Bachelor's Thesis

Design, Simulation, and Evaluation of a Load Balancing Algorithm for Peer-to-Peer Networks Based on Push-Pull Sum and Deal-Agreement-Based Algorithms

Emre Bayazıtoğlu

Examiner: Prof. Dr. Christian Schindelhauer
Advisers: Saptadi Nugroho

University of Freiburg

Faculty of Engineering

Department of Computer Science

Chair of Computer Networks and Telematics

February 28th, 2025

Writing Period

18. 12. 2024 – 28. 02. 2025

Examiner

Prof. Dr. Christian Schindelhauer

Advisers

Saptadi Nugroho

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Load balancing algorithms are designed to evenly distribute loads across networks, typically modeled as undirected graphs. In such networks, nodes exchange loads with their neighbors to achieve a balanced state. This thesis analyzes different load balancing approaches. The Push-Pull Sum algorithm, introduced in [1], combines elements of the Push-Sum [2] and Pull-Sum algorithms. The Push-Sum algorithm, proposed by Kempe et al., is a load balancing method where each node randomly selects a neighbor and transfers half of its current sum and weight to that neighbor. The Single-Proposal Deal-Agreement-Based load balancing algorithm, as presented in [3], incorporates a deal-agreement mechanism into load transfers in order to achieve fair load transfers between two nodes.

In this thesis, I propose and implement a variation of the Push-Pull Sum algorithm that integrates principles from the Deal-Agreement-Based algorithm and adaptive thresholding. This adaptation modifies and extends certain properties of the traditional Push-Pull Sum algorithm. For the Adaptive Threshold Push-Pull Sum algorithm, I provide pseudocode, implement this load balancing approach in a Peer-to-Peer network, and analyze simulation outcomes across different topologies. The objective is to find a compromise solution including overall good performance in different topologies. The performance of this algorithm is evaluated using the mean squared error (MSE) reduction over time as a convergence metric. Results are presented using log-log and log-linear graphs to compare the efficiency of error reduction in various scenarios. The slopes of the MSE curves give insights into how effectively

the algorithms distribute loads across the network. Additionally, the data is fitted to different models to assess the rate of convergence per region.

The findings suggest that the proposed modifications to the Push-Pull Sum algorithm achieve a more efficient and adaptable load balancing strategy for most of the scenarios tested in the experiments. The adaptive threshold mechanism added to the Push-Pull Sum algorithm dynamically adjusts the threshold based on the state of imbalance in the network. In some of the topologies under test, the Adaptive Threshold Push-Pull Sum algorithm acts as an intermediate solution between the Deal-Agreement-Based and Push-Pull Sum algorithms.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

1 Introduction

In times where computational tasks are becoming increasingly heavy, many systems require multiple servers or computers to complete these tasks. These servers, or computers, often referred to as nodes, interact directly with each other, are decentralized, and form a so-called Peer-to-Peer network. As depicted in figure 1, each node consists of a CPU, a memory module, and a network interface for communication with other nodes. Together, these nodes form a scalable network. The memory system is modeled as a distributed memory scheme, where each node has its own memory module, rather than a shared one, in order to avoid bottlenecks in memory access. [4]

Each computer is assigned a non-negative workload, referred to as load, which can represent various computational tasks such as CPU usage, memory utilization, or internet traffic [3]. The main objective when applying load balancing algorithms is to balance the state of the network, meaning that each node holds the average load of the network. This is accomplished by addressing overloading and underloading through the redistribution of load to other nodes. Heavily overloaded nodes risk failing to complete their tasks due to overheating. Therefore, load balancing enhances coordination in distributed systems and improves scalability.

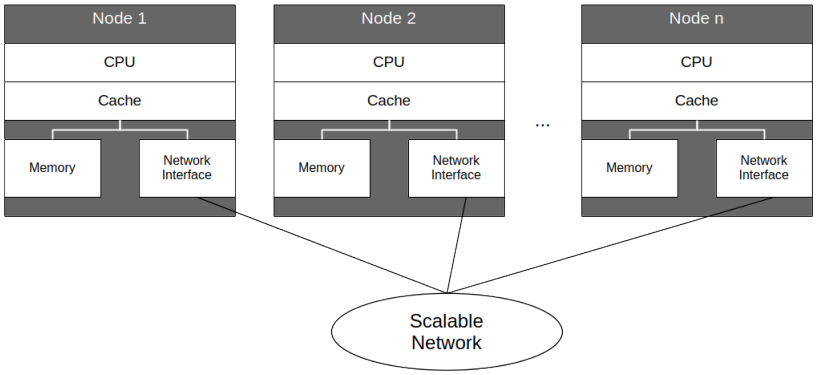


Figure 1: Overview of the setting

1.1 Preliminaries

The following definitions are taken from the lecture slides of Graph Theory of the summer term 2021 [5]:

Definition 1.1.1 (General Graph)**.** A graph G is defined as $G = (V, E \subseteq V \times V)$, where $V := \{v_0, v_1, ...\}$ represents the *vertices* (or nodes) of G and $E := \{e_0, e_1, ...\}$ represents the *edges*. The graph may consist of a single vertex and no edges.

Definition 1.1.2 (Order, Incident, Degree)**.** The number of vertices $|V|$ in the graph is referred to as the *order* of G.

A vertex and an edge are *incident* if the vertex is an endpoint of the edge.

The *degree* of a vertex is the number of incident edges.

Definition 1.1.3 (Regular Graphs)**.** A *regular graph* is a graph where each vertex has the same degree.

Definition 1.1.4 (Adjacent)**.** If an edge connects two vertices, these vertices are *adjacent* to each other, thus are neighboring vertices.

Definition 1.1.5 (Loop, Multiple Edges, Simple Graph)**.** A *loop* is an edge that connects a vertex to itself.

Multiple edges refer to edges that share the same pair of endpoints.

The *simple graph* is a graph that has no loops or multiple edges.

Definition 1.1.6 (Clique)**.** A *clique* is a set of pairwise adjacent vertices. So a subgraph of G that is complete.

Definition 1.1.7 (Path)**.** A *path* is a simple graph whose vertices can be ordered such that two vertices are adjacent if and only if they are consecutive in the list.

Definition 1.1.8 (Connected Graphs)**.** *Connected graphs* are graphs where each pair of vertices in the graph belongs to a path.

Definition 1.1.9 (Bipartite Graphs)**.** *Bipartite graphs* are graphs where the set of graph edges is the union of two disjoint independent sets.

Definition 1.1.10 (Distance, Diameter)**.** If G has a path from node u to node v, then the *distance* from u to v, $d_G(u, v)$, is the shortest length from a u, v-path.

The *diameter* of the graph $\text{diam}(G) := max_{u,v \in V} d_G(u, v)$ is the greatest length of any of the shortest paths between any two nodes.

Definition 1.1.11 (Peer-to-Peer Network)**.** A *Peer-to-Peer Network (P2P)* is a communication network between computers on the internet without a central control and without reliable partners [6].

1.2 Motivation

The motivation for this thesis stems from the observed performance discrepancies between the Single-Proposal Deal-Agreement-Based algorithm proposed by Yefim Dinitz, Shlomi Dolev, and Manish Kumar [3] and the Push-Pull Sum algorithm described in the paper by Saptadi Nugroho, Alexander Weinmann, and Christian Schindelhauer [1] across different network topologies. These observations were initially gathered during the student project titled "Comparative Analysis of Load Balancing Algorithms in General Graphs" [7]. According to the simulations conducted in the student project, the Push-Pull Sum algorithm performs better in reducing the MSE per round for the Complete graph and the Star graph compared to the Deal-Agreement-Based algorithm, which performs better in reducing the MSE per round for the Torus Grid graph and the Ring graph. The simulations were conducted for different network sizes, which also influenced the convergence speed, with larger network sizes requiring more rounds to achieve balance.

To address these challenges, this thesis introduces the Adaptive Threshold Push-Pull Sum algorithm, which combines the strengths of both the Continuous Single-Proposal Deal-Agreement-Based and the Push-Pull Sum algorithms and establishes a trade-off to lessen their respective drawbacks. The introduced algorithm uses the mechanic of adaptive thresholding in order to prevent load transfers with low effect in error reduction, since load transfers are pricey operations. This approach bridges the gap between the two algorithms by adjusting to the network's current state of balance.

1.3 Related Work

Nugroho et al. [1] proposed the Push-Pull Sum algorithm, which is essentially a combination of two algorithms: the Push-Sum algorithm, introduced by David Kempe, Alin Dobra, and Johannes Gehrke [2], and the Pull-Sum algorithm. The push and the

pull mechanisms are directly adopted from the Push-Pull Sum algorithm. Nugroho et al. use the MSE as a metric to evaluate the performance of their algorithm, an approach that will also be followed in this thesis. Their experiments were conducted on static general graphs, and this thesis similarly analyzes the behavior of the load balancing algorithms in static graphs.

Dinitz et al. [3] proposed two versions of the Single-Proposal Deal-Agreement-Based algorithm, as well as two variations of a multi-neighbor load balancing algorithm: a round-robin approach and a self-stabilizing load balancing algorithm. However, the only algorithms comparable to ours are the Single-Proposal Deal-Agreement-Based algorithms, which have two variations: one for the continuous setting and one for the discrete setting. The main difference between these two is that, in the continuous setting, any load may be transferred over the edges, whereas, in the discrete setting, all load transfers must involve integers. For multi-neighbor load balancing, nodes may transfer loads to several neighbors within a single round. In contrast, the Push-Pull Sum algorithm allows this only during pull actions, where a node responds to all requesting nodes by sending loads back. Additionally, the self-stabilizing and round-robin approaches are asynchronous algorithms, whereas both the Adaptive Threshold Push-Pull Sum and the Push-Pull Sum algorithms operate synchronously.

1.4 Hypothesis

The Adaptive Threshold Push-Pull Sum algorithm, which integrates key features of the Deal-Agreement-Based algorithm and the Push-Pull Sum algorithm, will demonstrate performances that are intermediate between the two in terms of MSE reduction across six network topologies. Specifically, it is expected to perform better than the Deal-Agreement-Based algorithm in reducing the error in high-degree networks and perform better than the Push-Pull Sum algorithm in reducing the error in low-degree networks.

This hypothesis will be evaluated through a comparative analysis of MSE across six topologies, some of which feature varying network sizes, to evaluate the algorithm's scalability. To analyze the data trends and draw conclusions about the convergence rate, model fitting is applied as an analysis technique. Additionally, slopes will be computed for three distinct time regions—*start*, *middle*, and *end*—to provide insights into the algorithm's performance at specific stages of execution.

1.5 Contribution

This thesis introduces a novel load balancing algorithm that combines the strengths of two established approaches: randomized load balancing and load balancing based on deal agreement. The introduced algorithm uses an adaptive threshold mechanism in order to adjust to the current state of the network to enhance performance. The push and pull mechanisms are directly adopted by the Push-Pull Sum algorithm.

2 Problem Overview

The load balancing problem is defined on an undirected general graph, where each node can transfer loads to its neighboring nodes via edges to achieve a balanced network state [3]. The problem setting and the approach to address the problem are elaborated on in this section.

2.1 Setting

Definition 2.1.1 (Continuous and Discrete Load Transfers)**.** In the *continuous* setting, nodes can transfer any amount of load over the edges, while in the *discrete* setting, all load transfers must consist of integer values [3].

Definition 2.1.2 (Synchronous and Asynchronous Message Delivery)**.** In the *synchronous* setting, the time for message delivery is constant (e.g., O(1)), whereas in the *asynchronous* setting, the message delivery time can be unpredictably large. However, it is possible to convert an asynchronous setting into a synchronous one by adjusting the time frame within which messages are expected to be delivered. [3]

Definition 2.1.3 (Static and Dynamic Graphs)**.** Load balancing algorithms can operate in either *static* or *dynamic* graph settings. In a dynamic graph, connections between nodes may change arbitrarily between rounds. In contrast, the connections and the nodes remain the same for the static graph [3].

The Peer-to-Peer network is modeled as a static general graph, meaning the set of edges remains unchanged during the application of the load balancing algorithms. The objective of load balancing is achieved using local algorithms, where each node gathers information only from its direct neighbors. The setting is a continuous setting. Additionally, a synchronous message delivery assumption is made, with a constant delivery time, e.g., O(1). The experiments are conducted on six network topologies, each contains of 2^{10} nodes.

2.2 Approach

This thesis consists of three steps: the design of a load balancing algorithm, simulations to test its ability to balance the network across different topologies, and a comparative analysis of the simulation outcomes using statistical methods. In prior research [7], the strengths and weaknesses of two load balancing algorithms were identified by simulating their performance on various topologies and network sizes to test the scalability and adaptability of the algorithms to different situations. The design of the algorithm builds upon these findings. Each simulation outcome includes 30 experiments to ensure statistical significance. The results are analyzed using the concept of model fitting to identify trends in MSE reduction. Slopes are calculated for different regions to assess the consistency of the performance of the algorithms. The results are presented in plots accompanied by explanations that provide insights into the behavior of the load balancing algorithms.

3 Algorithms

This thesis examines three load balancing algorithms: the Continuous Single-Proposal Deal-Agreement-Based algorithm, the Push-Pull Sum algorithm, and the proposed Adaptive Threshold Push-Pull Sum algorithm. The Adaptive Threshold Push-Pull Sums structure is described, along with how it incorporates features from the first two algorithms and the objective behind its design. The working mechanics of each algorithm are described along with pseudo-code. Furthermore, examples are provided to show how the algorithms achieve a balanced state in the network. To provide further light on the objectives of the Adaptive Threshold Push-Pull Sum method, the aspired results are described.

3.1 Characteristics

Load balancing algorithms may have different characteristics. These characteristics are elaborated on below:

Definition 3.1.1 (Static and Dynamic Load Balancing Algorithms)**.** Load balancing algorithms can be classified as either *static* or *dynamic* algorithms. Static algorithms assign tasks to the nodes at compile time, while dynamic algorithms assign tasks at run time. The main advantage that static load balancing algorithms have over dynamic load balancing algorithms is that they do not cause any run-time overhead. [8]

Definition 3.1.2 (Stochastic and Deterministic Load Balancing Algorithms)**.** *Stochastic* load balancing algorithms rely on randomness to select load transfer partners. *Deterministic* load balancing algorithms, on the other hand, follow predefined distribution rules in order to make load transfers. [4]

Definition 3.1.3 (Global and Local Load Balancing Algorithms)**.** *Local* load balancing algorithms allow nodes to transfer loads within their neighborhood, while *global* load balancing algorithms enable load balancing operations across the entire network [4].

Definition 3.1.4 (Monotonic Load Balancing Algorithms)**.** An algorithm is considered *monotonic* if each load transfer is from a higher-loaded node to a less-loaded node and the maximal load in the network never increases and the minimal load never decreases [3].

Definition 3.1.5 (Mass conservation property)**.** Some load balancing algorithms possess the *mass conservation property*, which guarantees that the values will converge to the correct aggregate of the network's ground truth [1].

Definition 3.1.6 (Anytime Property)**.** An *anytime* algorithm can be halted at any stage during the execution, and after stoppage the state of the network is not worse than in any previous rounds. The advantage that comes with an anytime algorithm is that the network in which the load balancing algorithm with this property is applied is guaranteed to show more feasible states with advanced rounds. [3]

Many of these properties are desirable for load balancing. For instance, monotonicity and the anytime property contribute to better performance and robustness, while locality reduces computational overhead. Similarly, determinism enhances the predictability and reliability of the algorithm's behavior.

3.2 Push-Pull Sum Algorithm

The Push-Pull Sum algorithm as proposed in [1] requires each node to hold sum $s_{i,r}$ and weight $w_{i,r}$ values as initial information. Initially, each node's weight is set to $w_{i,0} = 1$, and the sum of all weights is equal to the network size N at each round. Each node's initial sum value $s_{i,0}$ is equal to whatever the required input $x_i \in \mathbb{R}_0^+$ is. In the paper, the values for the sums are uniformly distributed values between 0 and 100 [1]. The algorithm consists of three main procedures: *RequestData*, *ResponseData*, and *Aggregate*, as detailed in algorithm 3.

Each round r, except the first, begins with the *Aggregate* procedure, where each node i collects messages $M_{i,r}$ sent by other nodes $\{(s_m, w_m)\}$ in the previous round $r-1$, requesting data. Each node then updates its sum and weight values as $\sum_{m \in M_{i,r}} s_m$ and $\sum_{m \in M_{i,r}} w_m$, respectively. The updated load is computed by dividing the sum by the weight. Next, each node calls the *RequestData* procedure. In this procedure, each node chooses a random neighbor node and executes a push operation, so each node sends half of its sum $\frac{s_{i,r}}{2}$ and half of its weight $\frac{w_{i,r}}{2}$ to the chosen neighbor and itself. Finally, each node executes the pull operation, which is described in the *ResponseData* procedure. Here, each node gathers the incoming requests per round r in a set $R_{i,r}$. Then, each node replies to each requesting node, including itself, by distributing half of its sum divided by the number of incoming requests $\frac{\frac{s_{i,r}}{2}}{|R_{i,r}|}$ to each requesting node.

The setting in [1] is similar to the one in this thesis. While their study focuses on a Complete graph with 10^4 nodes, this study examines network sizes of 2^{10} nodes and includes different topologies. In that paper, 50 experiments were conducted, each running for 30 rounds. Their findings demonstrated that the Push-Pull Sum algorithm reduces the expected potential Φ_r exponentially. The potential function

Algorithm 1 Push-Pull Sum algorithm

procedure RequestData
    Choose a random neighbor node v
    Send $(\frac{s_{u,r}}{2}, \frac{w_{u,r}}{2})$ to the chosen node v and the node u itself
end procedure
procedure ResponseData
    $R_{u,r} \leftarrow$ Set of the nodes calling u at a round r
    for all $i \in R_{u,r}$ **do**
        Reply to i with $\left(\frac{\frac{s_{u,r}}{2}}{|R_{u,r}|}, \frac{\frac{w_{u,r}}{2}}{|R_{u,r}|}\right)$
    end for
end procedure
procedure Aggregate
    $M_{u,r} \leftarrow \{(s_m, w_m)\}$ messages sent to u at round $r-1$
    $s_{u,r} \leftarrow \sum_{m \in M_{u,r}} s_m, w_{u,r} \leftarrow \sum_{m \in M_{u,r}} w_m$
    $f_{avg} \leftarrow \frac{s_{u,r}}{w_{u,r}}$
end procedure

for the Complete graph is defined as:

$$\Phi_r = \sum_{i,j} \left(v_{i,j,r} - \frac{w_{i,r}}{n} \right)^2, \tag{1}$$

where the $v_{i,j,r}$ component stores the fractional value of node j's contribution at round r. The equation:

$$\mathbb{E}[\Phi_{r+1} | \Phi_r = \phi] = (\frac{2e-1}{4e} - \frac{1}{4n})\phi \tag{2}$$

is the conditional expectation of Φ_{r+1} for the Push-Pull Sum algorithm. The Push-Pull Sum algorithm holds the mass conservation property and is classified as a stochastic load balancing algorithm due to its randomized neighbor selection process [1].

The Push-Pull Sum algorithm performed very well on Complete graphs, Ring of Cliques with large clique size, Lollipop graphs with large clique size, and Star graphs. In the case of the Star graph the central node acts as a distributor of the load. Since each leaf chooses the central node with 100% possibility as a "random" neighbor,

the central node is involved as an endpoint of $N-1$ external push operations and redistributes the load via pull operations to the leaves, where the sum is $\frac{\frac{s_{i,r}}{2}}{N-1}$ and accordingly the weight is $\frac{\frac{w_{i,r}}{2}}{N-1}$. For the Complete graph, the Ring of Cliques and the Lollipop graph, the density of the graph plays an important role. Since nodes select neighbors randomly, the high edge density allows the algorithm to spread loads efficiently and prevent bottlenecks. This explains why the Push-Pull Sum algorithm underperformed on Torus Grid and Ring graphs compared to the Single-Proposal Deal-Agreement-Based algorithm, as they are limited to a degree of 4 and 2, respectively.

3.2.1 Example

The example in figure 2 illustrates an execution of the Push-Pull Sum algorithm on a Complete graph with four nodes labeled *A, B, C* and *D*. Each node is initially assigned a sum and a weight value. The undirected graph represents the Peer-to-Peer network. The nodes are connected by edges. The solid directed edges depict the push operations, and the dashed directed edges visualize the pull operations. Each node maintains a set $R_{i,r}$ where i is the node ID and r is the round being examined. The load of the node is calculated by $\frac{s_{i,r}}{w_{i,r}}$. The example shows the first round of an execution of the Push-Pull Sum algorithm. The push and pull operations are distinct. The left-hand side of figure 2 depicts the behavior of the nodes while executing the push operations, while the right-hand side side illustrates the pull operations. During the push phase, each node randomly selects a neighbor to transfer load to. In this example:

- Node A selects node C and pushes half of its sum and weight to both node C and itself. (Self-loops are omitted from the figure for readability.)

- Node B selects node A as its trading partner.

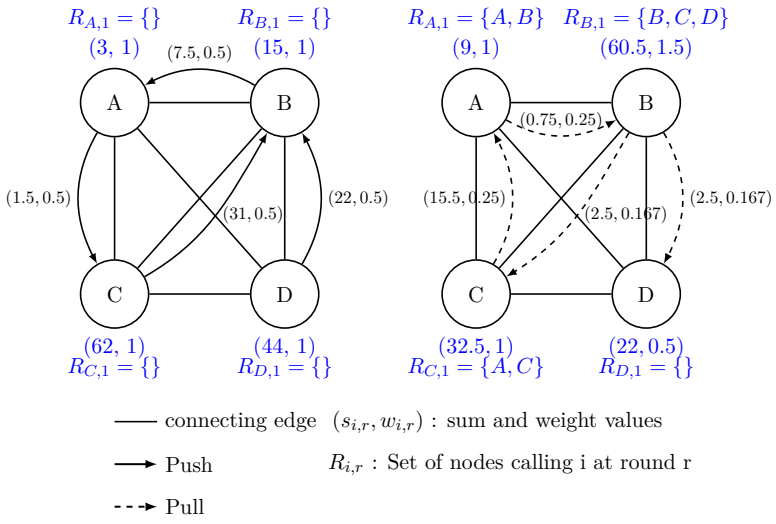


Figure 2: Push-Pull Sum: push and pull actions

- Nodes C and D push their values to node B and themselves.

Given the push operations for each node, the set $R_{i,r}$ is computed. Since node B pushed load to node A and node A pushed load to itself, the requesting nodes for node A are $R_{A,1} = \{A, B\}$. The updated sum and weight values after the push phase can be inspected on the right-hand side of figure 2.

Following the push actions, each node proceeds with the *ResponseData*-procedure, executing the pull phase. Each node in $R_{i,1}$ receives a response based on the respective pull values. Since node A has two nodes in its set $R_{A,1}$, it responds to both node B and itself with $\left(\frac{\frac{3}{2}}{2}, \frac{\frac{1}{2}}{2}\right)$, as indicated by a dashed directed edge. Similarly, the remaining nodes *B, C, and D* execute their pull operations. Following the push and pull operations, the nodes update their sum and weight values. The state of the network after round one is depicted in figure 3. The MSE at the beginning of round 1 is 542.50. After applying one round of the Push-Pull Sum algorithm, the MSE is reduced to 63.49.

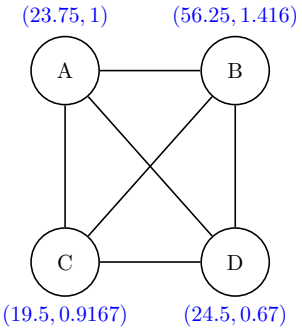


Figure 3: Push-Pull Sum: setting after round 1

3.3 Continuous Single-Proposal Deal-Agreement-Based Algorithm

The Continuous Single-Proposal Deal-Agreement-Based algorithm proposed by Dinitz et al. [3] achieves the goal of load balancing based on deterministic deal agreements, where one node proposes a load to one neighboring node, and the neighboring node accepts the transfer proposal either fully or partially. Dinitz et al. proved that the algorithm has the anytime property, meaning it never worsens the state of the network during execution. Their study examined the algorithm in a dynamic setting, where each node has access to a set of neighboring nodes, including the node's loads. The algorithm is divided into three phases: *proposal*, *deal*, and *summary*. In the *proposal*-phase, each node u identifies its minimally loaded neighbor v and sends a proposal to that neighbor if v has a lower load. The proposal is of value:

$$\frac{load_r(u) - load_r(v)}{2} \tag{3}$$

which is labeled as a *fair* proposal. Since the load transfer is fair, the resulting load of u is not lower than that of v. $load_r(u)$ represents the load of node u at round r. In the *deal*-phase, nodes evaluate the deals proposed to them. A node accepts the deal of the node that proposes the maximal load transfer. The actual transfer happens, and the nodes update their loads. Finally, in the *summary*-phase, each

node informs their neighbors regarding their updated load values. [3]

Algorithm 2 Continuous Single-Proposal Deal-Agreement-Based algorithm

Input: An undirected graph $G = (V, E, load)$
Output: A load state with discrepancy at most ϵ on G

for $r = 1$ and on **do**
  for every node u **do**
    Find a neighbor, v, with the minimal load
    if $load_r(u) - load_r(v) > 0$ **then**
      u sends to v a transfer proposal of value $(load_r(u) - load_r(v))/2$
    end if
  end for
  for every node u **do**
    if there is at least one transfer proposal to u **then**
      Find a neighbor, w, proposing to u the maximal transfer
      Node u makes a deal: informs node w on accepting its proposal
      The actual transfer from w to u is executed
    end if
  end for
  for every node u **do**
    Node u sends the updated value of its load to its neighbors
  end for
end for

The potential for a node u is defined as:

$$p(u) = (load(u) - L_{avg})^2 \tag{4}$$

where L_{avg} represents the current load average in the network. The potential for the graph $p(G)$ is defined as:

$$p(G) = \sum_{i \in V} p(i), \tag{5}$$

which is the sum of the potentials of each node in the graph G. Any fair load transfer of load l decreases the potential of the graph by at least $2l^2$. As a result of any round r of the Continuous Single-Proposal Deal-Agreement-Based algorithm, the graph potential decreases by at least $\frac{K_r^2}{2D_r}$, where K is the initial discrepancy and D is a bound for the graph diameter. [3]

The Single-Proposal Deal-Agreement-Based load balancing algorithm struggles to reduce the MSE as rapidly as the Push-Pull Sum algorithm for dense graphs like the Complete graph, the Lollipop graph with a large clique size, and the Ring of Cliques with a large clique size. This limitation arises because each node looks for the minimally loaded partner for a load transfer. For a Complete graph, the minimally loaded neighbor for each node is the same node with loads of L_{min} (the minimal load in the network). This causes each node that does not hold L_{min} to propose to the same node, which then evaluates the proposals and accepts exactly one transfer proposal, namely the maximal one. This situation is described in detail in the appendix in section 10.3 using an example. A similar scenario occurs in the Ring of Cliques and the Lollipop graph. However, in the Ring of Cliques, nodes do not all propose to the same minimally loaded neighbor, instead, they propose to the least-loaded neighbor within their respective cliques. For the Lollipop graph, the nodes in the path section balance their loads more quickly than the nodes in the clique, which is a bottleneck in this scenario. For the Torus Grid and the Ring graph, the Deal-Agreement-Based algorithm performs better in reducing the error compared to the Push-Pull Sum algorithm. In these scenarios, the proposals are more evenly distributed among nodes due to the lower density of the graphs. The Star graph is an exception to this case, since we have a similar bottleneck as in the Complete graph, as the central node is the common neighbor for each leaf, thus, each leaf proposes a load to the central node. The central node can accept only one proposal. Again, an example of this is provided in the appendix in section 10.3.

