The architectures studied include:

1. **FedAvg (Centralized)**: Standard centralized federated averaging algorithm
2. **FedProx (Centralized)**: Centralized federated learning with proximal regularization
3. **Decentralized Baseline**: Decentralized learning with random peer selection
4. **Decentralized Handshake - Asynchronous**: Decentralized learning with smart peer selection and sharing specific layers. Multiple versions of same have been implemented such as 1. Multi Peer interaction 2. Layer aggregation 3. Model specific param changes etc.
5. **Decentralized Handshake - Synchronous**: Decentralized learning with synchronous communication along with smart peer selection and sharing specific layers. Multiple versions of same have been implemented such as 1. Single vs Multi Peer interaction 2. Layer aggregation 3. Model specific param changes etc.
6. **MPLS List Scheduling** [3]: Custom decentralized protocol with similarity-based peer selection and handshake exchange as explained in [3] (Minor differences in implementation given the test setup was limited to a laptop).

The experiments were conducted using a Hybrid Clock mechanism that tracks both real compute time and simulate network delays (given the experiment is performed on a local computer), enabling fair comparison across different communication patterns.

Key Findings: -

- FedAvg achieved the highest final accuracy (96.20%) but required the most data transfer (24.68 MB) and highest virtual time (1205.05s)
- FedProx (when implemented) shows similar convergence to FedAvg with more virtual time and similar data transfer.
- Decentralized Baseline demonstrated faster convergence in terms of virtual time (945.42s) but lower final accuracy (43.36%) and reduced data transfer compared to centralized but still high at 19MB.
- Decentralized asynchronous handshake showed significant bandwidth savings (3.54 MB, 85% reduction) but required more control messages (588 messages) due to metadata overhead and achieved poor convergence (35.14% accuracy)
- Decentralized synchronous handshake showed bandwidth savings similar to decentralized asynchronous but there was lot of wait time and more messgaes due to metadata overhead and it achieved slightly better convergence than decentralized asynchronous (41.14% accuracy)
- Both the decentralized versions had some variations in results with different variations.
- Pseudo implementation of MPLS had a higher data transfer of around 8-10 MB but less than centralized versions and final accuracy with 50 rounds was around 70.34%.

As part of project, several critical implementation challenges associated with Federated Learning on Non-IID data were observed such as

- Model Convergence and Client Drift: The aggregation strategy (e.g., standard averaging vs. alternative techniques) is pivotal. We observed that merging weights from heterogeneous data sources can cause ‘client drift,’ leading to performance degradation rather than improvement.
- Transmission Strategy: Determining which model components to synchronize—specifically, choosing between convolutional layers, classification heads, or gradient updates—requires balancing communication costs with model efficacy.
- Communication Complexity: The architecture of peer-to-peer interaction (single vs. multi-peer) introduces significant complexity regarding deterministic behavior, handshake protocols, and peer identification mechanisms.

1. Problem Statement

The objective of this is to implement and compare the performance of centralized and decentralized Federated Learning architectures: Centralized Federated Learning (FedAvg), Centralized Federated Learning (FedProx), Decentralized Baseline (Random), synchronous and non synchronous decentralized smart peer handshake algorithms, and MPLS-style Decentralized Federated Learning.

The goal is to analyze the trade-offs between: - **Algorithmic convergence**: How quickly each method reaches high accuracy - **System speed**: Total virtual wall-clock time to convergence - **Network bandwidth**: Total data volume transferred - **Control message overhead**: Number of messages exchanged

This problem is implemented in Python utilizing inter-process communication. The system simulates a Non-IID data environment (label skew) where each node holds data from only 2 classes, demonstrating the robustness of smart peer selection algorithms.

1.1 Model Specifications

The system consists of N nodes, where each node p_i holds a partition of the training dataset (MNIST) and maintains a local LeNet-5 model. The system operates under one of five protocols, each with different communication patterns and aggregation strategies.

1.2 Hybrid Clock Mechanism

Since this simulation runs on a single machine where processes share a CPU, standard “Wall Clock” measurements (`time.time()`) are inaccurate due to OS scheduling overhead. We implement a **Virtual Hybrid Clock** for each node to track time accurately:

- **Real Compute Time (α_measured)**: Measured using `time.process_time()` to capture actual CPU time consumed by LeNet-5 training and peer selection algorithms

- **Simulated Network Time (β_calculated)**: Calculated transmission delay based on payload size and Link Bandwidth (B)

Virtual Time_i = Σ(α_measured + Payload_Size / Link_Bandwidth)

This ensures accurate time measurements in a single-machine simulation environment.

