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, as introduced in [1], combines the Push-Sum [2] and Pull-Sum algorithms. The Push-Sum algorithm, proposed by Kempe et al., is a load balancing algorithm where each node randomly selects a neighbor and transfers half of its current sum and weight to that neighbor. The Push-Pull Sum algorithm is a randomized load balancing algorithm. Load balancing algorithms are used to average loads in networks, modeled as undirected graphs. In these networks, nodes exchange loads with their neighbors in order to reach a balanced network state. The Single-Proposal Deal-Agreement-Based load balancing algorithm as proposed in [3], incorporates a Deal-Agreement into the load transfer to achieve only fair load transfers between two nodes.

In this thesis, I introduce and implement a variation of the Push-Pull Sum algorithm using principles of the Deal-Agreement-Based protocol and adaptive thresholding, designed to add or modify certain properties of the original Push-Pull Sum protocol. For the newly introduced load balancing approach, I provide the pseudocode, implement the algorithm in a peer-to-peer simulation tool, and analyze the simulation outcomes for different topologies. The objective is to find a compromise solution including overall good performance in different topologies.

The performance of this variations is evaluated using the mean squared error (MSE) reduction over time as a metric for convergence to the ground truth. The results are presented through log-log and log-linear graphs, which allow a comparison of the convergence rates and stability in different scenarios. The slope of the MSE curves

reflects how efficiently the protocols distribute the load across the network. The data is fitted into different models to get an insight into the rate of convergence.

The findings suggest that modification of the Push-Pull Sum algorithm incorporates a more efficient and more scalable load balancing strategy for most of the conducted experiments.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

1 Introduction

In times where computational tasks are getting more and more computational heavy, many systems require multiple servers/computers to complete tasks. These servers/computers (or also refered to as nodes) are often directly interacting 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 (to communicate with other nodes). All the nodes are composed to one scalable network. The memory system is modeled as 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 of the computers is assigned a non-negative workload, which from now on is refered to as load. Loads represent various computational taks 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 has the average load of the network. This objective is achieved by combating over- and underloading by redirecting load to alternate nodes in the system. Heavily overloaded nodes may fail to complete the assigned tasks due to overheating. In that sense, load balancing enhances coordination in distributed systems improves scalability and ensures high availability.

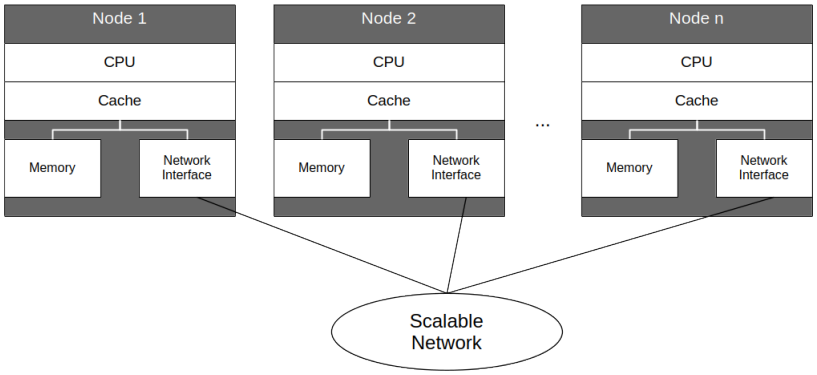


Figure 1: Overview of the Setting

1.1 Preliminaries

A graph $G = (V, E \subseteq V \times V)$, where $V := \{v_0, v_1, ..., v_{n-1}\}$ are the vertices of G and $E := \{e_1, e_2, ..., e_{n-1}\}$ are the edges. The graph may contain only one vertex and zero edges. The number of vertices $|V|$ is often called the order of G. A node and a edge are incident, if the node is an endpoint of the edge. The degree of a node is the number of incident edges. If a edge connects two vertices, these vertices are *adjacent* to each other, or also refered to as *neighboring vertices*. A loop is an edge with equal endpoints. Multiple edges are edges with same pair of endpoints. A simple graph has no loops, nor 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 G belongs to a path. A graph is classified as bipartite graph, if 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. $\text{diam}(G) := max_{u,v \in V} d_G(u, v)$ is the diameter of a graph. [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. The obervations were gathered in a the student project with the title "Comparative Analysis of Load Balancing Algorithms in General Graphs" [6]. Judging from the simulations conducted in the student project, the Push-Pull Sum algorithm seems to perform better in reducing the MSE per round for the complete graph and the star graph compared to the Deal-Agreement-Based algorithm performing 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 network size also played an role in faster convergence.