3.3.1 Example

Figure 4 illustrates two settings. The left-hand side of the figure depicts the initial state, and the right-hand side illustrates the state after one round of execution. The setting is the same as in the example above for the Push-Pull Sum, with the difference that each node is assigned a load value instead of sum and weight values.

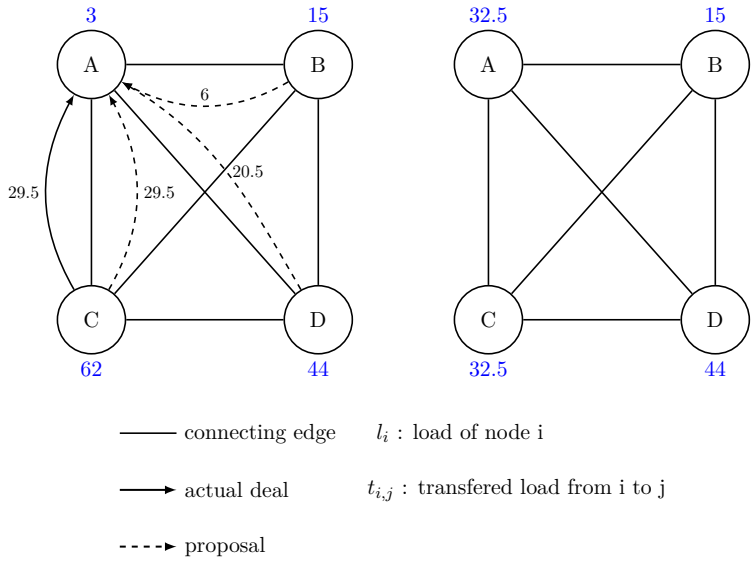


Figure 4: Deal-Agreement-Based: initial setup and setup after round 1

The dashed directed edges represent transfer proposals from one node to another, while solid directed edges indicate actual load transfers. The network consists of four nodes, labeled *A, B, C,* and *D*. Each node identifies its least-loaded neighbor as a potential transfer recipient. Nodes *B, C* and *D* determine node *A* as their minimally loaded neighbor. Node *A* identifies node *B* as the neighbor with the minimal load in its neighborhood, however, node *B* has more load than node *A*, so node *A* does not send a transfer proposal to node *B*. As a result, nodes *B, C*, and *D* each send a transfer proposal of value $\frac{(load_r(i) - load_r(A))}{2}$ to node *A*, where $i \in \{B, C, D\}$. Node *A* evaluates the proposal and accepts node *C*'s transfer proposal, as node *C* proposes the largest amount of load, namely 29.5. The actual transfer happens, and 29.5 of loads are transferred from node *C* to node *A*. The right-hand side of figure 4 shows the state of the network after round 1 of executing the Deal-Agreement-Based algorithm. Node *A* and *C* each have equal loads of 32.5, the loads of nodes *B* and *D* remain unchanged. The MSE at the beginning of round 1 is at 542.5. After executing the first round of the algorithm, the MSE decreases to a value of 107.375, which is approximately one-fifth of the initial MSE.

3.4 Adaptive Threshold Push-Pull Sum Algorithm

The Adaptive Threshold Push-Pull Sum algorithm is composed of key elements from both the Push-Pull Sum and Single-Proposal Deal-Agreement-Based algorithms, extended by the idea of adaptive thresholding. The Adaptive Threshold Push-Pull Sum consists of different procedures. In the *CheckThresholdsRequestData*, node u chooses a subset $RN_{u,r}$ with:

$$RN_{u,r} \subseteq neighborhood_u, |RN_{u,r}| = \lceil \log_2 (|neighborhood_u|) \rceil \tag{6}$$

random neighbors. This increases the likelihood of selecting a well-suited neighbor for a load transfer. The load difference between the node u and the nodes in $RN_{u,r}$ is computed and checked against a threshold θ. The threshold is computed in the *CalculateThresholds*-procedure. The threshold θ is calculated as:

$$\theta = k * \sqrt{MSE_{r-1}} \tag{7}$$

where k is some factor to adjust the sensitivity of the threshold. A larger k makes the threshold more sensitive, meaning that fewer nodes are eligible for load transfer, allowing only significant load transfer. Respectively, a smaller k makes the threshold less strict, so more nodes are eligible for load transfers. This condition ensures that only load transfers with meaningful impact happen, and load transfers between nodes with low impact on the balance of the network are avoided. The first eligible node in $RN_{u,r}$ that exceeds the threshold receives the sum of value ($\frac{s_{i,r}}{2}$) and the weight of value ($\frac{w_{i,r}}{2}$). The *ResponseData* and the *Aggregate*-procedure are directly adapted from the Push-Pull Sum algorithm. Also, the load distribution mechanism is directly taken from the Push-Pull Sum algorithm and extended by an adaptive threshold mechanism. Like the Single-Proposal Deal-Agreement-Based algorithm, the Adaptive Threshold Push-Pull Sum algorithm employs conditional load transfers, but with a threshold θ instead of requiring a strictly positive load difference. Instead of

initiating a load transfer with the maximally proposing node, the Adaptive Threshold Push-Pull Sum algorithm orders the nodes to initiate a load transfer with the first node proposing a load (the first node, where the load difference passes the threshold). The reason for that is to reduce computational overhead by avoiding that each node looks through its whole set RN and is required to evaluate each proposal. So the first node proposing a load transfer is accepted as a transfer partner.

Although formally not shown, the Adaptive Threshold Push-Pull Sum algorithm is expected to hold the mass conservation property, as it converged to the correct ground truth in all experiment settings, similar to the Push-Pull Sum algorithm. Also, given that the push and pull mechanisms are directly adapted, these operations are the only operations that let nodes transfer or receive nodes in the algorithm. The Adaptive Threshold Push-Pull Sum algorithm is a stochastic algorithm, as it inherits its stochastic nature from the Push-Pull Sum algorithm by choosing a subset of neighbors randomly.

3.4.1 Example

Figure 5 depicts a similar setting as described in section 3.2.1. According to the Adaptive Threshold Push-Pull Sum algorithm, each node chooses $\lceil \log_2 (|neighborhood_i|) \rceil$ neighbors. For this setting, each node chooses two neighbors, which are added to the set $RN_{i,1}$ for each node i. For instance, node A computes $RN_{A,1}$ as $\{B, C\}$. The load differences between node A and nodes B and C are computed. In this example, k is set to 0.01, thus the threshold θ is given by $0.01 * \sqrt{542.5} \approx 0.23$. Since both load differences between nodes A and B and nodes A and C exceed this threshold, both nodes are eligible to propose a load transfer with node A. In this scenario, node A selects node B as a transfer partner and pushes half of its sum 1.5 and weight 0.5 to node B and itself. This process is repeated for each node accordingly. Similar to the example in section 3.2.1, the nodes then execute the pull operation. The result is depicted in figure 6. The MSE dropped from 542.5 initially to 251.86. The

Algorithm 3 Adaptive Threshold Push-Pull Sum algorithm

procedure CALCULATETHRESHOLDS
  $\theta \leftarrow k * \sqrt{MSE_{r-1}}$
end procedure
procedure CHECKTRESHOLDREQUESTDATA
  $RN_{u,r} \leftarrow$ choose $\lceil \log_2 (|neighborhood(u)|) \rceil$ random neighbor
  for every node $v_i \in RN_{u,r}$ **do**
    $\Delta_{u,v_i} \leftarrow |(load(u) - load(v_i))|$
    if $\Delta_{u,v} > \theta$ **then**
      Send $(\frac{s_{u,r}}{2}, \frac{w_{u,r}}{2})$ to first node v fulfilling condition and the node u itself
    end if
  end for
end procedure
procedure RESPONSEDATA
  $R_{u,r} \leftarrow$ Set of the nodes calling u at a round r
  for all $i \in R_{u,r}$ **do**
    Reply to i with $\left(\frac{\frac{s_{u,r}}{2}}{|R_{u,r}|}, \frac{\frac{w_{u,r}}{2}}{|R_{u,r}|} \right)$
  end for
end procedure
procedure AGGREGATE
  $M_{u,t} \leftarrow \{(s_m, w_m)\}$ messages sent to u at a round $r-1$
  $s_{u,t} \leftarrow \sum_{m \in M_{u,r}} s_m, w_{u,r} \leftarrow \sum_{m \in M_{u,r}} w_m$
  $load(u) \leftarrow \frac{s_{u,r}}{w_{u,r}}$
end procedure

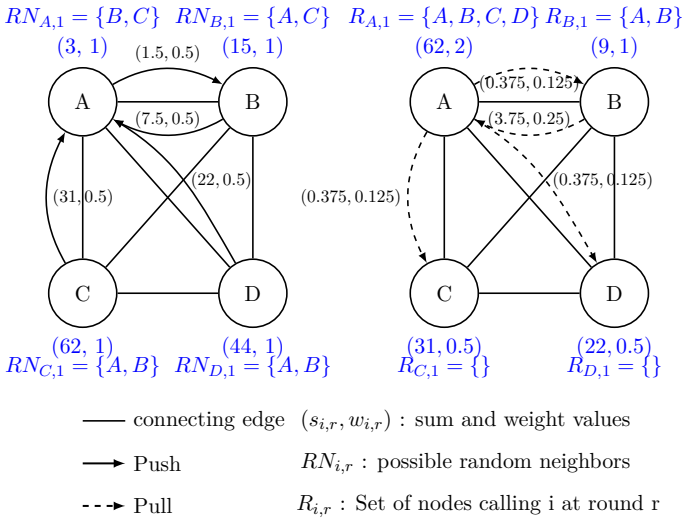


Figure 5: Adaptive Threshold Push-Pull Sum: push and pull actions

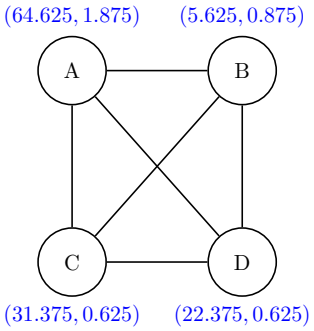


Figure 6: Adaptive Threshold Push-Pull Sum: setting after round 1

	Initial	**Dinitz et al.**	**Nugroho et al.**	**Bayazitoglu**
Node A	3	32.5	(23.75, 1) = 23.75	(64.625, 1.875) = 34.47
Node B	15	15	(56.25, 1.416) = 39.72	(5.625, 0.875) = 6.43
Node C	62	32.5	(19.5, 0.9167) = 21.27	(31.375, 0.625) = 50.2
Node D	44	44	(24.5, 0.67) = 36.57	(22.375, 0.625) = 35.8
MSE	542.50	107.375	63.49	251.86

Table 1: State of loads after one round

error reduction depends on the sensitivity factor k. If k is set to 1, the load transfer between nodes A and B would not have happened. Instead, nodes A and C would have exchanged loads, leading to a larger impact on the balance of the network.

3.4.2 Results

The final results of the first load balancing step can be taken from table 1. The Push-Pull Sum algorithm shows the lowest MSE after round one, followed by the Deal-Agreement-Based algorithm and the Adaptive Threshold Push-Pull Sum algorithm. A small k value was used for demonstrative purposes (as the simulations in chapter 6 were conducted with small k values). A higher k value would have a greater impact in the first round. In section 6, we will see that the Adaptive Threshold Push-Pull Sum algorithm proceeds to enhance in reducing error in later rounds, as the MSE and thus the thresholds adjust.

4 Topologies

The simulations were conducted for six network topologies. Each network has 2^{10} (1024) nodes. The behavior of the algorithms in different topologies is observed, since each topology has diverse characteristics, different performances exploiting the specials of the topologies are expected. The topologies contain *Complete graph*, *Torus Grid graph*, *Ring graph*, *Star graph*, *Lollipop graph*, and *Ring of Cliques*. In the following, the topologies including these characteristics are presented.

4.1 Complete Graph

The *Complete graph* K_N, as illustrated in figure 7, is a graph where each pair of distinct nodes is connected by an edge. The Complete graph, also referred to as a fully connected graph, has N nodes and $\frac{N*(N-1)}{2}$ edges. The Complete graph is a regular graph, where each node has a degree of $N-1$. As it contains the maximum possible number of edges for a given set of nodes, it is considered a dense graph [5]. The diameter of a Complete graph is 1, as each node is directly connected to every other node. Since each node has the same degree and connectivity, algorithms can treat all nodes uniformly.

The Complete graph is used in financial trading systems and military communications, where low latency is critical. It ensures optimal routing paths between nodes due to its low diameter. [9]

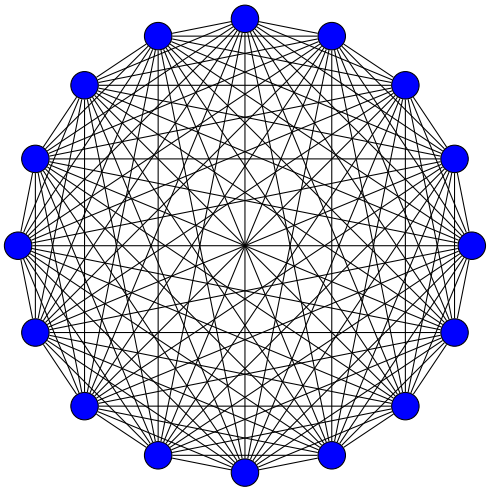


Figure 7: Complete graph: network size 16

4.2 Star Graph

A *Star graph* S_N, as illustrated in figure 8, is a bipartite graph [10] that is structured like a tree with a single central node connected to $N-1$ nodes, or leaves. A Star graph with N nodes has $N-1$ edges. In this structure, every leaf node has a degree of 1, meaning it is connected only to the central node. The diameter of a Star graph is 2, as the path from one leaf node through the central node to another leaf node takes two steps. From a load balancing perspective, the central node acts as a point of redistribution.

This topology is particularly suitable for master-slave or client-server models where the central node delegates tasks. A challenge to face when using the Star topology is that the central node may become overloaded in high-load scenarios. To face this challenge a careful network design is crucial. A hybrid structure where high-bandwidth nodes act as Peer-to-Peer servers (super nodes) helps to combat overloading of the central node [6]. A common usage for the Star topology is a LAN (Local Area Network) in home networks, where all devices are connected to a hub or the router [11]. A drawback when dealing with Star graphs is that the failure of the central node (hub

or router) shuts down the whole network. The advantage of such a setting is that adding new devices is simple, and failures of leaf nodes do not affect the whole network.

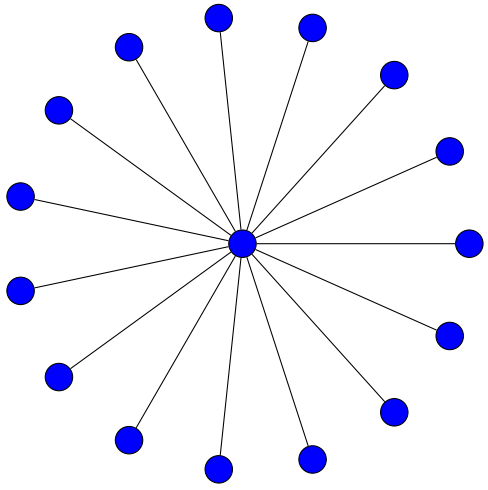


Figure 8: Star graph: network size 16

4.3 Ring Graph

The *Ring graph* R_N is a regular graph, where each node has a degree of 2, forming a cycle. The Ring graph can be constructed by building a Path graph, where the first and the last nodes are connected by an edge as depicted in figure 9. The graph consists of N nodes and N edges. The diameter of a ring is given by $\lfloor \frac{N}{2} \rfloor$.

Most of the Ring structures use a token-based communication model. The node that holds the token may transmit data. Most of the current high-speed LANs have a Ring topology. The advantage of a Ring structure is that it is easy to troubleshoot when faults occur, as the node that is faulty hinders the whole traffic. However, a significant drawback is that a single outage of a node may disrupt the whole network activity. [12]

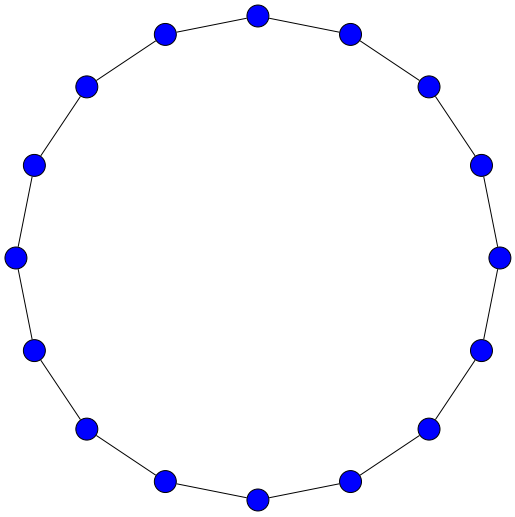


Figure 9: Ring graph: network size 16

4.4 Torus Grid Graph

The two-dimensional *Torus Grid graph* $T_{k,m}$, also referred to as the $(k \times m)$-Torus graph, is a graph that is built like a two-dimensional mesh with wrap-around edges as depicted in figure 10 [13]. k represents the height, and m denotes the width of the grid. The graph consists of $k * m$ nodes and $2 * k * m$ edges (if $k, m > 2$) and is a regular graph, where each node has a degree of 4. The graph's diameter is given by $\lfloor \frac{k}{2} \rfloor + \lfloor \frac{m}{2} \rfloor$. Tori are scalable, as the number of connections per node is constant, regardless of the graph size. In the previous research [7], the simulation results showed to not vary over different network sizes.

The Torus topology is widely used in supercomputers like IBM Blue Gene and Cray systems for high-performance computing [4]. It is also implemented in Network-on-Chip (NoC) designs for its ability to handle data distribution with low latency and high fault tolerance [9].

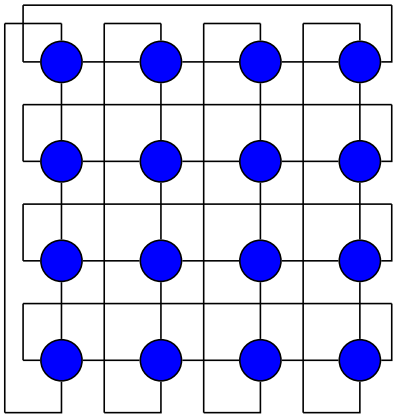


Figure 10: Torus Grid graph: network size 16

4.5 Lollipop Graph

A (k, m)-*Lollipop graph* $L_{k,m}$ is a graph that consists of a clique and a Path graph as depicted in figure 11. The clique and the Path graph are connected by a bridge node, thus a single edge. The (k, m)-Lollipop graph consists of a clique with k nodes and a path size of m nodes. A Lollipop graph has N nodes, where $N = k + m$ and $(\frac{k*(k-1)}{2}) + m$ edges [14].

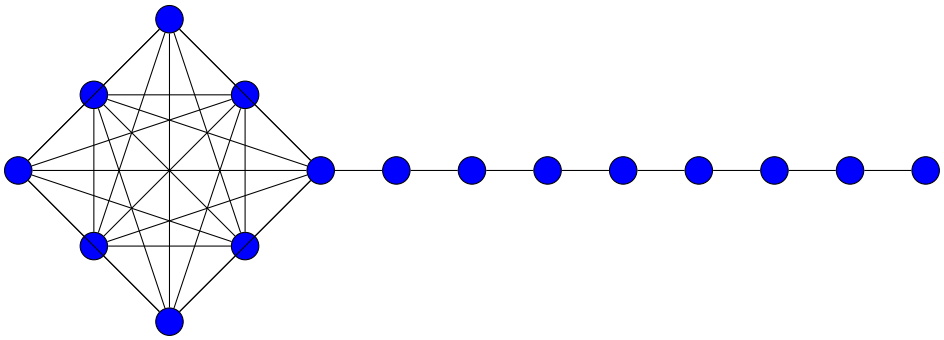


Figure 11: Lollipop graph: network size 16

4.6 Ring of Cliques

The $(k \times m)$-*Ring of Cliques* $ROC_{k,m}$ consists of k cliques, each containing m nodes. The cliques are connected to form a ring structure. To create the ring, one edge from each clique is removed, and the endpoints of these removed edges are connected to form a regular graph [13]. A $(k \times m)$-Ring of Cliques has $\left(k * \left(\frac{m*(m-1)}{2} - 1\right)\right) + k$ edges. The connectivity of the graph increases with larger clique sizes and decreases with smaller clique sizes.

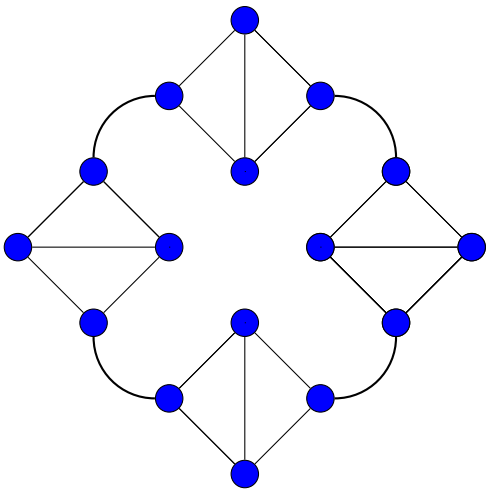


Figure 12: Ring of Cliques: network size 16

4.7 Expected Outcomes

Complete graph: The high connectivity of the Complete graph provides a variety of load transfer opportunities for each node, which will benefit randomized algorithms such as the Push-Pull Sum-based load balancing algorithms (Push-Pull Sum and Adaptive Threshold Push-Pull Sum), as they spread the loads uniformly across the entire network. However, the Complete graph creates a bottleneck for the Deal-Agreement-Based algorithm, as all nodes propose to the same subset of neighbors (if

there are multiple nodes that hold L_{min}). Since only the highest proposal is accepted per node, the number of transfers per round is significantly limited.

Star graph: In the Star graph, the central node acts as a point of redistribution. This benefits the Push-Pull Sum-based algorithms, as each node pushes to the central node (for the Adaptive Threshold Push-Pull Sum algorithm, this is only the case if the load discrepancies surpass the threshold). The central node redistributes loads to each leaf. For the Deal-Agreement-Based algorithm, the Star graph creates a similar bottleneck as the Complete graph topology, since each leaf has only one neighbor (the central node), all nodes propose to the same central node. The central node initiates the load transfer, accepting the maximal proposing load, again resulting in a very limited number of load transfers.

Ring graph: The Ring graph has a sequential structure of data transfers, which makes it slower for loads to propagate through the entire Ring. The Deal-Agreement-Based algorithm draws an important advantage over the Push-Pull Sum-based algorithms as it chooses the optimal load transfer partner out of its two immediate neighbors. In contrast, the Push-Pull Sum-based algorithms rely on random neighbor selection, which is suboptimal in a Ring structure. The Push-Pull Sum algorithm selects one of two nodes randomly, as does the Adaptive Threshold Push-Pull Sum algorithm, because each node chooses a subset of $\lceil \log_2(2) \rceil$ neighbors and ultimately ends up with one neighbor in the RN set. This neighbor might not be the optimal one to conduct a load transfer.

Torus Grid graph: Many load balancing algorithms perform well on 2D Tori due to the focus on local redistribution. The wrap-around edges eliminate boundary effects. Due to its low regular degree, the Deal-Agreement-Based algorithm will perform very well, as the nodes distribute load efficiently between each of its four neighbors. The Adaptive Threshold Push-Pull Sum algorithm improves upon the traditional Push-Pull Sum algorithm by selecting a subset of neighboring nodes and restricting load transfers to those with significant impact on the network balance. This approach

utilizes smaller neighborhoods more effectively than a purely randomized strategy.

Lollipop graph: The Lollipop graph is particularly interesting since it combines a dense clique structure with a sparse Path graph, which introduces a mixed topology challenge. The bridge node becomes a bottleneck since it connects two differently dense regions. Within the clique, load balancing is highly efficient for Push-Pull Sum-based algorithms, while the Deal-Agreement-Based algorithm struggles due to a limited amount of load transfers. Load transfers in the Path graph are sequential, where a deterministic approach like the one of the Deal-Agreement-Based algorithm shows to be very effective, while randomized algorithms like the Push-Pull Sum algorithm will show a moderate performance. The relative sizes of the clique and path affect algorithm performance, a larger clique benefits Push-Pull Sum-based approaches, while an extended path favors the Deal-Agreement-Based algorithm.

Ring of Cliques: Within each clique, Push-Pull Sum-based algorithms perform well due to dense intra-clique connectivity. However, inter-clique balancing poses a challenge for the Push-Pull Sum algorithm. The Deal-Agreement-Based algorithm benefits from its deterministic load distribution by transferring loads via the bridging nodes to other cliques once internal balancing is achieved. The Adaptive Threshold Push-Pull Sum algorithm selects neighbors based on a threshold after internal balancing is achieved. This mechanism favors inter-clique exchanges.

5 Implementation and Technology Stack

A variety of technologies were utilized to achieve the underlying results. Simulations were conducted using *PeerSim*, a simulation framework for *Java*. As a result of each simulation, an output file is generated that contains different simulation parameters, settings, and performance metrics. The data analysis part of this project is implemented using *Python* as a programming language.