1.3 Protocols & Execution

Each protocol follows a two-phase execution model:

Phase 1: COMPUTE - Train local model for E epochs - Select peer (protocol-dependent) - Prepare payload (full model or partial layers) - Measure real CPU time using `time.process_time()`

Phase 2: NETWORK - Calculate payload size in bits - Simulate network delay: `network_delay = payload_bits / Link_Bandwidth` - Update virtual clock

---

2. Design and Implementation

2.1 System Architecture

The implementation uses a multi-process architecture with shared memory for inter-process communication:

- **Process Management**: `torch.multiprocessing` with 'spawn' method
- **Shared Memory**: Manager-based shared dictionaries and lists for model exchange
- **Synchronization**: Lock-based synchronization for atomic operations
- **Logging**: Per-process logging to individual log files

2.2 Algorithm Implementations

2.2.1 FedAvg (Federated Averaging)

Algorithm: Federated Averaging (FedAvg) - McMahan et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data", AISTATS 2017

Protocol: Centralized Federated Learning with Federated Averaging

Protocol Details: - **Architecture**: Star topology with coordinator (rank 0) - **Payload**: Full model (upload) + Full model (download) - **Overhead**: 0 messages (no metadata) - **Aggregation**: Weighted average of all N models at coordinator

Algorithm Steps:

1. Each worker trains local model for E epochs

2. Workers upload full model to coordinator

3. Coordinator aggregates: `w_global = (1/N) * Σ(w_i)`

4. Coordinator broadcasts aggregated model to all workers

5. Workers update local model with global model

Message Complexity: 2N messages per round (N uploads + N downloads)

Data Volume: 2 × Model_Size × N per round

2.2.2 FedProx (Federated Proximal)

Algorithm: Federated Proximal (FedProx) - Li et al., “Federated Optimization in Heterogeneous Networks”, MLSys 2020

Protocol: Centralized Federated Learning with Proximal Regularization

Protocol Details: - **Architecture**: Star topology with coordinator (rank 0) - **Payload**: Full model (upload) + Full model (download) - **Overhead**: 0 messages (no metadata) - **Aggregation**: Weighted average with proximal term

Key Difference from FedAvg: - Local objective includes proximal term: `F_k(w) + (μ/2)||w - w_global||²` - This regularization improves convergence under Non-IID data distribution - Proximal coefficient μ = 0.1 (configurable)

Algorithm Steps:

1. Coordinator broadcasts global model w_global

2. Each worker trains with proximal regularization

3. Workers upload updated models to coordinator

4. Coordinator aggregates and broadcasts new global model

Message Complexity: 2N messages per round (same as FedAvg)

Data Volume: 2 × Model_Size × N per round (same as FedAvg)

2.2.3 Decentralized Baseline (Random Peer Selection)

Algorithm: Decentralized Federated Learning with Random Peer Selection (Baseline)

Protocol: Peer-to-peer decentralized learning with uniform random peer selection

Protocol Details: - **Architecture**: Peer-to-peer mesh topology - **Payload**: Full model (bidirectional exchange) - **Overhead**: 0 messages (no metadata) - **Peer Selection**: Uniform random selection from available peers

Algorithm Steps:

1. Each node trains local model for E epochs

2. Each node randomly selects a peer from available nodes

3. Nodes exchange full models with selected peers

4. Each node aggregates: `w_local = 0.5 × w_local + 0.5 × w_peer`

5. Nodes update local models

Message Complexity: 2N messages per round (N send + N receive)

Data Volume: 2 × Model_Size × N per round

2.2.4 Decentralized Federated Learning with Asynchronous Handshake-Based Exchange

Protocol: Peer-to-peer decentralized learning with cosine similarity-based smart peer selection and asynchronous handshake-based partial model exchange

Protocol Details: - **Architecture**: Peer-to-peer mesh topology (asynchronous execution) - **Payload**: Partial model (layers: fc2, fc3) + Metadata - **Overhead**: 2C messages per round (C query requests + C query responses) - **Peer Selection**: Cosine similarity-based selection from candidate set of size C

