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

January 25th, 2025

Writing Period

18. 12. 2024 – 18. 03. 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

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. 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. 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 original Push-Pull Sum algorithm. For the Adaptive Threshold Push-Pull Sum algorithm, I provide pseudocode, implement the different load balancing approaches 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 scalable 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, making it efficient in dense graphs, aswell as low regular degree graphs. 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

1 Introduction **1**

1.1 Preliminaries 2

1.2 Motivation 3

1.3 Related Work 4

1.4 Hypothesis 4

1.5 Contribution 5

2 Problem Overview **6**

2.1 Setting 6

2.2 Approach 7

3 Algorithms **8**

3.1 Characteristics 8

3.2 Classic Push-Pull Sum Algorithm 9

3.2.1 Example 12

3.3 Continunous Single-Proposal Deal-Agreement-Based Algorithm . . . 14

3.3.1 Example 16

3.4 Adaptive Threshold Push-Pull Sum Algorithm 18

3.4.1 Example 19

3.4.2 Aspired Outcome 21

4 Topologies **23**

4.1 Complete Graph 23

4.2 Torus Grid Graph 24
4.3 Ring Graph 25
4.4 Star Graph 26
4.5 Lollipop Graph 27
4.6 Ring of Cliques 28
4.7 Expected Outcome 28

5 Implementation and Technology Stack **31**
5.1 Programming Languages 31
5.2 Simulation Framwork 31
5.3 Implementation Details 34

6 Simulation Outcomes **36**
6.1 Complete Graph 38
6.2 Star Graph 42
6.3 Ring Graph 45
6.4 Torus Grid Graph 49
6.5 Lollipop Graph 53
6.5.1 (512, 512) Lollipop Graph 53
6.5.2 (128, 896) Lollipop Graph 57
6.5.3 (896, 128) Lollipop Graph 65
6.6 Ring of Cliques 69
6.6.1 32x32 Ring of Cliques 69
6.6.2 128x8 Ring of Cliques 71
6.6.3 8x128 Ring of Cliques 74

7 Conclusion **83**

8 Acknowledgments **84**

9 Appendix **85**

Bibliography **93**

List of Figures

1 Overview of the setting 2

2 Push-Pull Sum: push and pull actions 13
3 Push-Pull Sum: setting after round 1 14
4 Deal-Agreement-Based: initial setup and setup after round 1 17
5 Adaptive Threshold Push-Pull Sum: push and pull actions 20
6 Adaptive Threshold Push-Pull Sum: setting after round 1 21

7 Complete graph: network size 16 24
8 Torus Grid graph: network size 16 25
9 Ring graph: network size 16 26
10 Star graph: network size 16 27
11 Lollipop graph: network size 16 27
12 Ring of Cliques: network size 16 28

13 Project structure 34
14 Process model: methodic 35

15 Complete graph: mean squared error per rounds (log-log) 39
16 Complete graph - exponential regression fit: DAB 41
17 Complete graph - exponential regression fit: PPS 41
18 Complete graph - exponential regression fit: ATPPS 42
19 Complete graph: heat map of slopes per region 43
20 Star graph: mean squared error per rounds (log-log) 45

21 Star graph - exponential regression fit: DAB 46
22 Star graph - exponential regression fit: PPS 46
23 Star graph - exponential regression fit: ATPPS 47
24 Star graph: heat map of slopes per region 48
25 Ring graph: mean squared error per rounds (log-log) 49
26 Ring graph - polynomial regression fit: DAB 50
27 Ring graph - polynomial regression fit: PPS 50
28 Ring graph - polynomial regression fit: ATPPS 51
29 Ring graph: heat map of slopes per region 51
30 Torus Grid: mean squared error per rounds (log-log) 54
31 Torus Grid - polynomial regression fit: DAB; rounds 10-39 and 40-100 54
32 Torus Grid - polynomial regression fit: PPS; rounds 10-39 and 40-100 55
33 Torus Grid - polynomial regression fit: ATPPS; rounds 10-39 and 40-100 55
34 Torus Grid: heat map of slopes per region 56
35 (512, 512)-Lollipop graph: mean squared error per rounds (log-linear) 58
36 (512, 512)-Lollipop graph - polynomial regression fit: DAB 58
37 (512, 512)-Lollipop graph - polynomial regression fit: PPS 59
38 (512, 512)-Lollipop graph - polynomial regression fit: ATPPS 59
39 Lollipop graph: heat map of slopes per region 60
40 (128, 896)-Lollipop graph: mean squared error per rounds (log-log) . 62
41 (128, 896)-Lollipop graph - polynomial regression fit: DAB 62
42 (128, 896)-Lollipop graph - polynomial regression fit: PPS 63
43 (128, 896)-Lollipop graph - polynomial regression fit: ATPPS 63
44 (128, 896)-Lollipop graph: heat map of slopes per region 64
45 (896, 128)-Lollipop graph: mean squared error per rounds (log-log) . 66
46 (896, 128)-Lollipop graph - polynomial regression fit: DAB 66
47 (896, 128)-Lollipop graph - polynomial regression fit: PPS 67
48 (896, 128)-Lollipop graph - polynomial regression fit: ATPPS 67
49 (896, 128) Lollipop Graph: heat map of slopes per region 68

50 (32×32)-Ring of Cliques: mean squared error per rounds (log-linear and log-log) 71
51 (32×32)-Ring of Cliques - polynomial regression fit: DAB 72
52 (32×32)-Ring of Cliques - polynomial regression fit: PPS 72
53 (32×32)-Ring of Cliques - logarithmic regression fit: ATPPS 73
54 (32×32)-Ring of Cliques: heat map of slopes per region 73
55 (128×8)-Ring of Cliques: mean squared error per rounds (log-linear and log-log) 74
56 (128×8)-Ring of Cliques - polynomial regression fit: DAB 75
57 (128×8)-Ring of Cliques - polynomial regression fit: PPS 75
58 (128×8)-Ring of Cliques - polynomial regression fit: ATPPS 76
59 (128×8)-Ring of Cliques: heat map of slopes per region 76
60 (8×128)-Ring of Cliques: mean squared error per rounds (log-linear and log-log) 78
61 (8×128)-Ring of Cliques - polynomial regression fit: rounds 20-50 and 55-100 78
62 (8×128)-Ring of Cliques - polynomial regression fit: PPS 79
63 (8×128)-Ring of Cliques - polynomial regression fit: ATPPS 79
64 (8×128)-Ring of Cliques: heat map of slopes per region 80

65 Example Ring Graph 90

List of Tables

1 Overview over example outcomes 22

2 Simulation overview per topology: fitted model, slopes per region, final MSE 86

3 Simulation overview per topology: fitted model, slopes per region, final MSE 87

4 Simulation overview per topology: fitted model, slopes per region, final MSE 88

5 Simulation overview per topology: fitted model, slopes per region, final MSE 89

List of Algorithms

1 Push-Pull Sum algorithm 10
2 Continuous Single-Proposal Deal-Agreement-Based algorithm 15
3 Adaptive Threshold Push-Pull Sum algorithm 20

Listings

5.1 Example configuration 32

6.1 Snippet of simulation outcomes ATPPS: Experiment 2 78

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. 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 not only enhances coordination in distributed systems but also improves scalability and ensures high availability.

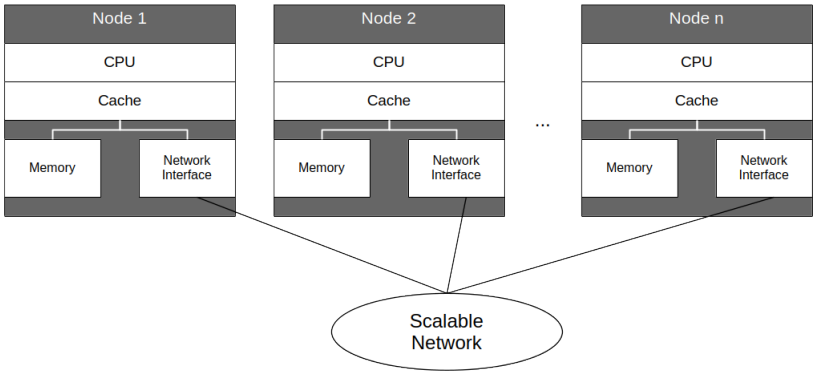


Figure 1: Overview of the setting

1.1 Preliminaries

A graph G is defined as $G = (V, E \subseteq V \times V)$, where $V := \{v_0, v_1, ..., v_{n-1}\}$ represents the vertices (or nodes) of G and $E := \{e_1, e_2, ..., e_{n-1}\}$ represents the edges. The graph may consist of a single vertex and no edges. The number of vertices $|V|$ is often 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. If an edge connects two vertices, these vertices are adjacent to each other, thus are neighboring vertices. A loop is an edge that connects a vertex to itself. Multiple edges refer to edges that share the same pair of endpoints. A simple graph has no loops or multiple edges. A clique is a set of pairwise adjacent vertices. 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. A graph is connected if each pair of vertices in the graph belongs to a path. Bipartite graphs are graphs where the set of graph edges is the union of two disjoint independent sets. 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. [5]

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" [6]. 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. The simulations were conducted across networks of varying sizes, which also influenced the convergence speed, with larger network sizes generally requiring more rounds to achieve balance.

To address these challenges, this research introduces a novel algorithm, called the Adaptive Threshold Push-Pull Sum algorithm, which leverages the strengths of both the Continuous Single-Proposal Deal-Agreement-Based and the Push-Pull Sum algorithms while establishing a trade-off to lessen their respective drawbacks. The newly introduced algorithm uses the mechanic of adaptive thresholding in order to prevent load transfers with low effect in error reduction, since load transfers are pricy operations. By adapting to the current state of balance of the investigated networks, this solution aims to deliver robust performance across a wide range of scenarios, effectively bridging the gap between the two algorithms.

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 research similarly employs 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 distinct network topologies. Specifically, it is expected to perform better than the Deal-Agreement-Based algorithm in high-degree networks and perform better than the Push-Pull Sum algorithm in low-degree networks, achieving a balance that improves overall adaptability and efficiency compared to both.

This hypothesis will be evaluated through a comparative analysis of MSE across six distinct topologies, some of which feature varying network sizes, to assess the algorithm's robustness and scalability in different network environments. 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 detailed insights into the algorithm's performance at specific stages of execution.

1.5 Contribution

This study introduces a novel load balancing algorithm that combines the strengths of two established approaches: randomized load balancing and deterministic load balancing based on deal agreement. By integrating an adaptive threshold mechanism, the algorithm dynamically adjusts to the current state of the network to enhance performance. It leverages the push and pull mechanisms to facilitate convergence to a balanced state.

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. The problem setting and the approach to address the problem are elaborated on in this section.

2.1 Setting

Continuous and Discrete: 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.

Synchronous and Asynchronous: 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.

Static or Dynamic: 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.

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 distinct network topologies, each comprising 2^{10} nodes.

2.2 Approach

This research consists of three steps: the design of a load balancing algorithm, the simulation phase to test its ability to balance the state of the network across different topologies, and a comparative analysis of the simulation outcomes using statistical methods. In prior research [6], the strengths and weaknesses of two distinct 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 novel adaptive threshold load balancing algorithm builds upon these findings. Each simulation outcome includes 30 distinct experiments to ensure statistical significance. The results are analyzed using model fitting to identify trends in MSE reduction, with slopes calculated for different regions to assess the consistency of performance. The findings are presented in plots, accompanied by explanations that provide insights into the observed 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 newly 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 rationale behind its development. 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 desired results are also described.

3.1 Characteristics

Like the graphs, load balancing algorithms may have different characteristics. These characteristics are elaborated on below:

Static and Dynamic: 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 [7].

Stochastic and Deterministic: 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]

Global and Local: Local load balancing algorithms allow nodes to transfer loads within their domain or neighborhood, while global load balancing algorithms enable load balancing operations across the entire network [4].

Monotonic and Non-monotonic: 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].

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].

Anytime: 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 shows more feasible states with intermediate 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 Classic 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. The sum of all initial $s_{i,0}$ 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 collects messages $M_{i,r}$ sent by other nodes $\{(s_m, w_m)\}$ in the previous round $r-1$, requesting data. The nodes then update their 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.

Algorithm 1 Push-Pull Sum algorithm

1: **procedure** RequestData
2: Chose a random neighbor node v
3: Send $(\frac{s_{u,r}}{2}, \frac{w_{u,r}}{2})$ to the chosen node v and the node u itself
4: **end procedure**
5: **procedure** ResponseData
6: $R_{u,r} \leftarrow$ Set of the nodes calling u at a round r
7: **for all** $i \in R_{u,r}$ **do**
8: Reply to i with $\left(\frac{\frac{s_{u,r}}{2}}{|R_{u,r}|}, \frac{\frac{w_{u,r}}{2}}{|R_{u,r}|}\right)$
9: **end for**
10: **end procedure**
11: **procedure** Aggregate
12: $M_{u,r} \leftarrow \{(s_m, w_m)\}$ messages sent to u at a round $r-1$
13: $s_{u,r} \leftarrow \sum_{m \in M_{u,r}} s_m, w_{u,r} \leftarrow \sum_{m \in M_{u,r}} w_m$
14: $f_{avg} \leftarrow \frac{s_{u,r}}{w_{u,r}}$
15: **end procedure**

The setting in [1] is similar to ours. 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 is defined as:

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

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

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

is the conditional expectation of $\Phi_r + 1$ 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 internal node acts as a distributor of the load. Since each leaf chooses the internal node with 100% possibility as a "random" neighbor the internal node is involved as a 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 for the Complete graph, the Ring of Cliques and the Lollipop graph, the density of the graph plays an crucial role. Since nodes select neighbors randomly, the high edge density allows the algorithm to spread loads efficiently, preventing bottlenecks and reducing reliance on deterministic paths. 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. For these topologies redundant communication may occur, as two nodes may push back and fourth, with the sum being $\frac{s_{i,r}}{2}$ and the weight being $\frac{w_{i,r}}{2}$ for the push and pull operations. The result of this action is that their load values interact in such a way that one node u adopts the previous round's state of another node v, leading to:

$$load_r(u) = load_{r-1}(v)$$

and vice versa. This phenomenon is described in chapter 9. $load_r(u)$ represents the load of node u at round r.

3.2.1 Example

The example in figure 2 illustrates a execution of the Push-Pull Sum algorithm on a Complete graph K_4 with 4 nodes labeled *A, B, C* and *D*. Each node is initially assigned a sum and a weight value. The undirected graph represents the network topology, showing connections between neighboring nodes.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 depicts the first round of a execution of the Push-Pull Sum algorithm. The push and pull operations are distinct. The left-hand side of figure 2 depicts the behaviour of the nodes while executing the push operations, while the right 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.

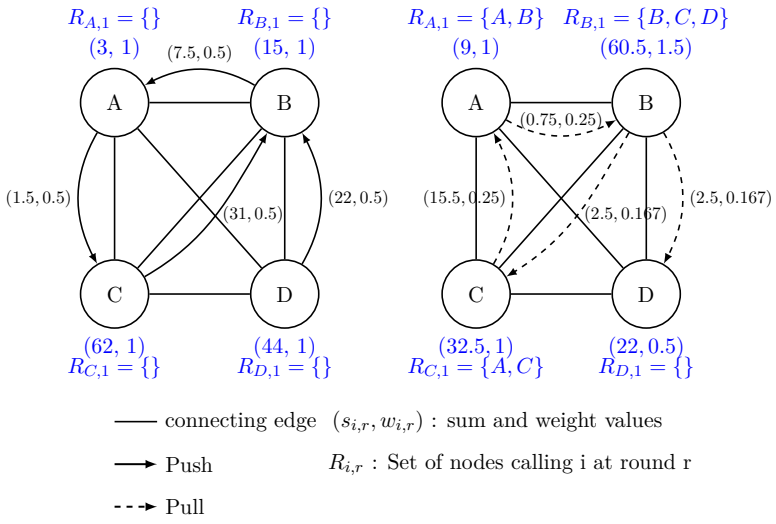


Figure 2: Push-Pull Sum: push and pull actions

- 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, we get $R_{A,1} = \{A, B\}$. 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 responsive 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.

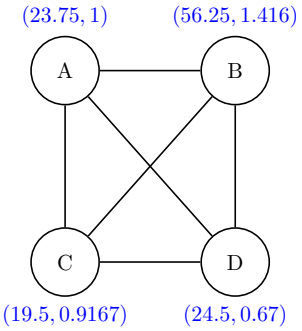


Figure 3: Push-Pull Sum: setting after round 1

3.3 Continunous Single-Proposal Deal-Agreement-Based Algorithm

The Continuous Single-Proposal Deal-Agreement-Based algorithm proposed by [3] is unlike the Push-Pull Sum algorithm, which is not a diffusion-based algorithm. The goal of load balancing is achieved based on deterministic deal agreements, where one node proposes a load to one neighboring node, and the neighboring node either 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. 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

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

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. 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 load values of 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 analysis in Dinitz et al.[3] is based on a potential function that measures the network. The potential for a node u is defined as

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

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

$$p(G) = \sum_{u \in V} p(u)$$

, which is the sum of all potential of each node in the graph G. Any fair load transfer of load l decreases the potential of the graph by at least $2 * l^2$. 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 seeks the minimally loaded partner for load transfer. For a Complete graph, the minimal load partner is the same node with loads of L_{min} (the minimally loaded node in the network). This causes each node to propose to the same node, which then evaluates the proposals and accepts exactly one transfer proposal, namely the maximal one. 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 minimal neighbor; instead, they propose to the least-loaded neighbor within their respective cliques. For the Lollipop graph, the nodes in the path graph 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. In these scenarios, the proposals are more evenly distributed among nodes due to the lower density of the graphs, allowing a greater number of nodes to participate in load transfers. The Star graph is an exception to this case, since we have a similar bottleneck as in the Complete graph, since the internal node is the common neighbor for each leaf, and thus each leaf proposes a load to the internal node. The internal node can accept only one proposal. The simulation results in the student project [6] indicate the performances of each load balancing algorithm for each topology very clearly.

3.3.1 Example

Figure 4 illustrates two settings. The left-hand side of the figure depicts the initial state and the right-handside illustrates the state after one round of execution. The setting is the same as in the example above in the Push-Pull Sum section, with the

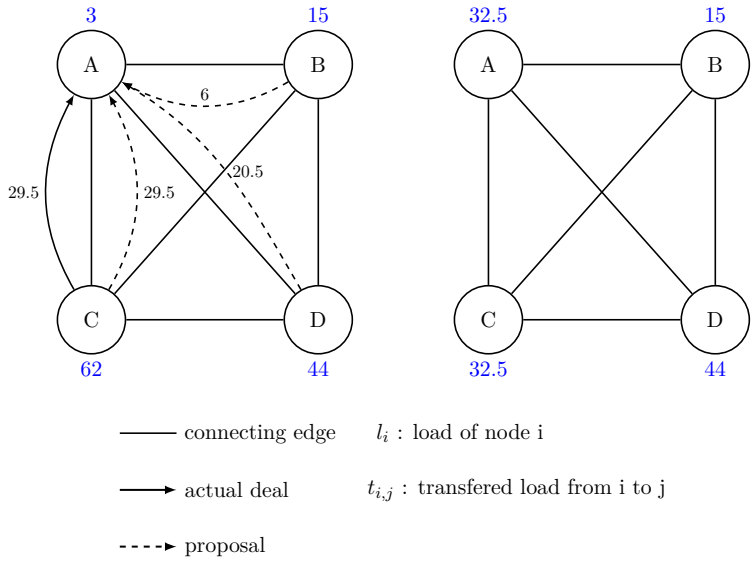


Figure 4: Deal-Agreement-Based: initial setup and setup after round 1

difference that each node is assigned a load value, instead of sum and weight values. In the figure, 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 minimal loaded neighbors. 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 an 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 transfered 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*, each node u chooses a subset $RN_{u,r}$ of $\log_2(|neighborhood_u|)$ random neighbors. This increases the likelihood of selecting a well-suited neighbor for load transfer. Selecting only one random neighbor lowers the chance to find an optimal or good neighbor to execute a load transfer. Then the load difference between the node u and each node 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

$$k * \sqrt{MSE_r - 1}$$

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; thus, 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. The way the loads are distributed, namely the push and the pull mechanisms, 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 load. 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 Threhsold Push-Pull Sum algorithm inherits is an 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 $\log_2(|neighborhood|)$ neighbors. For this setting, each node chooses 2 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 error reduction depends on the sensitivity factor k. If k is set to 1, the load transfer

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 $v_i \in RN$ **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

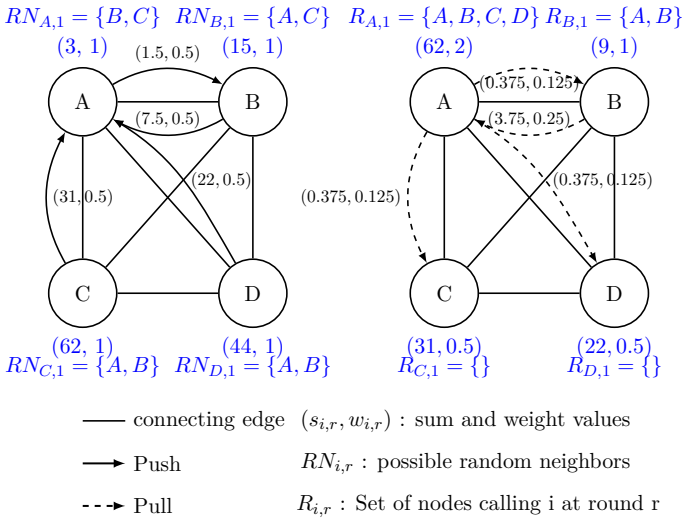


Figure 5: Adaptive Threshold Push-Pull Sum: push and pull actions

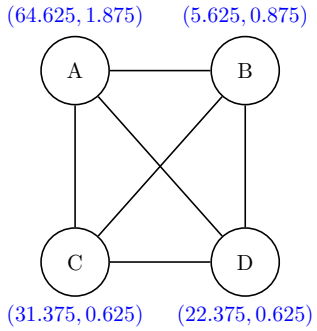


Figure 6: Adaptive Threshold Push-Pull Sum: setting after round 1

between node 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 Aspired Outcome

Considering the simulation results presented in [6], the discrepancy between the MSE reduction abilities per round for the different topologies shows that the algorithms perform either very well or mediocre to bad. The primary motivation behind designing the Adaptive Threshold Push-Pull Sum algorithm was to find a compromise solution that enhances adaptability across various network topologies.