To address this, the proposed research introduces a novel algorithm, the Adaptive Threshold Push-Pull Sum algorithm that leverages the strengths of both algorithms, creating a Trade-off to mitigate their individual weaknesses. 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 structural characteristics of the investigated networks, this new solution aims to achieve robust performance across a wide range of scenarios, bridging the gap between the Deal-Agreement-Based algorithm and the Push-Pull Sum algorithm.

1.3 Related Work

Nugroho et al. [1] proposed the Push-Pull Sum algorithm, which essentially is a composition of two algorithms: the Push-Sum algorithm proposed by David Kempe,

Alin Dobra and Johannes Gehrke [2] and the Pull-Sum algorithm. The Push and the Pull mechanics are directly adopted from the Push-Pull Sum algorithm. Nugroho et al. used the mean squared error as a metric to evaluate the performance of their algorithm. In this thesis, I will follow an similar approach. Nugroho et al. conducted their experiments in static general graphs. This research also uses a static graph. Dinitz et al. [3] suggested two versions of the Single-Proposal Deal-Agreement-Based algorithm and two versions of a multi-neighbor Load Balancing algorithm, a round robin approach and a self-stabilizing load balancing algorithm. However the only comparable algorithm with ours are the Single-Proposal Deal-Agreement-Based algorithms, from which there exist two variations. One for the contiunous setting and one for the discrete setting. The main difference between these two is that in the contiunous setting any load may be transfered over the edges and in the discrete setting all load transfers must contain integers. For multi-neighbor load balancing the nodes may transfer loads to several neighbors in one round. For the Push-Pull-Sum algorithm this is only the case for the the pull actions where one node responds to every calling node by sending loads back. The self-stabilizing and the round robin approach are asynchronous algorithms, while the Adaptive-Threshold Push-Pull Sum and the Push-Pull Sum algorithms are synchronous algorithm.

1.4 Hypothesis

The Adaptive Threshold Push-Pull Sum load-balancing algorithm, which integrates key features of the Deal-Agreement-Based algorithm and the Push-Pull Sum algorithm, will demonstrate performances that is 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 algoithm in low-degree networks, achieving a balance that improves overall adaptability and efficiency compared to the both.

This hypothesis will be tested through comparative analysis of the MSE for six distinct topologies (some with different network sizes), demonstrating the algorithm's robustness and scalability across network environments. Model fitting is applied as a analysis technique to see the trend of the data and to be able to make statements regarding the convergence rate. Slopes are computed for three different regions labeled as *Start*, *Middle* and *End*, in order to get details of the performance in specific time frames.

1.5 Contribution

This study introduces a novel load balancing algorithm that combines the strengths of two established approaches randomized load balancing and deal agreement-based balancing while integrating an adaptive threshold mechanism to enhance performance by adapting to the current state of the network. The load balancing algorithm uses the push and pull mechanics for convergence to a balanced state.

2 Problem Overview

The load balancing problem is defined in an undirected general graph, where each load may transfer loads over the edges to their neighboring nodes in order to balance the state of the network. The setting and the approach are elaborated on in this section.

2.1 Setting

- **Continuous and Discrete**: In the Continuous setting, load balancing algorithms may transfer any amount of load over the edges, while in the discrete setting all load transfers should contain integers.

- **Synchronous and Asynchronous**: In the synchronous setting the time of message delivery is constant (e.g. O(1)), while in the asynchronous setting the time of message delivery may be unpredictably large. However, it is possible to make the asynchronous setting synchronous, by simply adjusting the time frame in which messages should be delivered.

- **Static or Dynamic**: The graphs in which the load balancing algorithms operate in, may be static or dynamic. A dynamic graph setting may change arbitrarily between the rounds, while the static connections and the nodes remain the same for the static graph.

The Peer-to-peer network is modeled as a static general graph, meaning that the set of edges may not change during the application of the load balancing algorithms. The objective of balancing load will be achieved with local algorithms, so that each node only collects information of nodes in their direct neighborhood. The setting is a continuous setting, where data transfer over the edges may contain any amount not just integers. The assumption of an synchronous message delivery is made, where the time of message delivery is constant e.g. $O(1)$ time and thus is in a synchronous setting. There are six distinct network topologies under test, each consists of 2^{10} nodes.