Algorithm Steps:

1. Each node trains local model for E epochs

2. Each node stores full model state in shared memory for candidate queries

3. **Smart Peer Selection**: - Sample C random candidates from available peers - For each candidate, read their model from shared memory - Calculate cosine similarity on fc3 layer: `similarity = dot(w1, w2) / (||w1|| × ||w2||)` - Select peer with minimum similarity (for diversity)

4. **Asynchronous Handshake Phase**:

Node A prepares partial model (fc2, fc3 layers only)

Node A places partial model in shared memory slot

Node A polls selected peer's shared memory slot (asynchronous waiting, 0.2s intervals)

Node B independently places its partial model when ready

When both models available, both nodes read each other's partial models - Nodes clear slots after reading

Timeout: if peer's model not available within 30s, skip round

5. **Partial Aggregation**: Aggregate only fc2 and fc3 layers: `w_layer = 0.5 × w_local + 0.5 × w_peer` 6. Update local model with aggregated partial layers

Message Complexity: (2C + 2) × N messages per round - 2C messages for metadata queries (C candidates × 2 messages each) - 2 messages for partial model exchange (send + receive)

Data Volume: (Partial_Model_Size + C × Metadata_Size) × N per round - Partial model: Only fc2 and fc3 layers - Metadata: 1024 bits per candidate query

2.2.5 Decentralized Federated Learning with synchronous Handshake-Based Exchange

Algorithm: Decentralized Federated Learning with Synchronized Multi-Peer Handshake and Gradient-Based Layer Selection

Protocol: Peer-to-peer decentralized learning with gradient divergence-based peer selection, gradient-based layer selection, multi-peer aggregation, and synchronized barriers

Protocol Details: - **Architecture**: Peer-to-peer mesh topology with synchronization barriers -

Payload: Partial model (selected layers: fc1, fc2, fc3, optionally conv layers) + Metadata

Overhead: 2C messages per round (C query requests + C query responses)

Peer Selection: Gradient divergence-based selection from candidate set of size C (selects multiple peers)

Layer Selection: Gradient magnitude-based or divergence-based layer selection
Synchronization: Deadlock-free barriers at training, model placement, exchange, and round completion

Algorithm Steps:

1. Each node trains local model for E epochs

2. **Synchronization Barrier**

    1: Wait for all nodes to complete training (quorum-based, 50% minimum)

    2. **Synchronization Barrier** Wait for all nodes to complete training (quorum based, 50% minimum)

    3. **Layer Selection**: - Compute gradient magnitudes for each layer - Select layers with highest gradient activity (e.g., fc1, fc2, fc3) - Re-select every 5 rounds based on current gradients

    4. **Peer Selection**: - Sample C random candidates from available peers - For each candidate, calculate gradient divergence per layer: `divergence(l) = ||∇_local(l) - ∇_peer(l)||` - Select multiple peers (default: 1-3) with highest total divergence - Calculate peer selection probabilities based on divergence

5. **Layer Selection (Divergence-Based)**: - Aggregate layer divergences across selected peers - Select layers with maximum divergence across peers - Verify partial state contains all expected layers

6. **Synchronization Barrier 2**: Wait for all nodes to place models in shared memory

7. **Multi-Peer Exchange**: - Place partial model in shared memory for all selected peers - Wait for partial models from all selected peers (with timeout) - Receive and verify partial models from multiple peers

8. **Synchronization Barrier 3**: Wait for all nodes to complete exchange

9. **Multi-Peer Aggregation**: - Aggregate layers from multiple peers: `w_local(l) = Σ_s (weight_s × w_peer_s(l))` - Weights based on data size or uniform (0.5 per peer) - Only selected layers are aggregated, others remain unchanged

10. **Synchronization Barrier 4**: Wait for all nodes to complete round before next round 11. Update local model with aggregated layers

Message Complexity: (2C + 2P) × N messages per round - 2C messages for metadata queries (C candidates × 2 messages each) - 2P messages for multi-peer exchange (P peers × 2 messages: send + receive)

Data Volume: (Partial_Model_Size + C × Metadata_Size) × P × N per round - Partial model: Selected layers (fc1, fc2, fc3, optionally conv) - Metadata: 1024 bits per candidate query - P = number of selected peers per node - Total: Partial model size × P peers per node × N nodes