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 1, 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 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.

	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: Overview over example outcomes

4 Topologies

The simulations were conducted for six distinct network topologies. Each network has 2^{10} (1024) nodes. The behavior of the algorithms in these different topologies is observed, since each topology has different 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 $\frac{N\times(N-1)}{2}$ 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 high reliability and low latency are critical. The Complete graph ensures optimal routing paths between nodes, due to its low diameter [8].

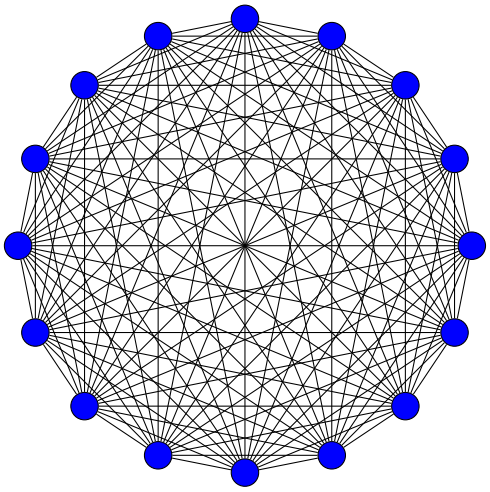


Figure 7: Complete graph: network size 16

4.2 Torus Grid Graph

The two-dimensional Torus Grid graph, also referred to as the $k \times m$-Torus graph or $T_{k,m}$, is a graph that is built like a two-dimensional mesh with wrap-around edges as depicted in figure 8 [9]. Here, k represents the height, while m denotes the width of the grid. The graph consists of $k \times m$ nodes and $2 \times k \times m$ 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 $\frac{\min(k,m)}{2}$. Tori are scalable, as the number of connections per node is constant, regardless of the graph size. In the previous research [6], the simulation results showed to not vary over different network sizes. Torus topology is widely used in supercomputers like IBM Blue Gene and Cray systems for high-performance computing. It is also implemented in Network-on-Chip (NoC) designs for its ability to handle data distribution with low latency and high fault tolerance [8].

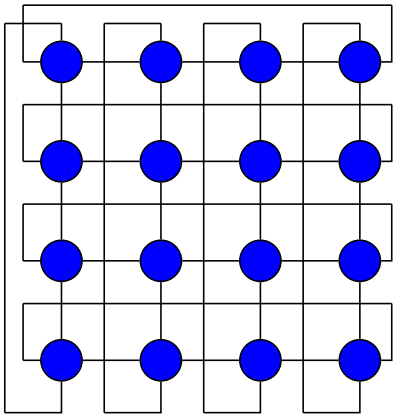


Figure 8: Torus Grid graph: network size 16

4.3 Ring Graph

The Ring graph R_N is a regular graph, where each node has a degree of two, 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 $\lfloor \frac{N}{2} \rfloor$. The regularity ensures that no single node has an advantage, making algorithm design and guaranteeing fairness in load balancing easier. 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 [10]. 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.

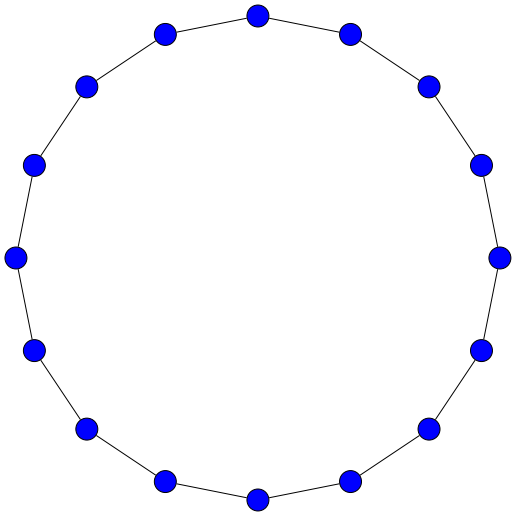


Figure 9: Ring graph: network size 16

4.4 Star Graph

A Star graph S_N as illustrated in figure 10, is a bipartite graph [11] that is structured like a tree structure 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 internal node. The central node has a degree of $N-1$. The diameter of a Star graph is 2, as the path from one leaf node through internal node to another leaf node takes two steps. From a load balancing perspective the internal 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 and collects results. A challenge to face when using the Star topology is that the central node may become overloaded in high-load scenarios, requiring careful design to prevent bottlenecks. 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 [12]. 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.

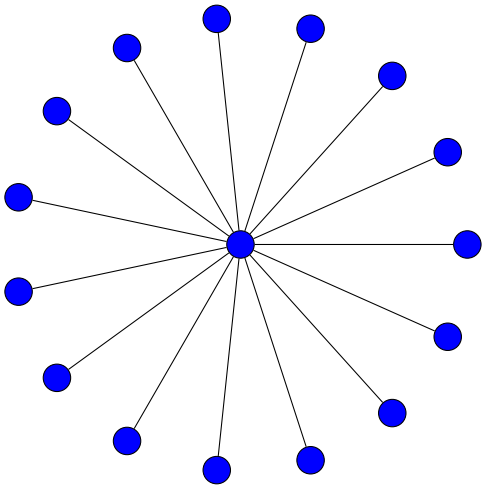


Figure 10: Star graph: network size 16

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 [13].

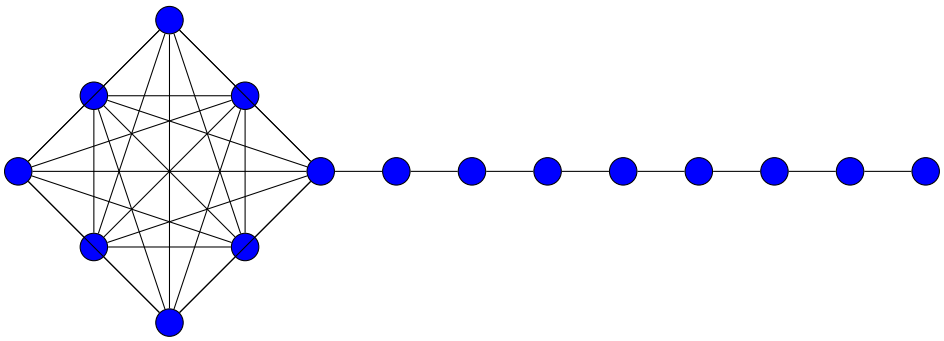


Figure 11: Lollipop graph: network size 16

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 [9]. A $k \times m$ ring of cliques has $\left(k \times \left(\frac{m \times (m-1)}{2} - 1\right)\right) + k$ edges. The connectivity of the graph increases with larger clique sizes and decreases with smaller clique sizes.

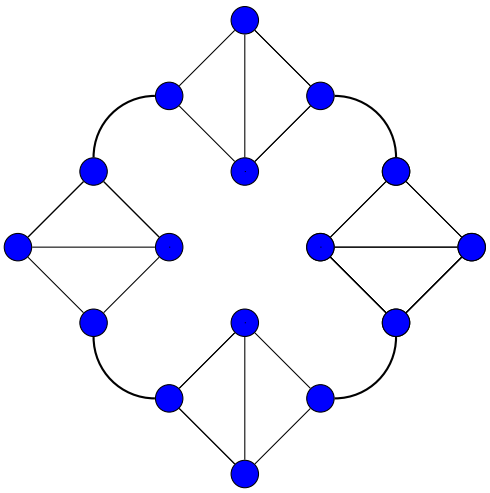


Figure 12: Ring of Cliques: network size 16

4.7 Expected Outcome

- **Complete graph**: The high-connectivity of the Complete graph provides a variety of load transfer opportunities for each node, which will benefit randomized algoithms such as the Push-Pull Sum based load balancing algorithms, 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.

- **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 and provide balanced redistribution across the entire grid. 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 utilises smaller neighborhoods more effectively than a purely randomized strategy.

- **Ring graph**: Unlike Complete graphs or Tori, the Ring graph possesses a sequential nature of data transfers, which makes it slower for loads to propagate across the entire ring. The Deal-Agreement-Based algorithm draws an important advantage over the Push-Pull 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 $\log_2(2)$ and ultimately ends up with one neighbor.

- **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 bottleneck, since each leaf has only one neighbor (the central node), all nodes propose to the same central node. The internal neighbor initiates the load transfer, accepting the maximal proposing load, resulting in exactly one load transfer.

- **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 node linking the clique and the Path graph often becomes a bottleneck, since it bridges two different connectivity regions. Within the clique, load balancing is highly efficient for Push-Pull Sum based algorithms, while the Deal-Agreement-Based algorithm struggles due to excessive proposals. 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 perforamce. 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 based algorithms. The Deal-Agreement-Based algorithm benefits from its deterministic load distribution, efficiently transferring loads via the bridging nodes to other cliques once internal balancing is achieved. Push-Pull Sum algorithms struggle due to randomized inter-clique communication. The Adaptive Threshold Push-Pull Sum algorithm mitigates this by selecting neighbors based on a threshold, often favoring inter-clique exchanges, once internal balancing is achieved. The shared nodes between cliques act as bottlenecks, regulating load transfer and potentially becoming overloaded.

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 containing different simulation parameters, settings, and performance metrics. The data analysis part of this project is primarily implemented using *Python* as a programming language.

5.1 Programming Languages

Python *3.12.6* is used for several tasks, mostly for preparing and analyzing 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* is used.

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

5.2 Simulation Framwork

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 our 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  # Synchrounous 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 # The 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*, and a shared *loadBalancingParameters*-class, 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 [14].

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.

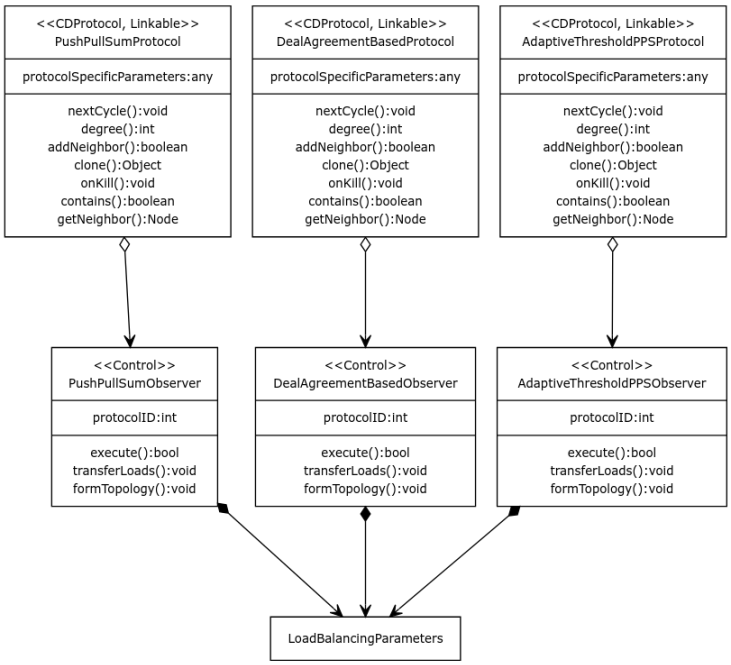


Figure 13: Project structure

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. [14]

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 distinct experiments were created to improve statistical significance. Then a Java script reads these configuration files and assigns an initial load value to each node. Then 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 network (e.g., which

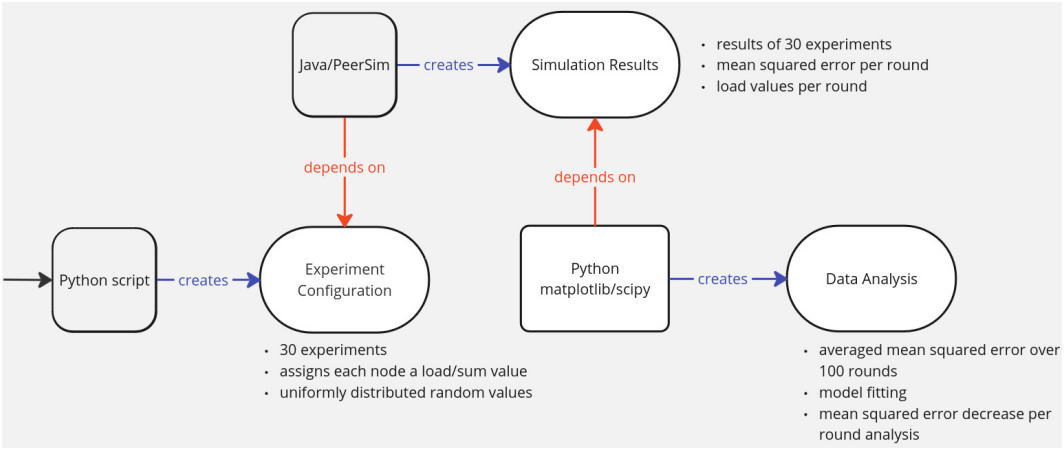


Figure 14: Process model: methodic

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 further 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 distinct topologies (some with varying structural symmetry, e.g., $L_{32,32}$, $L_{128,8}$, and $L_{8,128}$) to test the performance of the load balancing algorithms. 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 (logs for the MSE data 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 overall slope computed across all 100 rounds. 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

$$MSE_r = m * r + b,$$

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

$$MSE_r = a_0 + a_1 * r + a_2 * r^2 + a_3 * r^3 + ... + a_n * r^n.$$

This model is utilized when MSE reduction per round follows a power law relationship. It captures non-linear relationships between the independent variable r and dependent variable MSE_r [15]. Polynomial regression can model non-linear trends like diminishing returns (e.g., rapid MSE reduction initially, then slower reduction). It fits the curvature of MSE decay, even if it's not strictly exponential or linear. Higher-degree polynomials can capture intricate patterns in the reduction of MSE over rounds.

Exponential Regression: The exponential regression model is given by

$$MSE_r = a * e^{-br}.$$

Exponential models capture the initially steep drop in MSE at early rounds, followed by slower reductions 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, thus a faster convergence, while values closer to 0 indicate slower error reduction.

Logarithmic Regression: The logarithmic regression models data as

$$\log(MSE_r) = a + b * \log(r),$$

where a is the initial MSE when $\log(r) = 0$ and b is the rate of reduction per unit increase in $\log(r)$. A large positive b decreases the MSE faster. If b is small, the error reduction slows down as the rounds progress.This model is effective in analyzing load balancing algorithms where initial rounds see significant MSE reductions, followed by progressively smaller improvements.

6.1 Complete Graph

Figure 15 presents the MSE reduction per round for the three load balancing algorithms simulated on the Complete graph on a log-log graph. The DAB curve (black curve) shows a linear trend and a gentle decrease of the MSE over the 100 rounds. The DAB performs poorly since it uses a fixed, 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 minimal neighbor only accepts one proposal per round, the number of load transfers is heavily limited, resulting in slow MSE reduction. The PPS and ATPPS algorithms leverage the high connectivity of the network more effectively, due to their randomized nature. The PPS curve (red curve) exhibits a steady decrease in MSE, therefore showing efficient MSE reduction over time. Since all nodes are equally connected, no structural constraints slow down the process of pushing and pulling loads from neighbors. However, PPS does not distinguish

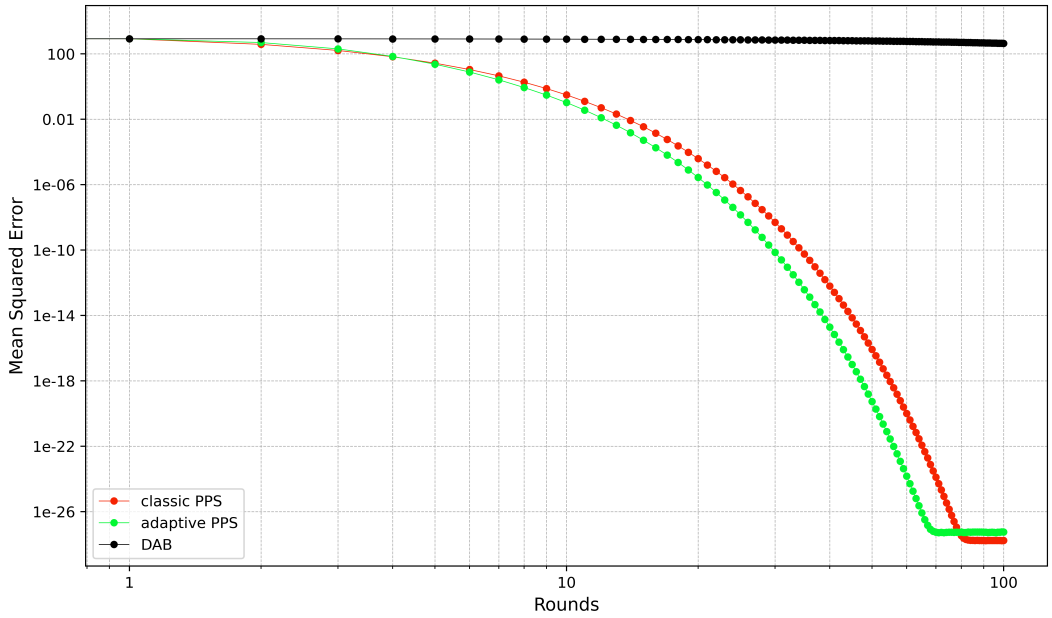


Figure 15: Complete graph: mean squared error per rounds (log-log)

between nodes that are highly imbalanced and those that are nearly balanced, leading to unnecessary load transfers in later rounds, which slows convergence. The ATPPS algorithm (green curve) achieves even faster MSE reduction than PPS. As the system nears equilibrium, ATPPS reduces redundant exchanges, causing the red curve to stagnate after a certain point in time. In summary, DAB underperforms due to limitations on the amount of load transfers; PPS improves upon this but lacks adaptive control, while ATPPS optimally balances load while avoiding unnecessary exchanges, making it the most effective approach in this setting.

Figure 16 shows the exponential regression fit for the MSE data when the DAB algorithm as a load balancing algorithm is applied to the network. Even though the curve might not suggest an exponential decay, the MSE data fits with the exponential regression model following the equation $MSE_r = 844.63 * e^{-0.01*r}$. The decay rate of -0.01 suggests a very slow decrease in MSE. The fitted curve and model seem very suitable for the MSE data, since the fitted curve aligns with the MSE data. The MSE data of the PPS is fitted to the exponential regression model for rounds

10 to 80. Over time, repeated randomized exchanges smooth out load imbalances, leading to an exponential decay in MSE, as seen in figure 17. The best fit follows the equation $MSE_r = 2530.41 * e^{-0.9*r}$. In Figure 18, the exponential regression fit is visualized for the MSE data of the ATPPS load balancing algorithm graph in the complete graph for rounds 10 to 65. The fitted curve is expressed by the equation $MSE_r = 4309.94 * e^{-1.06*r}$. A steep error reduction is indicated by the decay rate of -1.06. As the PPS algorithm, the ATPPS algorithm reduces the error very effectively in an exponential manner. Rounds 66 to 100 show a plateauing of the MSE data. The adaptive mechanism provides a faster decline in error indicated 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 (rates of change) for the three load balancing algorithms across different regions of the graph. The values reflect how steeply the MSE decreases over rounds. PPS and ATPPS have steep negative slopes in the start region, rounds 1-10 with a value of -92, indicating rapid initial improvement. However, their slopes in the middle (rounds 11-65) and end regions (rounds 66-100) approach zero, suggesting a plateau in performance. DAB has shallower negative slopes overall, reflecting a more gradual and consistent error reduction across all regions. The general slopes (rounds 1-100) for PPS and ATPPS (-8.4) are much steeper than DAB (-4.4), suggesting that PPS and ATPPS converge faster on average. The end region is chosen as 66-100 to catch the trend of plateauing for the ATPPS and also for PPS, which is beginning to plateau in error reduction in later rounds (rounds 80-100). 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.