5.1 Programming Languages

Python 3.12.6 is used for several tasks, mostly for preparing data necessary to conduct simulations and post-simulation analysis. For plotting purposes, *matplotlib v.3.9.1* is used and for handling data and data analysis *SciPy v.1.14.1* and *NumPy v.2.0.0* are used.

For conducting the simulations, *Java Oracle OpenJDK 21.0.1* and *PeerSim v.1.0.5* are utilized.

5.2 Simulation Framework

PeerSim is a simulation tool developed in Java, which is composed of two engines: the cycle-driven engine and the event-driven engine. I chose PeerSim for the simulations

because it is well-suited for large-scale Peer-to-Peer simulations. In previous projects, I was able to conduct simulations up to 2^{14} nodes and could probably push the boundaries by scaling to even higher dimensions. PeerSim is designed with pluggable components, implemented as interfaces, which are intuitive to use generally. Running a simulation in PeerSim follows four steps. The first step is setting up a *configuration file*, which defines key parameters such as network size and the load-balancing protocols being simulated.

```
1  # network size declaration and initialization
2  SIZE = 1024
3  network.size SIZE
4
5  # synchronous CD-protocol
6  CYCLES = 100
7  simulation.cycles CYCLES
8
9  # classic PPS protocol definition
10 protocol.loadBalancingProtocols loadBalancingProtocols.
11 PushPullSumProtocol
12 protocol.loadBalancingProtocols.linkable loadBalancingProtocols
13
14 # control classic PPS
15 control.avgo loadBalancingProtocols.PushPullSumObserver
16
17 # protocol to operate on
18 control.avgo.protocol loadBalancingProtocols
19 control.avgo.numberOfCycles CYCLES
```

Listing 5.1: Example configuration

Listing 5.1 shows a section of a configuration file used in this project. This configura-

tion sets up a simulation of the Push-Pull Sum algorithm for a network with 1024 nodes (**lines 2 and 3**) for 100 rounds (**lines 6 and 7** in Listing 5.1). From **lines 10 to 12**, the Push-Pull Sum algorithm is loaded. The way the algorithms are implemented, one algorithm consists of at least two Java classes, the *<ProtocolName>Protocol.java* and the *<ProtocolName>Observer.java*, and a shared *loadBalancingParameters.java*, which contains parameters like the cycle counter or topology-specific parameters as depicted in figure 13. At **line 15**, the *control* class is declared, and its usage follows in **lines 18 and 19**, where it is assigned parameters of type *protocol* and *numberOfCycles*. This process simultaneously handles steps two and three of creating a simulation by selecting the protocols to simulate and specifying control objects. Following that, the last step is to invoke the simulator class *peersim.Simulator.class*. For that, the IDE may be configured to call the simulator class on execution of the program code. Alternatively, a command line in the terminal can also invoke the simulator class. More on this in the PeerSim documentation [15].

PeerSim provides a wide range of classes and interfaces for simulations. In the cycle-driven approach, the framework offers the *CDProtocol* interface, which defines the *nextCycle* method. The *nextCycle* method is executed at the beginning of every simulation round.

Nodes in PeerSim are implemented as containers that hold various protocols. Each node is uniquely identified by a *nodeID* and interacts with the *Linkable* interface, which provides access to neighboring nodes. A class implementing the Linkable interface can override several methods, such as:

- **getNeighbor()**: Retrieves a neighbor with a specified ID.
- **degree()**: Returns the number of connections (or neighbors) a node has.
- **addNeighbor()**: Adds a neighbor to the node's set of neighbors.

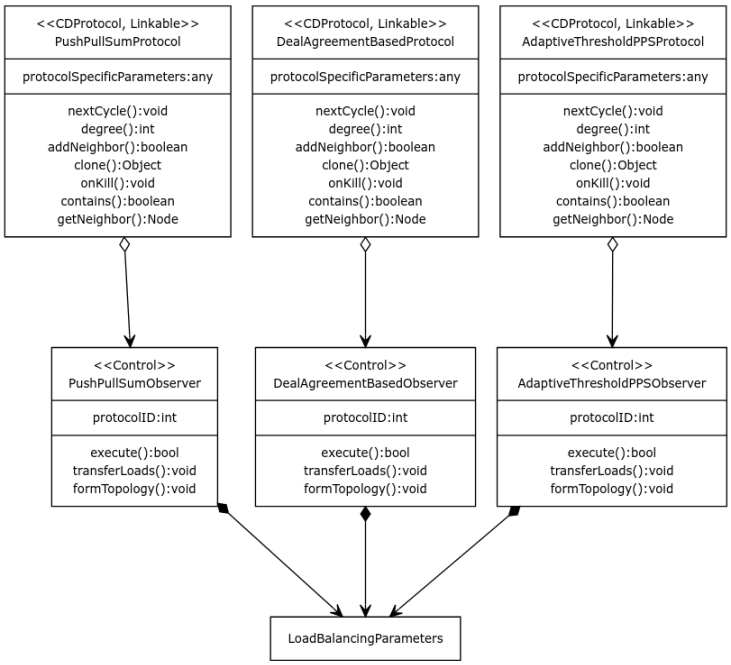


Figure 13: Project structure

To monitor or modify simulations, control objects are required. A class implementing the *Control* interface must define the execute method, which can be used to observe or alter the simulation at each round. [15]

5.3 Implementation Details

Figure 14 depicts a process model, modeling the method chosen to transition from experiment creation to data analysis and visualization. The methodology follows three steps. First, I wrote a Python script that generates configuration files where each node has uniformly distributed random load/sum values. 30 experiments were created to improve statistical significance. Then a Java script reads these configuration files and assigns an initial load value to each node. After that, the simulations are conducted. Each simulation outputs a file containing the simulation results, mainly the MSE per round, the loads per round, and the configuration of the

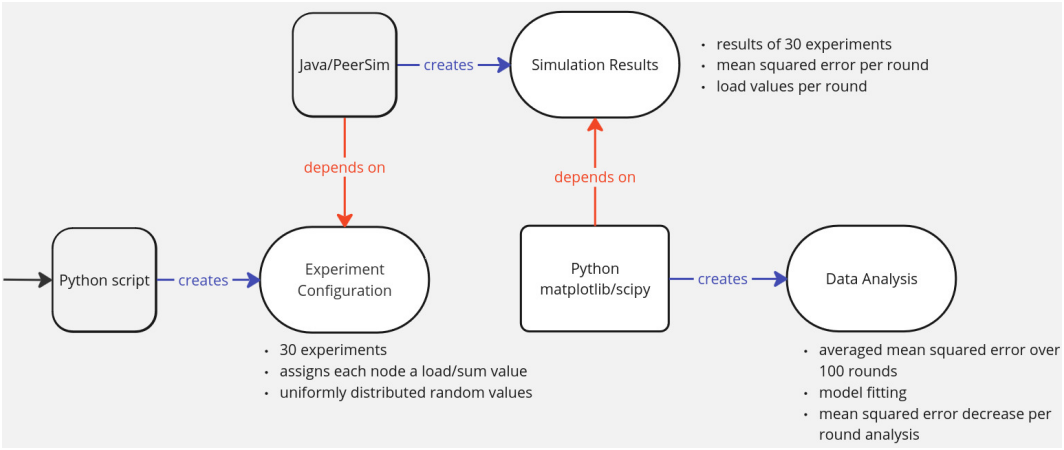


Figure 14: Process model: methodic

network (e.g., which topology is chosen, network size, etc.). The simulation results are averaged per round and then analyzed. Out of the simulation results, plots are generated, showing the MSE reduction per round in log-log or log-linear graphs. Additionally, model-fitting techniques are applied to analyze trends in the data.

6 Simulation Outcomes

When analyzing load balancing algorithms, the stability and efficiency of the algorithms are considered. Stability refers to how well an algorithm balances any initial load distribution across a network [4]. In the following, this is tested 1. by conducting 30 independent experiments for each topology, each with different initial load values per node 2. by choosing six different topologies (some with varying structural symmetry, e.g., $L_{32,32}$, $L_{128,8}$, and $L_{8,128}$). Efficiency measures how fast an algorithm achieves a balanced state in the network. To test this, the MSE is chosen as a metric and compared over the rounds to see which algorithm achieves low error values faster. A lower MSE at an earlier stage indicates higher efficiency.

For clarity, the load balancing algorithms are abbreviated:

DAB - Single-Proposal Deal-Agreement-Based algorithm

PPS - Push-Pull Sum algorithm (classic or traditional PPS)

ATPPS - Adaptive Threshold Push-Pull Sum algorithm (adaptive PPS)

The simulation outcomes are presented in log-log or log-linear graphs (logarithms are base 10), since the MSE varies significantly over the 100 simulation rounds. Each simulation outcome includes an analysis of the slopes for three regions of the x-axis (simulation rounds) and the general slope computed over all 100 rounds. The slopes are computed in either the log-log representation or the log-linear representation.

The plots in the log-log representation help analyze power-law relationships of the form:

$$MSE_r = a * r^b, \tag{8}$$

where relative differences matter more than absolute differences. A slope of b means that a 1% increase in r leads to a b% increase in MSE. A slope greater than 1 indicates that the MSE increases faster than r. A slope between 0 and 1 means the MSE increases slower than r. A negative slope indicates that the MSE decreases as r increases. [16]

The slopes of the log-log representation are calculated as:

$$(\frac{\log_{10} MSE_{r_2} - \log_{10} MSE_{r_1}}{\log_{10} r_2 - \log_{10} r_1}). \tag{9}$$

The slopes for the log-linear representation are calculated as:

$$(\frac{\log_{10} MSE_{r_2} - \log_{10} MSE_{r_1}}{r_2 - r_1}) \tag{10}$$

[17]. The MSE data over time is modeled using linear regression, polynomial regression, exponential regression, or logarithmic regression, depending on the best-fitting model.

Linear Regression: Linear regression models the relationship between MSE and simulation rounds as:

$$MSE_r = m * r + b, \tag{11}$$

where m is the slope, calculated as $m = \frac{\Delta MSE}{\Delta r} = (\frac{MSE_{r_2} - MSE_{r_1}}{r_2 - r_1})$. In this context the slope is mostly negative, as MSE_r decreases in comparison to MSE_{r-1}. b is the initial MSE at round $r = 0$, which represents the initial imbalance of the network before any load balancing is applied.

Polynomial Regression: The polynomial regression model is expressed as:

$$MSE_r = a_0 + a_1 * r + a_2 * r^2 + a_3 * r^3 + ... + a_n * r^n. \tag{12}$$

This model captures non-linear relationships between the independent variable r and dependent variable MSE_r [18]. Polynomial regression can model trends with rapid MSE reduction initially and slower reduction in advanced rounds.

Exponential Regression: The exponential regression model is given by:

$$MSE_r = a * e^{-b*r}. \tag{13}$$

Exponential models capture an initially steep MSE reduction, followed by slow MSE reduction in later rounds. a represents the initial MSE value. A larger a indicates a higher initial load imbalance in the network. b is the decay rate. It captures how quickly the MSE decreases per round r. Since MSE decreases over time, b is negative. A larger negative value for b indicates a faster error reduction in the network, while values closer to 0 indicate slower error reduction.

Logarithmic Regression: The logarithmic regression models data as:

$$\log(MSE_r) = a + b * \log(r), \tag{14}$$

where a is the initial MSE and b is the rate of reduction per unit increase in $\log(r)$. A large negative b decreases the MSE faster. If b is small, the error reduction slows down as the rounds progress. This model is effective in analyzing trends of load balancing algorithms where, in initial rounds the MSE drops significantly, followed by smaller reduction in later rounds. [19]

A sensitivity factor of $k = 0.1$ was selected for all simulations. This may not be the best choice for sparse topologies, but a uniform value was chosen for all topologies to ensure a consistent choice.

6.1 Complete Graph

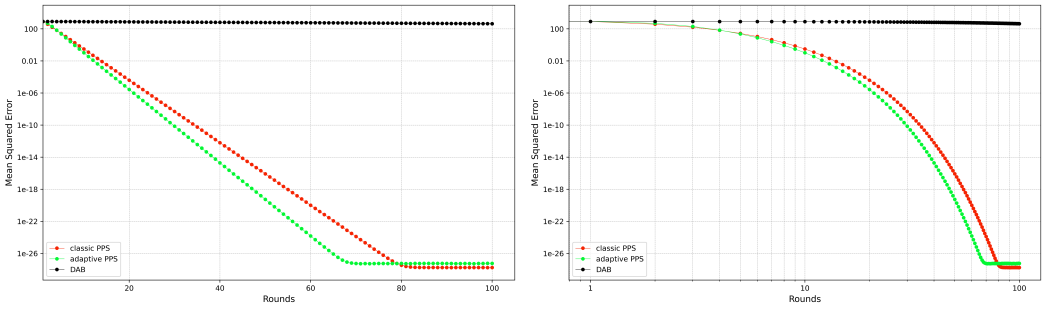


Figure 15: Complete graph - mean squared error per rounds - log-linear and log-log

Figure 15 a) presents the MSE reduction per round for the three load balancing algorithms simulated on the Complete graph K_{1024} on a log-linear graph. Figure 15 b) shows the log-log graph of the same. The DAB curve (black curve) shows a supposedly linear trend and a slight decrease of the MSE over the 100 rounds. The DAB performs poorly since it uses a deterministic load redistribution rule for its nodes, where each node selects the minimally loaded neighbor and proposes a load transfer. In a Complete graph where each node is interconnected, the same few nodes that hold the load of amount L_{min} are the nodes that are receiving all the proposals. Since each minimally loaded neighbor only accepts one proposal per round, the number of load transfers is heavily limited, resulting in slow MSE reduction. The PPS and ATPPS, which use a randomized transfer partner selection, exploit the high connectivity of the network more effectively. The PPS curve (red curve) exhibits a steep decrease in MSE. As the Complete graph is a regular graph, where each pair of distinct nodes is connected, no structural constraints slow down the mechanism of pushing and pulling loads from neighbors. However, PPS does not distinguish between nodes that are highly imbalanced and those that are nearly balanced, which leads to unnecessary load transfers in later rounds. The ATPPS (green curve) achieves even faster MSE reduction than PPS. As the system nears equilibrium, ATPPS reduces redundant exchanges with low impact. This avoids

further computational overhead.

The stagnation of the red and green curves at around 1.8×10^{-19} (-2^{64}) in the simulation is due to the limitations of Java's double precision, which cannot represent all integers exactly beyond a threshold. As a result, small increments become too insignificant to alter the MSE value, which causes the stagnation of the curve. In summary, DAB underperforms due to limitations on the amount of load transfers. PPS improves upon this but lags behind ATPPS in error reduction. ATPPS balances load while avoiding unnecessary exchanges, making it the most effective approach in this setting.

Figure 16 a) shows the linear regression fit for the MSE data of the DAB. The data aligns well with the model, but the exponential regression fit suits better in this case. Figure 16 b) shows the exponential regression fit for the MSE data when the DAB as a load balancing strategy is applied to the network. The MSE data fits with the exponential regression model following the equation:

$$MSE_r = 844.63 * e^{-0.01r}. \tag{15}$$

The decay rate of -0.01 indicates a very slow decrease in MSE. The MSE has hardly decreased over the 100 rounds of simulations. The MSE data of the PPS is fitted to the exponential regression model for rounds 10 to 80, as seen in figure 17. The best-fit follows the equation:

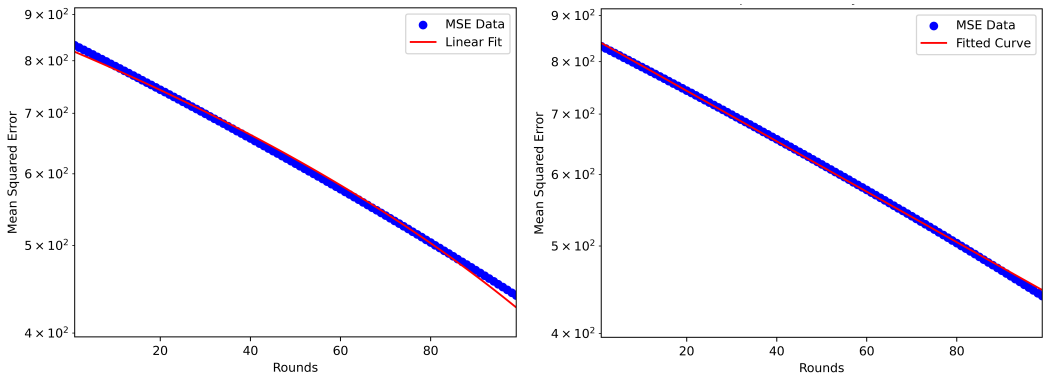
$$MSE_r = 2530.41 * e^{-0.9r}. \tag{16}$$

The decay rate of -0.9 suggests a steep decline in error. In Figure 18, the exponential regression fit is visualized for the MSE data of the ATPPS curve in the Complete graph for rounds 10 to 65. The fitted curve is expressed by the equation:

$$MSE_r = 4309.94 * e^{-1.06r}. \tag{17}$$

A steep error reduction is indicated by the decay rate of -1.06. As the PPS, the ATPPS reduces the error very effectively in an exponential manner. Rounds 66 to 100 show a plateauing of the MSE data, caused by precision problems of the double datatype in Java. The adaptive mechanism provides a faster decline in error outlined by the decay rate of -1.06 versus the one of the PPS of -0.9.

Figure 19 visualizes a heat map of the slopes in both the log-linear a) and log-log b) representations for the three load balancing algorithms across different regions of the graph. The values reflect how steeply the MSE decreases over rounds. In the start region (rounds 1 to 10), PPS and ATPPS achieve a decay rate of -0.38 and -0.43, while DAB only reaches a value of -2.6×10^{-3}. In this phase, PPS and ATPPS have a much faster exponential decay in MSE than DAB, meaning they reduce error more effectively at the start. In the middle region (rounds 11 to 65), the PPS and ATPPS algorithms slightly accelerate the error reduction to -0.39 for the PPS and -0.46 for the ATPPS. DAB proceeds to reduce the error in a slow manner. The decay rate is as low as -2.7×10^{-3} for the DAB. DAB has shallower negative slopes overall. The stagnation behavior described above can also be seen in the decay rate, which is close to zero in the final region, i.e., precisely when the ATPPS curve begins to stagnate. The PPS curve does not stagnate until round 80, so the decay rate is still relatively high. The general decay rates (rounds 1-100) for PPS and ATPPS (-0.3) are steeper than DAB (-2.8×10^{-3}), which suggests that PPS and ATPPS converge much faster on average. The MSE is reduced significantly by the PPS and ATPPS. The initial MSE value of 832 is reduced to 1.72×10^{-28} for the PPS and 5.63×10^{-28} for the ATPPS, while the DAB reduced the error to nearly half of the initial value, achieving a value of 436.84.



(a) **(b)**

Figure 16: Complete graph - linear and exponential regression - DAB

6.2 Star Graph

The behavior of the curves for the Star graph S_{1024} simulations in figure 20 is similar to those for the Complete graph, with the difference that this time the PPS curve falls more steeply than the ATPPS curve. Figure 20 a) shows the log-linear representation of the MSE data, while figure 20 b) shows the log-log representation. Again, DAB shows a much slower MSE reduction compared to PPS and ATPPS, as indicated by the flatter slope of its curve. It converges minimally, with MSE reducing in a slow manner over rounds. This points out that DAB is less efficient in balancing load in a Star graph. An explanation for that is that, in the Star graph, all the leaves have the central node as a neighbor, meaning each leaf node selects the same node to propose load transfers. When the central node has a higher load than the leaf requesting a transfer, no load transfer occurs between these two nodes. The number of load transfers, therefore, is heavily limited. Both PPS and ATPPS exhibit steep MSE reductions early on, as seen in the sharp downward trends in the log-log plot. The PPS balances load more efficiently than the ATPPS since it reduces MSE faster and reaches lower MSE values sooner (as of round 45). Again, the PPS and ATPPS curves start to plateau as a result of the precision of the double datatype in Java. The Push-Pull Sum-based algorithms draw an advantage from the Star graph, as

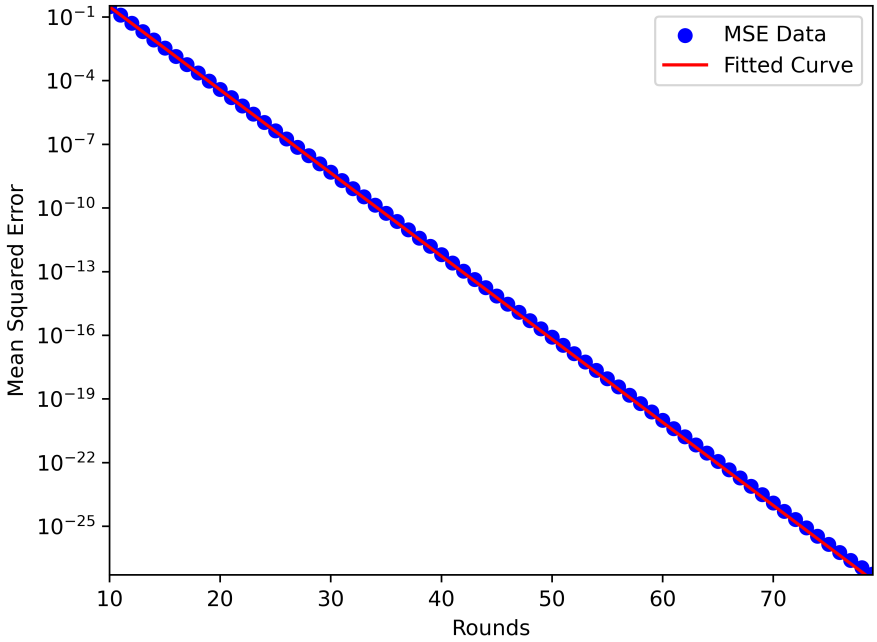


Figure 17: Complete graph - exponential regression fit - PPS

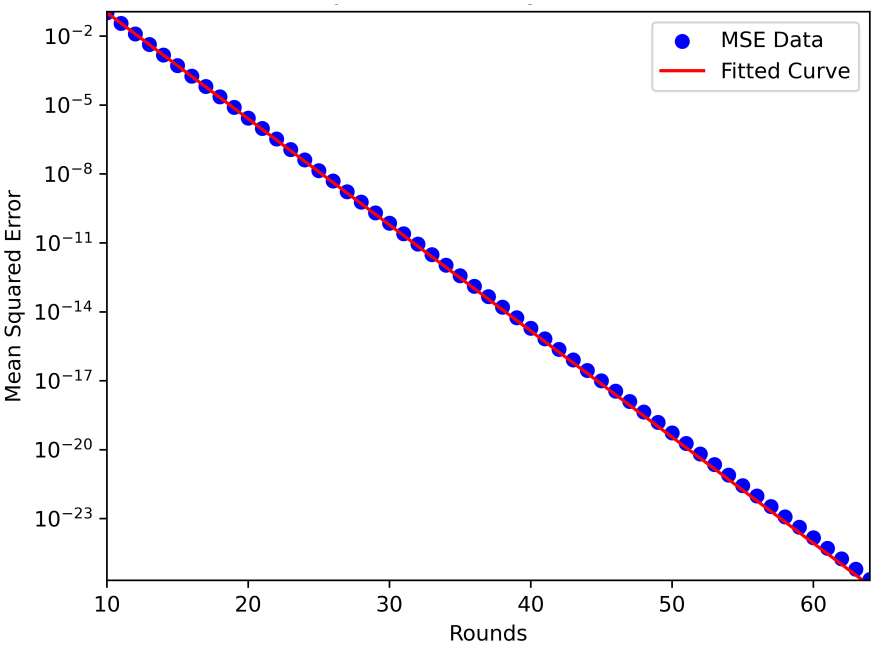


Figure 18: Complete graph - exponential regression fit - ATPPS

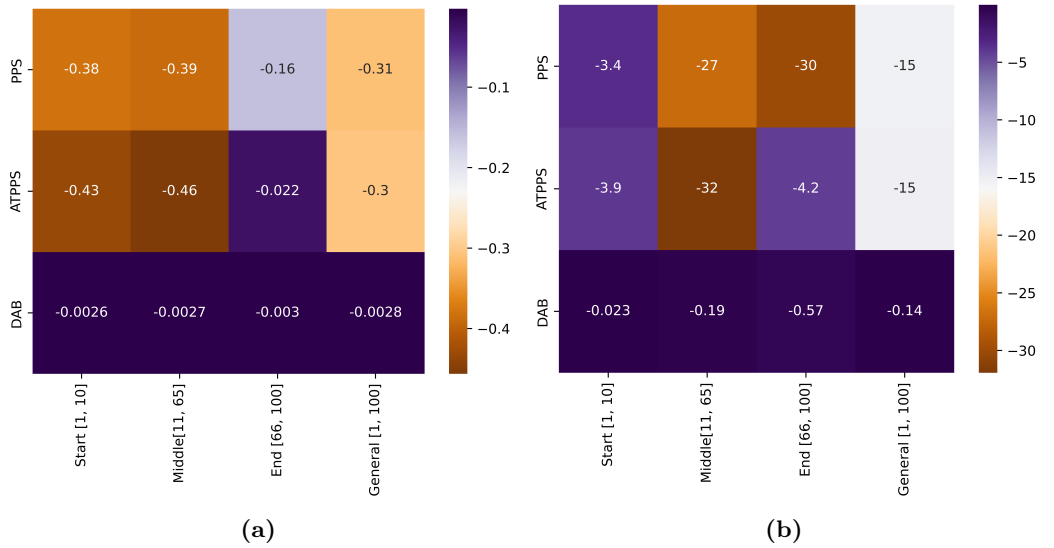


Figure 19: Complete graph - heat map of slopes per region - log-linear and log-log

the redistribution happens through the central node, which is chosen by every leaf node as a push destination. Following that, the central node redistributes parts of the load to the leaf nodes in magnitude of $\left(\frac{\frac{s_{i,r}}{2}}{N-1}, \frac{\frac{w_{i,r}}{2}}{N-1}\right)$.