2.2 Approach

This research consists of three steps, the design of a load balancing algorithm, the simulation step to test the ability to balance the state of the network for distinct topologies, and a comparative analysis step, where the simulation outcomes are evaluated, using methods from statistics. In the previous research "Comparative Analysis of Load Balancing Algorithms in General Graphs" [6], the strengths and weaknesses of two distinct load balancing algorithms were determined, by simulating them in different topologies for different network sizes to test the scalability and adaptability of the algorithms to different situations. The design of a novel adaptive threshold load balancing algorithm is based on this knowledge. Each simulation contains of 30 distinct experiments, for further statistical significance. The simulation outcomes are finally evaluated using model fitting to determine the trend of the algorithms regarding MSE reduction, and slopes are calculated per region to see the consistency in MSE reduction. The results are presented in comprehensive plots, with ellaboration on why the plots look like they do.

3 Algorithms

This thesis examines three load balancing algorithms. First, the classic Push-Pull Sum algorithm and the Continuous Single-Proposal Deal-Agreement-Based algorithm are introduced. Next, the proposed Adaptive Threshold Push-Pull Sum algorithm is presented as a novel approach. The composition of this new algorithm is explained, including how it integrates elements of the first two algorithms and the rationale behind its design. For each algorithm, pseudo-code and a detailed description of their operational mechanics are provided. Additionally, examples are included to illustrate how the algorithms achieve load balancing. For the Adaptive Threshold Push-Pull Sum algorithm, the intended outcomes are also outlined to provide further clarity on its objectives.

3.1 Characteristics

Like the graphs, load balancing algorithms may have different characteristics. In the following these characteristics are elaborated on:

- **Static and Dynamic**: Load balancing algorithms can be categorized into static and dynamic algorithms. Static load balancing algorithms assign the tasks to the nodes at compile time. Dynamic load balancing algorithms assign the 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 use randomness in order to choose a load transfer partner. Deterministic load balancing algorithms on the other hand, use some predefined distribution rules in order to make load transfers. [4]

- **Global and Local**: For local load balancing algorithms nodes may only transfer loads within their domain/neighborhood, while a global load balancing algorithm enables the nodes in the network to perform load balancing operations across the whole network [4].

- **Monotonic and Non-monotonic**: A load balancing algorithm is characterized as 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**: Load balancing algorithms may posses the mass conservation property, which ensures that the values will converge to the correct aggregate of the ground truth [1].

- **Anytime**: An *anytime* load balancing algorithm can be stopped at any time during the execution, and after stopage the state of the network is not worse than one of its preceding states. The advantage that comes with an anytime algorithm is that the network in which the load balancing algorithm with this property is applied to shows more feasible states with intermediate rounds. [3]

Many of these properties and characteristics are desirable for the load balancing algorithms to make the load balancing more efficient, like monotonicity or anytimeness. Others contribute to computational lightness like locality, and others guarantee predictability of the behaviour like determinism.

3.2 Classic Push-Pull Sum Algorithm

The Push-Pull Sum algorithm as proposed in [1] requires each node to have sum $s_{i,r}$ and weight $w_{i,r}$ values as initial information. The initial weight $w_{i,0}$ for each node is equal to 1. The sum of all weights is equal to the network size N. The sum of $s_{i,0}$ 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 pseudo code is to be found in figure 3. The Push-Pull Sum algorithm is composed of three different procedures, namely the *RequestData* procedure, the *ResponseData* procedure and the *Aggregate* procedure. In every round r the *Aggregate* procedure is called, except for the first round, since there is no information to be aggregated. In this procedure, every node gatheres all incoming messages $M_{i,r}$ sent by other nodes $\{(s_m, w_m)\}$ in the previous round $r-1$, requesting data. Also, nodes update their sum values which essentially is $\sum_{m \in M_{i,r}} s_m$ and weight values $\sum_{m \in M_{i,r}} w_m$. The respective loads are calculated by dividing sum by weight. Following that, 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 node and itself. The pull mechanism 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) with the half of its sum value divided by the number of incoming requests, so $\frac{\frac{s_{i,r}}{2}}{|R_{i,r}|}$.