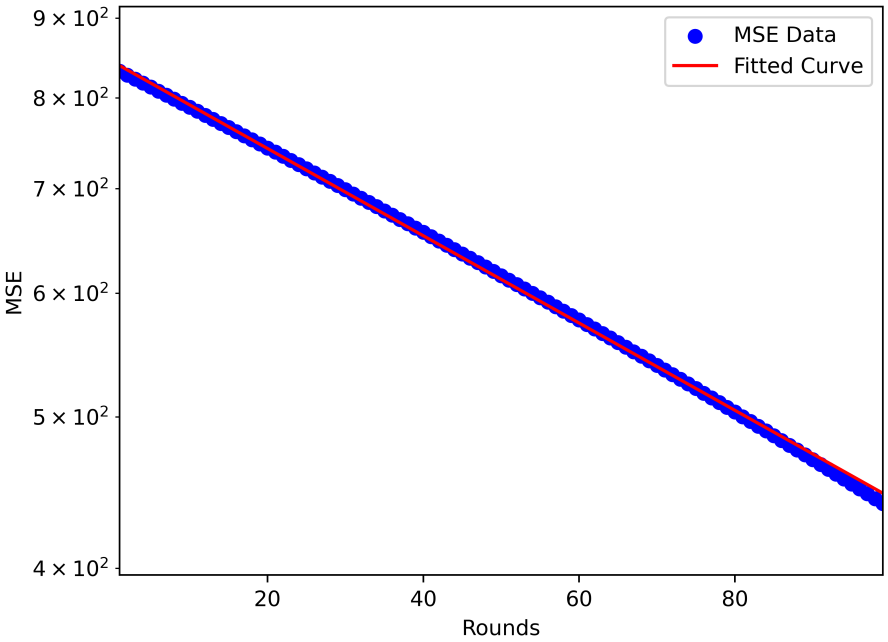


Figure 16: Complete graph - exponential regression fit: DAB

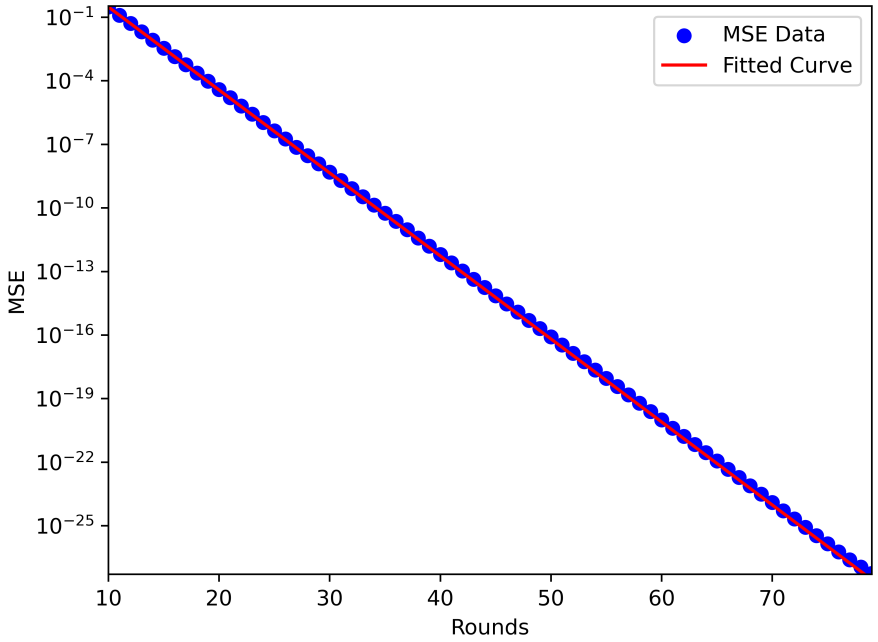


Figure 17: Complete graph - exponential regression fit: PPS

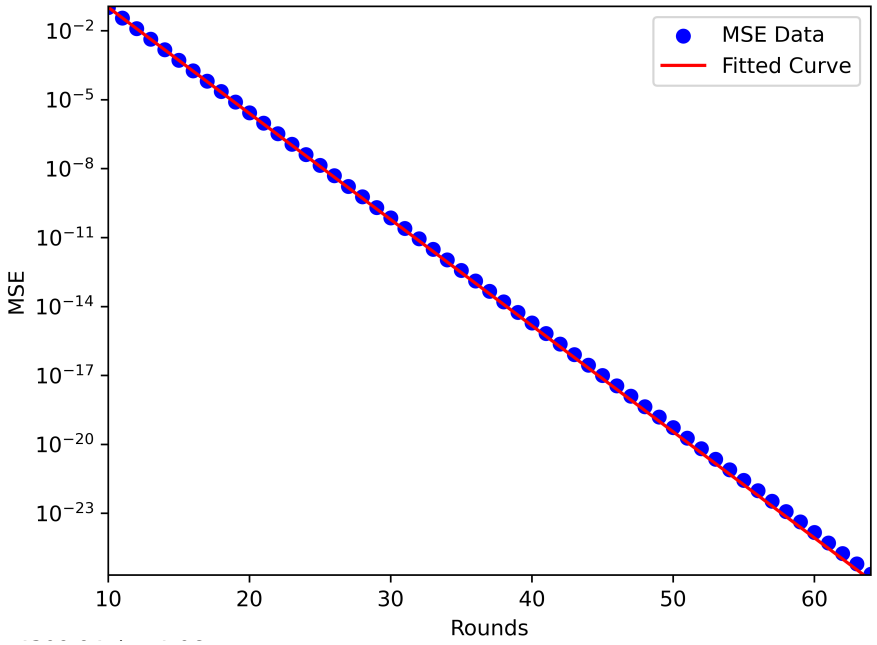


Figure 18: Complete graph - exponential regression fit: ATPPS

6.2 Star Graph

The behavior of the curves in figure 20 is similar to the ones for the Complete graphs, with the difference that the PPS curve falls more steeply than the PPS curve. Again, DAB shows a much slower MSE reduction compared to the Push-Pull Sum based algorithms, as indicated by the flatter slope of its curve. It converges minimally, with MSE remaining constant over rounds after an initial reduction. This indicates that DAB is less efficient in balancing load in a Star graph, likely due to its deterministic nature and lack of dynamic adaptability. In the Star graph all the leaves have the central node as a partner, and thus each node chooses the same node to propose load to. When the central node has a higher load than the leaf requesting a load transfer, than no load transfer is happening between these two nodes (bottleneck). While, both Push-Pull Sum based algorithms exhibit steep MSE reductions early on, as seen in the sharp downward trends in the log-log plot. The PPS algorithm outperforms the ATPPS algorithm, as it reduces MSE faster and reaches lower MSE values sooner

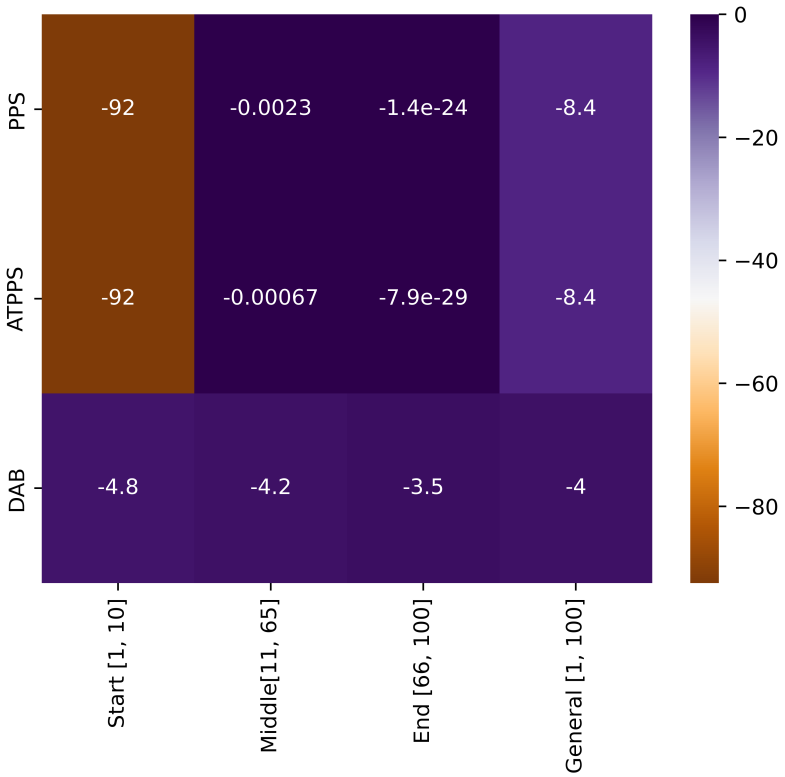


Figure 19: Complete graph: heat map of slopes per region

(as of round 45) before the graph plateaus. The Push-Pull Sum based algorithms draw an advantage of the Star graph, as the redistribution happens through the central node, which is chosen by every leaf node as push destination. Following that the central node redistributes parts of the load to the leaf nodes in magnitude of $\frac{\frac{s_i,r}{2}}{N-1}$.

As seen in figure 21 the MSE data for the DAB balanced network aligns nearly perfectly with the fitted curve of the exponential regression model with equation $MSE_r = 840.42 * e^{-0.01*r}$ for the rounds 1 to 100, eventhough the decay rate of -0.01 indicates a very slow decay. From round to round the improvement in the network is significantly minimal and thus highlights the inability of the DAB to balance the network within 100 rounds to a satisfying magnitude. Figures 22 and 23 show the fitted curves for the MSE data of the PPS and ATPPS respectively. The best-fit model for the MSE data of the PPS load balancing algorithm for the rounds 10 to 45 follows the equation $MSE_r = 29794.60 * e^{-1.39*r}$. The rounds 10 to 45 show the steepest decay and for that reason been fitted to exponential regression model. The decay rate of -1.39 indicates a fast error reduction in the network, especially if compared to the DAB. The ATPPS MSE data fits best with the exponential model following the equation $MSE_r = 9329.40 * e^{-1.05*r}$ between the rounds 18 to 60. In Star graphs, the central node dominates communication, and the load balancing heavily depends on that node. Threshold adjustments do not sufficiently impact performance under these conditions.

Overall the scenario is similiar to the Complete graph, where the DAB reduces the error exponentially with a very slow rate, while the Push-Pull Sum based algorithms perform a very fast error reduction. This behaviour is captured in the heat map of figure 24. While the DAB reduces the error in average per region -3.75 ± 0.55, the Push-Pull Sum based algorithms exhibit a error reduction of -92 for the first 10 rounds. The slopes approach zero in the middle and end regions, implying almost negligible MSE reduction in later stages. This is due to the fact that the network

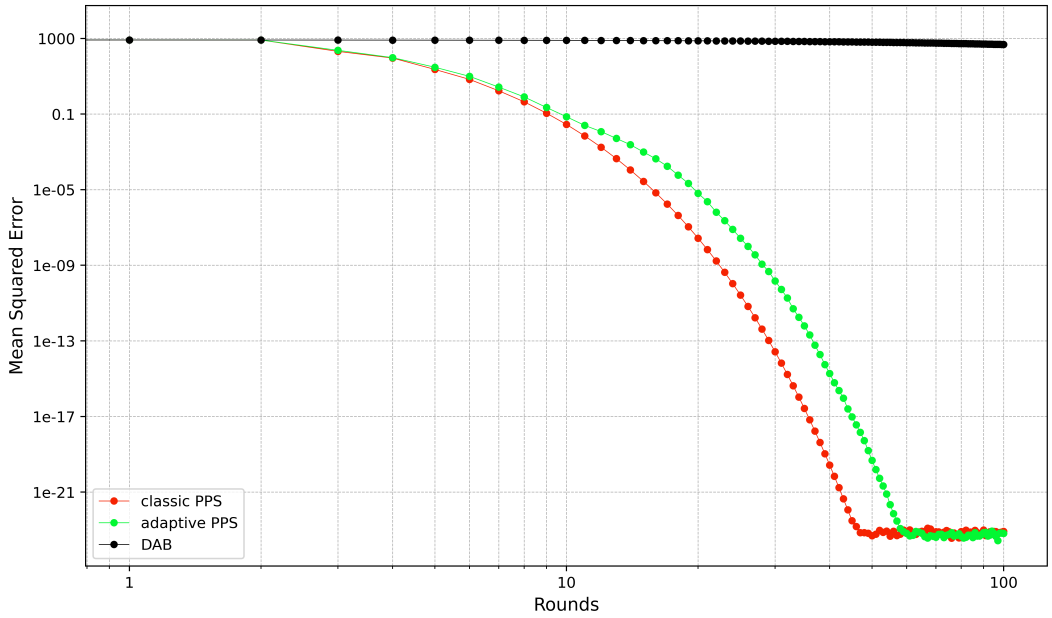


Figure 20: Star graph: mean squared error per rounds (log-log)

already is balanced to a degree where the MSE will not improve significantly further for later rounds. Both algorithms converge early and maintain a near-steady state afterward. The DAB's deterministic nature and lack of adaptability lead to slower convergence and less effective load balancing, as reflected in its consistently shallow slopes. The MSE reduced from 832 initially to a value of ~ 480.47 8.3×10^{-24} for the PPS and $\sim 6.5 \times 10^{-24}$ for the ATPPS.

6.3 Ring Graph

The DAB curve in figure 25 shows the steepest decline in error in the first few rounds of the simulation. The slope of the first 10 rounds is -84 for the DAB curve compared to -82 for each Push-Pull Sum based algorithm as depicted in figure 29. The near-overlap of PPS and ATPPS across the 100 rounds of simulation indicates that the adaptive mechanism provides limited additional benefit in a Ring graph, where choosing a random neighbor might already target the optimal load transfer

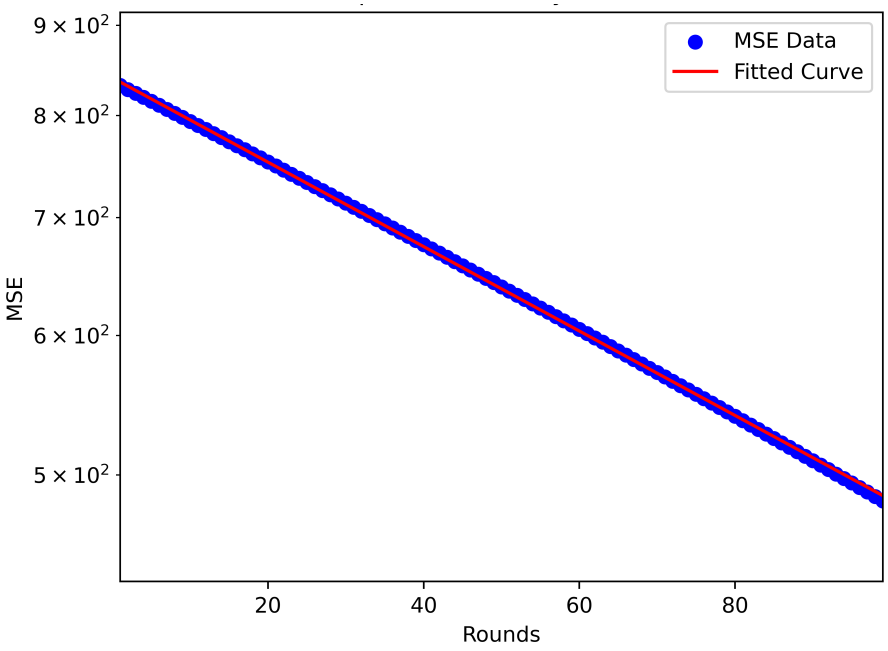


Figure 21: Star graph - exponential regression fit: DAB

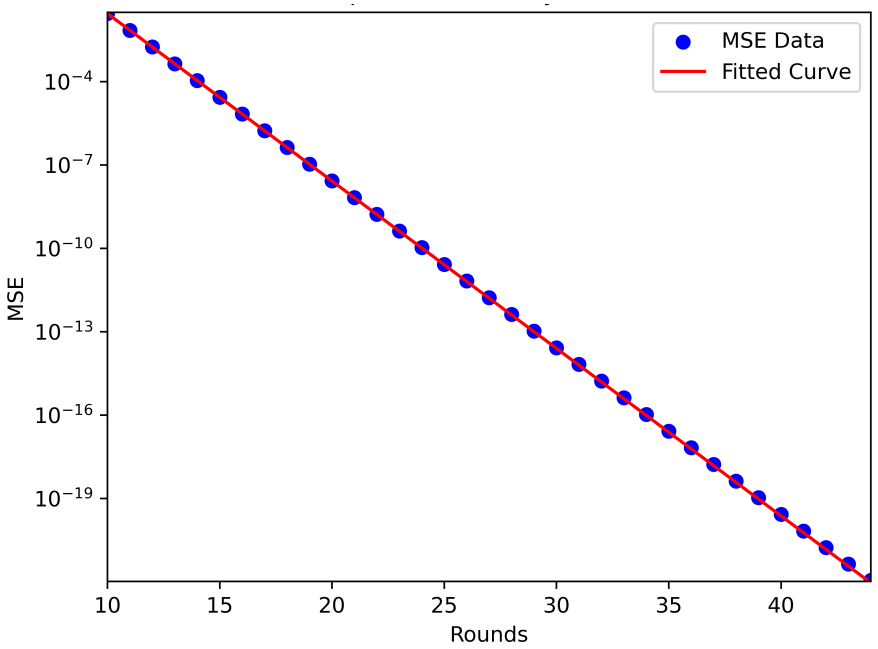


Figure 22: Star graph - exponential regression fit: PPS

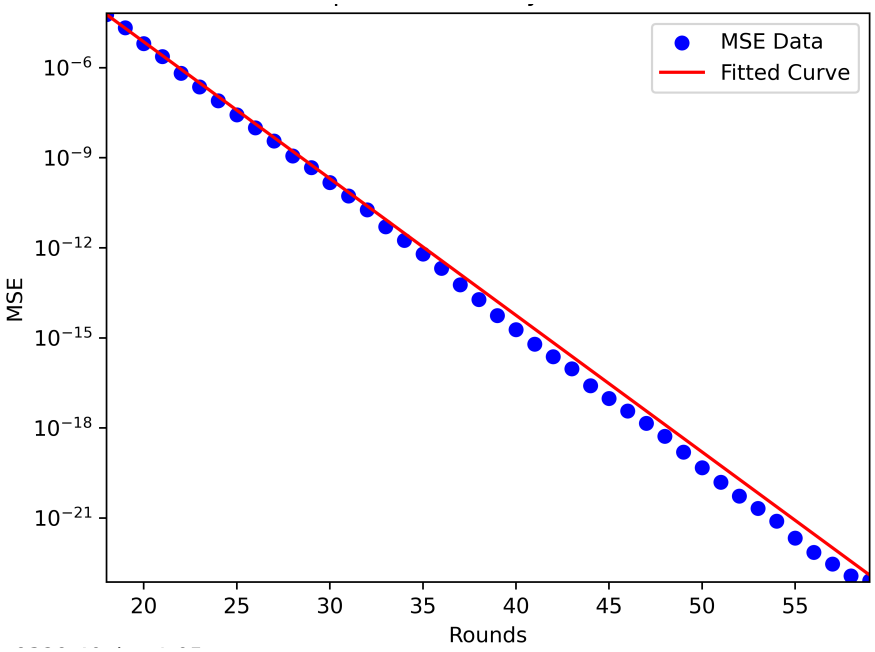


Figure 23: Star graph - exponential regression fit: ATPPS

partner. In the sense that in a Ring topology, each node only has access to its two neighbors. The adaptive threshold mechanism relies on meaningful differences in load between nodes to trigger exchanges, but in a Ring graph, local differences might not vary significantly enough to make the threshold mechanism advantageous for each neighborhood. For the PPS, every node communicates with its neighbors in every round. This constant exchange ensures rapid propagation of load updates. The adaptive threshold might reduce some of these exchanges, but in a Ring graph, where the diameter is large, reducing communication might delay convergence rather than improve it. Thus, the threshold mechanism could counteract its own benefits, limiting the advantage of the ATPPS over the PPS. The DAB functions way better in this scenario, since it always interacts with the optimal partner to exchange loads. The randomness benefit vanishes for a network topology where each node only has two neighbors, and a deterministic approach actually performs better.