2.2.6 MPLS List Scheduling

Algorithm: Decentralized Federated Learning with List Scheduling-Based Layer Assignment

Protocol: Peer-to-peer decentralized learning with multi-factor peer selection and list scheduling algorithm for optimal layer-to-peer assignment

Protocol Details:

Architecture: Peer-to-peer mesh topology with multi-peer aggregation

Payload: Selected layers from multiple peers (different layers from different peers) -

Overhead: 2C messages per round (C query requests + C query responses)

Peer Selection: Multi-factor probability-based selection (bandwidth + data divergence) -

Layer Assignment: List scheduling algorithm maps layers to peers to minimize communication delay

Algorithm Steps:

1. Each node trains local model for E epochs

2. Each node stores full model state in shared memory for queries

3. **Peer Selection**: - Sample C random candidates from available peers - For each candidate, calculate bandwidth (Bsk) and data divergence (DDs = Σ_c |pic - psc|) - Calculate peer selection probability: `psk = (τ1 × Bsk + τ2 × DDs) / Σ_s' (τ1 × Bs'k + τ2 × DDs')` - Select peers probabilistically based on psk

4. **Layer Selection**: - For each layer l and each peer s, calculate gradient variation: `gst',t(l) = ||wst(l) - wst'(l)||_2` - Calculate layer probability: `qsk(l) = gst',t(l) / Σ_s' gs't',t(l)`

5. **List Scheduling Algorithm**: - Calculate combined probability: μ2(s,l) = psk × qsk(l) - Calculate download time: μ1(s,l) = Ml / Bsk (Ml = layer size, Bsk = bandwidth) - Calculate efficiency metric: E(s,l) = φ(l) / μ1(s,l) where φ(l) = min_s μ1(s,l) - Assign layers to peers using list scheduling to minimize total communication delay - Result: Assignment map π(l) = s (layer l comes from peer s)

6. **Multi-Peer Aggregation**: - Download each layer l from its assigned peer s - Aggregate: `w_local(l) = 0.5 × w_local(l) + 0.5 × w_peer_s(l)` - Unassigned layers remain unchanged 7. Update local model with aggregated layers

Message Complexity: (2C + L) × N messages per round - 2C messages for metadata queries (C candidates × 2 messages each) - L messages for layer downloads (one per assigned layer)

Data Volume: Σ_l Ml × N per round - Ml = size of layer l - N = number of nodes - Total: Sum of sizes of all assigned layers (typically full model size)

2.2.7 Decentralized Federated Learning with Deterministic Pairing (not pursued)

Algorithm: Custom Decentralized Federated Learning Protocol

Inspired by: Deterministic pairing/matching strategies used in synchronous decentralized learning (similar to deterministic gossip, match-allreduce, or structured peer-exchange matrices in distributed optimization literature)

Protocol: Peer-to-peer decentralized learning with deterministic round-based pairing and partial model aggregation

Protocol Details: - **Architecture**: Peer-to-peer mesh topology with global coordination - **Payload**: Partial model (layers: fc2, fc3) - **Overhead**: Coordination messages for pairing (minimal) - **Peer Selection**: Deterministic pairing using round-based shuffling

Algorithm Steps:

1. Each node trains local model for E epochs

2. **Deterministic Pairing**: - All nodes signal readiness in shared queue - When all N nodes are ready, coordinator shuffles and pairs nodes - Each node receives assigned peer from pairing map

3. Nodes exchange partial models with assigned peers

4. **Partial Aggregation**: Aggregate only fc2 and fc3 layers 5. Update local model with aggregated partial layers

Message Complexity: ~2N messages per round (pairing coordination + model exchange)

Data Volume: 2 × Partial_Model_Size × N per round

Pairing Strategy: - Nodes are shuffled deterministically each round - Pairs are formed: (shuffled[0], shuffled[1]), (shuffled[2], shuffled[3]), … - Ensures all nodes are paired without conflicts

2.3 Implementation Details

2.3.1 LeNet-5 Model Architecture

```
Input (28×28×1)
  ↓
Conv2d(1→6, kernel=5, padding=2) + ReLU
  ↓
AvgPool2d(kernel=2, stride=2)
  ↓
Conv2d(6→16, kernel=5) + ReLU
  ↓
AvgPool2d(kernel=2, stride=2)
  ↓
Flatten → Linear(400→120) + ReLU
  ↓
Linear(120→84) + ReLU
  ↓
Linear(84→10) → Output
```