As seen in figure 21 b), the MSE data for the DAB-balanced network aligns nearly perfectly with the fitted curve of the exponential regression model given by the equation:

$$MSE_r = 840.42 * e^{-0.01r} \tag{18}$$

for rounds 1 to 100. The decay rate of -0.01 indicates a very slow reduction in error. Since a linear decrease is suspected, the MSE data is also fitted to the linear regression model as presented in 21 a). However, the exponential regression model aligns better with the DAB MSE data. From round to round, the improvement in the network remains minimal. This highlights the inability of the DAB to balance the network within 100 rounds to a satisfactory level. Figures 22 and 23 show the fitted curves for the MSE data of the PPS and ATPPS algorithms, respectively. The

best-fit model for the MSE data of the PPS load balancing algorithm between rounds 10 and 45 follows the equation:

$$MSE_r = 29794.60 * e^{-1.39r}. \tag{19}$$

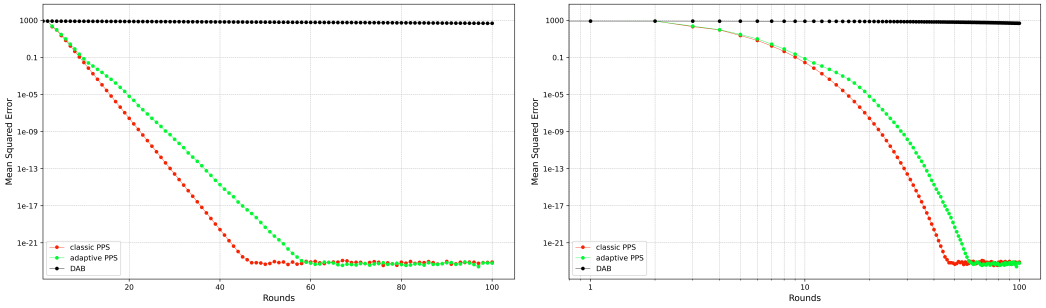
The rounds 10 to 45 exhibit the steepest decline in MSE, and for that reason, they have been fitted to an exponential regression model. The decay rate of -1.39 indicates a fast reduction in error, especially in comparison to DAB. For the ATPPS, the MSE data fits best with the exponential model in the range of rounds 18 to 60, following the equation:

$$MSE_r = 9329.40 * e^{-1.05r}. \tag{20}$$

In Star graphs, the central node dominates communication, and load balancing heavily depends on that node. The adaptive threshold mechanism does not significantly impact performance under these conditions. The discrepancy between the two Push-Pull Sum-based algorithms can be explained by the fact that the ATPPS limits the number of messages due to the conditional threshold-based load transfer, which means that there is less interaction compared to the PPS.

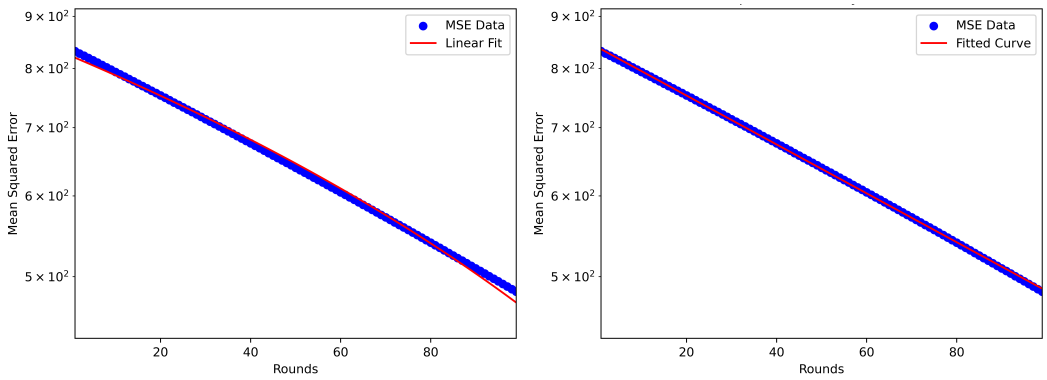
Overall, the situation is similar to the Complete graph, with the Push-Pull Sum-based algorithms performing quick error reduction and the DAB reducing the error exponentially at a very slow rate. This behavior is captured in the heat map of figure 24, where both trends, log-linear a) and log-log b) are presented. In the first 10 rounds, the PPS achieves the fastest decay with a decay rate of -0.5, followed by the ATPPS with a decay rate of -0.45. The DAB experiences a very slow decay with a value of -2.3×10^{-3}. Between rounds 11 and 45, the PPS even reaches a faster decay rate of -0.6. The ATPPS and DAB decay rates do not change in this region. Similar to the Complete graph, the PPS and ATPPS curves stagnate again. Now with the difference that the PPS stagnates in an earlier round than the ATPPS. This can be seen again in the final region, where the decay rate of the PPS approaches

zero. The DAB converges slowly, as reflected in its consistently shallow decay rates. The MSE decreases from an initial value of 832 to approximately 480.47 for the DAB, while for the PPS and ATPPS algorithms, the MSE reaches values of 8.3×10^{-24} for the PPS and 6.5×10^{-24} for the ATPPS.



(a) **(b)**

Figure 20: Star graph - mean squared error per rounds - log-linear and log-log



(a) **(b)**

Figure 21: Star graph - linear and exponential regression - DAB

6.3 Ring Graph

For the Ring graph R_{1024}, the DAB curve in figure 25 shows the steepest decline in error in the first few rounds of the simulation. The slopes in figure 29, represent the exponents in the power-law relationship between the MSE and r. The slope of the

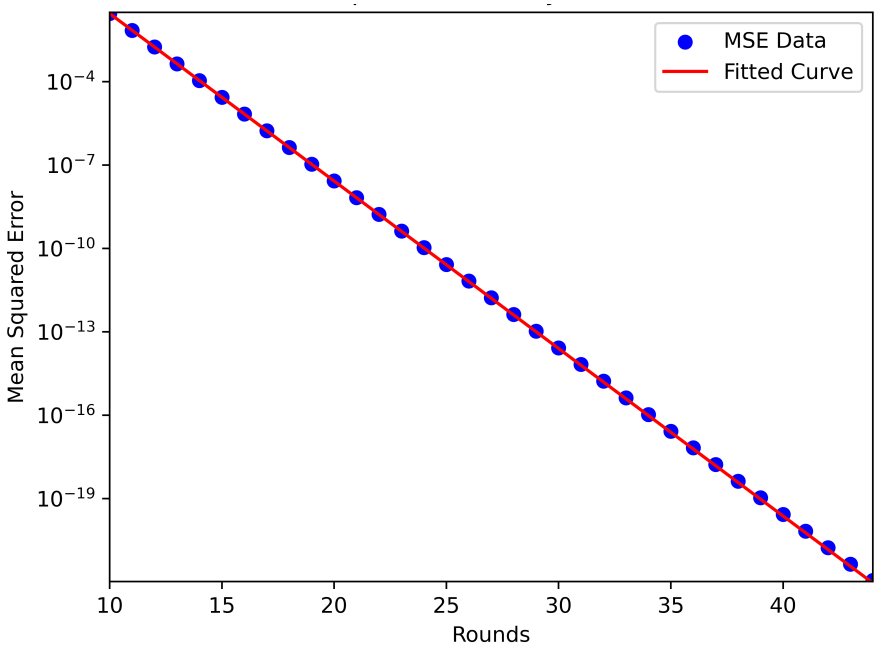


Figure 22: Star graph - exponential regression fit - PPS

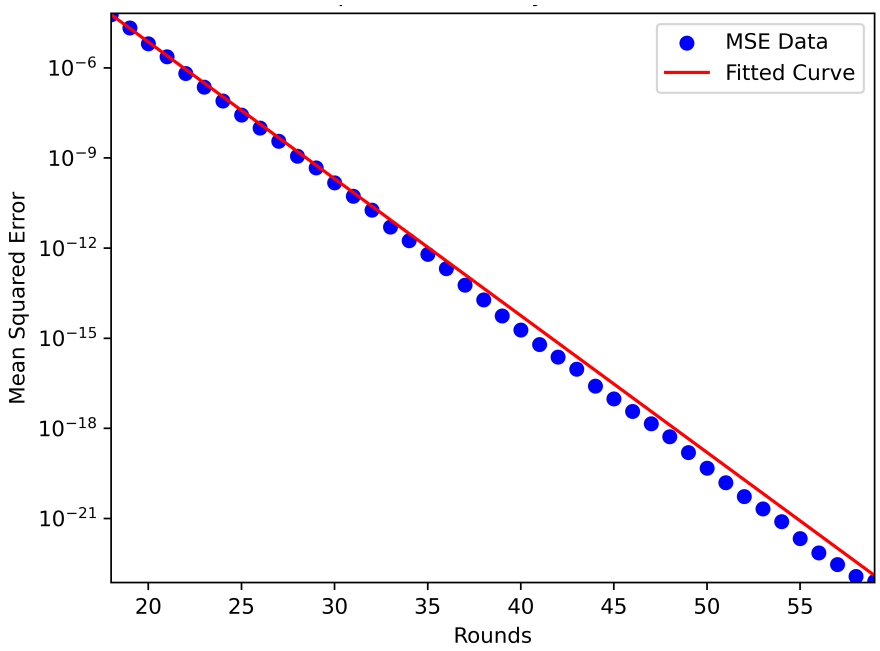


Figure 23: Star graph - exponential regression fit - ATPPS

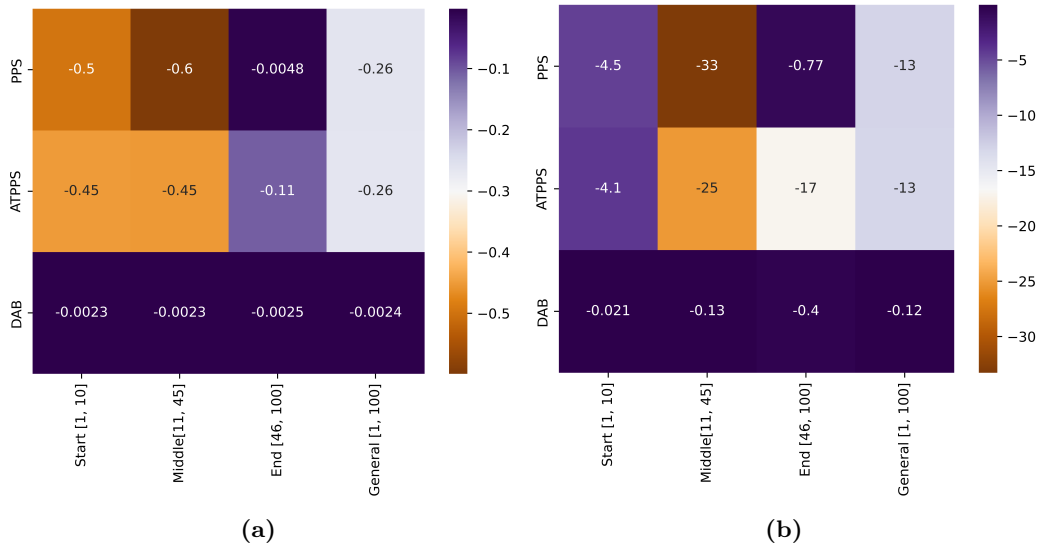


Figure 24: Star graph - heat map of slopes per region - log-linear and log-log

first 10 rounds is -1.1 for the DAB curve compared to -0.94 ± 0.1 for each Push-Pull Sum-based algorithm. The slopes decrease for all three algorithms to -0.53 ± 0.3 in the middle region and remain at these values for the end region. The near-overlap of the PPS and ATPPS curves across the 100 rounds of simulation suggests that the adaptive mechanism provides no additional benefit in a Ring graph, where selecting a random neighbor might already target an optimal load transfer partner. In a Ring topology, each node only interacts with its two immediate neighbors. The RN set for a node, therefore, has exactly one neighboring node in it, as $\lceil log_2(2) \rceil = 1$. The adaptive threshold mechanism relies on significant load differences between nodes to trigger transfers, but in a Ring graph, local load differences might not be significant enough to make the threshold mechanism advantageous. For the PPS, every node communicates with its neighbors in every round. The adaptive threshold might reduce some of these exchanges. However, in a Ring graph, where the network diameter is large, reducing communication might delay convergence rather than improve it. The threshold mechanism therefore cannot draw an advantage over the PPS. The DAB performs better in this scenario since it always interacts with the

optimal partner to exchange loads. The benefit of randomness vanishes for a network topology where each node only has two neighbors, and a deterministic approach actually performs better.

The MSE data from each algorithm's simulations were fitted to a polynomial regression model of degree 4. The best-fit model for the DAB MSE data for rounds 10 to 60 follows the equation:

$$MSE_r = 1.72 \times 10^{-5} r^4 - 2.30 \times 10^{-3} r^3 + 0.19 r^2 - 5.99 r + 114.83 \tag{21}$$

as shown in figure 26. For small r, the constant term dominates. As r increases, the higher-degree terms influence the behavior of the curve more and more. The MSE data of the PPS is fitted to a fourth-degree polynomial model:

$$MSE_r = 2.99 \times 10^{-5} r^4 - 5.0 \times 10^{-3} r^3 + 0.32 r^2 - 9.68 r + 166.30 \tag{22}$$

as shown in figure 27. The ATPPS MSE data is also fitted to a fourth-degree polynomial:

$$MSE_r = 3.04 \times 10^{-5} r^4 - 5.0 \times 10^{-3} r^3 + 0.32 r^2 - 9.64 r + 161.86 \tag{23}$$

as depicted in figure 28. The polynomial fit suggests that MSE reduction slows down over time.

6.4 Torus Grid Graph

Figure 30 shows the MSE reduction over rounds on a Torus Grid graph $T_{32,32}$, plotted on a log-log scale. The DAB curve has a slightly faster initial reduction compared to PPS' and ATPPS' curves in the beginning (rounds 1 to 7). The slope in this region is superior for the DAB with a value of -2.1 compared to approximately -1.5 for the

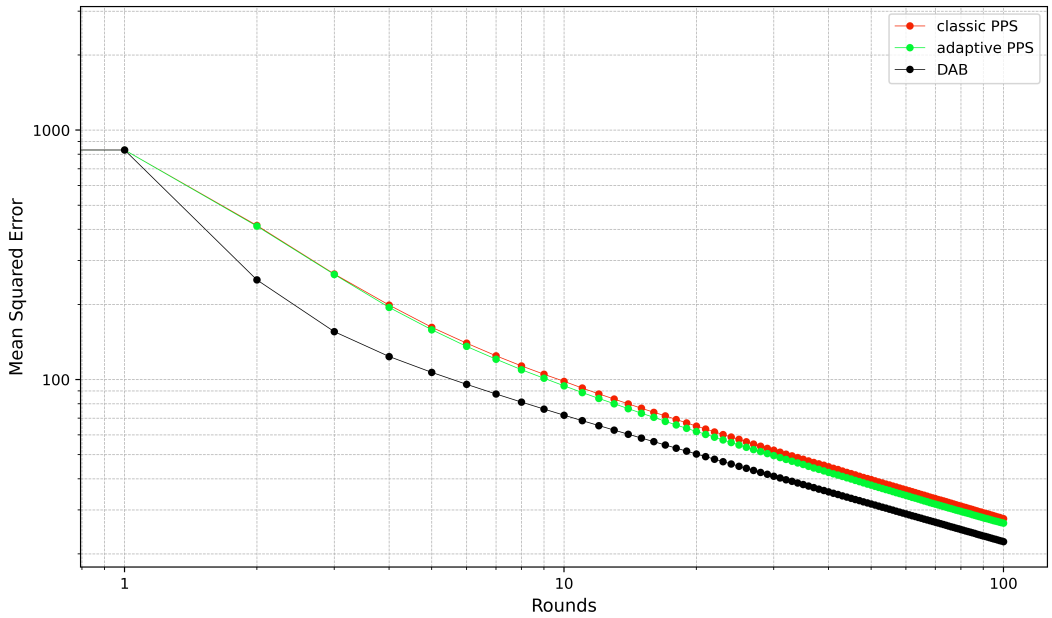


Figure 25: Ring graph - mean squared error per rounds - log-log

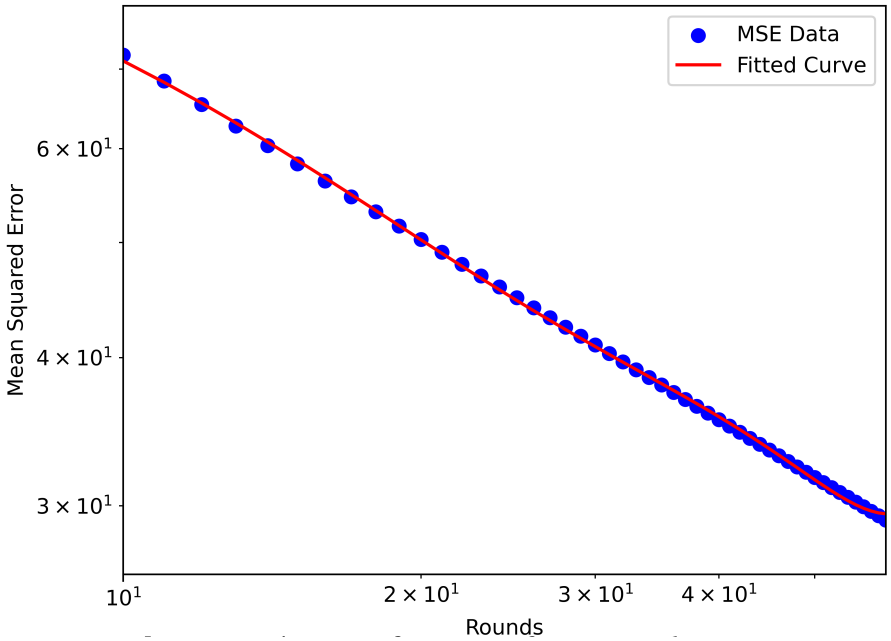


Figure 26: Ring graph - polynomial regression fit - DAB

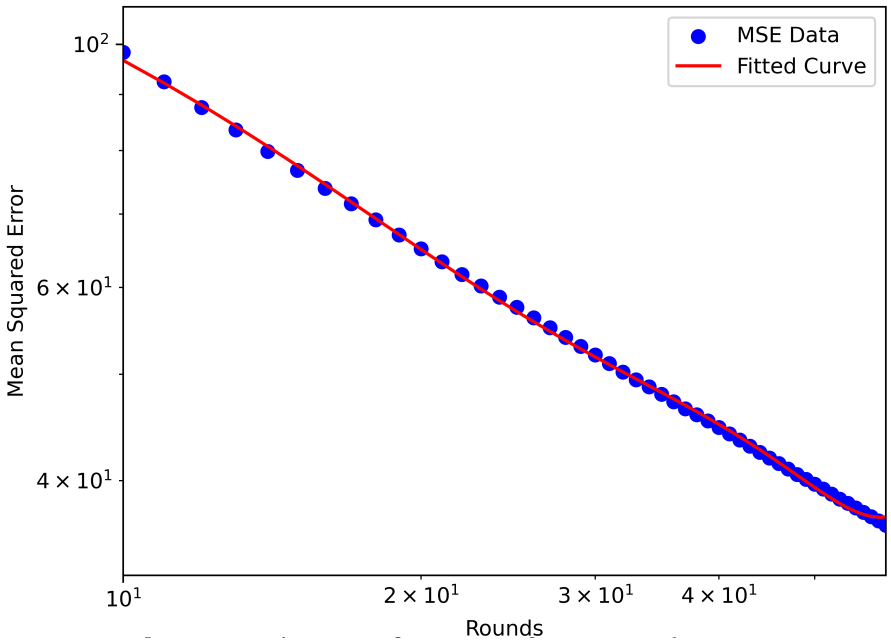


Figure 27: Ring graph - polynomial regression fit - PPS

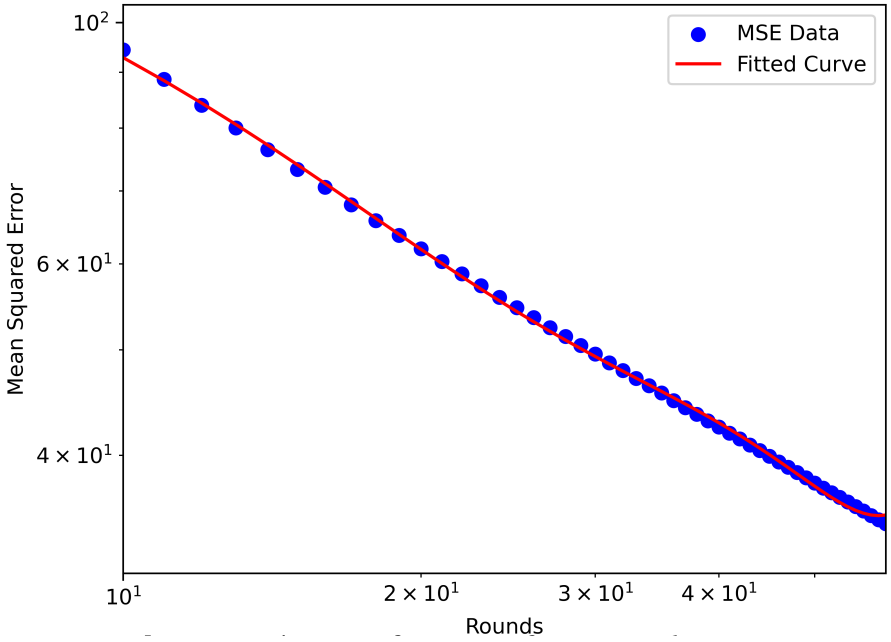


Figure 28: Ring graph - polynomial regression fit - ATPPS

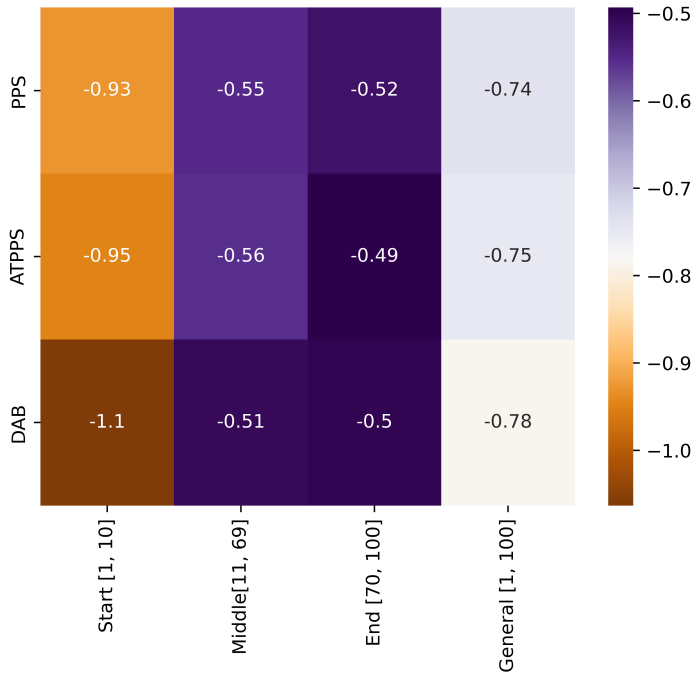


Figure 29: Ring graph - heat map of slopes per region - log-log

Push-Pull Sum-based algorithms (figure 34 - log-log). The PPS and ATPPS curves maintain nearly identical performances during the middle region (rounds 8 to 40) and reduce MSE at a similar rate, with a slope of -1.2 for the PPS curve and -1.3 for the ATPPS curve. DAB shows a similar slope as the PPS (both have a slope of -1.2). In the final region, DAB scores the highest slope with a value of -2, followed by the ATPPS with a value of -1.7. The PPS achieves a slope of -1.4 in this region. This suggests that DAB and ATPPS adapt more efficiently to the graph's structure during advanced rounds. PPS and ATPPS lag behind the DAB in reaching such low MSE values after 100 rounds. DAB shows a superior performance in Tori. The ATPPS achieves a balanced trade-off in this scenario. It outperforms the PPS approach, particularly in later rounds, where its adaptive mechanism prevents redundant load transfers and prioritizes exchanges that have a more significant impact on error reduction.

The uniform neighborhood structure of Tori ensures that DAB's deterministic deci-

sions (e.g., always choosing the minimally loaded neighbor and proposing to it) are effective in this scenario. The algorithm's deterministic neighbor selection prevents suboptimal decisions introduced by probabilistic neighbor choices, which makes it well-suited to the topology. Both PPS and ATPPS protocols rely on randomly selecting a neighbor for load exchange. While this randomness is beneficial in irregular or dense graphs (e.g., Star or Complete graphs), the DAB's deterministic approach seems to perform better in the more structured Torus Grid. The Push-Pull Sum-based algorithms do not always target the most unbalanced areas. This means that load propagation can sometimes "stall" in certain regions, where they require more rounds to achieve global balance. The ATTPS draws its benefit over the PPS (especially in later rounds) by deciding which option of the available subset is the best. The discrepancy between the two load balancing algorithms widens in the last few rounds. Trades between two nodes with higher load differences are more impactful once the network is mostly balanced.

The fitted polynomial curve of degree 5 matches the MSE data for DAB. For rounds 10 to 39, the MSE data is fitted to the polynomial regression model following the equation:

$$MSE_r = -1.35 \times 10^{-6} r^5 + 1.89 \times 10^{-4} r^4 - 0.01 r^3 + 0.30 r^2 - 4.6 r + 34.10 \quad (24)$$

as shown in figure 31 a). This suggests that in these rounds, the MSE reduction follows a complex pattern, as the loads propagate to different regions of the graph thanks to the wrap-around edges of the Tori. In later rounds, the performance of the DAB can be captured by a polynomial curve of degree 3 with the equation:

$$MSE_r = -6.01 \times 10^{-6} r^3 + 1.66 \times 10^{-3} r^2 - 0.16 r + 6 \quad (25)$$

(figure 31 b)). At this stage, load balancing stabilizes, and the reduction in MSE becomes more linear or gradual. Thus, a lower-degree polynomial suffices to model the behavior in the later rounds. The PPS and ATPPS curves are fitted for rounds

10 to 39 to the equations:

$$MSE_r = -5.54 \times 10^{-6} r^5 + 7.65 \times 10^{-4} r^4 - 0.04 r^3 + 1.16 r^2 - 16.81 r + 112.86 \tag{26}$$

for the PPS (figure 32 a)) and:

$$MSE_r = -3.65 \times 10^{-6} r^5 + 5.16 \times 10^{-4} r^4 - 0.03 r^3 + 0.83 r^2 - 12.52 r + 88.16 \tag{27}$$

for the ATPPS (figure 33 a)). For rounds 40 to 100 the MSE data of the Push-Pull Sum-based algorithms are fitted to the equations:

$$MSE_r = -1.15 \times 10^{-5} r^3 + 3.21 \times 10^{-3} r^2 - 0.33 r + 13.72 \tag{28}$$

for the PPS (figure 32 b)):

$$MSE_r = -9.99 \times 10^{-6} r^3 + 2.80 \times 10^{-3} r^2 - 0.28 r + 11.29 \tag{29}$$

and for the ATPPS (figure 33 b)).