The MSE data of each algorithms simulations were fitted to the polynomial regression model with degree of 4 each. The best-fit model for the DAB MSE data for the rounds

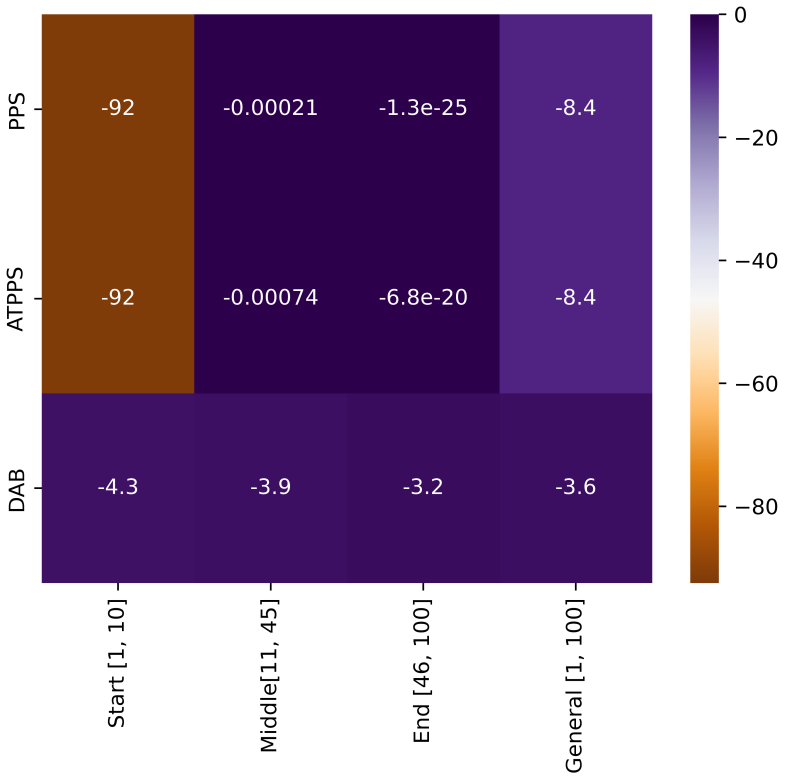


Figure 24: Star graph: heat map of slopes per region

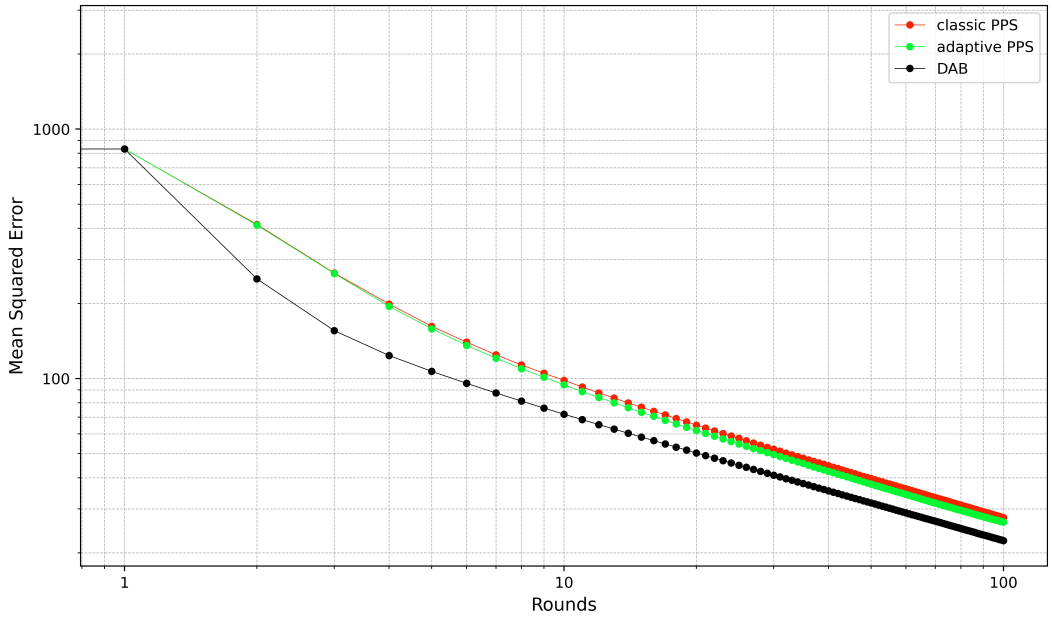


Figure 25: Ring graph: mean squared error per rounds (log-log)

10 to 60 follow the equation: $MSE_r = 1.72 \times 10^{-5} r^4 - 2.30 \times 10^{-3} r^3 + 0.19 r^2 - 5.99 r + 114.83$ (figure 26). A similiar equation is fitted for the PPS and ATPPS curves: $MSE_r = 2.99 \times 10^{-5} r^4 - 0.5 \times 10^{-2} r^3 + 0.32 r^2 - 9.68 r + 166.30$ (figure 27) for the PPS, and for the ATPPS: $MSE_r = 3.04 \times 10^{-5} r^4 - 0.5 \times 10^{-2} r^3 + 0.32 r^2 - 9.64 r + 161.86$ (figure 28).

6.4 Torus Grid Graph

Figure 30 shows the MSE reduction over rounds on a Torus Grid graph, plotted on a log-log scale. The DAB curve appears to have a slightly faster initial reduction compared to PPS' and ATPPS' curve in the beginning (rounds 1 to 7). The slope in this region is superior for the DAB algorithm with a value of -140 compared to -130 for the Push-Pull Sum based algorithms (figure 34). The PPS curve and ATPPS curve maintain nearly identical performances during the middle region (rounds 8 to 40), reducing MSE at a similar rate, with a slope of -1 for the PPS curve and

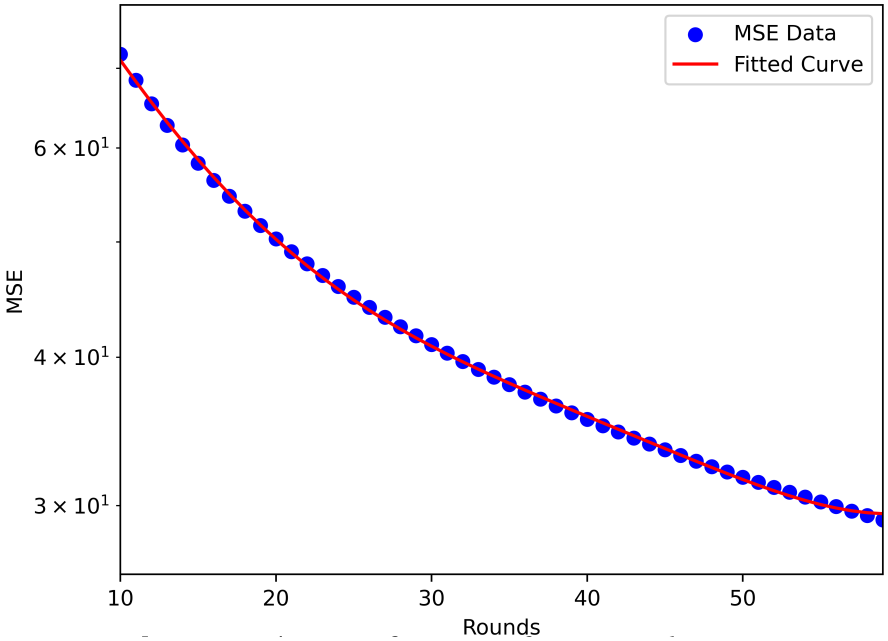


Figure 26: Ring graph - polynomial regression fit: DAB

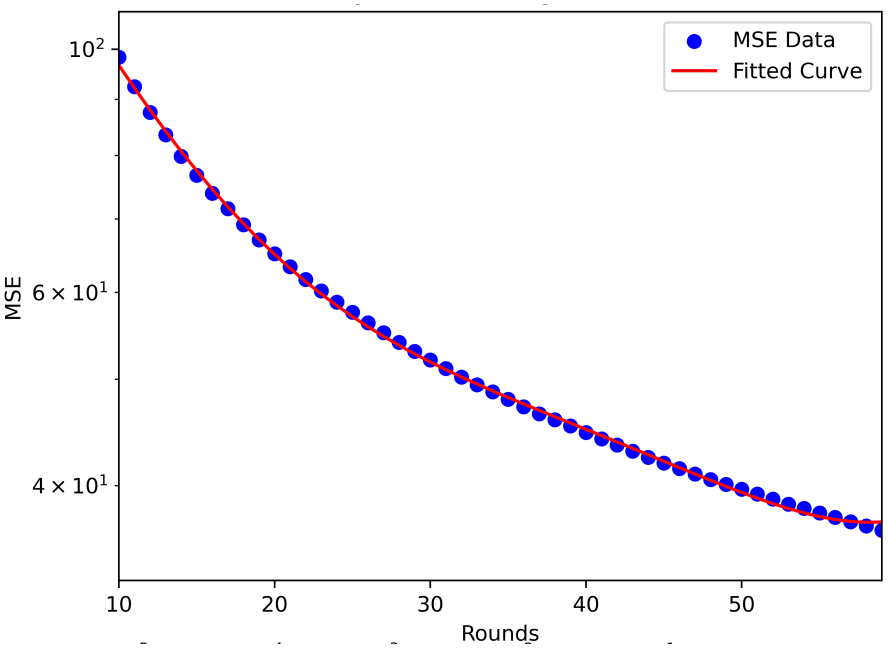


Figure 27: Ring graph - polynomial regression fit: PPS

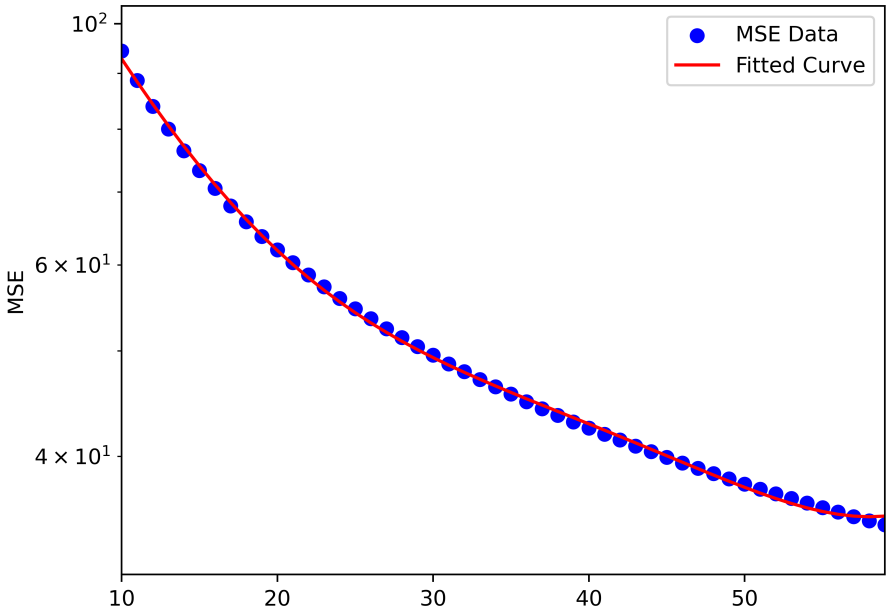


Figure 28: Ring graph - polynomial regression fit: ATPPS

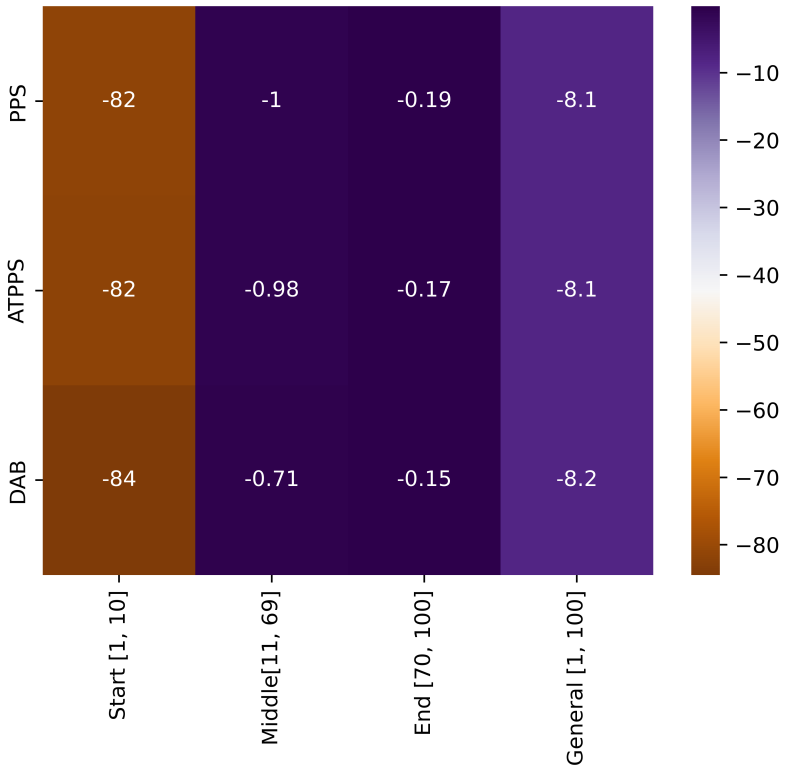


Figure 29: Ring graph: heat map of slopes per region

-0.83 for the ATPPS curve. DAB shows a noticeable improvement of reducing the imbalance compared to both, PPS and ATPPS, achieving lower MSE values consistently. This suggests that DAB adapts more efficiently to the graph's structure during the intermediate (rounds in the middle) rounds. DAB continues to reduce MSE more effectively than the Push-Pull Sum based protocols in the end region (rounds 41 to 100). PPS and ATPPS exhibit convergence, but they lag behind the DAB algorithm in reaching minimal MSE. A Torus grid has more structured connectivity compared to a Star or Ring graph, allowing better distribution of loads via localized interactions. DAB's deterministic approach benefits from leveraging this regularity, leading to its superior performance. The ATPPS algorithm achieves a compromise solution in this scenario. It performs better than the PPS approach judging from the simulation outcomes, especially in later rounds, where the adaptiveness condition prevents "redundant" load transfers from happening and prioritizing load transfers that reduce the error impactfully.

The uniform neighborhood structure of Tori ensure that DAB's deterministic decisions (e.g., always choosing the minimal neighbor) are consistently effective across the graph. The algorithm does not suffer from random suboptimal decisions introduced by probabilistic neighbor choices, making 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), it is less effective in structured topologies compared to the DAB like a 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, requiring 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 as trades between two nodes with higher load differences are more impactful once the network is already heavily balanced.

The fitted polynomial curve of degree 5 matches the MSE data for DAB effectively, capturing the non-linear dynamics during rounds 10 to 39 following the equation: $MSE_r = -1.35 \times 10^{-6}r^5 + 1.89 \times 10^{-4}r^4 - 0.01r^3 + 0.30r^2 - 4.6r + 34.10$ (figure 31 a)). This suggests that in the early rounds (rounds 10 to 39), the MSE reduction follows a complex pattern due to the DAB's optimization mechanism, such as its focus on minimizing neighbors loads dynamically. The higher degree captures these dynamics with more precision than simpler models. In later rounds the performance of the DAB can still be captured by a polynomial curve of degree 3 with equation: $MSE_r = -6.01 \times 10^{-06}r^3 + 1.66 \times 10^{-3}r^2 - 0.16r + 6$ (figure 31 b)). By this stage, the load balancing has stabilized, and the reduction in MSE is more linear or gradual. A lower-degree polynomial suffices for rounds 40 to 100 since the dynamics are simpler compared to the earlier rounds. The curve behavior for the PPS and ATPPS are similar to that of the DAB curve expressed for rounds 10 to 39 by the equations: $MSE_r = -5.54 \times 10^{-6}r^5 + 7.65 \times 10^{-4}r^4 - 0.04r^3 + 1.16r^2 - 16.81r + 112.86$ for the PPS (figure 32 a)) and: $MSE_r = -3.65 \times 10^{-6}r^5 + 5.16 \times 10^{-4}r^4 - 0.03r^3 + 0.83r^2 - 12.52r + 88.16$ for the ATPPS (figure 33 a)). The equations that describe the behavior of the two Push-Pull Sum based algorithms are fitted for the rounds 40 to 100 for the PPS is: $MSE_r = -1.15 \times 10^{-5}r^3 + 3.205 \times 10^{-3}r^2 - 0.33r + 13.72$ (figure 32 b))and for the ATPPS is: $MSE_r = -9.99 \times 10^{-6}r^3 + 2.8034 \times 10^{-3}r^2 - 0.28r + 11.29$ (figure 33 b)).

6.5 Lollipop Graph

6.5.1 (512, 512) Lollipop Graph

Figure 35 shows the three curves of the load balancing algorithms. The ATPPS slightly outperforms the PPS. The DAB lacks to perform as good as the other two load balancing algorithms in reducing the error. The superior performance of PPS and ATPPS compared to DAB in the Lollipop graph can be explained by the

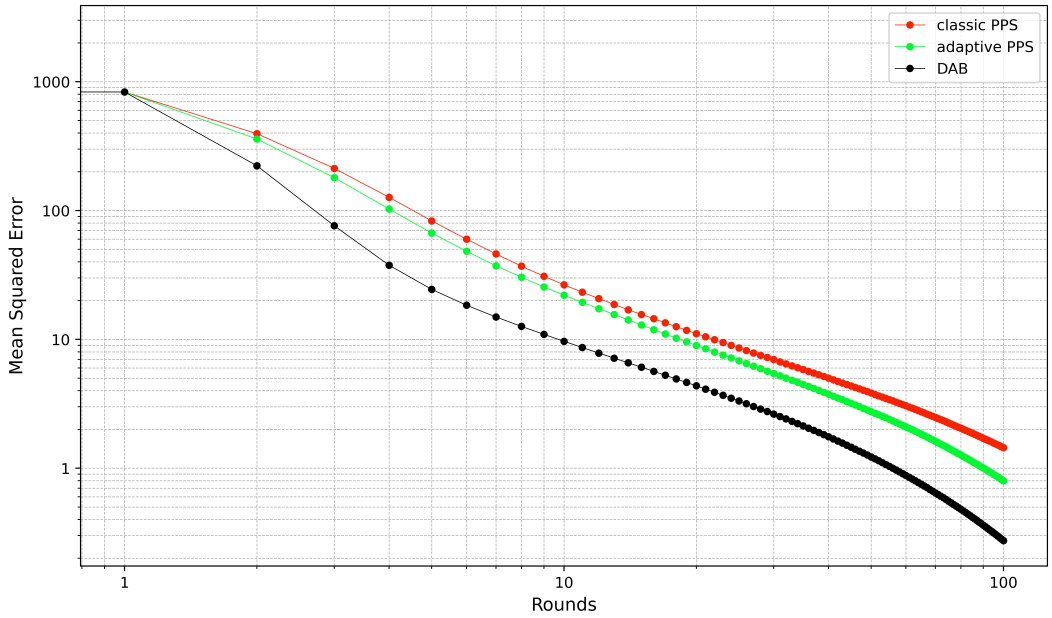


Figure 30: Torus Grid: mean squared error per rounds (log-log)

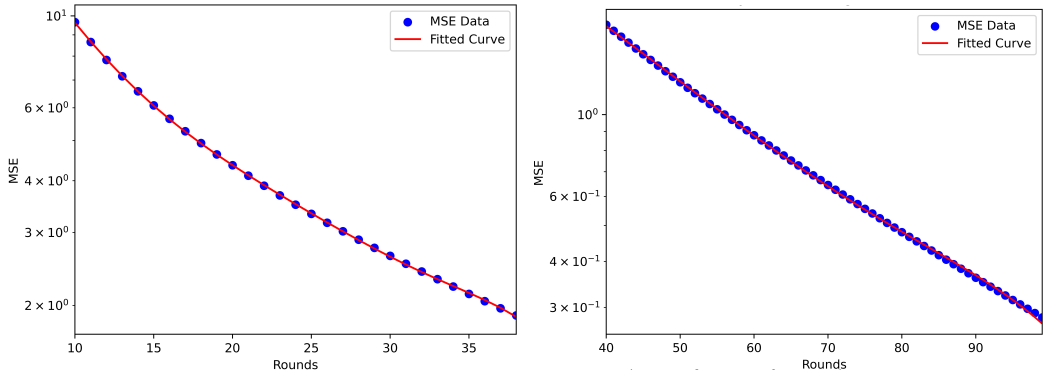


Figure 31: Torus Grid - polynomial regression fit: DAB; rounds 10-39 and 40-100

structure of the graph and the fundamental differences in how these algorithms operate. The clique region consists of high connectivity and enables rapid local balancing for the Push-Pull Sum based algorithms. While the path region consists of limited connectivity and slows down information propagation and balancing for the Push-Pull Sum based algorithms and favors the deterministic algorithm DAB. The initial discrepancy in the first 10 rounds is due to the potentialy fast decrease