Model Size: ~61,706 parameters - Full model: ~1.97 Mbits (32-bit floats) - Partial model (fc2 + fc3): ~0.30 Mbits

2.3.2 Non-IID Data Distribution

Each node receives data from exactly 2 classes (label skew): - Total classes: 10 (digits 0-9) - Classes per node: 2 - Distribution: Round-robin assignment ensuring all classes are covered

This creates heterogeneity that challenges convergence, especially for decentralized methods.

2.3.3 Hybrid Clock Implementation

```
class HybridClock:
    def start_compute(self):
        return time.process_time()

    def end_compute(self, t_start):
        compute_time = time.process_time() - t_start
        self.virtual_time += compute_time
        return compute_time

    def add_network_delay(self, payload_bits, bandwidth_mbps):
        network_delay = payload_bits / (bandwidth_mbps * 1e6)
        self.virtual_time += network_delay
        return network_delay
```

---

3. Experimental Setup

3.1 Hardware and Software Environment

- **Platform**: macOS (darwin 24.6.0), M2 Processor with 16 cores
- **Python Version**: 3.7+
- **Framework**: PyTorch with torch.multiprocessing
- **Dataset**: MNIST (60,000 training samples, 10,000 test samples)

3.2 Experimental Parameters

All experiments use the following fixed parameters (from `inp-params.txt`):

Parameter	Value	Description
N	10	Number of nodes/processes
R_max	50	Maximum communication rounds
E	5	Local training epochs per round
B	10.0	Link bandwidth (Mbps)
C	5	Candidate set size (for MPLS)
MU	0.1	FedProx proximal coefficient

3.3 Data Distribution

- **Training Data**: 60,000 samples distributed across 10 nodes
- **Test Data**: 10,000 samples (shared across all nodes for evaluation)

- **Non-IID**: Each node receives data from exactly 2 classes
- **Batch Size**: 32 (training), 64 (testing)

3.4 Training Configuration

- **Optimizer**: SGD (Stochastic Gradient Descent)
- **Learning Rate**: 0.01
- **Loss Function**: Cross-Entropy
- **Model**: LeNet-5 (61,706 parameters)

3.5 Evaluation Methodology

- **Accuracy Metric**: Global test accuracy on shared test set
- **Runs per Protocol**: 5 independent runs
- **Averaging**: Mean across runs for all metrics
- **Statistics Collected**:
    - Accuracy per round
    - Virtual time per round
    - Cumulative data volume per round
    - Cumulative message count per round