6.5 Lollipop Graph

6.5.1 (512, 512)-Lollipop Graph

Figure 35 displays the three curves of the load balancing algorithms in a log-log plot. The ATPPS slightly outperforms the PPS, while the DAB underperforms compared to both Push-Pull Sum-based methods in reducing the error. The superior performance of PPS and ATPPS compared to DAB in the Lollipop graph $L_{512,512}$ can be explained by the structure of the graph and the fundamental differences in how these algorithms operate. The clique region is characterized by high connectivity

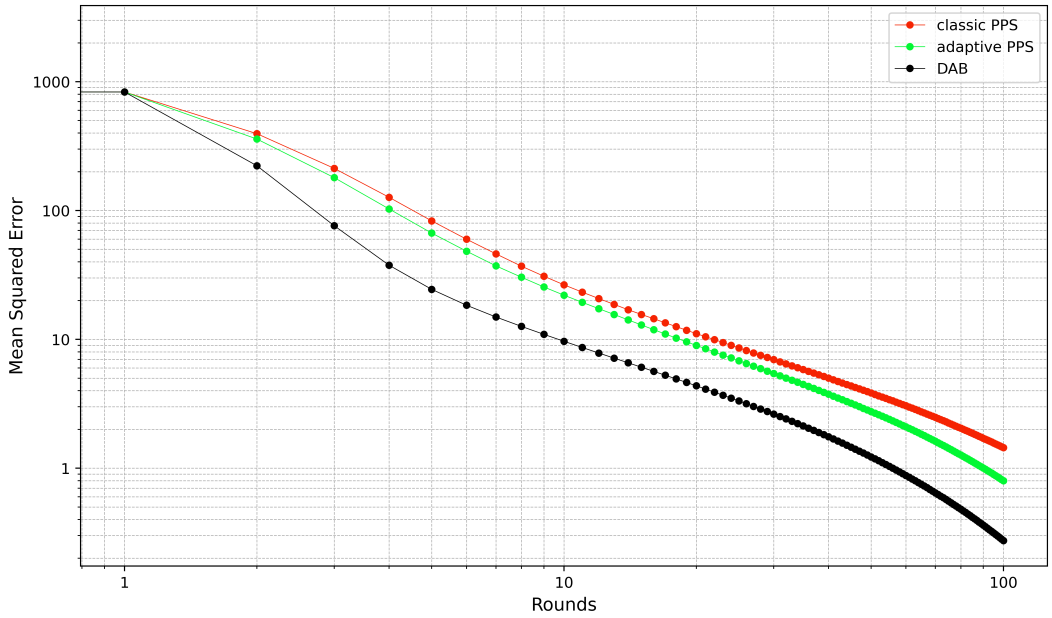


Figure 30: Torus Grid - mean squared error per rounds - log-log

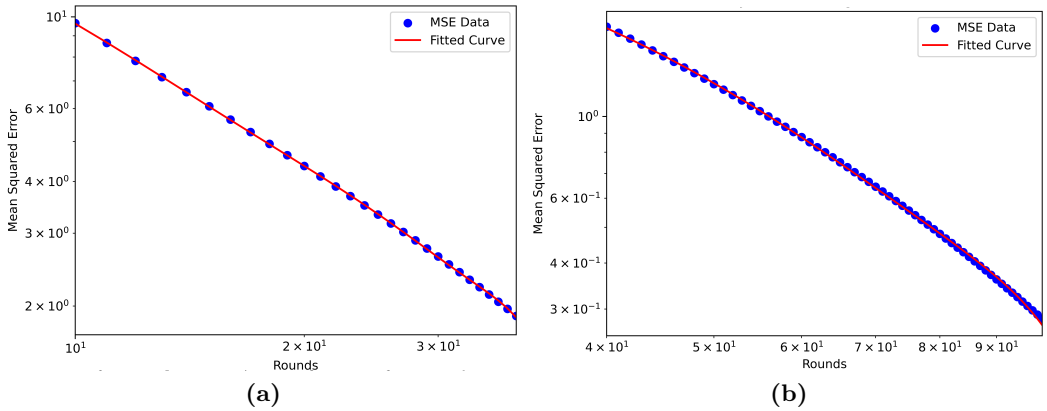
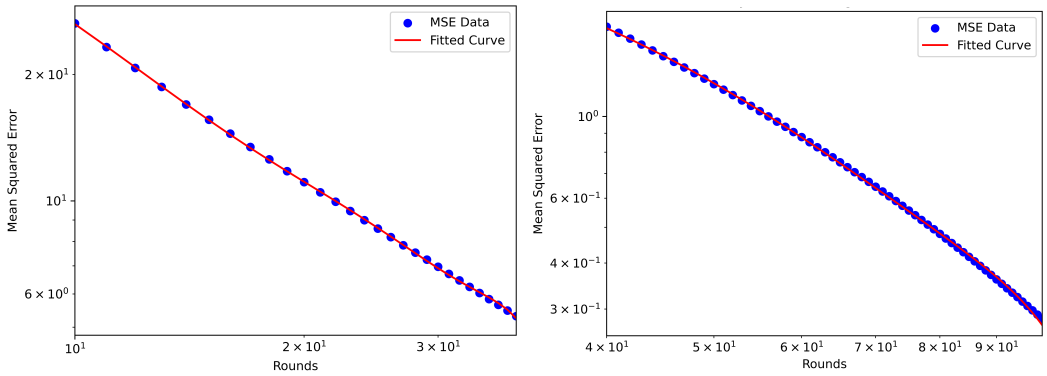


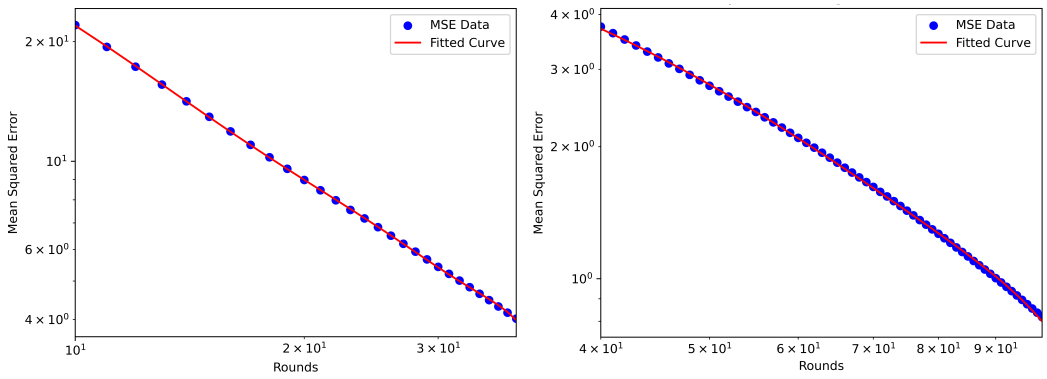
Figure 31: Torus Grid - polynomial regression fit - DAB; rounds 10-39 and 40-100

and enables fast local balancing due to the low diameter for the Push-Pull Sum-based algorithms. The path region with low connectivity slows down information propagation and balancing for the Push-Pull Sum-based algorithms and favors the deterministic DAB. The initial discrepancy between the DAB and the Push-Pull Sum-based algorithms in the first 10 rounds arises due to the fast error reduction achieved by PPS and ATPPS within the clique. The DAB struggles to perform well



(a) **(b)**

Figure 32: Torus Grid - polynomial regression fit - PPS; rounds 10-39 and 40-100



(a) **(b)**

Figure 33: Torus Grid - polynomial regression fit - ATPPS; rounds 10-39 and 40-100

for cliques, as observed in section 6.1. DAB performs rather well in the Path graph, which in theory is similar to the Ring graph without the edge connecting the first and last nodes as described in section 6.3. The proposal technique of the DAB prioritizes nodes with the least load, which can lead to inefficient propagation along the clique. The Push-Pull Sum-based algorithms achieve a steep downward slope with value -1.3 between rounds 1 and 7, while the DAB achieves a value of -0.34 (figure 39). In the mid-to-late phase, the slope flattens for both Push-Pull Sum-based algorithms. This is evident from the slope values dropping to as low as -0.56 for PPS and -0.57 for ATPPS. In this phase, the path section of the Lollipop graph dominates the

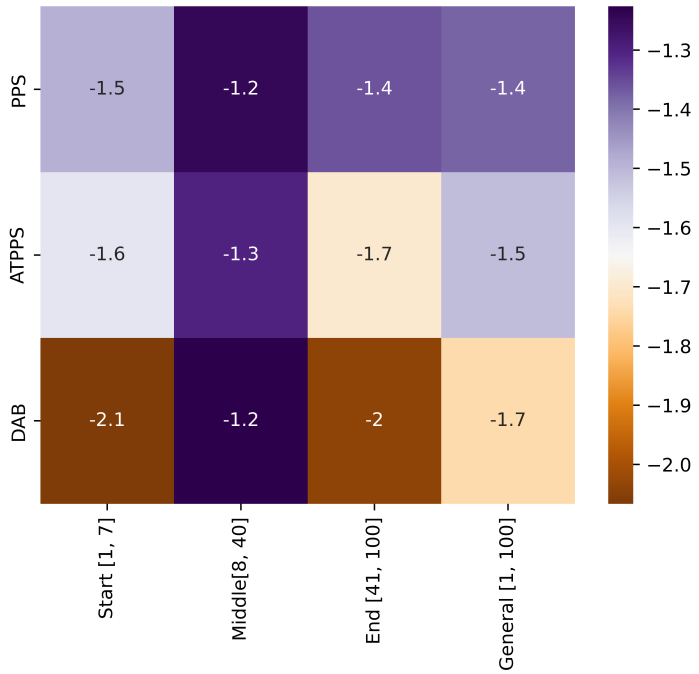


Figure 34: Torus Grid - heat map of slopes per region - log-log

residual imbalances. The ATPPS achieves nearly identical behavior to the PPS in the early rounds, as it starts with similar strategies where the threshold is relatively easy to surpass as the load differences in the network are still very high. ATPPS outperforms the PPS slightly in the later rounds due to its ability to dynamically adjust its balancing strategy based on the current state of load imbalances. This allows it to better address residual imbalances in the clique section of the Lollipop graph as showcased in section 6.3.

The MSE data for the DAB is best described by a polynomial of degree 3, following the equation:

$$MSE_r = -5.89 \times 10^{-5} r^3 + 0.03r^2 - 5.68r + 459.42 \tag{30}$$

as shown in figure 36. The MSE trends for the PPS and ATPPS algorithms are better captured by polynomials of degree 4. The best-fit equation for the PPS model

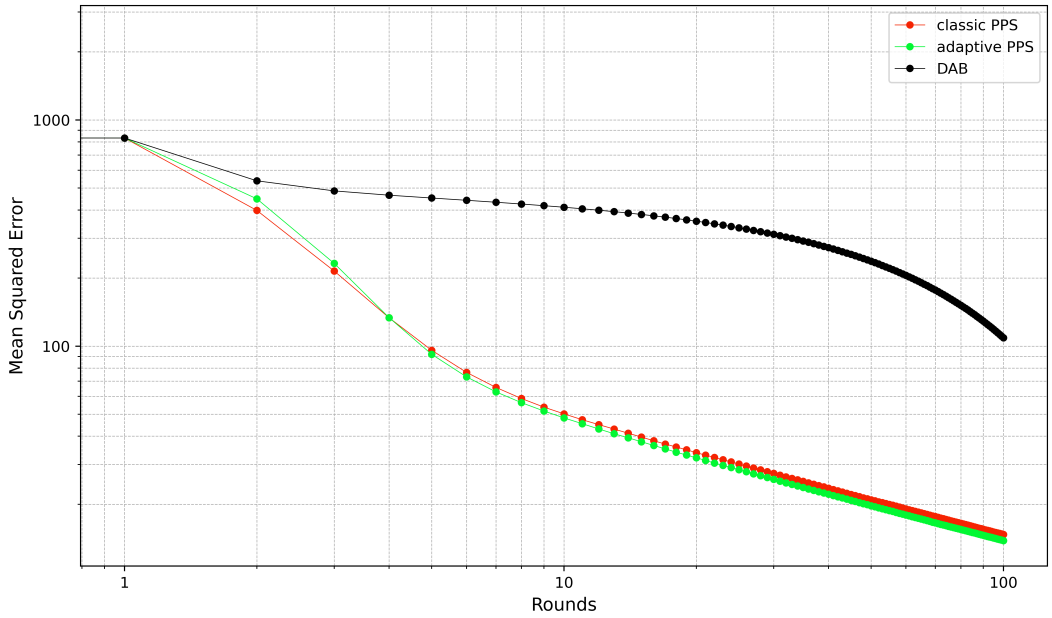


Figure 35: (512, 512)-Lollipop graph - mean squared error per rounds - log-log

is:

$$MSE_r = 8.44 \times 10^{-7} r^4 - 2.52 \times 10^{-4} r^3 - 0.03 r^2 - 1.64 r + 56.68 \tag{31}$$

(figure 37) and the ATPPS model fit follows the equation:

$$MSE_r = 8.69 \times 10^{-7} r^4 - 2.56 \times 10^{-4} r^3 + 0.03 r^2 - 1.62 r + 54.48 \tag{32}$$

(figure 38).

To analyze the impact of the path size and clique size on the simulation results, additional simulations were conducted with varying proportions of nodes assigned to each region. In subsection 6.5.2, the number of nodes assigned to the clique is reduced to 128, which is one-fourth of the previous value of 512. Consequently, the path section now consists of 896 nodes. The total network size remains unchanged. This adjustment allows us to observe how a larger path section, compared to a smaller clique, influences the load balancing behavior of the different algorithms. Similarly,

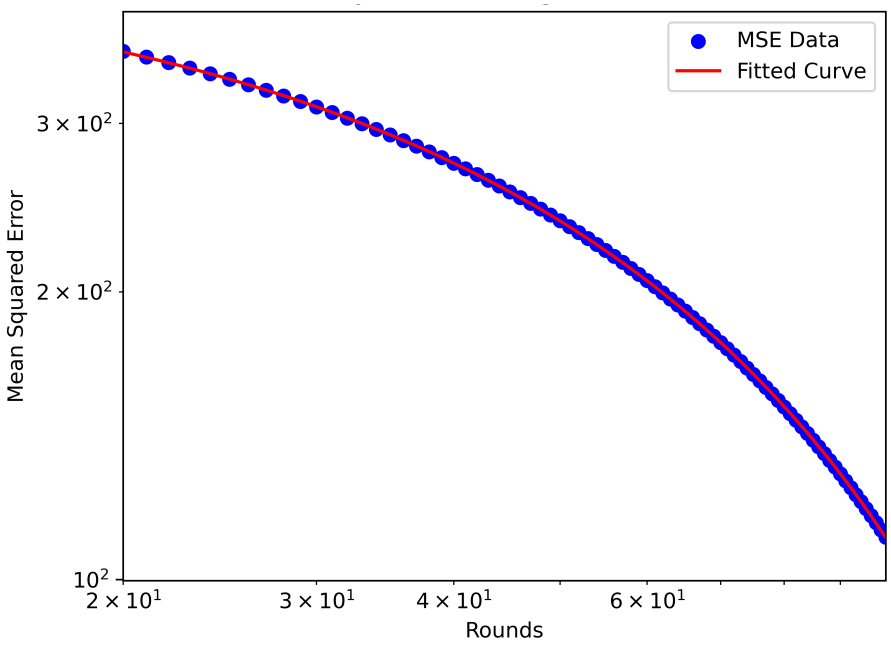


Figure 36: (512, 512)-Lollipop graph - polynomial regression fit - DAB

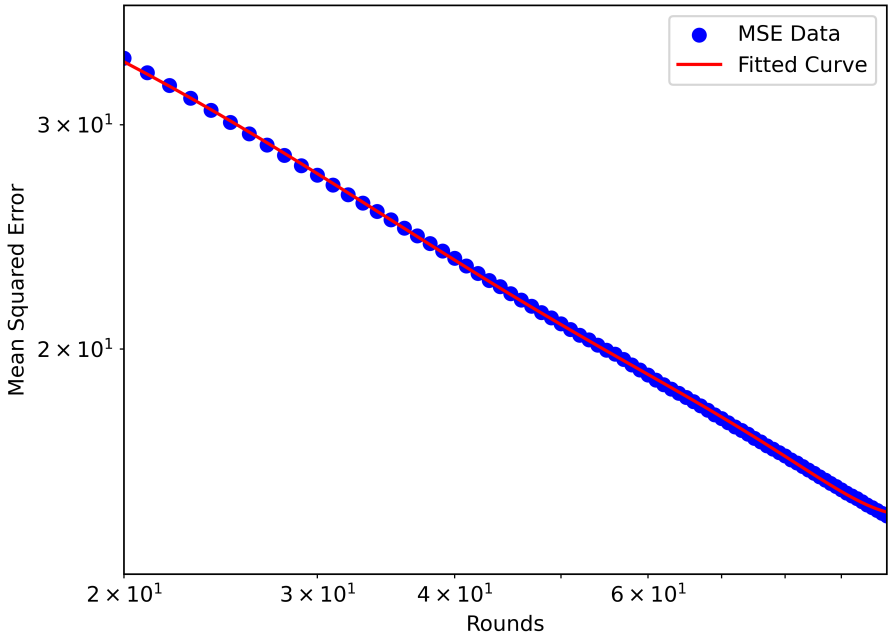


Figure 37: (512, 512)-Lollipop graph - polynomial regression fit - PPS

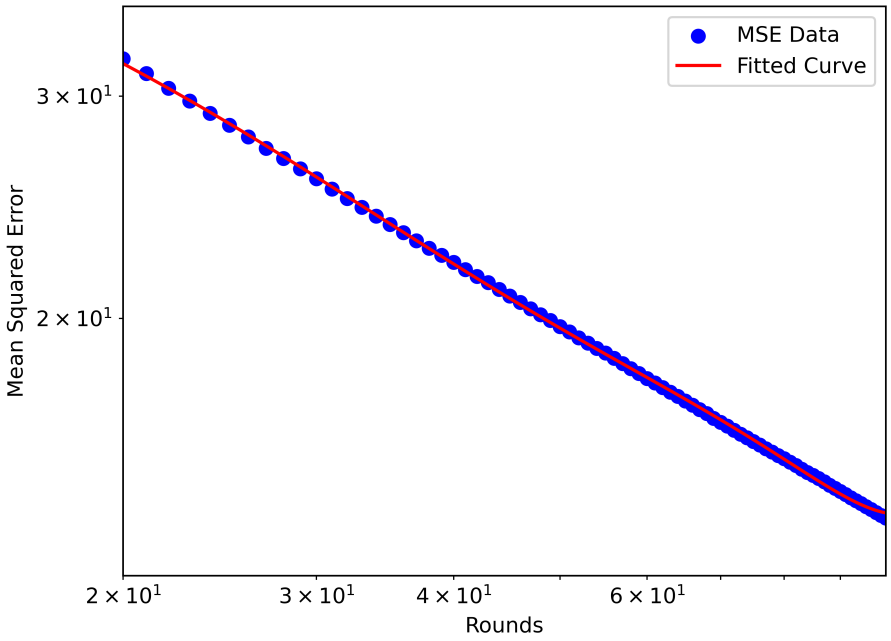


Figure 38: (512, 512)-Lollipop graph - polynomial regression fit - ATPPS

in subsection 6.5.3, the simulations are performed with a reduced path size, assigning 896 nodes to the clique and only 128 nodes to the path. This configuration provides insight into how a more densely connected clique affects the overall performance of the algorithms.

6.5.2 (128, 896)-Lollipop Graph

In the initial phase (the first 7 rounds) for the (128, 896)-Lollipop Graph $L_{128,896}$, the PPS and ATPPS start with a steep decline in MSE, both with a slope of -1.2, as shown in figure 44. DAB demonstrates a slightly slower initial convergence, with a slope of -0.86. Compared to the previous experiment, where the path size and clique sizes were equal, DAB's performance in the initial rounds has improved. The slope in this region in the previous experiment was -0.34 and improved to -0.86, which indicates a faster initial error reduction. In the middle region, rounds 8 to 50, the ATPPS shows a performance close to that of the PPS, where both curves maintain

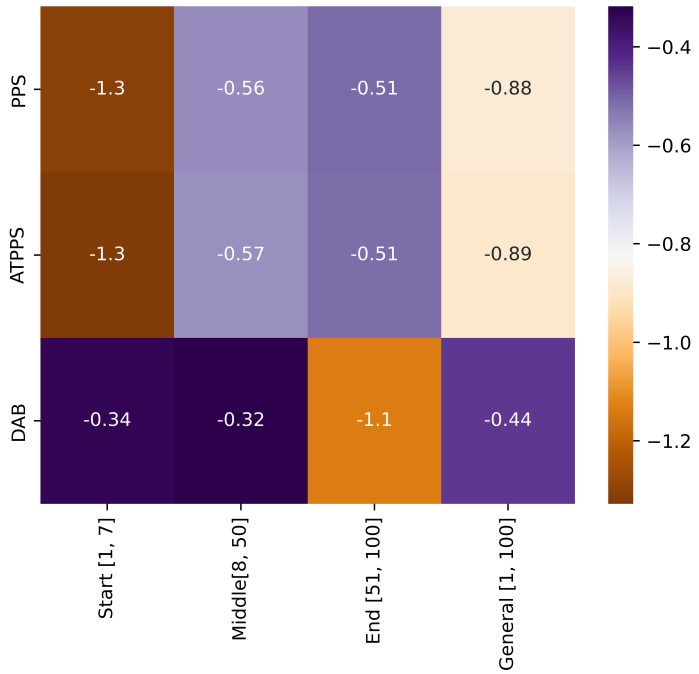


Figure 39: (512, 512)-Lollipop graph - heat map of slopes per region - log-log

a consistently steep slope. In the middle region, the PPS manages to increase its performance again slightly, as the slope has fallen by 0.1. The ATPPS manages to close the MSE discrepancy to the PPS in the end region. In this region, the ATPPS emits the largest drop in MSE. In the end phase, the DAB shows a slightly shallower decline (-0.78) in MSE compared to earlier rounds. This can be attributed to the fact that once most of the load has propagated through the path, the balancing process along the clique slows. Both Push-Pull Sum variants outperform DAB, as they achieve lower MSE values in fewer rounds due to their efficient clique-based load redistribution. Interestingly, PPS and ATPPS diverge between rounds 10 and 90, before finally intersecting (or nearly so) around round 100. The PPS and ATPPS MSE data are very close to each other in round 100, suggesting that the ATPPS does not lead to an improved load balancing behavior.

The MSE data of all three load balancing algorithms were fitted to polynomial regression models. The MSE data for DAB and ATPPS were best approximated by

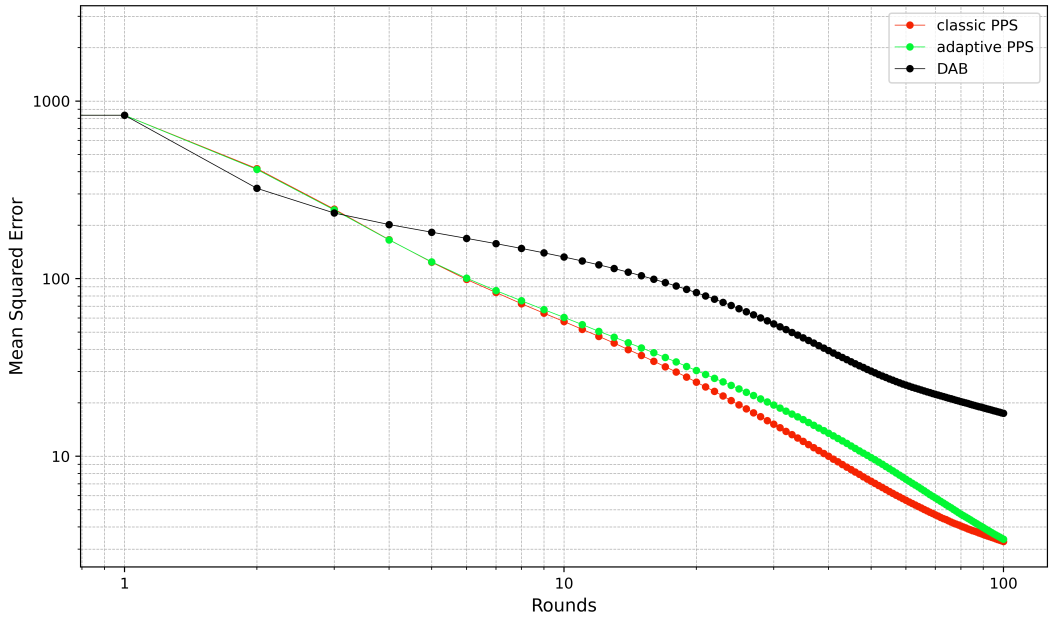


Figure 40: (128, 896)-Lollipop graph - mean squared error per rounds - log-log

polynomials of degree 4, while the PPS MSE data exhibited a slightly more complex behavior, following a polynomial of degree 5. The fitted model for the DAB MSE data follows the equation:

$$MSE_r = 3.46 \times 10^{-6} r^4 - 1.12 \times 10^{-3} r^3 + 0.14 r^2 - 7.69 r + 190.78 \tag{33}$$

(figure 41). For ATPPS, the polynomial fit is given by:

$$MSE_r = 1.59 \times 10^{-6} r^4 - 4.74 \times 10^{-4} r^3 + 0.05 r^2 - 2.94 r + 70.59 \tag{34}$$

(figure 42). The PPS MSE data, which required a higher-degree polynomial (degree 5), follows the equation:

$$MSE_r = -3.72 \times 10^{-8} r^5 + 1.31 \times 10^{-5} r^4 - 1.85 \times 10^{-3} r^3 + 0.13 r^2 - 4.96 r + 84.91 \tag{35}$$

(figure 43).