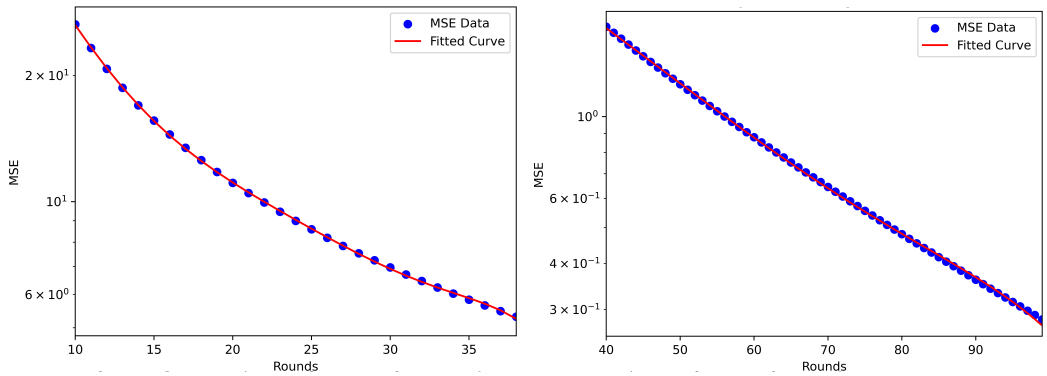


Figure 32: Torus Grid - polynomial regression fit: PPS; rounds 10-39 and 40-100

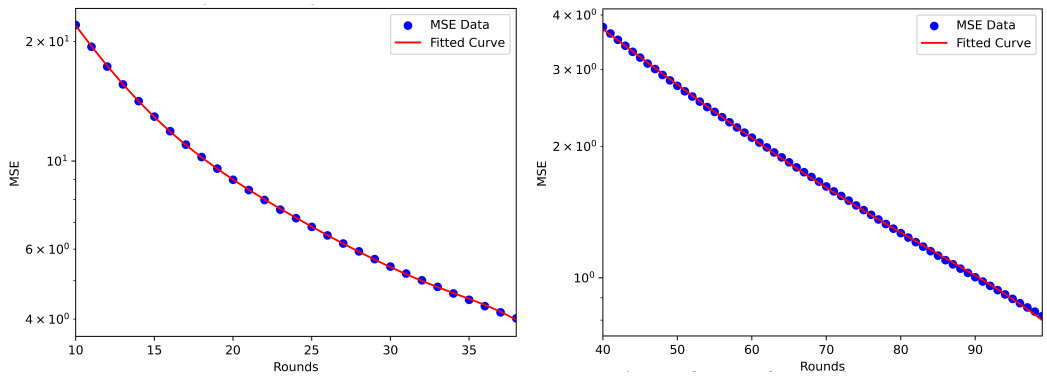


Figure 33: Torus Grid - polynomial regression fit: ATPPS; rounds 10-39 and 40-100

of error for the PPS in cliques. The DAB struggles to perform good 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. It prioritizes nodes with the least load, which can lead to inefficient propagation along the clique. Load imbalances in the clique region take longer to converge because DAB does not exploit the random spreading mechanism of Push-Pull Sum based algorithms. In the initial phase PPS rapidly reduces the MSE, demonstrating strong initial convergence. The Push-Pull Sum based algorithms achieve a steep downwards slope with value -130 between rounds

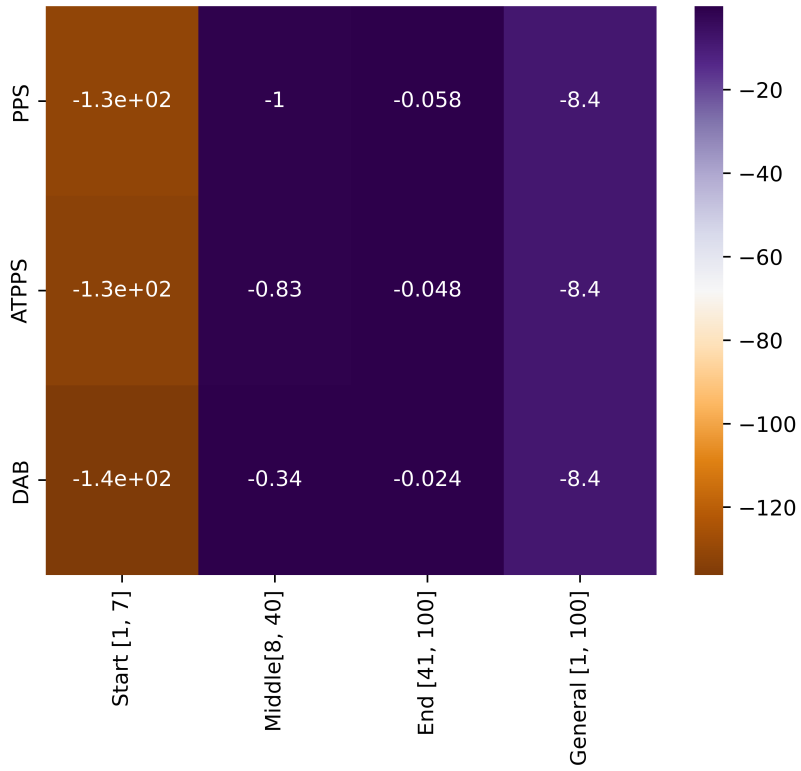


Figure 34: Torus Grid: heat map of slopes per region

1 to 7, while the DAB reaches nearly half of it with value -67 (figure 39) In the mid-to-late phase the slope flattens for both of the Push-Pull Sum based algorithms, indicating slower convergence this is also indicated by the slopes droping to as low as -0.12 to ~ -0.9. In this phase, the path section of the Lollipop graph dominates the 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 surpassed 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 path section of the Lollipop graph as showcased in section 6.3.

The DAB model fit follows the equation: $MSE_r = -5.89 \times 10^{-5} r^3 + 0.03r^2 - 5.68r + 459.42$ (figure 36), which is a 4-th degree polynomial equation. The PPS and ATPPS are a bit more complex and expressed by a polynomial equation of degree 5. The PPS model fit follows the equation: $MSE_r = 8.44 \times 10^{-7} r^4 - 2.52 \times 10^{-4} r^3 - 0.03r^2 - 1.64r + 56.68$ (figure 37) and the ATPPS model fit follows the equation: $MSE_r = 8.69 \times 10^{-7} r^4 - 2.56 \times 10^{-4} r^3 + 0.03r^2 - 1.62r + 54.48$ (figure 38).

To see the impact of the pathsize and cliquesize on the simulation results respectively, the simulations are conducted with less nodes assigned to the clique in subsection 6.5.2. The network sizes remains the same. The nodes assigned to the cliques are cut to 128 which is one fourth of the previous value of 512. Thus 896 nodes are assigned to the path size. And similarly the simulations are conducted with less nodes assigned to the path in subsection 6.5.3.

6.5.2 (128, 896) Lollipop Graph

In the initial region, the first 7 rounds, PPS and ATPPS start with a steep decrease in MSE with a slope of -86 as depicted in figure 44. DAB, on the other hand, has

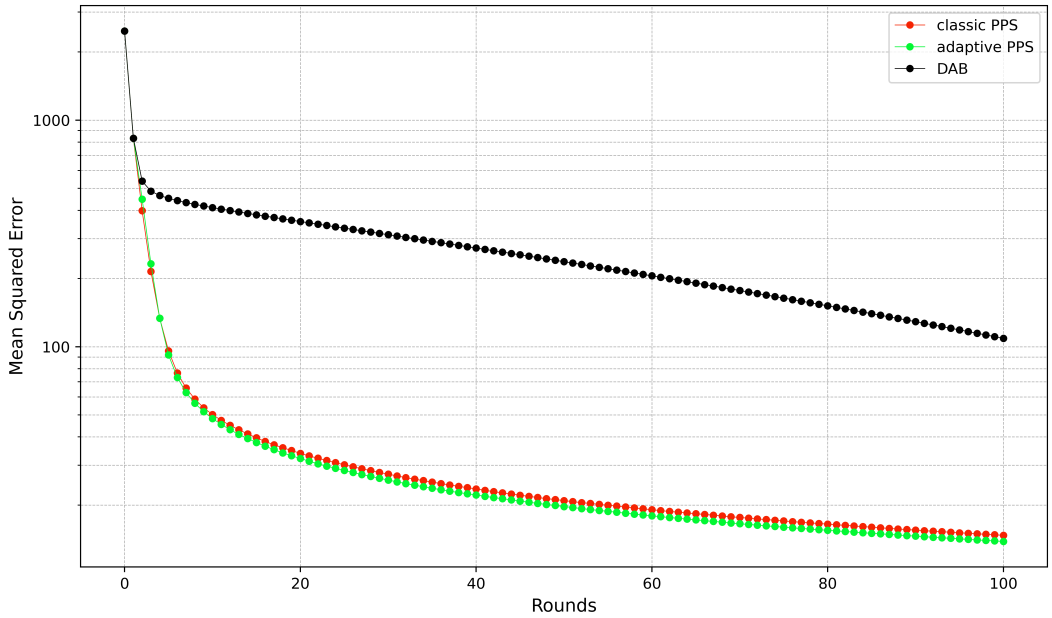


Figure 35: (512, 512)-Lollipop graph: mean squared error per rounds (log-linear)

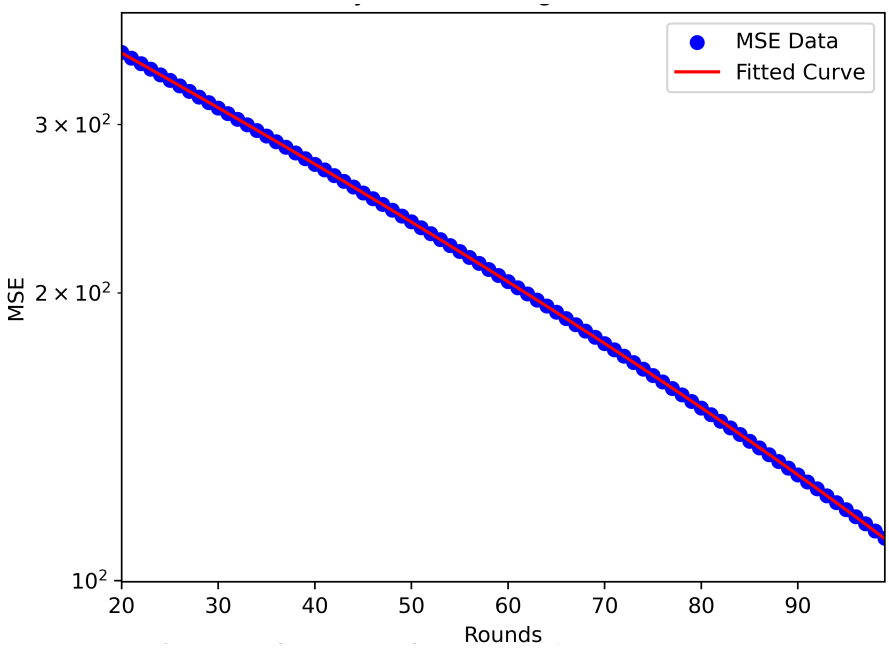


Figure 36: (512, 512)-Lollipop graph - polynomial regression fit: DAB

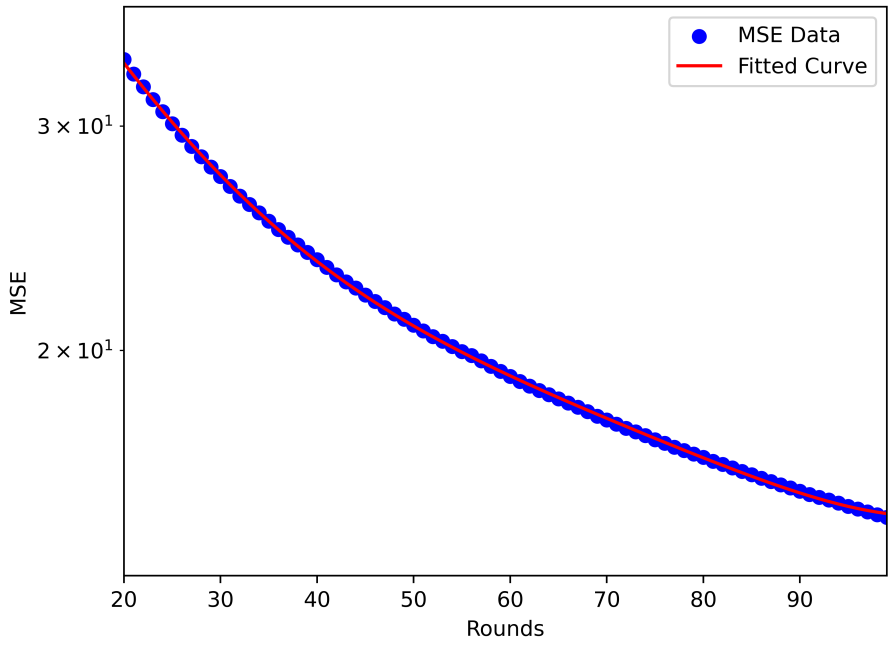


Figure 37: (512, 512)-Lollipop graph - polynomial regression fit: PPS

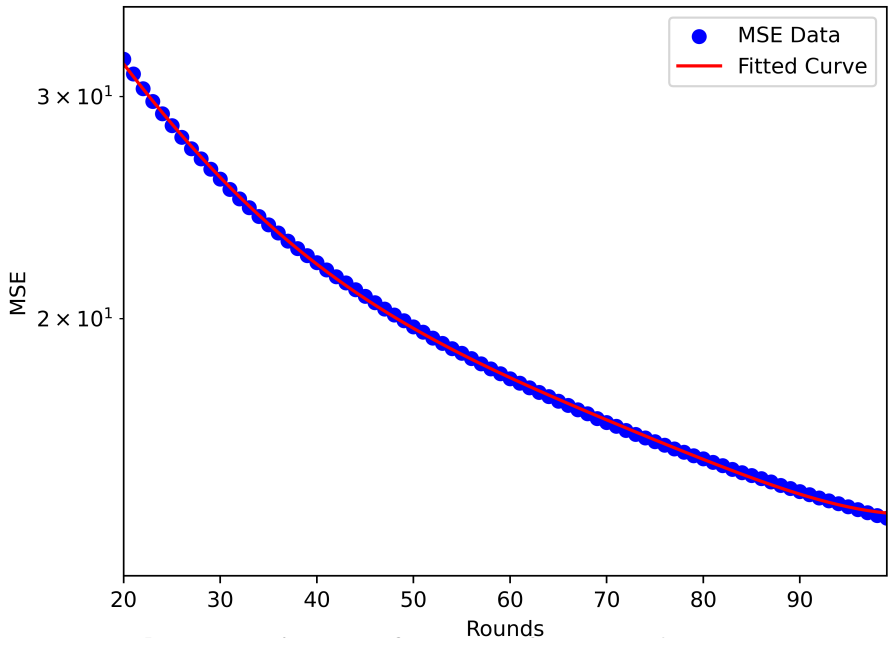


Figure 38: (512, 512)-Lollipop graph - polynomial regression fit: ATPPS

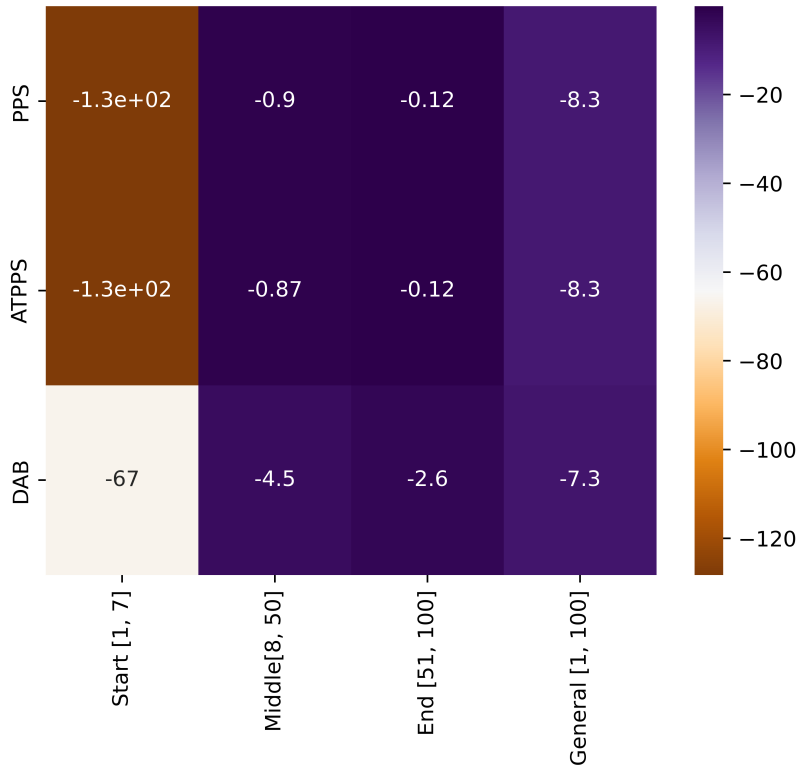


Figure 39: Lollipop graph: heat map of slopes per region

a slightly slower initial convergence compared to PPS showcasing a slope of -110. Compared to the slopes of the previous experiment where the path size and the clique size where equal, the convergece in the first 7 rounds improved for the DAB, achieving a steeper downwards slope. The slope in this region in the previous experiment was -68 and improved to -110. In the middle region, rounds 8 to 50, the ATPPS shows a performance close to that of the PPS where both curves maintain a consistently steep slope. In this region the DAB shows to have the steepest slope decreasing the error by -2.8 on average per round. The PPS and ATPPS already show low MSE values beforehand due to the steep decrease of error in the initial phase, thus only reduce the error by around ~ -1.5 per round on average. In the end region the DAB shows a slightly sharper decline in MSE compared to earlier rounds. This could be due to the longer path structure of the graph: once the majority of the load has propagated through the clique, the balancing process accelerates. Both Push-Pull Sum variants continue to outperform DAB, achieving lower MSE values in fewer rounds due to their clique-centric structure, which more effectively balances loads in the smaller clique. Interestingly the PPS and ATPPS slightly diverge between rounds 10 and 90 and finaly interesect (or nearly) in round 100.

The MSE data of all three load balancing algorithms fit a equation of the polynomial regression model. The MSE data of the DAB and ATPPS are fitted to a polynomial of degree 4 each, while the PPS MSE data seems to be a bit more complex following a polynomial of degree 5. The fitted model of the DAB MSE data follows the equation: $MSE_r = 3.46 \times 10^{-6} r^4 - 1.12 \times 10^{-3} r^3 + 0.14 r^2 - 7.69 r + 190.78$ (figure 41) and the one of the ATPPS follows the equation $MSE_r = 1.59 \times 10^{-6} r^4 - 4.74 \times 10^{-4} r^3 + 0.054 r^2 - 2.94 r + 70.59$ (figure 42). The polynomial fitted to the PPS MSE data follows the equation: $MSE_r = -3.72 \times 10^{-8} r^5 + 1.31 \times 10^{-5} r^4 - 1.85 \times 10^{-3} r^3 + 0.13 r^2 - 4.96 r + 84.91$ (figure 43).

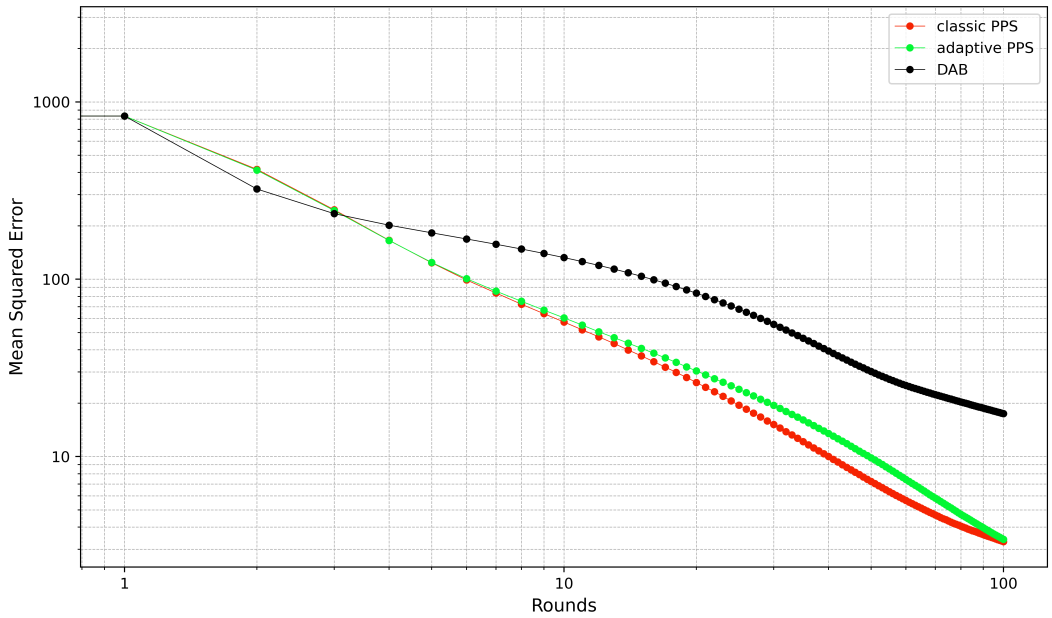


Figure 40: (128, 896)-Lollipop graph: mean squared error per rounds (log-log)