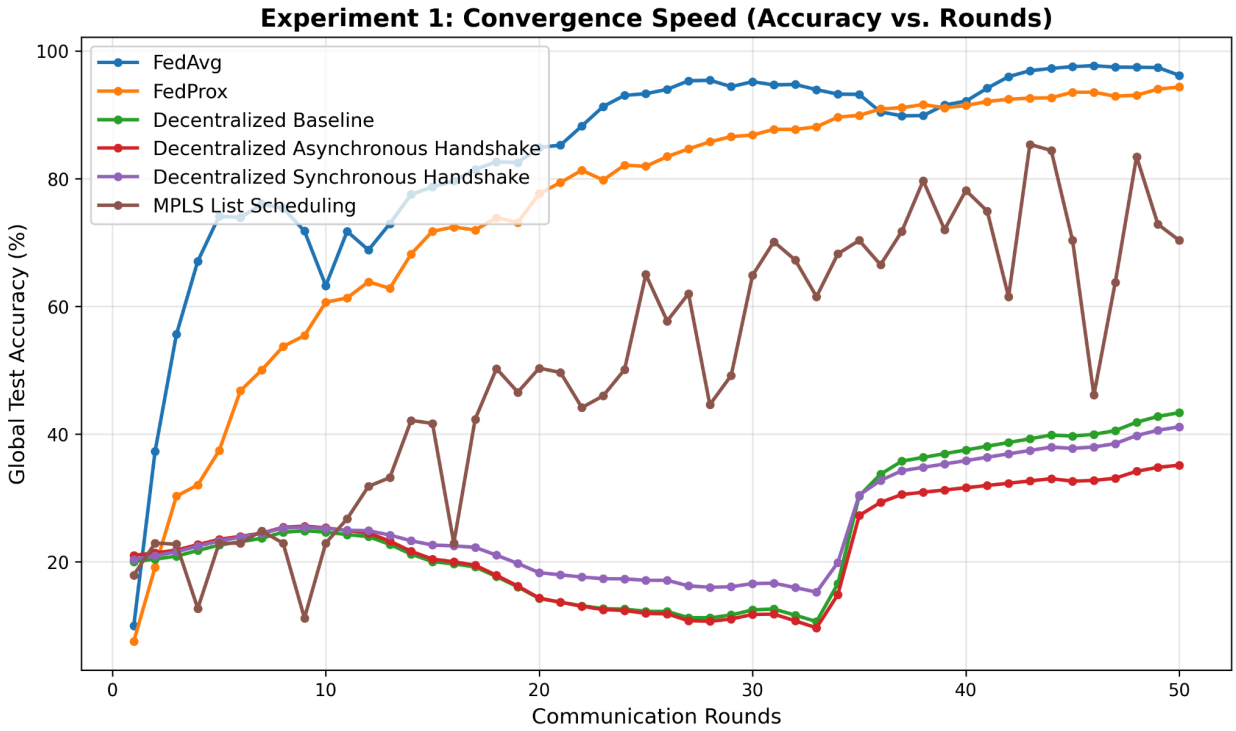
---

4. Experiments and Results

4.1 Experiment 1: Convergence Speed (Accuracy vs. Rounds)

Objective: Compare algorithmic efficiency - how quickly each method converges in terms of communication rounds.

Methodology: - X-axis: Communication Rounds (0 to R_max = 50) - Y-axis: Global Test Accuracy (%) - Each protocol runs for 50 rounds - Accuracy is measured at the end of each round on the global test set - Results are typically averaged across 2-4 runs.



Final Accuracy Summary - All Protocols (50 Rounds)

Protocol	Final Accuracy (Round 50)
FedAvg	96.20%
FedProx	94.36%
Decentralized Baseline	43.36%
Decentralized Asynchronous Handshake	35.14%
Decentralized Synchronous Handshake	41.14%
MPLS List Scheduling	70.34%

Observations:

- FedAvg and FedProx shows fastest convergence (centralized aggregation) Decentralized methods converge slower due to limited information exchange
- Zig Zag movement of accuracy per round is observed for decentralization, especially prominent for MPLS because each model weights convergence leading to lesser accuracy in the next round.
- Centralized ones showed better accuracy than decentralized ones.

- Partial model transfer limiting information exchange - Aggregation of only fc2 and fc3 layers may not be sufficient - Non-IID data distribution makes partial aggregation less effective

4.2 Experiment 2: Convergence Speed (Accuracy vs. Virtual Time)

Objective - Compare system efficiency using the Hybrid Clock mechanism. This is the experiment that reveals if bandwidth savings translate to time savings. Virtual time accounts for both compute time and network delays

Protocol	Time Taken (secs)	Final Accuracy (%)	Time taken to arrive at 70% accuracy
FedAvg	1205	96.2	~700
FedProx	1312	94.36	~700
Decentralized baseline	945	43.36	NA
Decentralized async handshake	615	35.14	NA
Decentralized sync handshake	15200	41.14	NA
MPLS list scheduling	1012	70.34	~1012

Observations:

- FedAvg and FedProx took more time to converge primarily because of virtual time considered for network transfer.
- Decentralized async handshake was transferring specific layers of models and hence lesser network traffic time and hence less time to converge.
- Decentralized sync handshake was stuck waiting on synchronization barriers for local training and model updates for long time.
- MPLS List scheduling worked fine.

4.3 Experiment 3: Bandwidth Efficiency (Accuracy vs. Data Volume)

Objective: Compare data consumption efficiency - how much data must be transferred to achieve a given accuracy level.