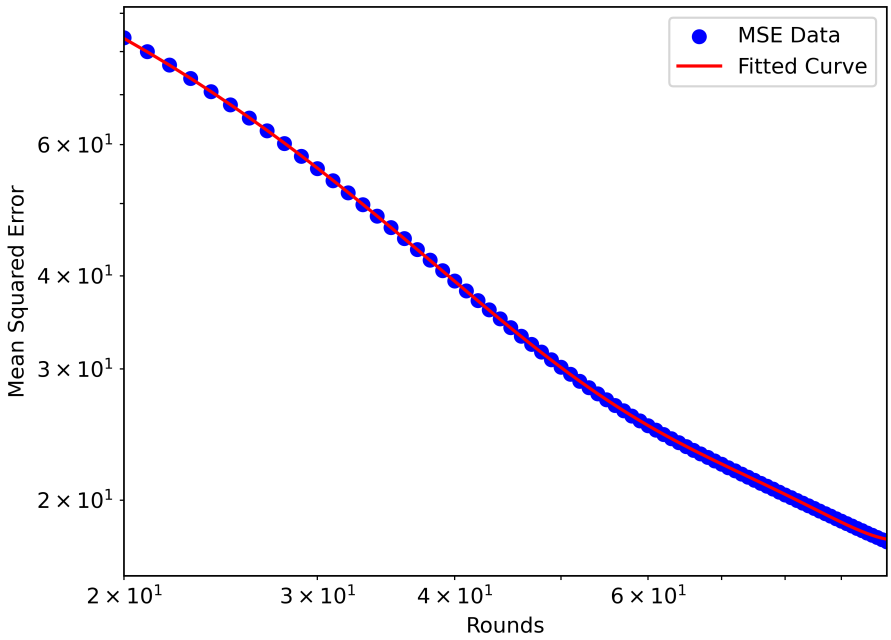


Figure 41: (128, 896)-Lollipop graph - polynomial regression fit - DAB

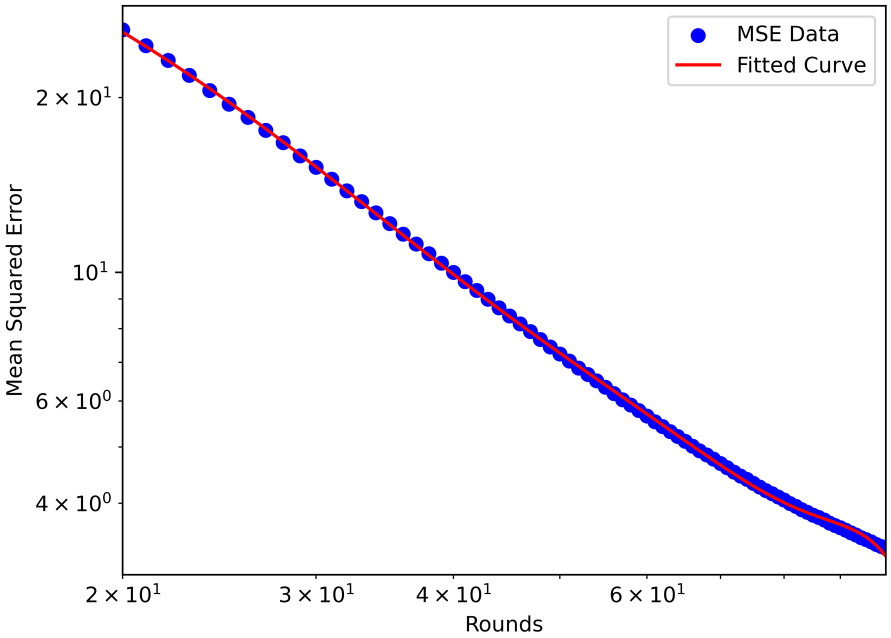


Figure 42: (128, 896)-Lollipop graph - polynomial regression fit - PPS

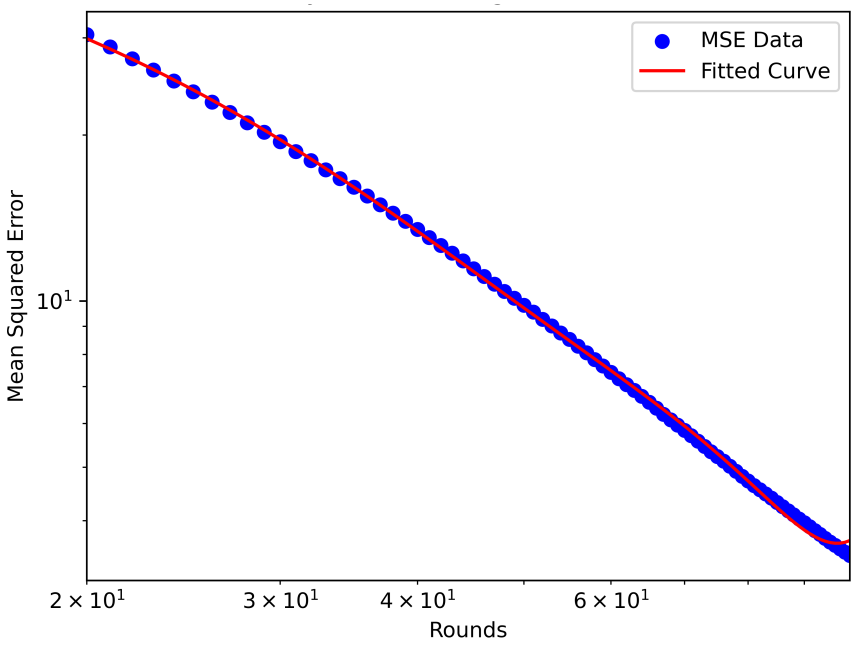


Figure 43: (128, 896)-Lollipop graph - polynomial regression fit - ATPPS

6.5.3 (896, 128)-Lollipop Graph

As more nodes are assigned to the clique and fewer to the Path graph in $L_{896,128}$ compared to $L_{512,512}$, the performance gap between the Push-Pull Sum-based algorithms and the DAB becomes more pronounced, as shown in figure 45. This is also reflected in the log-log slopes. In the start region, the Push-Pull Sum-based algorithms balance the network more quickly in $L_{896,128}$, achieving slopes of -2, compared to -1.3 in $L_{512,512}$ (figure 49). In the middle and end regions, the slopes are almost quartered compared to the start region. The DAB achieves consistently low slopes: -0.08 in the start region, -0.1 in the middle region, and -0.11 in the end region. The overall MSE is also lower for $L_{896,128}$, with the ATPPS reducing the error to 3.27 and the PPS to 3.59. In contrast, the MSE values for the $L_{512,512}$ graph are higher after 100 rounds, where ATPPS achieved a value of 13.83 and the PPS achieved a value of 14.71. Despite this, no significant advantage of the ATPPS over the PPS is observed. The DAB, however, struggles to reduce the error, with a substantial difference in MSE values after 100 rounds. In $L_{512,512}$, the DAB reaches an MSE of 108.90, while

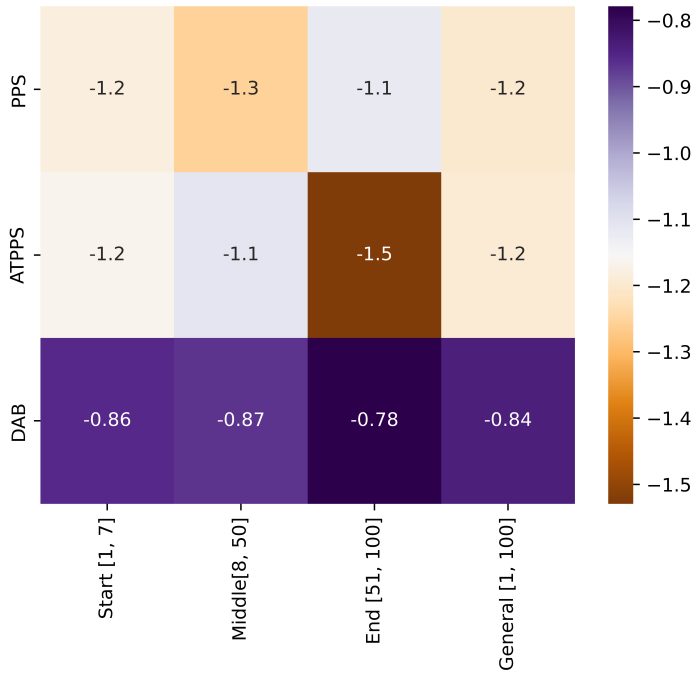


Figure 44: (128, 896)-Lollipop graph - heat map of slopes per region - log-log

in $L_{896,128}$, the MSE increases to 542.09 - almost five times higher.

The best-fit polynomials for the MSE data of the load balancing algorithms are of degree 3 for the DAB and degree 4 for the Push-Pull Sum-based algorithms. The polynomial for the DAB MSE data is expressed as:

$$MSE_r = -1.93 \times 10^{-4} r^3 + 0.05 r^2 - 5.33 r + 745.95 \tag{36}$$

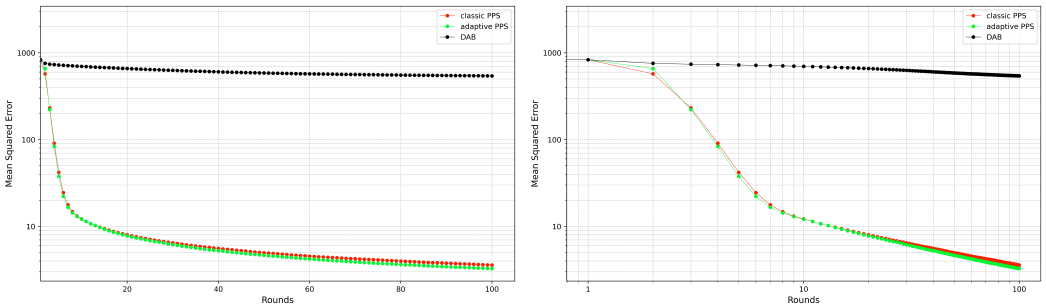
(figure 46). For the PPS, the polynomial is:

$$MSE_r = 2.00 \times 10^{-7} r^4 - 6.01 \times 10^{-5} r^3 + 6.95 \times 10^{-3} r^2 - 0.39 r + 13.44 \tag{37}$$

(figure 47) and for the ATPPS, is:

$$MSE_r = 2.28 \times 10^{-7} r^4 - 6.77 \times 10^{-5} r^3 + 7.68 \times 10^{-3} r^2 - 0.42 r + 13.62 \tag{38}$$

(figure 48).



(a) **(b)**

Figure 45: (896, 128)-Lollipop graph - mean squared error per rounds - log-linear and log-log

6.6 Ring of Cliques

6.6.1 (32 x 32)-Ring of Cliques

In the early rounds, PPS and ATPPS exhibit the fastest MSE reduction, as shown by a very steep downward trend between rounds 1 and 5 in figure 50 b). Both Push-Pull Sum-based approaches demonstrate faster convergence rates compared to DAB in the (32×32)-Ring of Cliques $ROC_{32,32}$. The initial steep downward slope of around -1.8 for each Push-Pull Sum-based algorithm, as depicted in figure 54, within the first 5 rounds is due to the fact that each clique is balanced efficiently by these algorithms. The Push-Pull Sum-based algorithms perform better than DAB in cliques with high degrees. Once the cliques start to balance, round by round, DAB catches up, especially between rounds 6 and 40, where the slopes are -2.1 for the DAB compared to approximately -0.5 for the Push-Pull Sum-based algorithms. Nodes within cliques tend to converge quickly internally due to their dense interconnections for the Push-Pull Sum-based algorithms. However, load balancing between cliques, especially when the nodes select random neighbors, is slower (this is especially the case for the PPS, as seen by the curve starting to stagnate between round 10 and

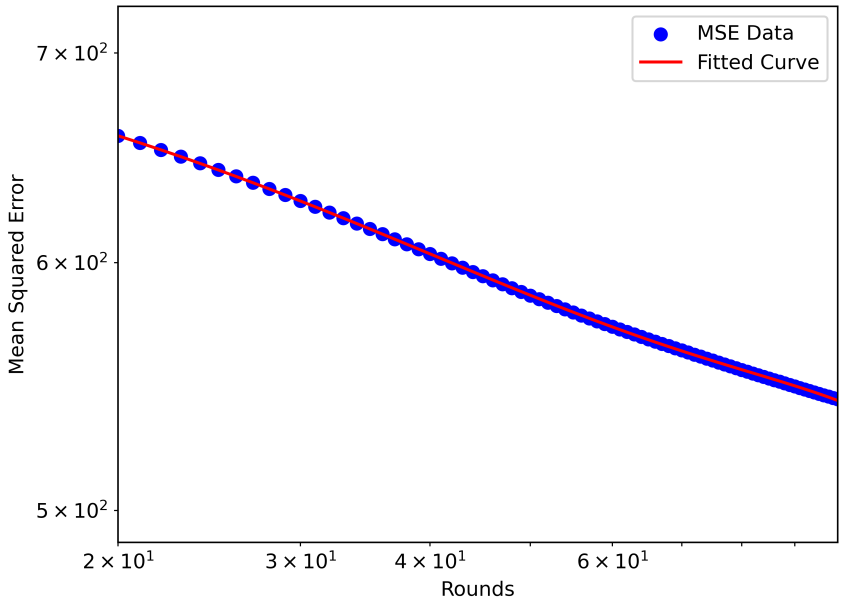


Figure 46: (896, 128)-Lollipop graph - polynomial regression fit - DAB

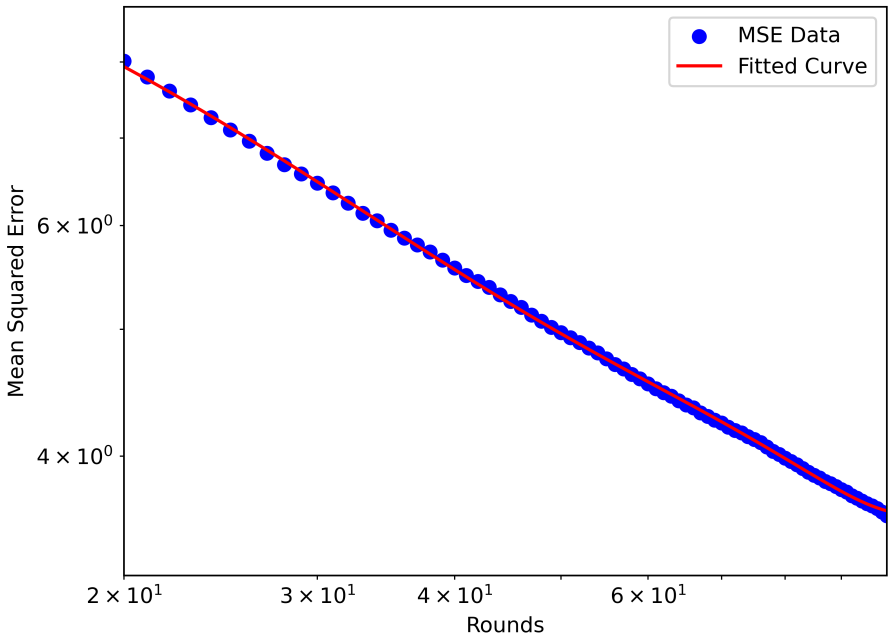


Figure 47: (896, 128)-Lollipop graph - polynomial regression fit - PPS

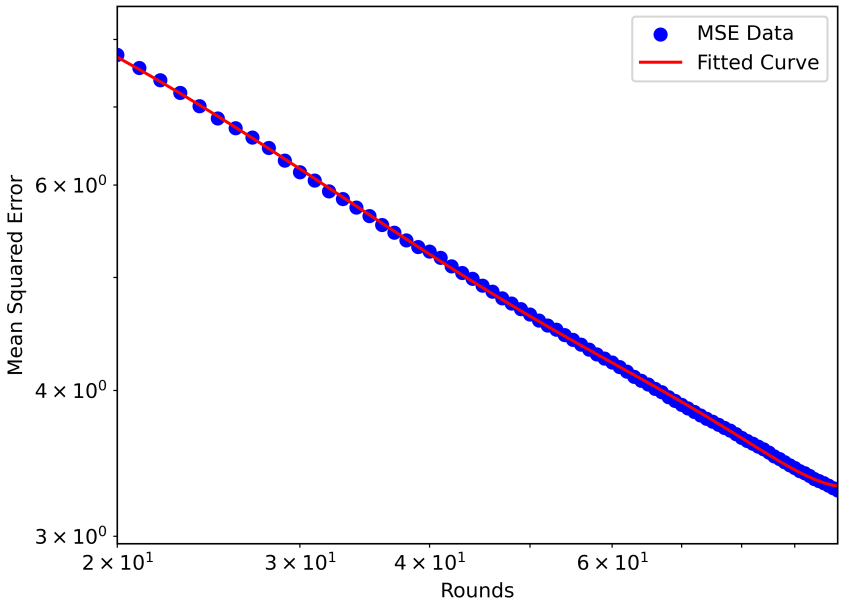


Figure 48: (896, 128)-Lollipop graph - polynomial regression fit - ATPPS

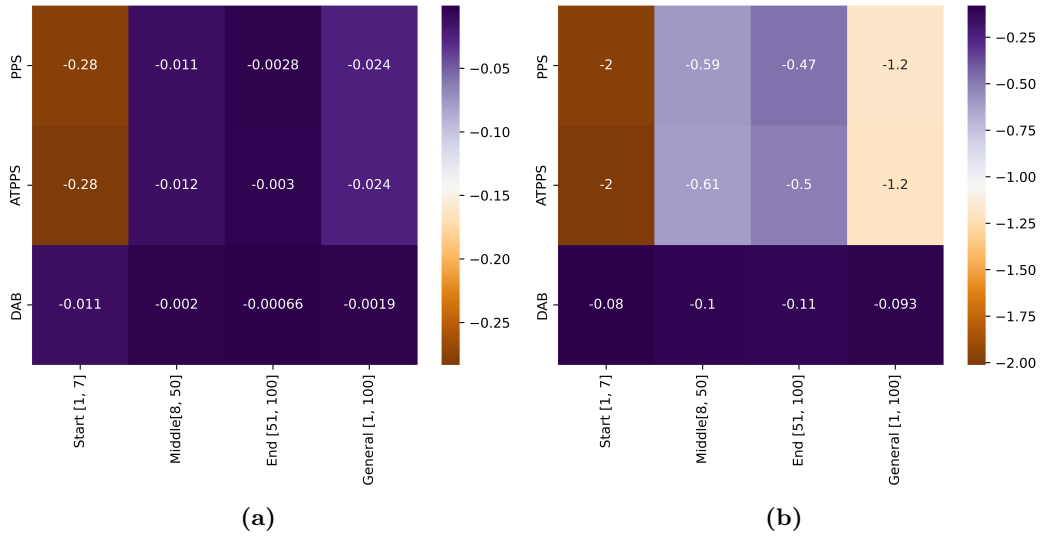


Figure 49: (896, 128)-Lollipop graph - heat map of slopes per region - log-linear and log-log

100). Thanks to the deterministic load balancing strategies of the DAB, the bridging nodes start to spread the load to other cliques. The ATPPS achieves better results compared to the PPS since it has a similar mechanism in this scenario like the DAB,

prioritizing communication between cliques (where differences in loads are more significant) over redundant communication within cliques (where loads are already close to balanced). This is reflected in the behavior of the curve after round 10. Again, the ATPPS acts as a compromise solution between the PPS and the DAB, achieving results close to those of the DAB. Overall, at round 100, the DAB achieved an MSE of approximately 6.55, the PPS an MSE of 19.80, and the ATPPS an MSE of 8.42.

The polynomial fit for the DAB is expressed as:

$$MSE_r = 1.04 \times 10^{-6} r^4 - 2.94 \times 10^{-4} r^3 + 0.03 r^2 - 1.66 r + 45.30 \tag{39}$$

(figure 51), which models the MSE data as a function of rounds with a fourth-degree polynomial. For the PPS, the MSE data from rounds 20 to 100 is fitted to a linear regression model:

$$MSE_r = -0.05 r + 24.70 \tag{40}$$

(figure 52). The negative slope indicates a consistent reduction in MSE with each round in this region, though the value -0.05 is relatively small. The linear fit indicates that PPS achieves only gradual improvement in balancing the load in $ROC_{32,32}$. The reason for that is that PPS uses the push and pull mechanisms, which rely on random neighbor selection. PPS lacks a mechanism to prioritize balancing in the inter-clique scenario once intra-clique balancing is achieved. As a result, its performance is bottlenecked by the topology. The logarithmic model for the ATPPS is given by:

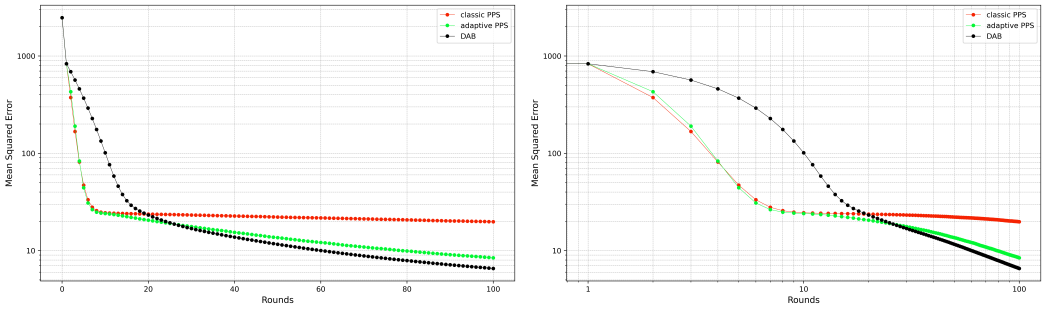
$$\log(MSE_r) = -7.59 \log(r) + 43.26 \tag{41}$$

(figure 53). By exponentiating this equation, the relationship between MSE and the

number of rounds can be written as:

$$MSE_r = r^{-7.59} * 10^{43.26}. \tag{42}$$

The steep negative slope of -7.59 in the log-log fit indicates a rapid decrease in MSE as the number of rounds increases. The fitted model suggests that ATPPS achieves exponentially faster convergence compared to PPS, particularly in the early rounds of load balancing.



(a) **(b)**

Figure 50: (32×32)-Ring of Cliques - mean squared error per rounds - log-linear and log-log

In the following experiments, the structure of the Ring of Cliques is reorganized. The number of cliques is increased from 2^5 to 2^7 in subsection 6.6.2, which results in a decrease in the size of each clique to 2^3. Subsection 6.6.3 covers the experiment where the clique size is increased, while the number of cliques is decreased by the same magnitude.

6.6.2 (128 x 8)-Ring of Cliques

The clique size plays an important role for the efficiency of the DAB. The decreased clique size favors the DAB, which reduces the error in the network faster than the Push-Pull Sum-based algorithms in the (128 x 8)-Ring of Cliques $ROC_{128,8}$, as seen in figure 55. All the load balancing algorithms show a fast decrease of error in

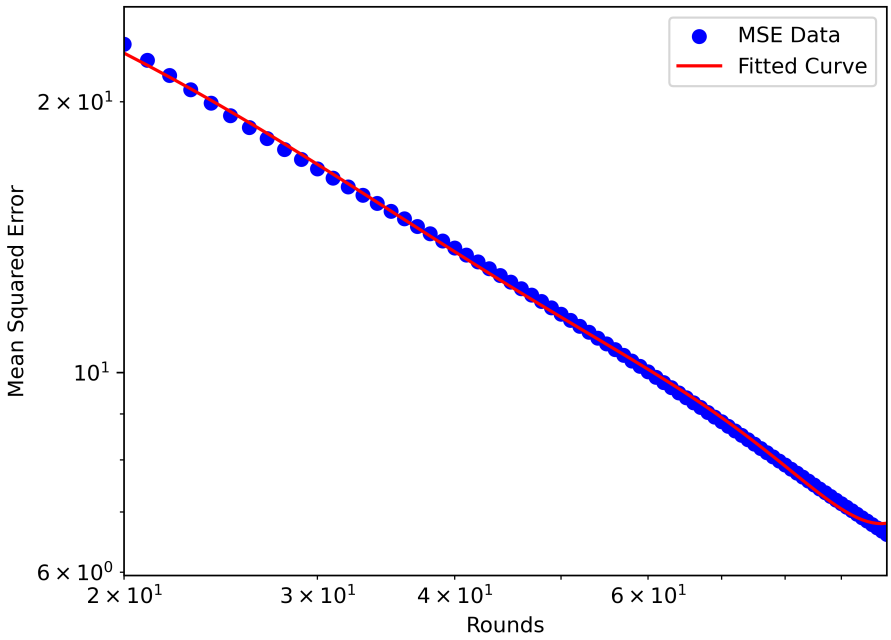


Figure 51: (32×32)-Ring of Cliques - polynomial regression fit - DAB

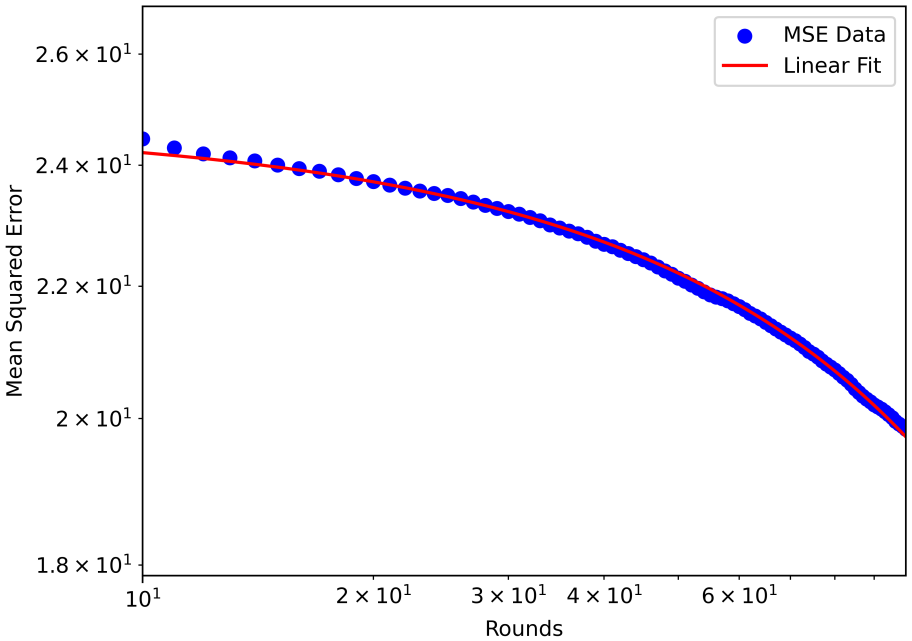


Figure 52: (32×32)-Ring of Cliques - linear regression fit - PPS

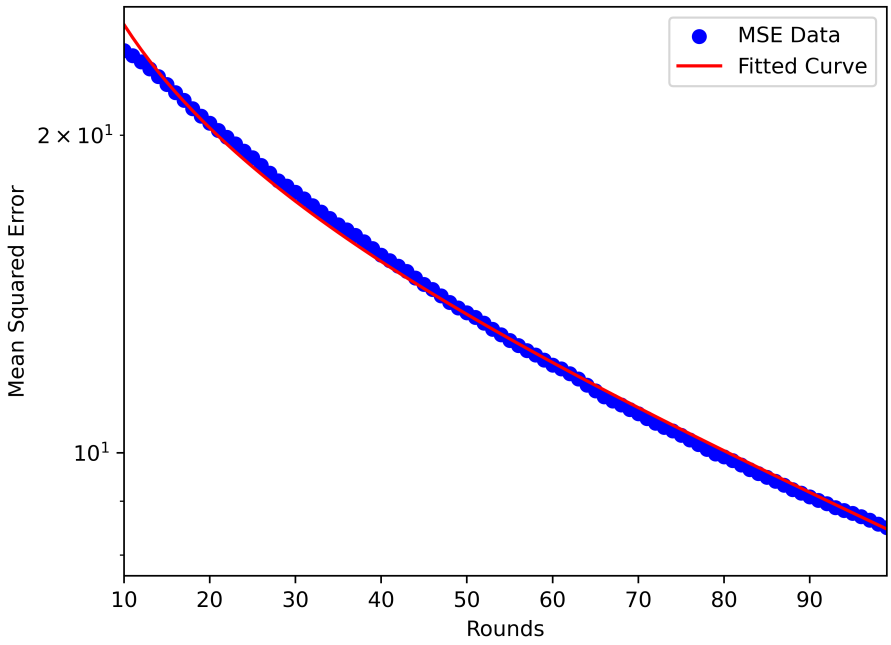


Figure 53: (32×32)-Ring of Cliques - logarithmic regression fit - ATPPS

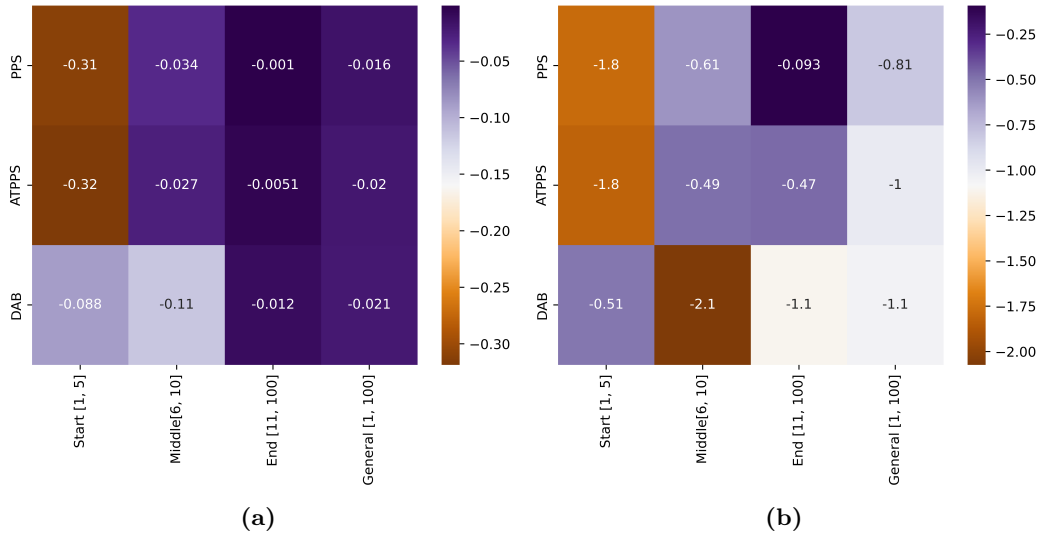


Figure 54: (32×32)-Ring of Cliques - heat map of slopes per region - log-linear and log-log