The setting in the paper [1] is similar to ours. While they only inspect a complete graph with 10^4 nodes, we have network sizes of 2^{10} nodes and more topologies under test. In that paper, 50 experiments each conducted for 30 rounds were performed. The paper showed that the Push-Pull Sum algorithm decreases the expected potential Φ_r exponentially. The potential function is defined as $\Phi_r = \sum_{i,j} \left(v_{i,j,r} - \frac{w_{i,r}}{n}\right)^2$. The $v_{i,j,r}$ component stores the fractional value of node j's contribution at round r. The conditional expectation of $\Phi_r + 1$ for the Push-Pull Sum algorithm is

Algorithm 1 Push-Pull Sum algorithm

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

$\mathbb{E}[\Phi_r + 1 | \Phi_r = \phi] = (\frac{2e-1}{4e} - \frac{1}{4n})\phi$. The Push-Pull Sum algorithm has the mass-conservation property. Due to the fact that the load balancing algorithm orders its nodes to choose a random neighbor the load balancing algorithm is characterized as a stochastic load balancing algorithm. [1]

The Push-Pull Sum algorithm performed very well for the Complete graph, the Ring of Cliques with large clique size and Lollipop graphs with large clique size, and the Star graph. For the Star graph the internal node acts as a distributor of the load. While every leaf chooses with 100% possibility the internal node 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 $\frac{\frac{w_{i,r}}{2}}{N-1}$. A different explanation applies for the Complete graph, the Ring of Cliques and the Lollipop graph, where the density of each node plays an crucial role. Nodes choose a random neighbor, since the dense graphs have many edges, randomly choosing a neighbor allows the algorithm to spread loads effectively without relying on a deterministic path, reducing the likelihood of bottlenecks or overloading a single node. That is the reason, why the

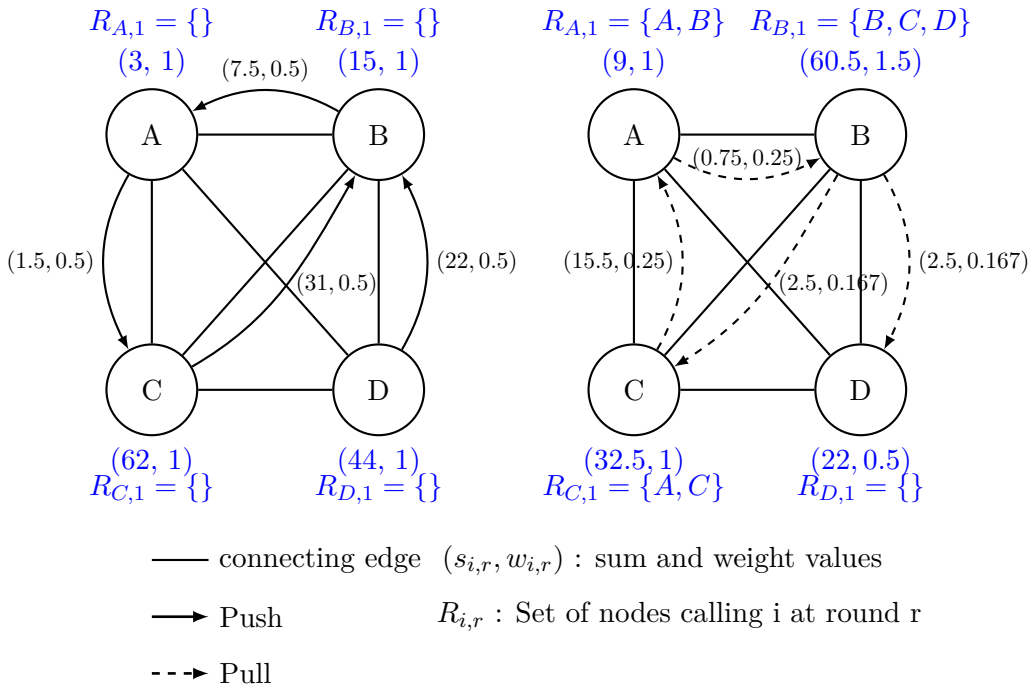


Figure 2: Left: Push actions; Right: Pull actions

Push-Pull Sum algorithm did not perform so well for the Torus Grid graph and the Ring graph compared to the Single-Proposal Deal-Agreement-Based algorithm, where the number of edges is limited to 4 and 2 respectively. For these topologies redundant communication may occur, since 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 the load values interact in such way that one node u adopts the state of another node v during the round, thus $load_r(u) = load_{r-1}(v)$ and vice versa, as 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 depicts a Complete graph K_4 (4 nodes) labeled from A to D. Each node has a initial sum and weight value assigned. The undirected graphs