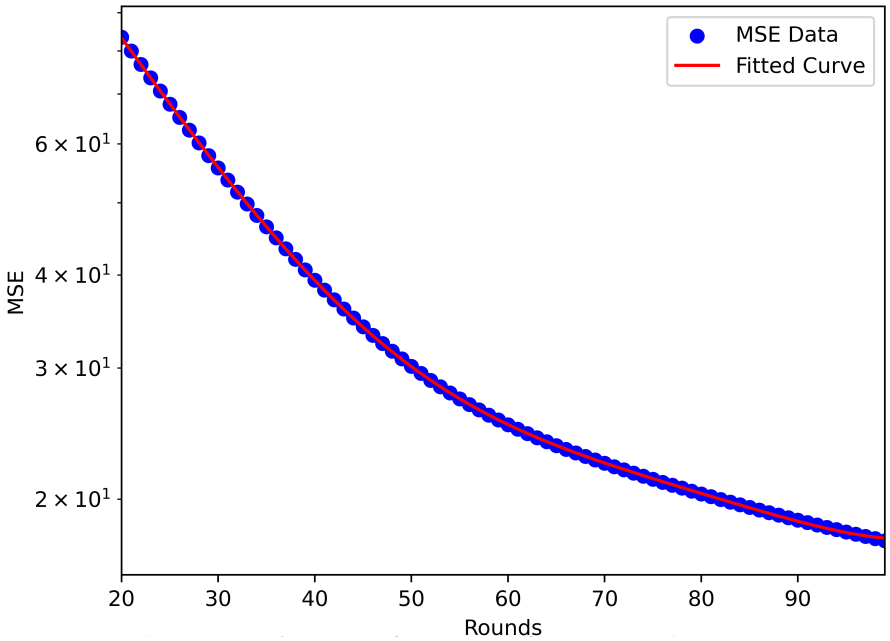


Figure 41: (128, 896)-Lollipop graph - polynomial regression fit: DAB

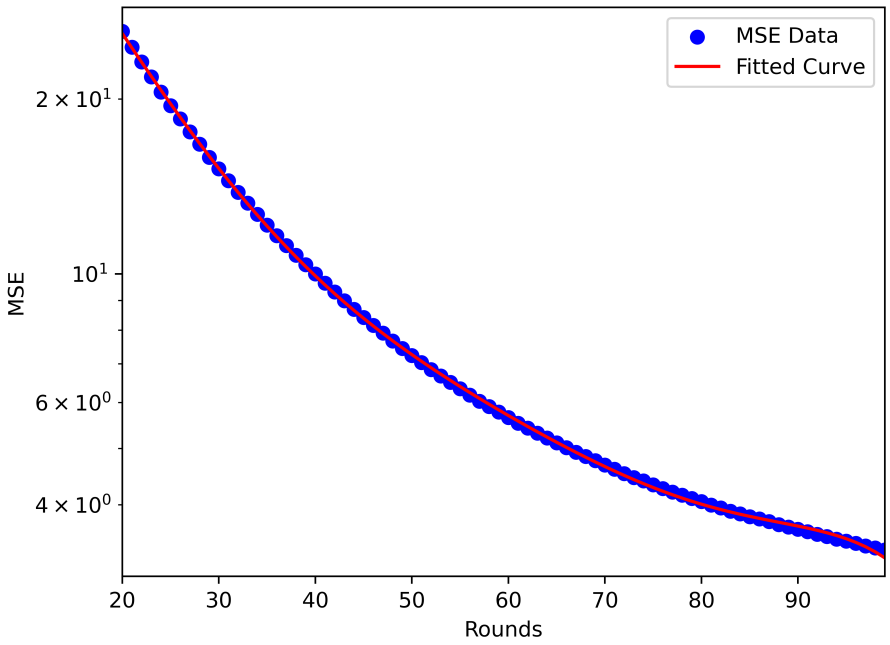


Figure 42: (128, 896)-Lollipop graph - polynomial regression fit: PPS

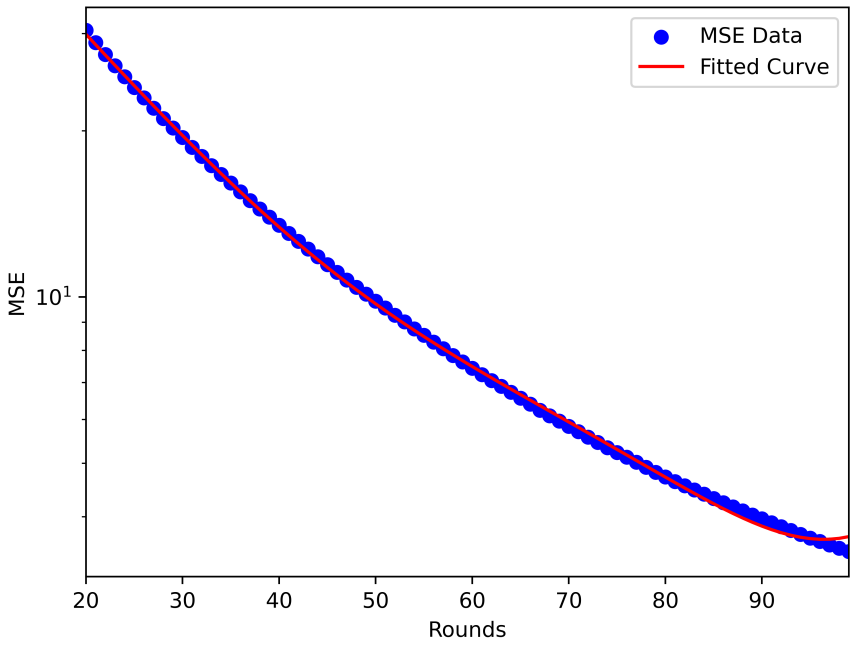


Figure 43: (128, 896)-Lollipop graph - polynomial regression fit: ATPPS

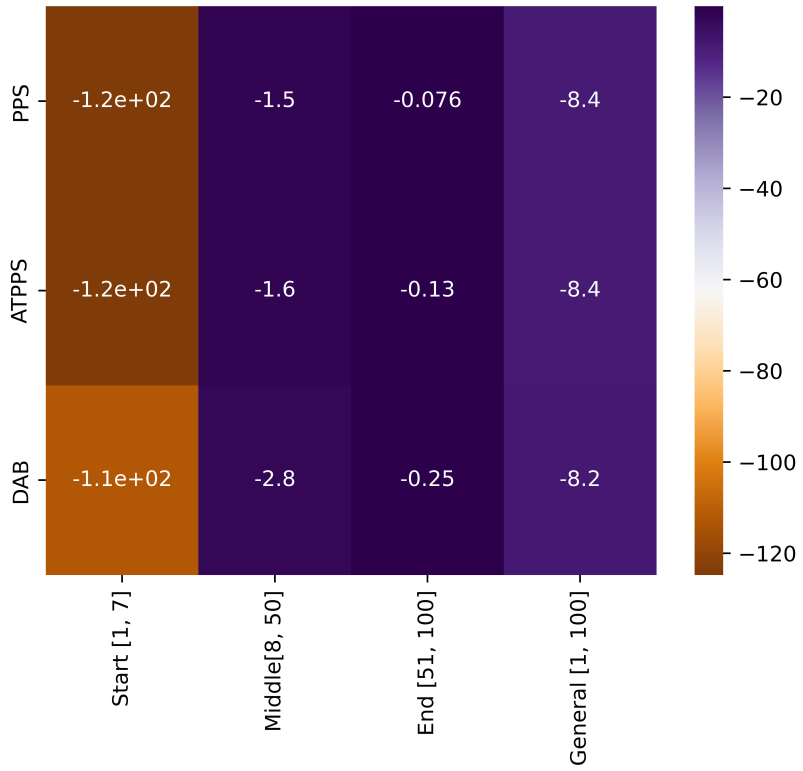


Figure 44: (128, 896)-Lollipop graph: heat map of slopes per region

6.5.3 (896, 128) Lollipop Graph

Now that more nodes are assigned to the clique and taken from the path compared to the initial experiment where the clique and the path had the same number of nodes, the discrepancy between the Push-Pull Sum based and the DAB algorithms is more obvious, as shown in figure 45. This is also indicated by the slopes. In the start region the Push-Pull Sum based algorithms both balance the network with the (896, 128)-Lollipop graph more quickly, achieving slopes of -140 in this initial region, compared to -130 (figure 49) in the (512, 512)-Lollipop graph. The overall MSE is also lower for the (896, 128)-Lollipop graph, where the error is reduced to a value of 3.27 for the network where the ATPPS is applied and 3.59 for the network where PPS is applied as a load balancing strategy. The MSE values for the (512, 512)-Lollipop graph are higher, where the MSE data of the ATPPS shows values of 13.83 and for the PPS 14.71. However, no significant advantage of the ATPPS over the PPS is observed. The DAB struggles to achieve good performance in reducing error, showcased by the immense discrepancy between the MSE values after 100 rounds. For the (512, 512)-Lollipop graph experiments the DAB achieved values of 108.90, in the experiment of (896, 128)-Lollipop graph the value that the network ends up with is at 542.09, which is nearly five times as much.

The best-fit polynomials fitted to the MSE data of the load balancing algorithms are of degree 3 for the DAB MSE data and of degree 4 for the PPS-based algorithms MSE data. The polynomial for the DAB MSE data follows the equation: $MSE_r = -1.936 \times 10^{-4} r^3 + 0.05 r^2 - 5.33 r + 745.95$ (figure 46). The polynomial for the Push-Pull Sum based algorithms follow the equation: $MSE_r = 2.00 \times 10^{-7} r^4 - 6.01 \times 10^{-5} r^3 + 6.95 \times 10^{-3} r^2 - 0.39 r + 13.44$ (figure 47) for the PPS and $MSE_r = 2.28 \times 10^{-7} r^4 - 6.77 \times 10^{-5} r^3 + 7.68 \times 10^{-3} r^2 - 0.42 r + 13.62$ for the ATPPS (figure 48).

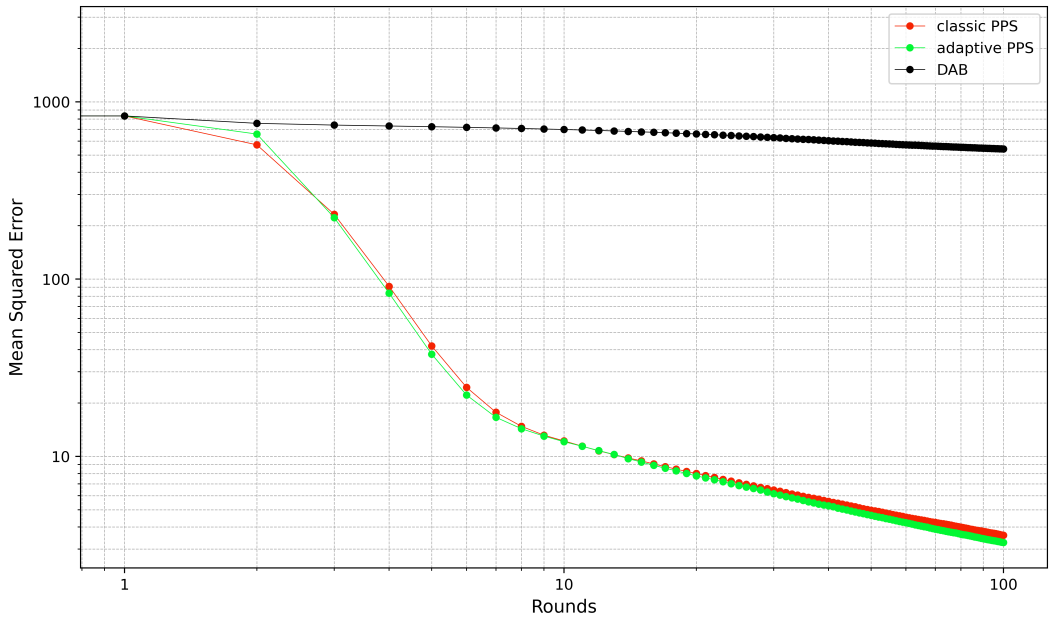


Figure 45: (896, 128)-Lollipop graph: mean squared error per rounds (log-log)

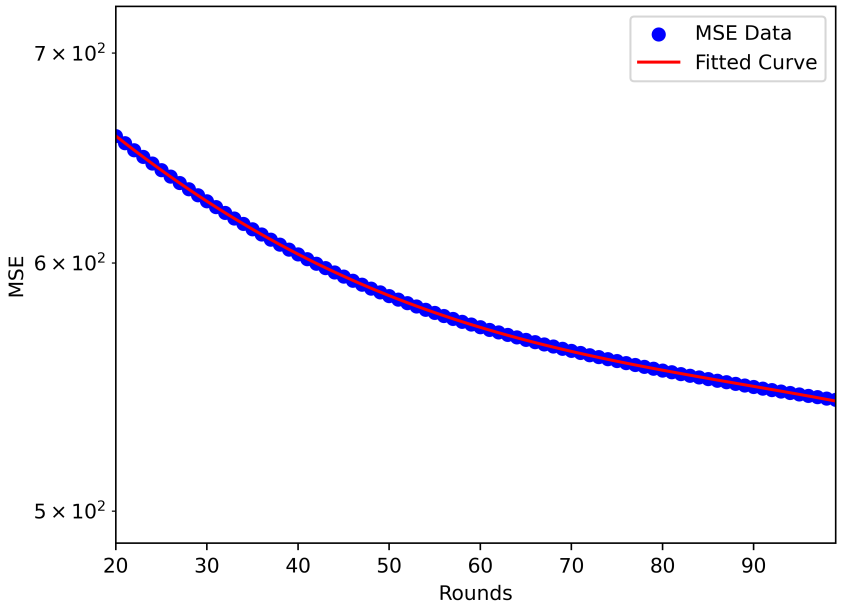


Figure 46: (896, 128)-Lollipop graph - polynomial regression fit: DAB

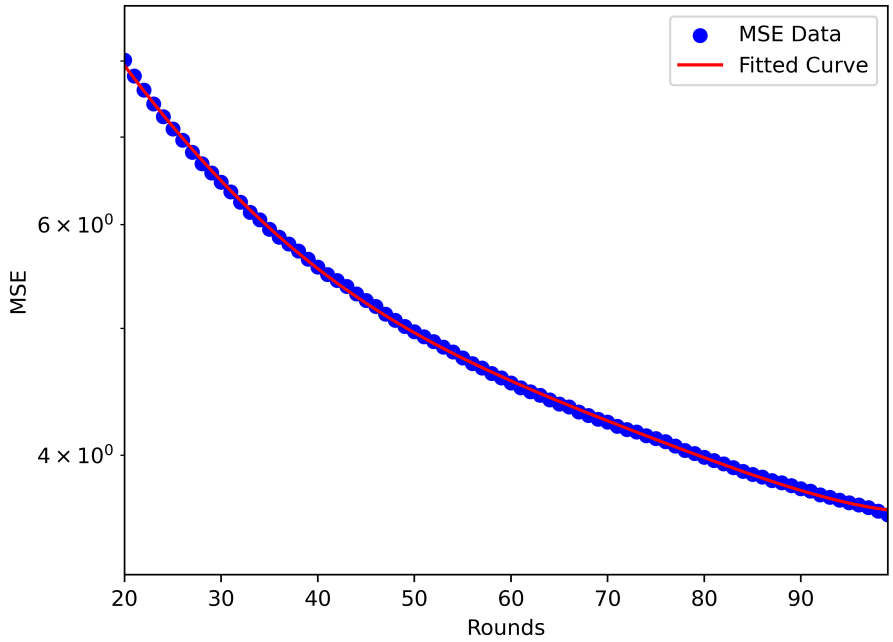


Figure 47: (896, 128)-Lollipop graph - polynomial regression fit: PPS

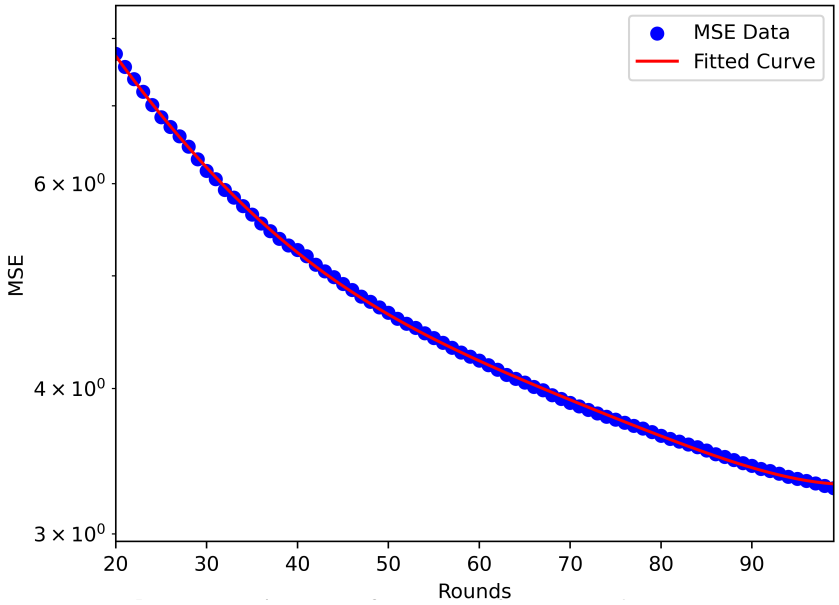


Figure 48: (896, 128)-Lollipop graph - polynomial regression fit: ATPPS

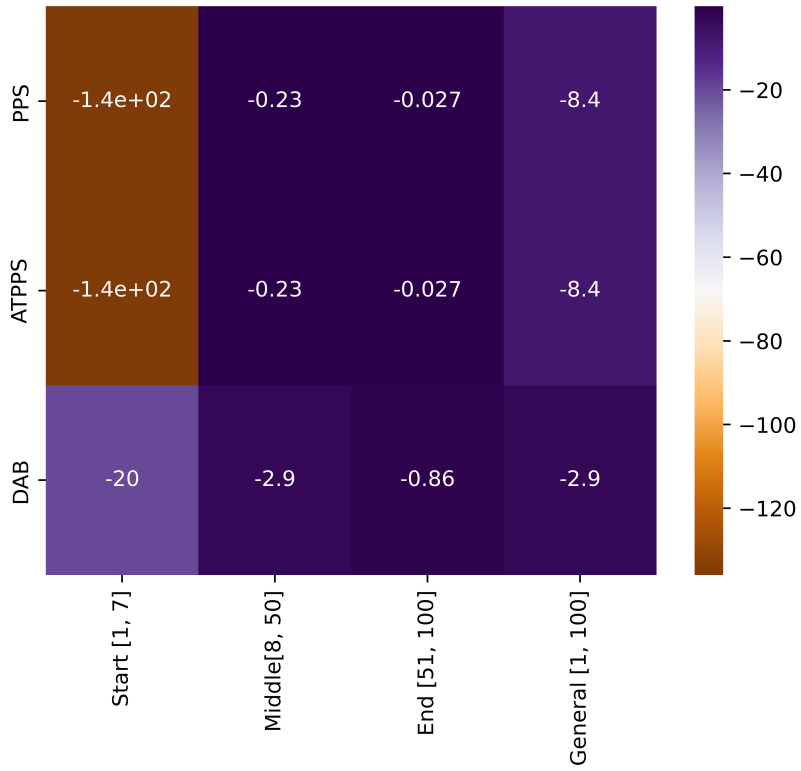


Figure 49: (896, 128) Lollipop Graph: heat map of slopes per region

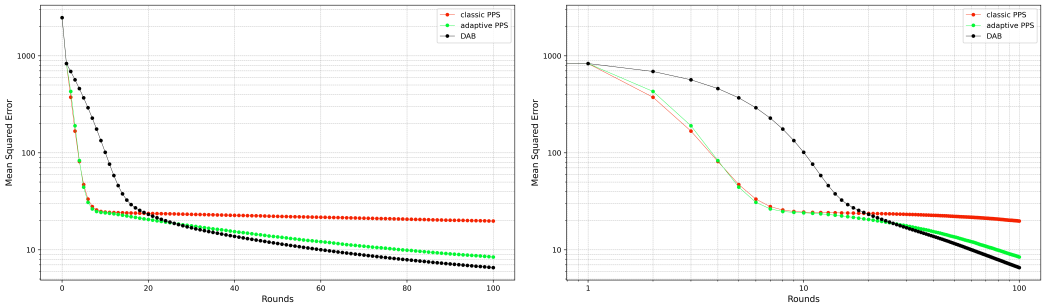
6.6 Ring of Cliques

6.6.1 32x32 Ring of Cliques

In the early rounds, PPS and ATPPS exhibit the fastest MSE reduction showcased by a very steep downwards trend between rounds 1 and 5 in figure 50 b). Both Push-Pull Sum based approaches start with faster convergence rates compared to DAB. The initial steep downwards slope of -200 for each Push-Pull Sum based algorithm as depicted in figure 54 within the first 5 rounds is due to the unbalanced state of each clique. The Push-Pull Sum based approaches have shown to be outperforming the DAB algorithm for Complete graphs. While the cliques are still unbalanced the Push-Pull Sum based algorithms achieve faster convergence. After the cliques are balanced round by round the DAB algorithm catches up especially between the rounds 6 to 40 where the slopes are -48 compared to ~ -2 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 for random neighbors is slower, creating bottlenecks for global convergence (this is especially the case for the PPS, whichs curve starts to stagnate between round 10 to 100). Due to the determistic nature of the DAB, the bridging nodes choose nodes outside of their cliques, once the load is balanced within the clique, and spread the load to other cliques. The ATPPS algorithm 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 mirrored 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 them of the DAB algorithm. Overall, at round 100 the DAB achieved a MSE of ~ 6.55, the PPS a MSE of 19.80 and the ATPPS a MSE of 8.42.