the network, shown by the steep negative slopes as visualized in figure 59 b). The Push-Pull Sum-based algorithms show an initial slope of -1.3, whereas the DAB

achieves an even steeper slope of -1.4 in the start region (rounds 1 to 5). After this initial sharp decline, the error reduction slows down. However, DAB still maintains the steepest slope, with an average reduction of -0.95 per round, followed by ATPPS (-0.89) and PPS (-0.79). The curves in this region are relatively flat, as observed in the log-log representation in figure 55 a). The decrease effect diminishes as the error of the network is already relatively low in the end region. At round 100, the final MSE values highlight the effectiveness of each algorithm. DAB achieves the lowest MSE of 10.64, ATPPS follows closely with 13.68. PPS results in the highest MSE of 22.33. Compared to $ROC_{32,32}$, the load balancing algorithms further reduced the error by 2 to 4 units. The ATPPS showed the most improvement, lowering the MSE by 5 units, followed by DAB (4 units) and PPS (2 units).

All three load balancing algorithms were fitted to fourth-degree polynomials. The best-fit equations are as follows: DAB's MSE data is fitted to a fourth-degree polynomial:

$$MSE_r = 5.45 \times 10^{-7} r^4 - 1.7 \times 10^{-4} r^3 + 0.02 r^2 - 1.33 r + 46.71 \tag{43}$$

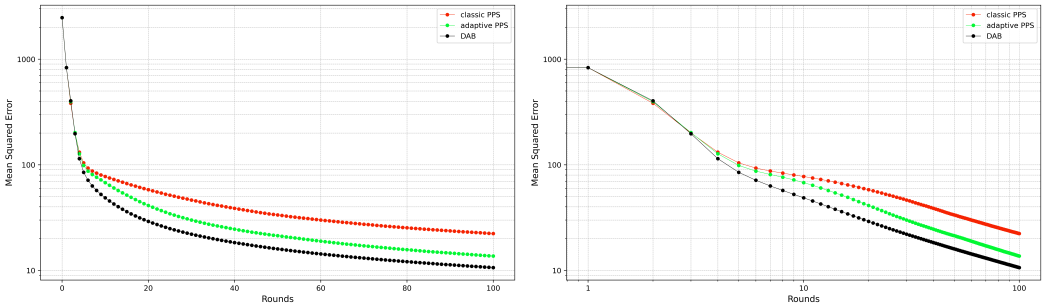
(figure 56), as are the PPS:

$$MSE_r = 1.04 \times 10^{-6} r^4 - 3.41 \times 10^{-4} r^3 + 0.04 r^2 - 2.73 r + 97.58 \tag{44}$$

(figure 57) and the ATPPS:

$$MSE_r = 9.54 \times 10^{-7} r^4 - 2.93 \times 10^{-4} r^3 + 0.03 r^2 - 2.05 r + 67.02 \tag{45}$$

(figure 58).



(a) **(b)**

Figure 55: (128 × 8)-Ring of Cliques - mean squared error per rounds - log-linear and log-log

6.6.3 (8 x 128)-Ring of Cliques

Increasing the clique size and reducing the number of cliques favors the Push-Pull Sum-based algorithms, which exhibit a steep decrease in error during the first 5 rounds of the simulation in the (8 x 128)-Ring of Cliques $ROC_{8,128}$ (figure 60). PPS and ATPPS achieve an error reduction of approximately -2 on average for the start region. DAB shows a more moderate decrease of -0.12 in the same period. The error reduction slows, since the error of the network is already very low, showing an MSE in this region of around -30 for the Push-Pull Sum-based algorithms. In the middle region, the state of the network managed by the DAB is still very unbalanced. Thus, the error reduction potential is higher compared to the Push-Pull Sum-based algorithms. The Push-Pull Sum-based algorithms reduce the error by around -0.13 for the ATPPS and -0.01 for the PPS in the final region, while the DAB shows a still very high value of approximately -2.1. The steep decrease between rounds 5 and 10 stems from the fast error reduction in the cliques. Compared to the experiment in the $ROC_{8,128}$ in 6.6.2, the error after 100 rounds drops to lower values. The final MSE values for the DAB are 5.18 compared to 5.95 for the PPS and 4.56 for the ATPPS. In the final phase (rounds 12 to 100), the DAB slowly catches up to the Push-Pull Sum-based algorithms. However, both PPS and ATPPS achieve a well-balanced network state by round 10 by reducing the error to an MSE of 7. In

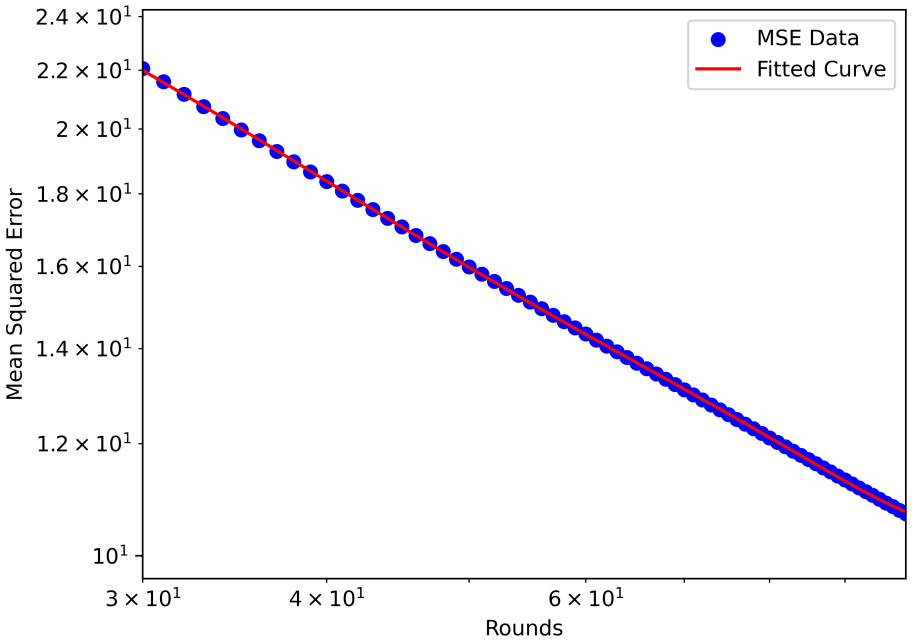


Figure 56: (128×8)-Ring of Cliques - polynomial regression fit - DAB

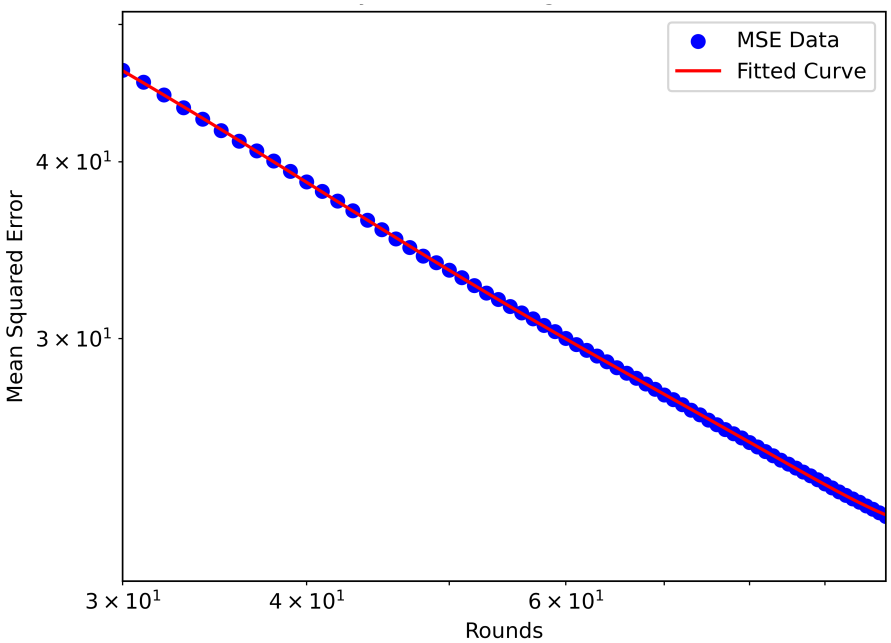


Figure 57: (128×8)-Ring of Cliques - polynomial regression fit - PPS

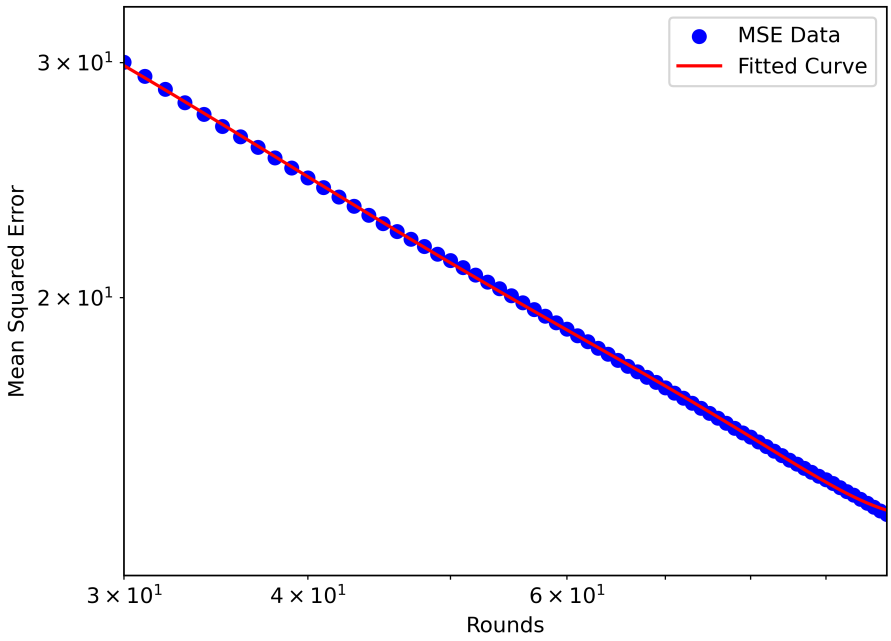


Figure 58: (128 × 8)-Ring of Cliques - polynomial regression fit - ATPPS

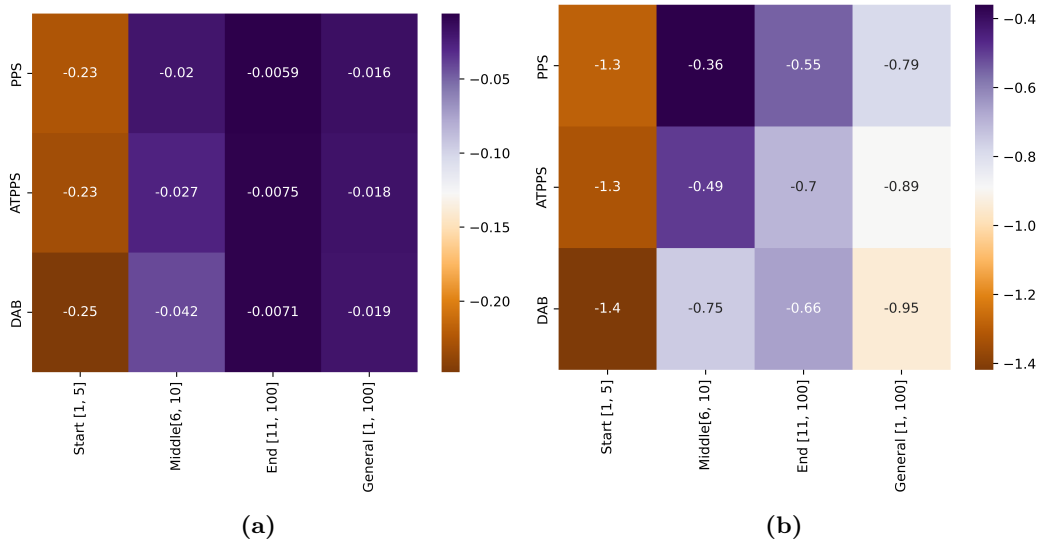


Figure 59: (128 × 8)-Ring of Cliques - heat map of slopes per region - log-linear and log-log

the following rounds the inter-clique error reduction follows. The PPS algorithm struggles in this phase due to its randomized selection of transfer partners, which

slows down further convergence. The ATPPS has an advantage compared to the PPS here since it prioritizes the bridging nodes once its neighbors within the clique are already balanced. This elaborates why the PPS curve is mostly stagnating, while the ATPPS curve shows a downward trend. The stagnating trend between rounds 20 and 100 is expressed by the linear model fit in figure 62. The best-fit model follows the equation:

$$MSE_r = -1.5 \times 10^{-3} r + 6.05. \tag{46}$$

The slope of -1.5×10^{-3} is very small. The error in the network is barely reduced in this area. The behavior of the PPS is captured by this simple model. It does not involve any accelerations like in the exponential model. The ATPPS curve shows a more complex relation between the MSE reduction over the rounds, namely a polynomial of degree 2 following the equation:

$$MSE_r = 6.07 \times 10^{-5} r^2 - 0.02r + 6.39 \tag{47}$$

(figure 62). The DAB does not follow a single power relationship in this region. Between rounds 15 and 50, the error reduction can be expressed by a third-degree polynomial:

$$MSE_r = -3.33 \times 10^{-3} r^3 + 0.63r^2 - 40.23r + 872.75 \tag{48}$$

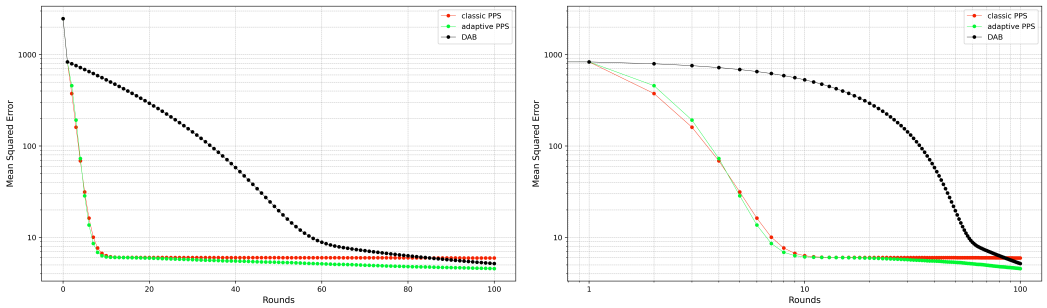
(figure 63 a)), while in later rounds the power relation is a bit more complex, expressed by a fourth-degree polynomial following the equation:

$$MSE_r = 2.96 \times 10^{-6} r^4 - 9.97 \times 10^{-3} r^3 + 0.13r^2 - 7.16r + 161.73 \tag{49}$$

(figure 63 b)).

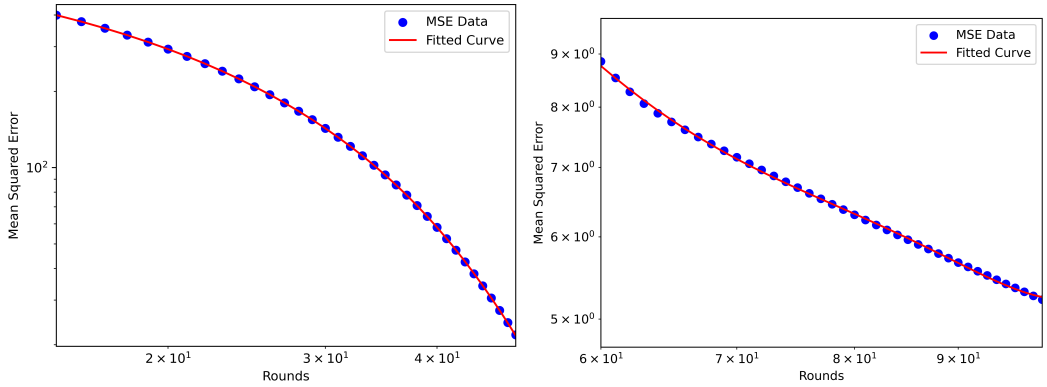
Snippets of the simulation results confirm that within each clique, loads are balanced.

However, inter-clique load balancing remains pending, which indicates that global equilibrium is not fully achieved. Listing 6.1 shows the simulation outcomes of two connected cliques balanced by the ATPPS in round 100. While the first clique converged to a value of around 46.35, the second clique averaged to approximately 47.90. The ground truth of the first clique is 45.89, and the ground truth of the second clique is 48.13. So the simulation outcomes and the ground truths align. However, the averages do not align with the ground truth of the Ring of Cliques (which is 49.09) itself. The PPS simulation outcomes show a similar behavior.



(a) **(b)**

Figure 60: (8 × 128)-Ring of Cliques - mean squared error per rounds - log-linear and log-log



(a) **(b)**

Figure 61: (8 × 128)-Ring of Cliques - polynomial regression fit - DAB; rounds 20-50 and 55-100

```
# start: first clique
```

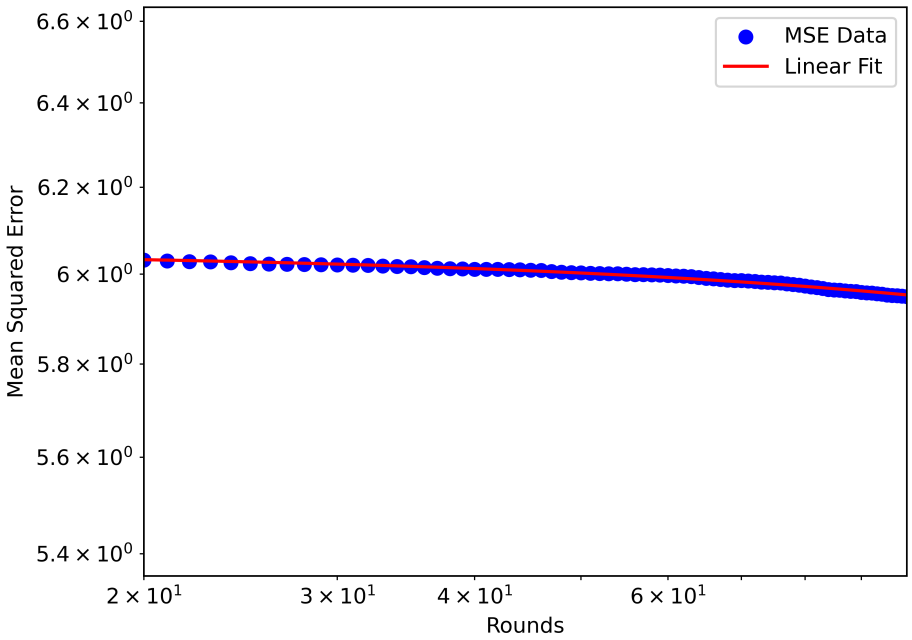


Figure 62: (8×128)-Ring of Cliques - linear regression fit - PPS

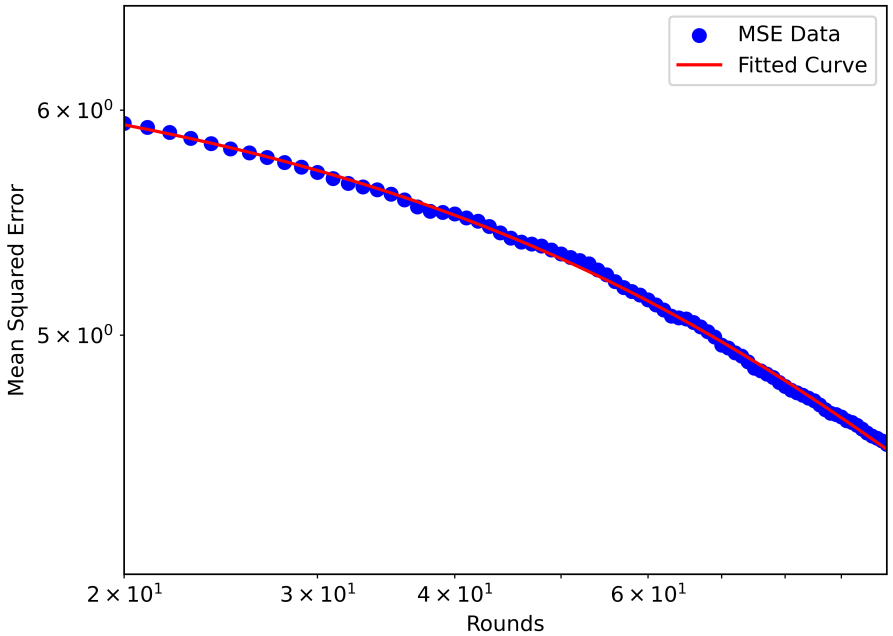
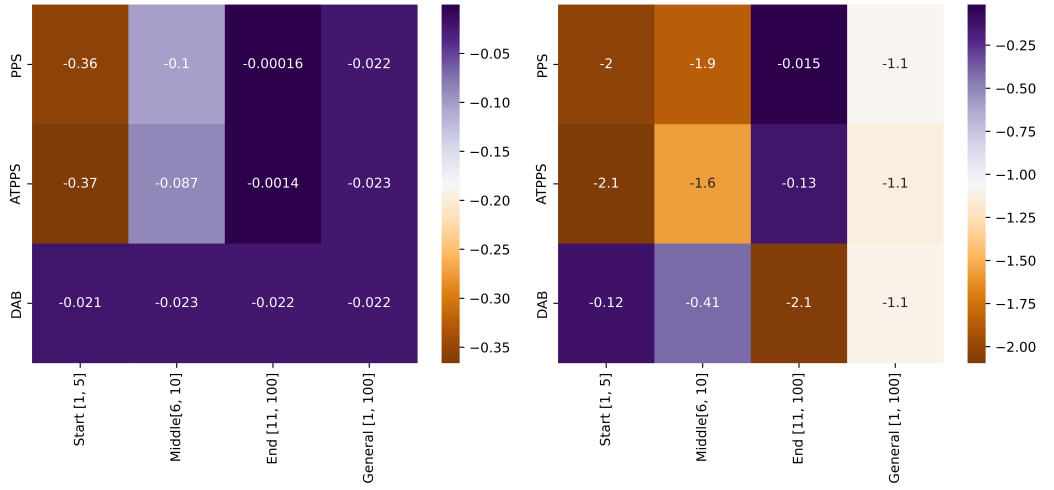