Protocol	Total data volumen (in MBs)	Final Accuracy (%)
FedAvg	~25	96.2
FedProx	~25	94.36
Decentralized baseline	~19	43.36
Decentralized async handshake	~3.5	35.14
Decentralized sync handshake	~3.5	41.14
MPLS list scheduling	~9	70.34

Observations

- All of model state was getting transferred in FedAvg, FedProx and Decentralized baseline and hence we see a higher data transfer when compared to other which passes specific layers.
- Decentralized async handshake achieves 85% bandwidth savings compared to FedAvg (3.54 MB vs 24.68 MB) - This is achieved by transferring only fc2 and fc3 layers (~15% of model size) - However, this bandwidth efficiency comes at the cost of convergence quality

4.4 Experiment 4: Message Complexity (Messages vs. Rounds)

Objective: Compare control message overhead - how many messages are required for coordination.

Observations:

Theoretically, we can easily arrive at number of messages per round and also explained and documented above in architecture.

Overall, we see many message exchanges for decentralized models especially MPLS, and others given it has to interact for metadata queries.

5. Insights, and Key findings

5.1 Key Findings

- FedAvg achieved the highest final accuracy (96.20%) but required the most data transfer (24.68 MB) and highest virtual time (1205.05s)
- FedProx (when implemented) shows similar convergence to FedAvg with more virtual time and similar data transfer.
- Decentralized Baseline demonstrated faster convergence in terms of virtual time (945.42s) but lower final accuracy (43.36%) and reduced data transfer compared to centralized but still high at 19MB.
- Decentralized asynchronous handshake showed significant bandwidth savings (3.54 MB, 85% reduction) but required more control messages (588 messages) due to metadata overhead and achieved poor convergence (35.14% accuracy)
- Decentralized synchronous handshake showed bandwidth savings similar to decentralized asynchronous but there was lot of wait time and more messgaes due to metadata overhead and it achieved slightly better convergence than decentralized asynchronous (41.14% accuracy)
- Both the decentralized versions had some variations in results with different variations.
- Pseudo implementation of MPLS had a higher data transfer of around 8-10 MB but less than centralized versions and final accuracy with 50 rounds was around 70.34%.

As part of project, several critical implementation challenges associated with Federated Learning on Non-IID data were observed such as

- Model Convergence and Client Drift: The aggregation strategy (e.g., standard averaging vs. alternative techniques) is pivotal. We observed that merging weights from heterogeneous data sources can cause ‘client drift,’ leading to performance degradation rather than improvement.
- Transmission Strategy: Determining which model components to synchronize—specifically, choosing between convolutional layers, classification

heads, or gradient updates—requires balancing communication costs with model efficacy.

- Communication Complexity: The architecture of peer-to-peer interaction (single vs. multi-peer) introduces significant complexity regarding deterministic behavior, handshake protocols, and peer identification mechanisms.
- There are multiple reasons for poor convergence of decentralized models like non IID distribution, partial model transfers, limited aggregation and synchronization issues.

5.2 Future Work

1. **Improve MPLS Convergence**:
    - Aggregate more layers (include conv layers)
    - Use adaptive layer selection based on gradient magnitudes
    - Implement multi-peer aggregation per round
2. **Better Peer Selection**:
    - Use gradient-based similarity instead of weight-based
    - Consider data distribution overlap
    - Implement reputation-based selection
3. **Hybrid Approaches**:
    - Combine centralized and decentralized phases
    - Use centralized aggregation periodically
    - Implement hierarchical aggregation
4. **Optimize Synchronization**:
    - Reduce lock contention
    - Implement asynchronous aggregation
    - Use message queues instead of shared memory

---

6. References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. 20th Int. Conf. Artif. Intell. Statist. (AISTATS)*, Fort Lauderdale, FL, USA, 2017, pp. 1273-1282.

[2] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in *Proc. 3rd MLSys Conf.*, Austin, TX, USA, 2020, pp. 429-450.

[3] Y. Xu, Z. Yao, H. Xu, Y. Liao, and Z. Xie, “MPLS: Stacking diverse layers into one model for decentralized federated learning,” School of Computer Science and Technology, University of Science and Technology of China, Hefei, China, and Suzhou

Institute for Advanced Research, University of Science and Technology of China, Suzhou, China.

[4] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich, “A unified theory of decentralized SGD with changing topology and local updates,” in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, Virtual Event, 2020, pp. 5381-5393.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[6] Y. LeCun, C. Cortes, and C. J. C. Burges, “The MNIST database of handwritten digits,” 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/