The polynomial fit for the DAB algorithm is: $MSE_r = 1.04 \times 10^{-6} r^4 - 2.941 \times 10^{-4} r^3 + 0.03 r^2 - 1.66 r + 45.30$ (figure 51). Thus the MSE as a function of rounds is modeled by a fourth degree polynomial. The PPS MSE data of rounds 20 to 100 is fitted to a linear regression model following the equation: $MSE_r = -0.05 r + 24.70$ (figure 52). The negative slope indicates a consistent reduction in MSE with each round in this region, but the value -0.05 is relatively small. This suggests that the PPS algorithm achieves slow and steady progress toward load balancing. The linear fit highlights that PPS achieves only gradual improvement in balancing the load in the Ring of Cliques topology. The reason for that is that PPS focuses on a push-pull mechanism that is, relying on random neighbor interactions. In the Ring of Cliques, inter-clique connections are sparse, and PPS lacks a mechanism to prioritize balancing between these sparsely connected regions. As a result, its performance is bottlenecked by the topology. The linearity of the model suggests a uniform rate of error reduction across rounds, without any acceleration observed in other methods (like DAB). The logarithmic model for the ATPPS algorithm is given as: $\log(MSE_r) = -7.59 \log(r) + 43.26$ (figure 53). By exponentiating this equation, the relationship between the MSE and the number of rounds is: $MSE_r = 10^{43.26} * r - 7.59$. The steep negative slope of value -7.59 in the log-log fit indicates a rapid decrease in MSE as the number of rounds increases. This suggests that ATPPS achieves exponentially faster convergence compared to PPS, especially in the early rounds of load balancing.

In the following the Ring of Cliques are newly organized. Experiments are conducted where the number of cliques is increased from 2^5 to 2^7 in subsection 6.6.2 and thus the clique size of each clique is decreased to 2^3. Subsection 6.6.3 covers the experiment where the clique size is increased and the number of cliques is decreased in the same magnitude as discussed before.



(a) **(b)**

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

6.6.2 128x8 Ring of Cliques

The clique size plays a crucial role for the efficacy of the DAB algorithm. The decreased clique size favors the DAB making it the load balancing algorithm to reduce the error in the network more rapidly than the Push-Pull Sum based algorithms as seen in figure 60. All the load balancing algorithms show an rapid decrease of error in the network, indicated by the steep negative slopes as visualized in figure 64. The Push-Pull Sum based algorithms show an decrease of -180, whereas the DAB shows a decrease of -190 in the start region (rounds 1 to 5). After the very steep decrease in the start region the load balancing algorithms decrease the error more moderately, where the DAB algorithm still has the steepest decrease of error of -5.2, followed by the ATPPS decreasing the error as low as -4.6, lastly the PPS follows with a slope of -3.6. The curve in this region is rather flat as seen in the log-log representation in figure 60 a) The decrease effect diminishes as the error of the network is already relatively low, in the end region. The MSE values at the end of round of 100 show the magnitude of error reduction. The DAB scored the lowest MSE value 10.64, followed by the ATPPS achieving very good results to with a MSE value of 13.68. The PPS value achieved the highest MSE value of 22.33. Compared to the (32×32)-Ring of Cliques the load balancing algorithms reduced the error even more by a margin of 2 to 4 units. The most improvement is drawn by the ATPPS

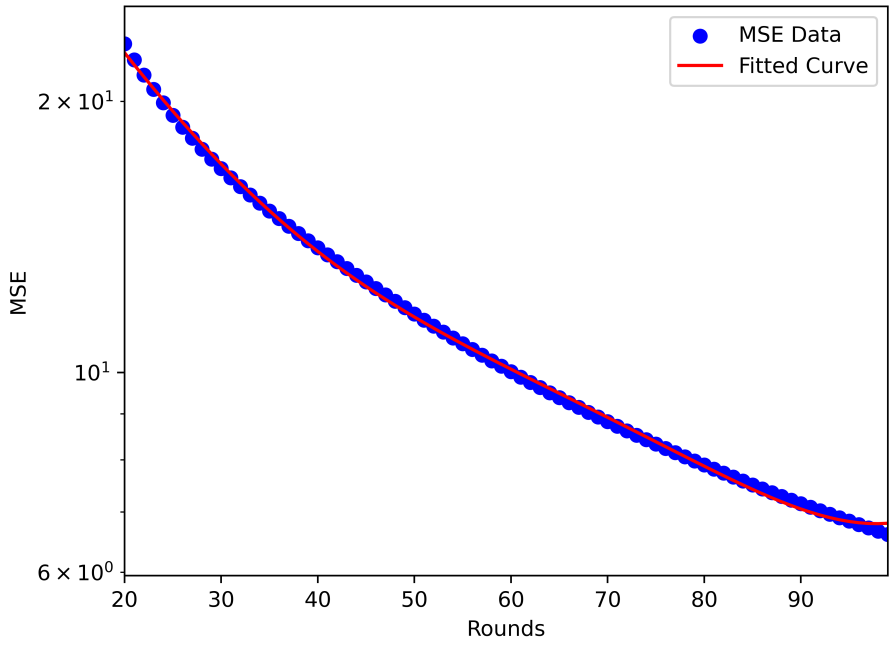


Figure 51: (32×32)-Ring of Cliques - polynomial regression fit: DAB

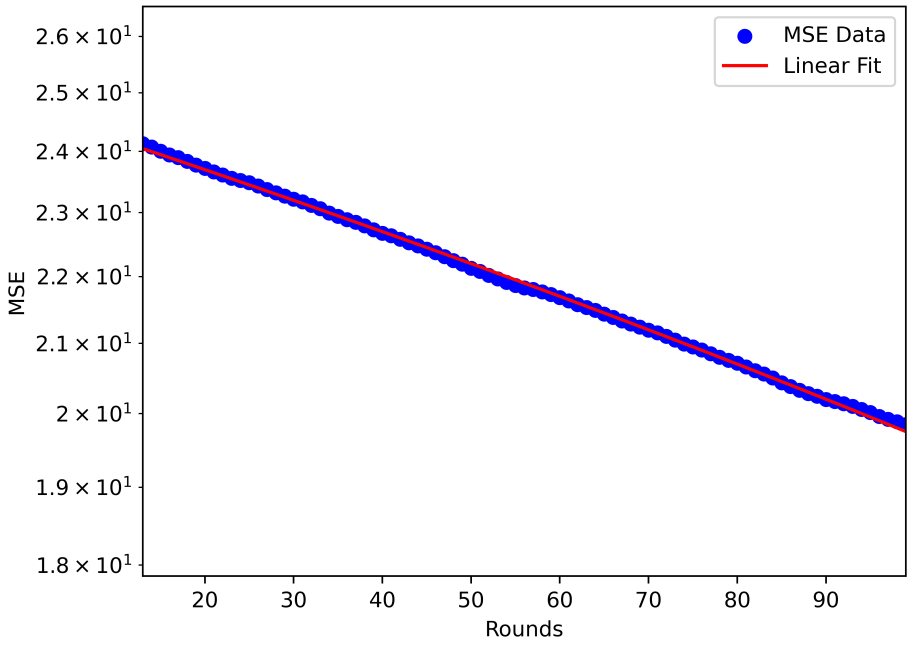


Figure 52: (32×32)-Ring of Cliques - polynomial regression fit: PPS

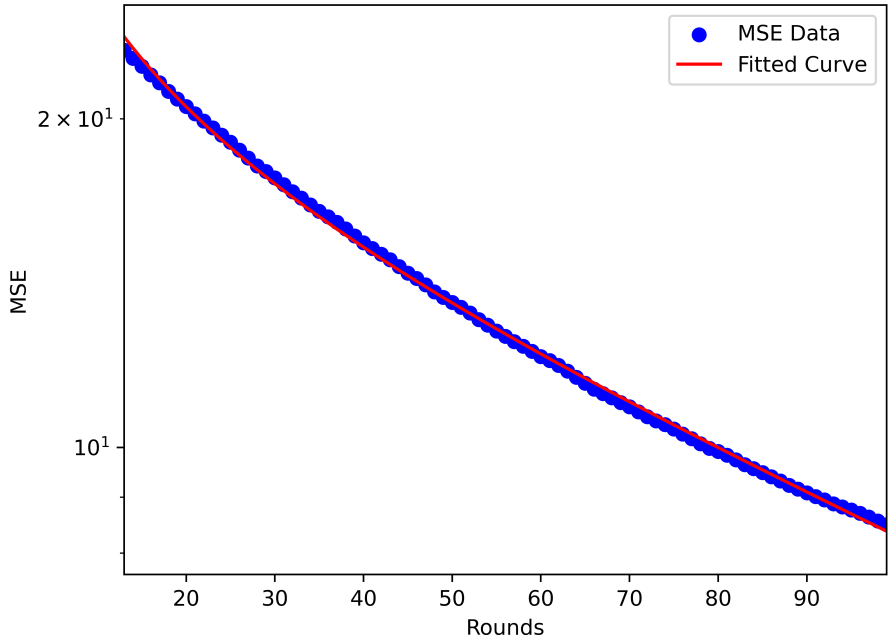


Figure 53: (32×32)-Ring of Cliques - logarithmic regression fit: ATPPS

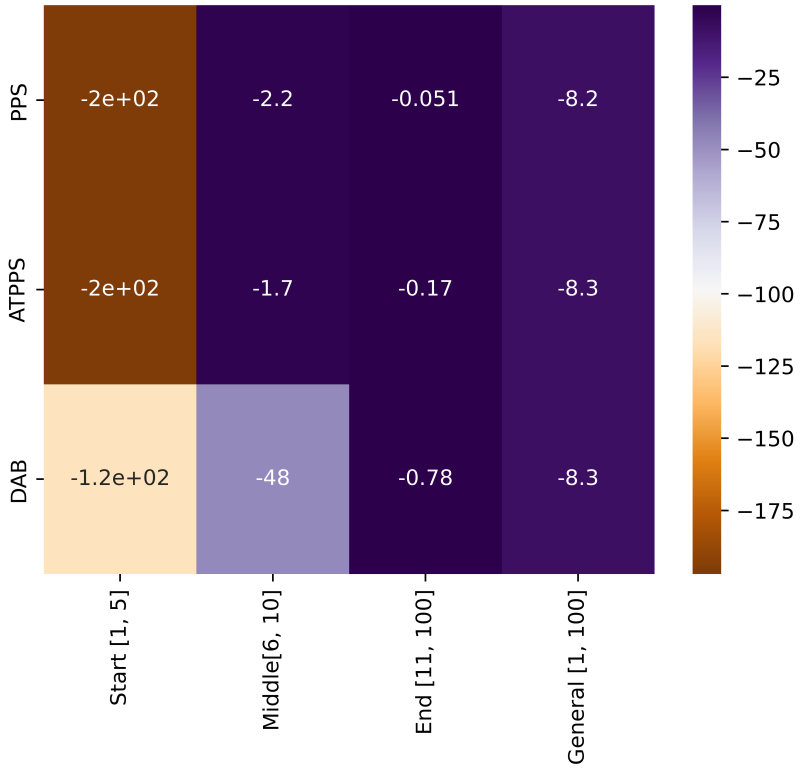
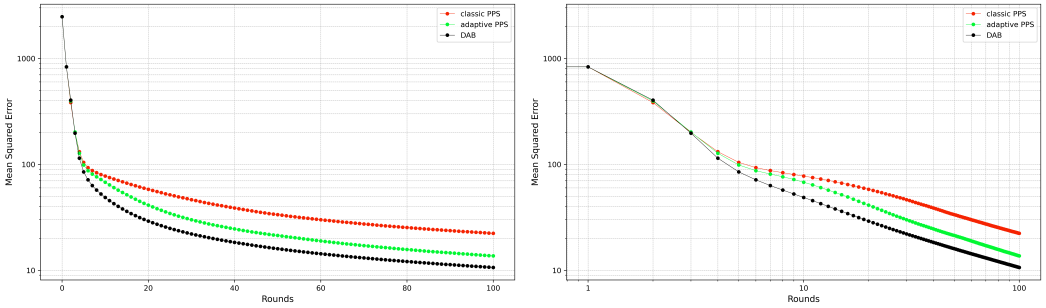


Figure 54: (32×32)-Ring of Cliques: heat map of slopes per region

with a lower value of around around 5 followed by the DAB, with a lower MSE of 4 and lastly the PPS with a decreased value of around 2.

All three load balancing algorithms were fitted against polynomials with degree 4. The equations are as follows: DAB $MSE_r = 5.45 \times 10^{-7} r^4 - 1.7 \times 10^{-4} r^3 + 0.02 r^2 - 1.33 r + 46.71$ (figure 56) PPS: $MSE_r = 1.04 \times 10^{-6} r^4 - 3.41 \times 10^{-4} r^3 + 0.04 r^2 - 2.73 r + 97.58$ (figure 57) and ATPPS: $MSE_r = 9.54 \times 10^{-7} r^4 - 2.93 \times 10^{-4} r^3 + 0.03 r^2 - 2.05 r + 67.02$ (figure 58).



(a) **(b)**

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

6.6.3 8x128 Ring of Cliques

Increasing the clique size and reducing the number of cliques favors the Push-Pull Sum based algorithms, which show a very steep decrease of error in the first 5 rounds of the simulation. The error is reduced by -200 on average in this region for each Push-Pull Sum based algorithm. The DAB data shows a more moderate decrease by -36 in this region. The error reduction reduces over the next regions, since the error of the network is already very low and the network is in a state of good balance showing a MSE in this region of around ~ -30 for the Push-Pull Sum based algorithms. After the first region the state of the network managed by the DAB is still very unbalanced and thus the balance potential is higher. This is shown

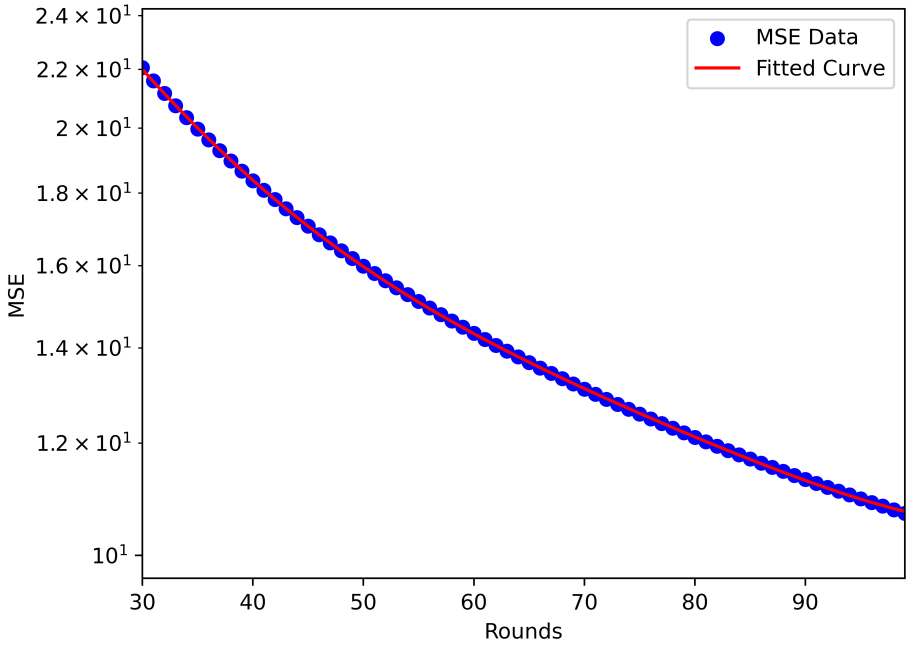


Figure 56: (128 × 8)-Ring of Cliques - polynomial regression fit: DAB

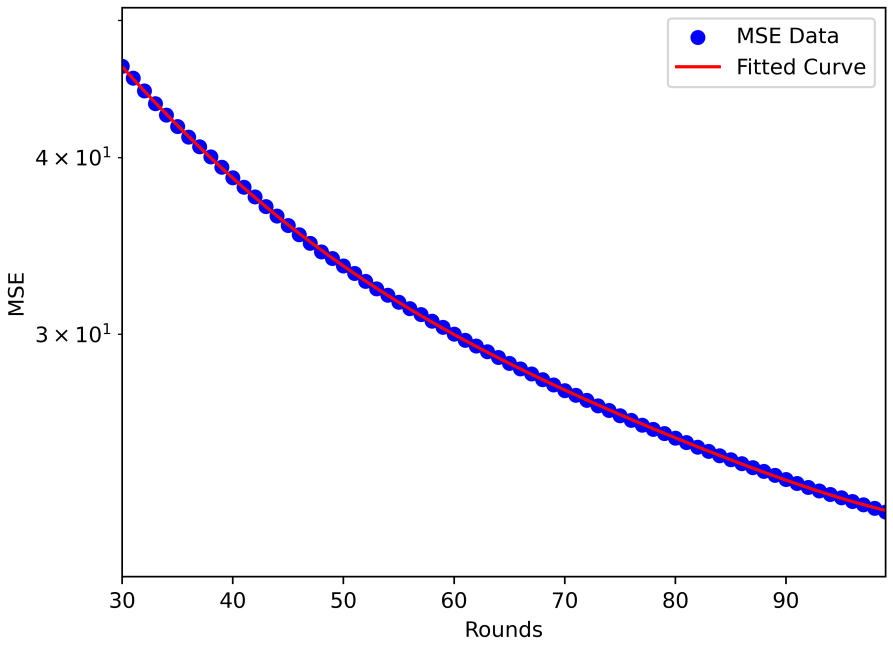


Figure 57: (128 × 8)-Ring of Cliques - polynomial regression fit: PPS

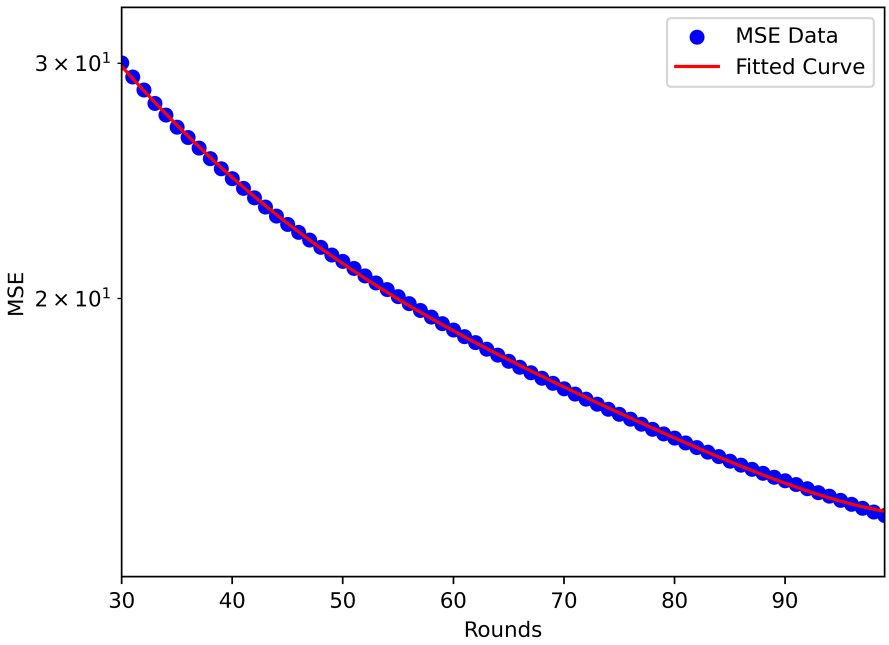


Figure 58: (128 × 8)-Ring of Cliques - polynomial regression fit: ATPPS

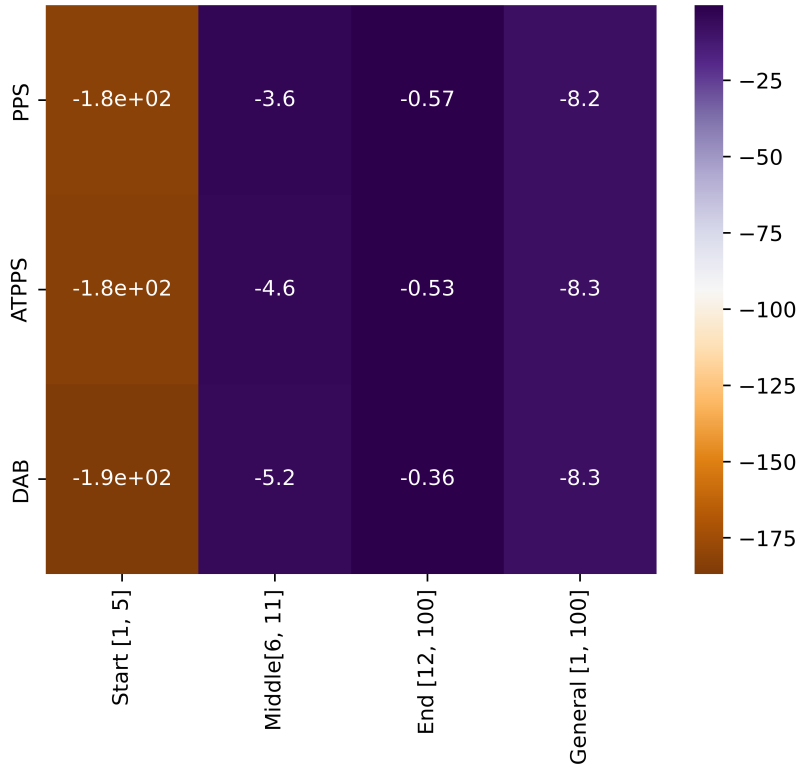
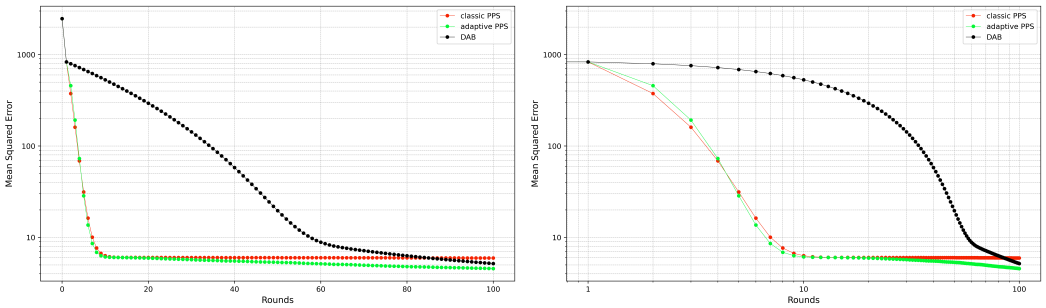