Figure 63: (8×128)-Ring of Cliques - polynomial regression fit - ATPPS



(a) **(b)**

Figure 64: (8×128)-Ring of Cliques - heat map of slopes per region - log-linear and log-log

```
ID 0      sum 23.703116631927735   weight 0.5112902995975737
Average 46.359410007551446
ID 1      sum 39.52476483621201    weight 0.852801600538612
Average 46.34696371494727
ID 2      sum 8.793798989811481    weight 0.18958634077240344
Average 46.38413798158778
...
ID 125    sum 94.3005473042508     weight 2.0354736681273184
Average 46.32855181615266
ID 126    sum 145.10578299145917   weight 3.1321941681390815
Average 46.32719914604474
ID 127    sum 149.43516229432458   weight 3.222997634840688
Average 46.365272092950555
# end: first clique


# start: second clique
```

```
ID 128    sum 198.97437468622982   weight 4.149869252421544
Average 47.947143050380134
ID 129    sum 168.24642313171216   weight 3.5107838866374186
Average 47.92275131832632
ID 130    sum 35.6266112221637     weight 0.7449330697151406
Average 47.82525124812511
...
ID 253    sum 63.87360335784297    weight 1.3317373880292274
Average 47.962611797185005
ID 254    sum 15.173810743452982   weight 0.31637895519911763
Average 47.96087253623784
ID 255    sum 19.982619800226413   weight 0.4175626540105257
Average 47.85538076334456
# end: second clique
```

Listing 6.1: Snippet of simulation outcomes - ATPPS: Experiment 2

7 Conclusion

The Adaptive Threshold Push-Pull Sum algorithm shows to be a compromise solution between the Push-Pull Sum and Single-Proposal Deal-Agreement-Based algorithms. The simulation results show that the Adaptive Threshold Push-Pull Sum algorithm performs well in dense graphs as well as in regular low-degree graphs. Especially for the Complete Graph K_{1024}, Ring of Cliques $ROC_{32,32}$ and $ROC_{8,128}$, as well as for Lollipop $L_{512,512}$ and $L_{896,128}$ topologies, the ATPPS algorithm proved to be the best solution, as the algorithm achieved the lowest MSE values after 100 rounds of simulation. The Adaptive Threshold Push-Pull Sum achieved good overall results. In cases where the PPS failed to show good balancing abilities, like in the $ROC_{32,32}$, the Adaptive Threshold Push-Pull Sum proved to be very efficient, achieving results close to the Deal-Agreement-Based algorithm. In cases where the Push-Pull Sum algorithm achieved lower MSE within 100 rounds, the difference to the Adaptive Threshold Push-Pull Sum algorithm was not significant as seen for the Star graph S_{1024}, the Lollipop graph $L_{128,896}$, as well as the Torus Grid $T_{32,32}$. However, when the Adaptive Threshold Push-Pull Sum outperforms the traditional Push-Pull Sum algorithm, the MSE discrepancies are more pronounced, like in the case of Ring of Cliques $ROC_{32,32}$ and $ROC_{128,8}$. Here, the Adaptive Threshold Push-Pull Sum algorithm manages to distribute loads efficiently by adapting to the network state. When the load differences within the cliques were no longer significant enough, clique-to-clique communication via the bridging nodes was favored. The Adaptive Threshold Push-Pull Sum algorithm also achieves the sharpest downward trend for K_{1024} until the curve finally stagnates (due to the precision of doubles in Java). No improvement of

the load balancing behavior was observed for the Ring structure R_{1024}. The reason for this is that each node i has exactly 2 neighbors, and thus only one neighbor is selected as $\lceil \log_2 (|neighborhood_i|) \rceil$ evaluates to 1. The behavior of this algorithm is therefore very similar to the traditional Push-Pull Sum algorithm, without prioritizing any nodes. Also for the Star topology, the Adaptive Threshold Push-Pull Sum algorithm does not achieve any advantage over the traditional Push-Pull Sum algorithm, as the leaf nodes all communicate with the central node. The condition with significant load transfers limits the number of requests to the central node compared to the traditional Push-Pull Sum algorithm. The result is that the Push-Pull Sum algorithm achieves a balanced state of the network earlier. Compared to the Push-Pull Sum algorithms, the Single-Proposal Deal-Agreement-Based algorithm has problems with dense graphs, such as the Complete graph, Lollipop graph with a large clique size, and Ring of Cliques with a large clique size. The MSE differences for dense graphs after 100 rounds of computation are extremely large between the Single-Proposal Deal-Agreement-Based algorithm and the Push-Pull Sum-based algorithms. For low-degree topologies such as the Ring graph and the Torus Grid graph the MSE differences are not as significant, especially between the best-performing Single-Proposal Deal-Agreement-Based algorithm and the Adaptive Threshold Push-Pull Sum algorithm. The simulation results depend mainly on the sensitivity factor k. A large k ensures a more sensitive selection of neighbors, while a smaller k allows a coarser selection of neighbors.

8 Outlook

The proposed load balancing algorithm can be used in many different scenarios, like in cloud environments where many different servers are connected to a network, many with different hardware specifications. The method of randomized neighbor selection and the push and pull mechanisms allow an even distribution of loads throughout the network. The adaptive threshold condition only allows load transfers with a high impact on the balance of the network. The weight value can, for instance, represent the server capacities, i.e., given the hardware of a server, how capable it is of balancing the loads relative to the other partners in the network.

It is important to note that the performance of the algorithm was only tested in specific static topologies. Future research can also inspect dynamic networks. Furthermore, the performance was only tested with the metric of mean squared error, and the trend of its decay was analyzed with model fitting. However, further research is planned to compare the number of messages sent to achieve low imbalance in the network after r rounds. This would provide an insight into how efficient the load balancing mechanisms of the individual algorithms are.

9 Acknowledgments

First and foremost, I would like to thank...

- Saptadi Nugroho for his guidance and support throughout this thesis.
- Prof. Dr. Christian Schindelhauer for giving me the opportunity to conduct this research at the chair of Computer Networks and Telematics. His critical questions and advice challenged me to refine my ideas and improved my work further.
- my family for their support, patience, and encouragement throughout this journey. Their belief in me has been a constant source of motivation.

10 Appendix

10.1 Model Fitting

The model fitting was performed using the *SciPy* and *NumPy* libraries in Python. For fitting data to an exponential model, *SciPy* provides the **curve_fit** method. This method uses the Levenberg-Marquardt method to solve non-linear least squares problems [20]. The linear regression was performed using the **linregress** method from *scipy.stats*, which calculates a linear least squares regression for two sets of data points for x and y [21]. For fitting the MSE data to a polynomial model of equation 12, *NumPy*'s **polyfit** method was used.

10.1.1 Levenberg-Marquardt Method

The Levenberg-Marquardt method is a hybrid of the Gauss-Newton method and the Levenberg method (Trust-Region approach). Starting from an initial guess for the parameters, the Levenberg-Marquardt method uses the Jacobian matrix to estimate how changes in parameters affect the fitted function. Then a damped least squares step is applied, where if the parameters are far from optimal, the Levenberg-Marquardt method behaves like the gradient descent, and if the parameters are near optimal, the Levenberg-Marquardt method acts like the Gauss-Newton method. It

minimizes the sum of the squared residuals:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x) = \frac{1}{2}||r||^2, \tag{50}$$

where $f(x)$ is the objective function that we are trying to minimize, $r_j(x)$ represents the residuals (which are the differences between the observed data and the model predictions), and $r(x)$ is the vector of residuals, where $r(x) = [r_1(x), r_2(x), ..., r_n(x)]^T$. The Gauss-Newton method is modified by a damping coefficient λ and is written as:

$$(J^T J + \lambda I)p = J^T r(x), \tag{51}$$

where J is the Jacobian matrix of $r(x)$ (the derivatives of the residuals), I is the identity matrix, and λ controls the step size. $J^T J$ is the approximation of the Hessian matrix. p is the update step. This equation is solved by iteratively reducing $f(x)$ until the parameters converge.

If λ is large, the equation behaves like the gradient descent:

$$\lambda I p = J^T r(x), \tag{52}$$

while a small λ causes the equation to behave like the Gauss-Newton method:

$$J^T J p = J^T r(x). \tag{53}$$

[22] [23] [24]

10.1.2 SciPy's linregress

SciPy's **linregress** method fits data to a simple linear model as depicted in equation 11. It achieves this by solving the *Ordinary Least Squares* regression, which minimizes

the sum of squared residuals. It returns the slope and the intercept of the fitted line, as well as the *R-value* (*Pearson's R*), *p-value*, standard error of the estimated slope, and the standard error of the estimated intercept. [21] [25]

10.1.3 NumPy's polyfit and polyval

The *numpy.polyfit* method fits a polynomial of a given degree to a set of data points using the least squares minimization method [26]. First, the Vandermonde Matrix is constructed, if given a degree d and a set of data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ of the form:

$$V = \begin{bmatrix} x_1^d & x_1^{d-1} & \ldots & x_1^1 & 1 \\ x_2^d & x_2^{d-1} & \ldots & x_2^1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^d & x_n^{d-1} & \ldots & x_n^1 & 1 \end{bmatrix}. \tag{54}$$

Each row represents one data point, and each column corresponds to a power of x. Following the construction of the Vandermonde matrix, the polynomial coefficients $c = [c_d, c_{d-1}, ..., c_0]$ are computed by solving:

$$c = (V^T V)^{-1} V^T y. \tag{55}$$

This method solves the linear least squares problem and ensures that the sum of squared residuals is minimized:

$$\sum_{i=1} (y_i - P(x_i))^2, \tag{56}$$

where $P(x_i)$ is the polynomial function. [26] [27] [28]

Once the polynomial coefficients are computed, the polynomial is evaluated using numpy's **polyval** method. This method internally uses *Horner's method* in order to

reduce the number of multiplications [29]. For the polynomial of the form:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \tag{57}$$

it outputs a polynomial of form:

$$P(x) = p_0 x^n + p_1^{n-1} + \cdots + p_n. \tag{58}$$

[30]

10.2 Overview of Simulation Outcomes

The simulation outcomes are presented in the tables 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11. The tables contain the equations for the fitted models, the slopes in all regions, and the MSE value after 100 rounds of execution of the algorithms.

10.3 Struggles of the Single-Proposal Deal-Agreement-Based Algorithm with Dense Graphs

Figure 65 illustrates a Complete graph with six nodes. The red node represents the minimally loaded node in the network that carries the load L_{min}. Using the Single-Proposal Deal-Agreement-Based algorithm, all blue nodes propose a load transfer to the red node. Since the red node now has five potential transfer options, it evaluates these options and accepts the option with maximal effect. This behavior also occurs with Complete graphs with a larger network size. In a graph where each node is interconnected, the balancing potential is usually high, as there are many options per node. However, in this stubborn deterministic way, this load balancing algorithm decreases the error only in very small steps.

Table 2: Simulation overview - K_{1024}: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-linear)** Rounds 1–10	Rounds 11–65	Rounds 66–100	Rounds 1–100	MSE_{100}
K_{1024}	DAB	**Rounds 1–100:** $MSE_r = 844.63 \cdot e^{-0.01r}$	-2.6×10^{-3}	-2.7×10^{-3}	-3.0×10^{-3}	-2.8×10^{-3}	436.85
	PPS	**Rounds 10–80:** $MSE_r = 2530.41 \cdot e^{-0.9r}$	-0.38	-0.39	-0.16	-0.31	1.73×10^{-28}
	ATPPS	**Rounds 10–65:** $MSE_r = 4309.94 \cdot e^{-1.06r}$	-0.43	-0.46	-0.02	-0.3	5.63×10^{-28}

Table 3: Simulation overview - S_{1024}: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-linear)** Rounds 1–10	Rounds 11–45	Rounds 46–100	Rounds 1–100	MSE_{100}
S_{1024}	DAB	**Rounds 1–100:** $MSE_r = 840.42 \cdot e^{-0.01r}$	-2.3×10^{-3}	-2.3×10^{-3}	-2.5×10^{-3}	-2.4×10^{-3}	480.48
	PPS	**Rounds 10–45:** $MSE_r = 29794.60 \cdot e^{-1.39r}$	-0.5	-0.6	-4.8×10^{-3}	-0.26	8.31×10^{-24}
	ATPPS	**Rounds 18–60:** $MSE_r = 9329.40 \cdot e^{-1.05r}$	-0.45	-0.45	-0.11	-0.26	6.52×10^{-24}

Table 4: Simulation overview - R_{1024}: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–10	Rounds 11–69	Rounds 70–100	Rounds 1–100	MSE_{100}
R_{1024}	DAB	**Rounds 10–60:** $MSE_r = 1.72 \times 10^{-5} r^4 - 2.30 \times 10^{-3} r^3 + 0.19 r^2 - 5.99 r + 114.83$	-1.1	-0.51	-0.5	-0.78	22.41
	PPS	**Rounds 10–60:** $MSE_r = 2.99 \times 10^{-5} r^4 - 5.0 \times 10^{-3} r^3 + 0.32 r^2 - 9.68 r + 166.30$	-0.93	-0.55	-0.52	-0.74	27.68
	ATPPS	**Rounds 10–60:** $MSE_r = 3.04 \times 10^{-5} r^4 - 5.0 \times 10^{-3} r^3 + 0.32 r^2 - 9.64 r + 161.86$	-0.95	-0.56	-0.49	-0.75	26.56

Table 5: Simulation overview - $T_{32,32}$: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–10	Rounds 10–39	Rounds 40–100	Rounds 1–100	MSE_{100}
$T_{32,32}$	DAB	**Rounds 10–39:** $MSE_r = -1.35 \times 10^{-6} r^5 + 1.89 \times 10^{-4} r^4 - 0.01 r^3 + 0.30 r^2 - 4.6 r + 34.10$ **Rounds 40–100:** $MSE_r = -6.01 \times 10^{-6} r^3 + 1.66 \times 10^{-3} r^2 - 0.16 r + 6$	-2.1	-1.2	-2	-1.7	436.85
	PPS	**Rounds 10–39:** $MSE_r = -5.54 \times 10^{-6} r^5 + 7.65 \times 10^{-4} r^4 - 0.04 r^3 + 1.16 r^2 - 16.81 r + 112.86$ **Rounds 40–100:** $MSE_r = -1.15 \times 10^{-5} r^3 + 3.21 \times 10^{-3} r^2 - 0.33 r + 13.72$	-1.5	-1.2	-1.4	-1.4	1.73×10^{-28}
	ATPPS	**Rounds 10–39:** $MSE_r = -3.65 \times 10^{-6} r^5 + 5.16 \times 10^{-4} r^4 - 0.03 r^3 + 0.83 r^2 - 12.52 r + 88.16$ **Rounds 40–100:** $MSE_r = -9.99 \times 10^{-6} r^3 + 2.80 \times 10^{-3} r^2 - 0.28 r + 11.29$	-1.6	-1.3	-1.7	-1.5	5.63×10^{-28}

Table 6: Simulation overview - $L_{512,512}$: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–7	Rounds 8–50	Rounds 51–100	Rounds 1–100	MSE_{100}
$L_{512,512}$	DAB	**Rounds 20–100:** $MSE_r = -5.89 \times 10^{-5} r^3 + 0.03 r^2 - 5.68 r + 459.42$	-0.34	-0.32	-1.1	-0.44	108.90
	PPS	**Rounds 20–100:** $MSE_r = 8.44 \times 10^{-7} r^4 - 2.52 \times 10^{-4} r^3 - 0.03 r^2 - 1.64 r + 56.68$	-1.3	-0.56	-0.51	-0.88	14.71
	ATPPS	**Rounds 20–100:** $MSE_r = 8.69 \times 10^{-7} r^4 - 2.56 \times 10^{-4} r^3 + 0.03 r^2 - 1.62 r + 54.48$	-1.3	-0.57	-0.51	-0.89	13.82

Table 7: Simulation overview - $L_{128,896}$: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–7	Rounds 8–50	Rounds 51–100	Rounds 1–100	MSE_{100}
$L_{128,896}$	DAB	**Rounds 20–100:** $MSE_r = 3.46 \times 10^{-6} r^4 - 1.12 \times 10^{-3} r^3 + 0.14 r^2 - 7.69 r + 190.78$	-0.86	-0.87	-0.78	-0.84	17.47
	PPS	**Rounds 20–100:** $MSE_r = -3.72 \times 10^{-8} r^5 + 1.31 \times 10^{-5} r^4 - 1.85 \times 10^{-3} r^3 + 0.13 r^2 - 4.96 r + 84.91$	-1.2	-1.3	-1.1	-1.2	3.31
	ATPPS	**Rounds 20–100:** $MSE_r = 1.59 \times 10^{-6} r^4 - 4.74 \times 10^{-4} r^3 + 0.05 r^2 - 2.94 r + 70.59$	-1.2	-1.1	-1.5	-1.2	3.41

Table 8: Simulation overview - $L_{896,128}$: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–7	Rounds 8–50	Rounds 51–100	Rounds 1–100	MSE_{100}
$L_{896,128}$	DAB	**Rounds 20–100:** $MSE_r = -1.93 \times 10^{-4} r^3 + 0.05 r^2 - 5.33 r + 745.95$	-0.08	-0.1	-0.11	-0.09	542.09
	PPS	**Rounds 20–100:** $MSE_r = 2.00 \times 10^{-7} r^4 - 6.01 \times 10^{-5} r^3 + 6.95 \times 10^{-3} r^2 - 0.39 r + 13.44$	-2	-0.59	-0.47	-1.2	3.59
	ATPPS	**Rounds 20–100:** $MSE_r = 2.28 \times 10^{-7} r^4 - 6.77 \times 10^{-5} r^3 + 7.68 \times 10^{-3} r^2 - 0.42 r + 13.62$	-2	-0.61	-0.5	-1.2	3.27

Table 9: Simulation overview - $ROC_{32,32}$: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–5	Rounds 6–10	Rounds 11–100	Rounds 1–100	MSE_{100}
$ROC_{32,32}$	DAB	**Rounds 20–100:** $MSE_r = 1.04 \times 10^{-6} r^4 - 2.94 \times 10^{-4} r^3 + 0.03 r^2 - 1.66 r + 45.30$	-0.51	-2.1	-1.1	-1.1	6.55
	PPS	**Rounds 10–100:** $MSE_r = -0.05 r + 24.70$	-1.8	-0.61	-0.09	-0.81	19.80
	ATPPS	**Rounds 10–100:** $MSE_r = r^{-7.59} * 10^{43.26}$	-1.8	-0.49	-0.47	-1	8.42

Table 10: Simulation overview - $ROC_{128,8}$: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–5	Rounds 6–11	Rounds 12–100	Rounds 1–100	MSE_{100}
$ROC_{128,8}$	DAB	**Rounds 30–100:** $MSE_r = 5.45 \times 10^{-7} r^4 - 1.7 \times 10^{-4} r^3 + 0.02 r^2 - 1.33 r + 46.71$	-1.4	-0.75	-0.66	-0.95	10.64
	PPS	**Rounds 30–100:** $MSE_r = 1.04 \times 10^{-6} r^4 - 3.41 \times 10^{-4} r^3 + 0.04 r^2 - 2.73 r + 97.58$	-1.3	-0.36	-0.55	-0.79	22.33
	ATPPS	**Rounds 30–100:** $MSE_r = 9.54 \times 10^{-7} r^4 - 2.93 \times 10^{-4} r^3 + 0.03 r^2 - 2.05 r + 67.02$	-1.3	-0.49	-0.7	-0.89	13.68

Table 11: Simulation overview - $ROC_{8,128}$: fitted model, slopes per region, and final MSE

Topology		**Fitted Model**	**Slope (log-log)** Rounds 1–5	Rounds 6–11	Rounds 12–100	Rounds 1–100	MSE_{100}
$ROC_{8,128}$	DAB	**Rounds 15–50:** $MSE_r = -3.33 \times 10^{-3} r^3 + 0.63 r^2 - 40.23 r + 872.75$ **Rounds 60-100:** $MSE_r = 2.96 \times 10^{-6} r^4 - 9.97 \times 10^{-3} r^3 + 0.13 r^2 - 7.16 r + 161.73$	-0.12	-0.41	-2.1	-1.1	5.18
	PPS	**Rounds 20–100:** $MSE_r = -1.5 \times 10^{-3} r + 6.05$	-2	-1.9	-0.01	-1.1	5.95
	ATPPS	**Rounds 20–100:** $MSE_r = 6.07 \times 10^{-5} r^2 - 0.02 r + 6.39$	-2.1	-1.6	-0.13	-1.1	4.56

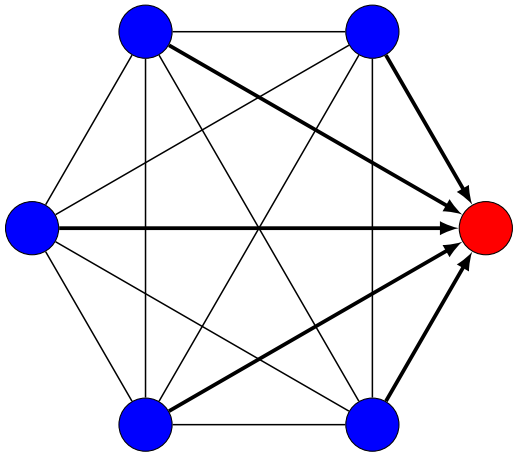


Figure 65: Complete graph: network size 6

A similar problem arises in the Star graph, as shown in figure 66. All nodes have the central node as a neighbor. Now all leaves propose a load transfer to the central node, only now with the difference that the central node is not necessarily the minimally loaded node in the network (i.e., it does not carry L_{min}). The minimally loaded node in the example of figure 66 is a leaf node. The central node now receives a maximum of 4 proposals (i.e., exactly when the central node holds the second smallest load in the network) and accepts exactly one. If the central node is the maximally loaded node (i.e., the node that carries L_{max}), then there will be no proposals from the leaf nodes. Only the central node proposes the minimally loaded node, and that is the only load transfer.

In both scenarios, the number of load transfers is heavily limited.

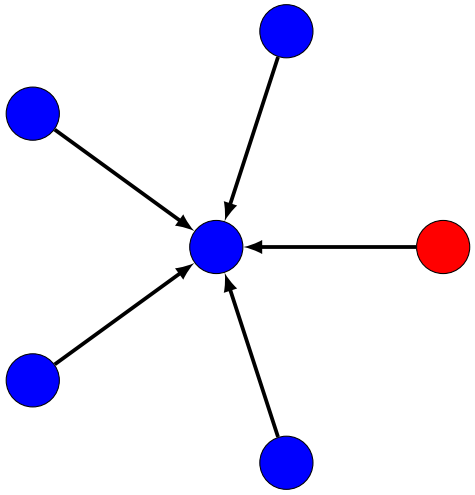


Figure 66: Star graph: network size 6

Bibliography

[1] S. Nugroho, A. Weinmann, and C. Schindelhauer, *Adding Pull to Push Sum for Approximate Data Aggregation.* Springer, 2023.

[2] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pp. 482–491, 2003.

[3] Y. Dinitz, S. Dolev, and M. Kumar, "Local deal-agreement algorithms for load balancing in dynamic general graphs," *Theory of Computing Systems*, vol. 67, pp. 348–382, Apr 2023.

[4] C. Xu and F. C. Lau, *Load Balancing in Parallel Computers: Theory and Practice.* USA: Kluwer Academic Publishers, 1997.

[5] C. Schindelhauer, "Lecture notes in graph theory," 2021.

[6] C. Schindelhauer, "Lecture notes in peer-to-peer networks," 2023.

[7] E. Bayazitoglu, "Comparative analysis of load balancing algorithms in general graphs." University of Freiburg, 2024.

[8] M. A. Iqbal, J. H. Saltz, and S. H. Bokhari, "A comparative analysis of static and dynamic load balancing strategies," in *International Conference on Parallel Processing, ICPP'86, University Park, PA, USA, August 1986*, pp. 1040–1047, IEEE Computer Society Press, 1986.

[9] S. Banerjee and D. Sarkar, "Hypercube connected rings: A scalable and fault-tolerant logical topology for optical networks," *Computer Communications*, vol. 24, pp. 5–6, 2001.

[10] D. B. West, *Introduction to Graph Theory*. Pearson; Pearson Education, Inc., 2nd, reprint ed., 2002. Includes solution manual.

[11] A. O. Jayeola and P. T. Ayomide, "Modelling and assessment of the star network topology using opnet simulation techniques," *International Journal of Scientific Research in Computer Science and Engineering*, vol. 11, no. 1, pp. 14–22, 2023.

[12] V. P. Vidomenko, "The traffic self-guidance for multimedia communications," *Automation of the Russian Academy of Sciences*, 1997. Accessed on January 11, 2025.

[13] P. Mahlmann, *Peer-to-peer networks based on random graphs*. PhD thesis, University of Paderborn, 2010. Paderborn, Univ., Diss., 2010.

[14] J. Jonasson, "Lollipop graphs are extremal for commute times," *Random Structures & Algorithms*, vol. 16, no. 2, pp. 131–142, 2000.

[15] Drakos, N. and Moore, R., *PeerSim: HOWTO: Build a new protocol for the PeerSim 1.0 simulator*, December 2005. Accessed: August 13, 2024.

[16] R. Sabin and F. Bertolotti, "A master equation for power laws," *R. Soc. Open Sci.*, vol. 9, 2022.

[17] J. Lorenzo, "Graphing by hand and on computer," 2000. Accessed: 2025-02-10.

[18] H. Motulsky and A. Christopoulos, *Fitting Models to Biological Data Using Linear and Nonlinear Regression: A practical guide to curve fitting*. 10 2023.

[19] LibreTexts Mathematics, "Exponential and logarithmic regressions," 2022. Accessed: 2025-02-23.

[20] S. Community, *SciPy curvefit Documentation*, 2024. Accessed: 2025-02-12.

[21] S. Community, *SciPy linregress Documentation*, 2024. Accessed: 2025-02-12.

[22] H. P. Gavin, "The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems," 2020. Accessed: 2025-02-11.

[23] J. J. Moré, "The levenberg-marquardt algorithm: Implementation and theory," in *Numerical Analysis* (G. A. Watson, ed.), (Berlin, Heidelberg), pp. 105–116, Springer Berlin Heidelberg, 1978.

[24] T. Brox, "Lecture notes in optimisation, lecture 3: Newton- and quasi-newton methods," 2014-2020.

[25] A. Wooditch, N. J. Johnson, R. Solymosi, J. Medina Ariza, and S. Langton, *Ordinary Least Squares Regression*, pp. 245–268. Springer International Publishing, 2021.

[26] N. Community, *NumPy polyfit Documentation*, 2024. Accessed: 2025-02-12.

[27] Y. Li and X. Ding, "Vandermonde determinant and its applications," *Journal of Education and Culture Studies*, vol. 7, p. p16, 10 2023.

[28] G. I. of Technology, *Least Squares Approximation*, 2024. Accessed: 2025-02-12.

[29] N. Community, *NumPy polyval Documentation*, 2024. Accessed: 2025-02-20.

[30] W. D. Project, "Horner's method," 2024. Accessed: 2024-02-08.