Figure 59: (128 × 8)-Ring of Cliques: heat map of slopes per region

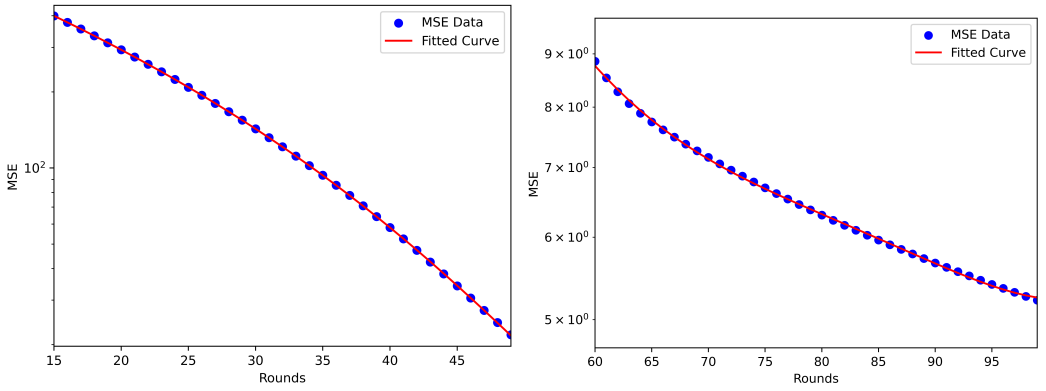
by the slopes in the middle region. The Push-Pull Sum based algorithms reduce the error by ~ -2 while the DAB shows a still very high value of ~ -30. The reason behind this behavior is due to the impact of the clique sizes. The Push-Pull Sum based algorithms achieve good performances in reducing error in this scenario compared to the DAB which struggles in the context of dense graphs. The steep decrease and the low MSE values after rounds 5 and 10 stem from the fast error reduction in the cliques while still having to spread loads from clique to clique through the briding nodes. Compared to the experiment in the (8×128)-Ring of Cliques in 6.6.2 the error after rounds 100 drop to lower values. After rounds 100 the DAB achieves a MSE value of 5.18 compared to 5.95 for the PPS and 4.56 for the ATPPS. especially in the last region (rounds 12 to 100) the DAB catches up to the PPS-based algorithms. However, the Push-Pull Sum based algorithms achieved a broadly balanaced state of the network after round 10 dropping the error to a MSE value of 7. In the following rounds the inter -clique error reduction follows. Here the PPS algorithm has difficulties, since the algorithms orders the nodes of the network to choose their transfer partner by random. 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 downwards trend. The stagnating trend between rounds 20 to 100 is expressed by the linear mode fit in figure 62. The best-fit model follows the equation. $MSE_r = -0.0015x + 6.05$. The PPS curve shows a complexer relation between the MSE reduction over the rounds, namely a polynomial of degree 2 following the equation $MSE_r = 6.07 \times 10^{-5}r^2 - 0.02r + 6.39$ (figure 62). The DAB does not follow a single power relation ship in this region. Between the rounds 15 to 50 the error reduction can be expressed by a third degree polynomial $MSE_r = -3.33 \times 10^{-3}r^3 + 0.63r^2 - 40.23r + 872.75$ (figure 63 a)) while in later rounds the power relation is a bit complexer, expressed by a fourth degree polynomial following the equation $MSE_r = 2.96 \times 10^{-6}r^4 - 9.97 \times 10^{-3}r^3 + 0.13r^2 - 7.16r + 161.73$ (figure 63 b)).

Snippets of the simulation outcomes after the 100th rounds of each load balancing algorithm verify that within the cliques the load is balanced, however the inter clique connection seems to be pending. 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 ~ 46.35 the second clique averaged to ~ 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 similiar 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: rounds 20-50 and 55-100

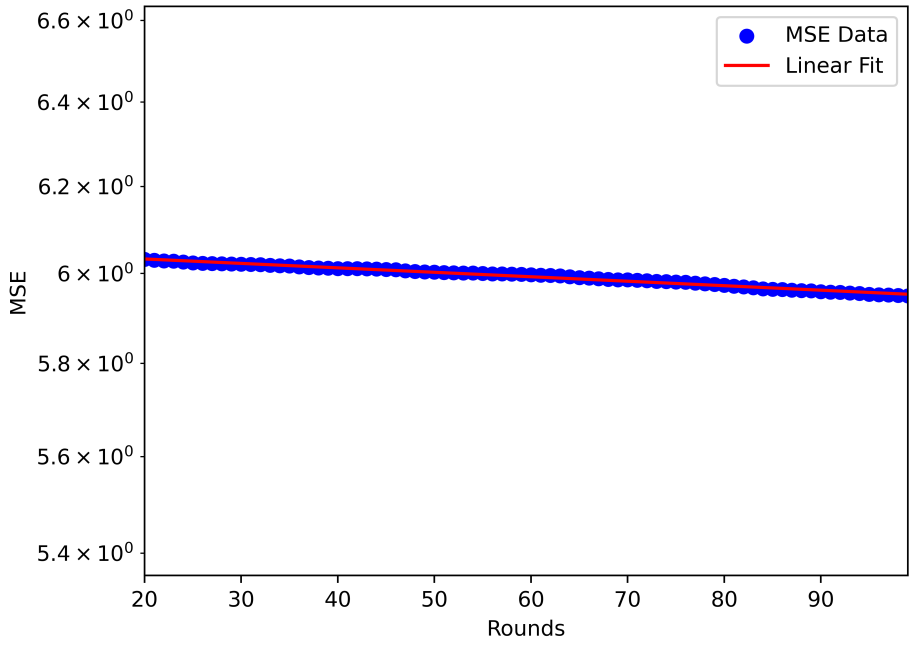


Figure 62: (8 × 128)-Ring of Cliques - polynomial regression fit: PPS

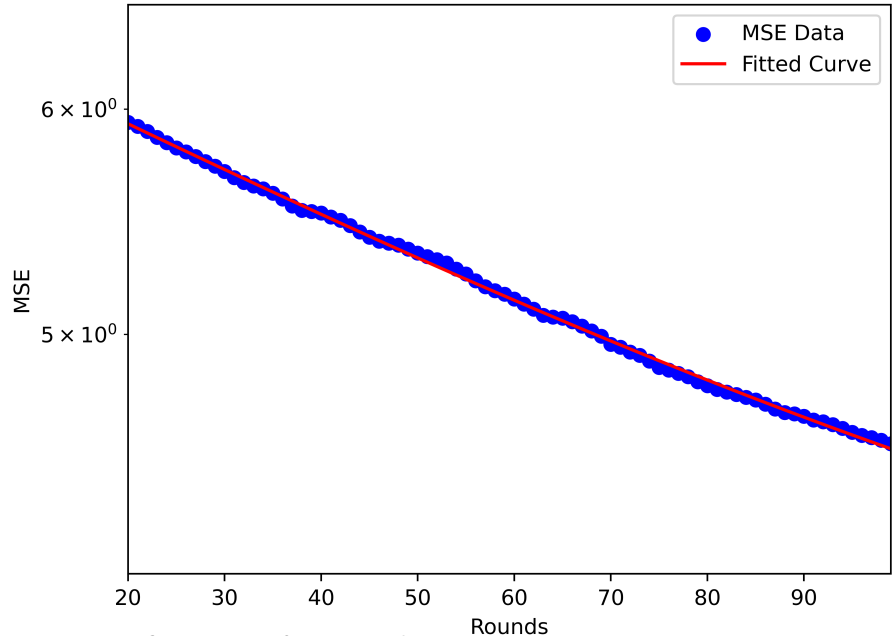


Figure 63: (8 × 128)-Ring of Cliques - polynomial regression fit: ATPPS

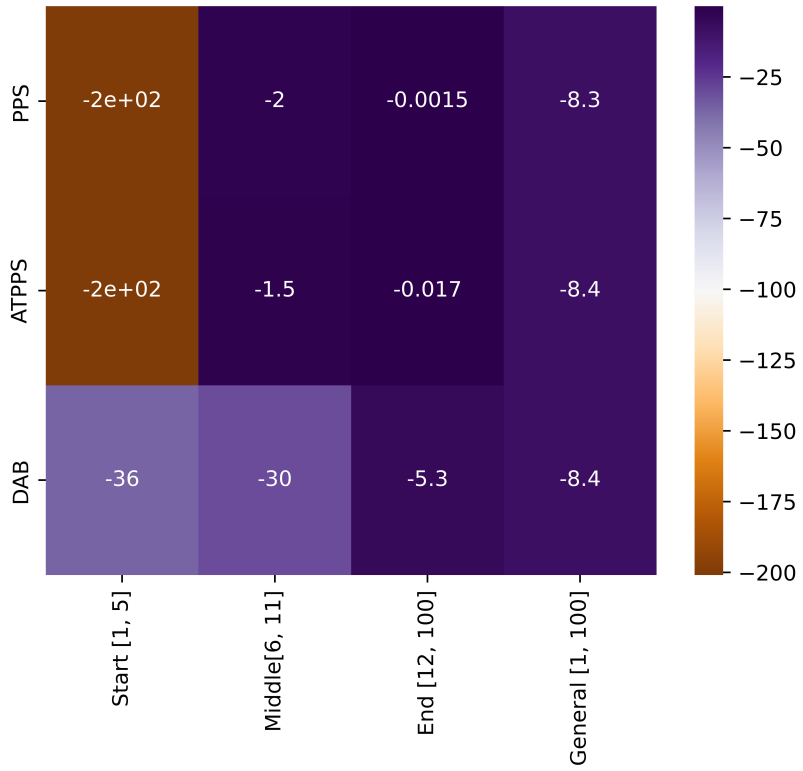


Figure 64: (8×128)-Ring of Cliques: heat map of slopes per region

```
# First Clique starts here
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
# First Cliques ends here

# Second Clique starts here
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
# Second Cliques ends here
```

Listing 6.1: Snippet of simulation outcomes ATPPS: Experiment 2

7 Conclusion

8 Acknowledgments

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

- advisers
- examiner
- person1 for the dataset
- person2 for the great suggestion
- proofreaders

9 Appendix

Table 2: Simulation overview per topology: fitted model, slopes per region, final MSE

Topology		**Fitted Model**	**Slope Region 1**	**Slope Region 2**	**Slope Region 3**	**Slope General**	**final MSE**
K_{1024}			**rounds 1-10**:	**rounds 11-65**:	**rounds 66-100**:		
	DAB	**rounds 1-100**: $MSE_r = 844.63 * e^{-0.01*r}$	-4.8	-4.2	-3.5	-4	436.85
	PPS	**rounds 10-80**: $MSE_r = 2530.41 * e^{-0.9*r}$	-92	-2.3×10^{-3}	-1.4×10^{-24}	-8.4	1.73×10^{-28}
	ATPPS	**rounds 10-65**: $MSE_r = 4309.94 * e^{-1.06*r}$	-92	-6.7×10^{-4}	-7.9×10^{-29}	-8.4	5.63×10^{-28}
S_{1024}			**rounds 1-10**:	**rounds 11-45**:	**rounds 46-100**:		
	DAB	**rounds 1-100**: $MSE_r = 840.42 * e^{-0.01*r}$	-4.3	-3.9	-3.2	-3.6	480.48
	PPS	**rounds 10-45**: $MSE = 29794.60 * e^{-1.39*r}$	-92	-2.1×10^{-4}	-1.3×10^{-25}	-8.4	8.31×10^{-25}
	ATPPS	**rounds 18-60**: $MSE_r = 9329.40 * e^{-1.05*r}$	-92	-7.4×10^{-4}	-6.8×10^{-20}	-8.4	6.52×10^{-24}
R_{1024}			**rounds 1-10**:	**rounds 11-69**:	**rounds 70-100**:		
	DAB	**rounds 10-60**: $MSE_r = 1.72 \times 10^{-5} r^4 - 2.30 \times 10^{-3} r^3 + 0.19 r^2 - 5.99 r + 114.83$	-82	-0.71	-0.15	-8.2	22.41
	PPS	**rounds 10-60**: $MSE_r = 2.99 \times 10^{-5} r^4 - 0.5 \times 10^{-2} r^3 + 0.32 r^2 - 9.68 r + 166.30$	-82	-1	-0.19	-8.1	27.68
	ATPPS	**rounds 10-60**: $MSE_r = 3.04 \times 10^{-5} r^4 - 0.5 \times 10^{-2} r^3 + 0.32 r^2 - 9.64 r + 161.86$	-82	-0.98	-0.17	-8.1	26.56

Table 3: Simulation overview per topology: fitted model, slopes per region, final MSE

Topology		**Fitted Model**	**Slope Region 1**	**Slope Region 2**	**Slope Region 3**	**Slope General**	**final MSE**
$L_{512,512}$			**rounds 1-7**:	**rounds 8-50**:	**rounds 51-100**:		
	DAB	**rounds 20-100**: $MSE_r = -5.89 \times 10^{-5} r^3 + 0.03 r^2 - 5.68 r + 459.42$	-67	-4.5	-2.6	-7.3	108.90
	PPS	**rounds 20-100**: $MSE_r = 8.44 \times 10^{-7} r^4 - 2.52 \times 10^{-4} r^3 - 0.03 r^2 - 1.64 r + 56.68$	-130	-0.9	-0.12	-8.3	14.71
	ATPPS	**rounds 20-100**: $MSE_r = 8.69 \times 10^{-7} r^4 - 2.56 \times 10^{-4} r^3 + 0.03 r^2 - 1.62 r + 54.48$	-130	-0.87	-0.12	-8.3	13.82
$L_{128,896}$			**rounds 1-7**:	**rounds 8-50**:	**rounds 51-100**:		
	DAB	**rounds 20-100**: $MSE_r = 3.46 \times 10^{-6} r^4 - 1.12 \times 10^{-3} r^3 + 0.14 r^2 - 7.69 r + 190.78$	-110	-2.8	-0.25	-8.2	3.41
	PPS	**rounds 20-100**:$MSE_r = -3.72 \times 10^{-8} r^5 + 1.31 \times 10^{-5} r^4 - 1.85 \times 10^{-3} r^3 + 0.13 r^2 - 4.96 r + 84.91$	-120	-1.5	-7.6×10^{-2}	-8.4	3.31
	ATPPS	**rounds 20-100**: $MSE_r = 1.59 \times 10^{-6} r^4 - 4.74 \times 10^{-4} r^3 + 0.054 r^2 - 2.94 r + 70.59$	-120	-1.6	-0.13	-8.4	17.47
$L_{896,128}$			**rounds 1-7**:	**rounds 8-50**:	**rounds 51-100**:		
	DAB	**rounds 20-100**: $MSE_r = -1.936 \times 10^{-4} r^3 + 0.05 r^2 - 5.33 r + 745.95$	-20	-2.9	-0.86	-2.9	542.09
	PPS	**rounds 20-100**:$MSE_r = 2.00 \times 10^{-7} r^4 - 6.01 \times 10^{-5} r^3 + 6.95 \times 10^{-3} r^2 - 0.39 r + 13.44$	-140	-0.23	-2.7×10^{-2}	-8.4	3.59
	ATPPS	**rounds 20-100**: $MSE_r = 2.28 \times 10^{-7} r^4 - 6.77 \times 10^{-5} r^3 + 7.68 \times 10^{-3} r^2 - 0.42 r + 13.62$	-140	-0.23	-2.7×10^{-2}	-8.4	3.27

Table 4: Simulation overview per topology: fitted model, slopes per region, final MSE

Topology		**Fitted Model**	**Slope Region 1**	**Slope Region 2**	**Slope Region 3**	**Slope General**	**final MSE**
$ROC_{32,32}$			**rounds 1-5**:	**rounds 6-10**:	**rounds 11-100**:		
	DAB	**rounds 20-100**: $MSE_r = -5.89 \times 10^{-5} r^3 + 0.03r^2 - 5.68r + 459.42$	-120	-48	-0.78	-8.3	6.55
	PPS	**rounds 10-100**: $MSE_r = 8.44 \times 10^{-7} r^4 - 2.52 \times 10^{-4} r^3 - 0.03r^2 - 1.64r + 56.68$	-200	-2.9	-5.1×10^{-2}	-8.2	19.80
	ATPPS	**rounds 10-100**: $MSE_r = 8.69 \times 10^{-7} r^4 - 2.56 \times 10^{-4} r^3 + 0.03r^2 - 1.62r + 54.48$	-200	-1.7	-0.17	-8.3	8.42
$ROC_{128,8}$			**rounds 1-5**:	**rounds 6-11**:	**rounds 12-100**:		
	DAB	**rounds 30-100**: $MSE_r = 5.45 \times 10^{-7} r^4 - 1.7 \times 10^{-4} r^3 + 0.02r^2 - 1.33r + 46.71$	-190	-5.2	-0.36	-8.3	10.64
	PPS	**rounds 30-100**:$MSE_r = 1.04 \times 10^{-6} r^4 - 3.41 \times 10^{-4} r^3 + 0.04r^2 - 2.73r + 97.58$	-180	-3.6	-0.57	-8.2	5.95
	ATPPS	**rounds 30-100**: $MSE_r = 9.54 \times 10^{-7} r^4 - 2.93 \times 10^{-4} r^3 + 0.03r^2 - 2.05r + 67.02$	-180	-4.6	-0.53	-8.3	22.33
$ROC_{8,128}$			**rounds 1-5**:	**rounds 6-11**:	**rounds 12-100**:		
	DAB	**rounds 15-50**: $MSE_r = -3.33 \times 10^{-3} r^3 + 0.63r^2 - 40.23r + 872.75$ **rounds 60-100** $MSE_r = 2.96 \times 10^{-6} r^4 - 9.97 \times 10^{-3} r^3 + 0.13r^2 - 7.16r + 161.73$	-36	-30	-5.3	-8.4	5.18
	PPS	**rounds 20-100**:$MSE_r = 6.07 \times 10^{-5} r^2 - 0.02r + 6.39$	-200	-2	-1.5×10^{-3}	-8.3	5.95
	ATPPS	**rounds 20-100**: $MSE_r = -0.0015x + 6.05$	-200	-1.5	-1.7×10^{-2}	-8.4	4.56

Table 5: Simulation overview per topology: fitted model, slopes per region, final MSE

Topology		**Fitted Model**	**Slope Region 1**	**Slope Region 2**	**Slope Region 3**	**Slope General**	**final MSE**
$T_{32,32}$			**rounds 1-10**:	**rounds 10-39**:	**rounds 66-100**:		
	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**: $MSE_r = -6.01 \times 10^{-06} r^3 + 1.66 \times 10^{-3} r^2 - 0.16 r + 6$	-140	-0.34	-2.4×10^2	-8.4	436.85
	PPS	**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**: $MSE_r = -1.15 \times 10^{-5} r^3 + 3.205 \times 10^{-3} r^2 - 0.33 r + 13.72$	-130	-1	-5.8×10^{-2}	-8.4	1.73×10^{-28}
	ATPPS	**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**: $MSE_r = -9.99 \times 10^{-6} r^3 + 2.8034 \times 10^{-3} r^2 - 0.28 r + 11.29$	-130	-0.83	-4.8×10^{-2}	-8.4	5.63×10^{-28}

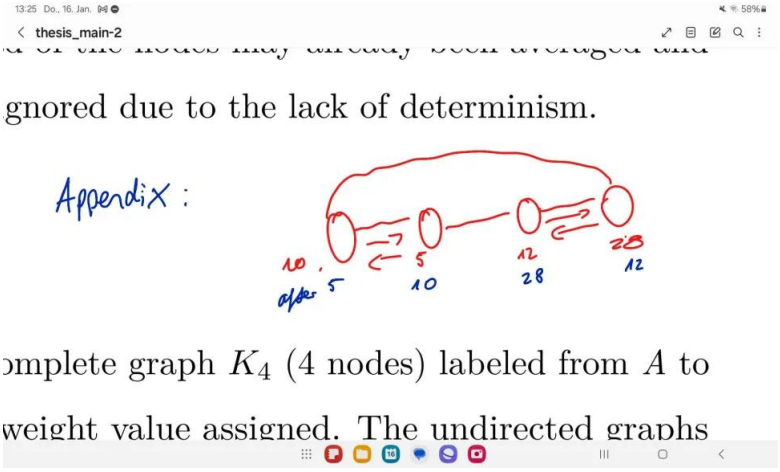


Figure 65: Example Ring Graph

ToDo Counters

To Dos: 0;

Parts to extend: 0;

Draft parts: 0;

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

[7] 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.

[8] 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.

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

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

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

[12] 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.

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

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

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