depict the connections between the nodes, indicating which nodes are neighboring nodes. The solid directed edges depict the push operations and the dashed directed edges depict the pull operations. Each node has a set $R_{i,r}$ where i is the node id and r is the round where we inspect the set. 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 depicts the behaviour of the nodes while executing the push operations, while the right-hand side depicts the case of pull operations. Each node chooses a random neighbor to push load to. Node A selected node C and pushes half of its sum and weight to the chosen node C and to itself (the loops are not included into the graphics, due to the readability of the figures). Node B chose A as a trading partner, node C and D push to node B and themselves. Given the push operations for each node the set $R_{i,r}$ is computed. Due to node B pushing load to node A and node A pushing load to itself, the set $R_{A,1}$ evaluates to $\{A, B\}$. The updated sum and weight values can be inspected on the right hand side of figure 2. Following the push actions, each node proceeds with the *ResponseData*-procedure. For all the nodes in $R_{i,1}$ the nodes reply with the pull values. Since node A has two nodes in its set $R_{A,1}$, node A replies with $\left(\frac{\frac{3}{2}}{2}, \frac{\frac{1}{2}}{2}\right)$ to node B and itself, indicated by a dashed directed edge. Accordingly, the remaining nodes *B, C and D* execute the pull operations. Following the push and pull operations the nodes sum and weight values are updated. The setting after round one is depicted in figure 3. The mean squared error in the beginning of round 1 is 542.50, following one round of applying the Push-Pull Sum algorithm the mean squared error after round 1 is 63.49.

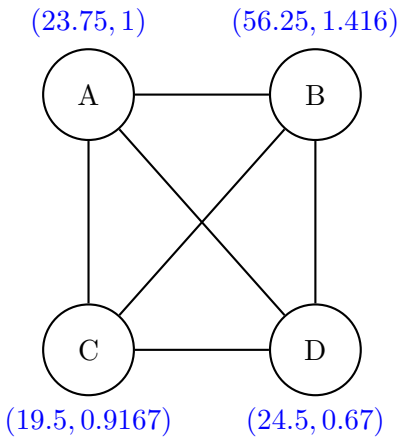


Figure 3: Setting after round 1

3.3 Contiunous Single-Proposal Deal-Agreement-Based Algorithm

The Continuous Single-Proposal Deal-Agreement-Based algorithm proposed by [3], is unlike the Push-Pull Sum algorithm not an diffusion-based load balancing algorithm. The goal of load balancing is achieved based on deterministic deal-agreements, where one node proposes node to one neighboring node and the neighboring node either accepts the transfer proposal either full or partialy. Dinitz et al. proofed that the algorithm is a anytime algorithm, as they never worsen the state of the network during execution. They studied the algorithm in a dynamic setting. Each node has a set of neighboring nodes and their including the node's initial load itself as initial information. The algorithm is divided into three phases. There is the *proposal*, *deal* and the *summary*-phase. In the *proposal*-phase each node u contacts the minimal loaded neighbor v and sends a proposal to that neighbor, if the neighbor is less loaded. 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. Following that, the nodes enter the *deal*-phase. In this 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 are being updated.

Finally, in the *summary*-phase each node informs their neighbors regarding their updated load values. [3]

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

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 this paper is based on a potential function. Dinitz et al. defines potential for a node u as $p(u) = (load(u) - L_{avg})^2$ where L_{avg} is 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 essentially 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 $2 * l^2$. And as a result of any round r of the Contiunous 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 seems to have difficulties in reducing the mean squared error as rapidly as the Push-Pull Sum algorithm for dense graphs like the a complete graph, the Lollipop graph with large

clique size, the Ring of Cliques with large clique sizes. This seems to be the case, since each node is looking for a minimal load partner. For a Complete graph (and for the cliques respectively) the minimal load partner is the same node with loads of L_{min}. 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. An analouge scenario happens for the Ring of Cliques and Lollipop graph, with the difference that for the Ring of Cliques not each node is proposing to the same minimal neighbor, but for the minimal laoded 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. Looking at the Torus Grid graph and the Ring graph, this Deal-Agreement-Based algorithm seems to perform very well in reducing the error, since the proposals are more distributed for the nodes due to the density of the graph, and thus more nodes are involved in load transfers. The Star graph makes an exception to this case, since we have a similar bottleneck as in the Complete graph, where the internal node is the common neighbor for each leaf and thus each leaf proposes load to the internal node. The internal node, then chooses only the maximal proposing transfer. 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 depicts two settings. The setting is the same as in the example above in the Push-Pull Sum section, with the difference that each node has a load value assigned, instead sum and weight values. The dashed directed edge in this example represent a proposal from one node to another. The solid directed edge represents the actual load transfered from one node to another. We again consider the nodes *A, B, C and D*. Each node looks for its minimal loaded neighbor. For nodes *B, C and D* this is node *A*. For node *A* this is node *B*, since node *B* has more load than node *A*, node

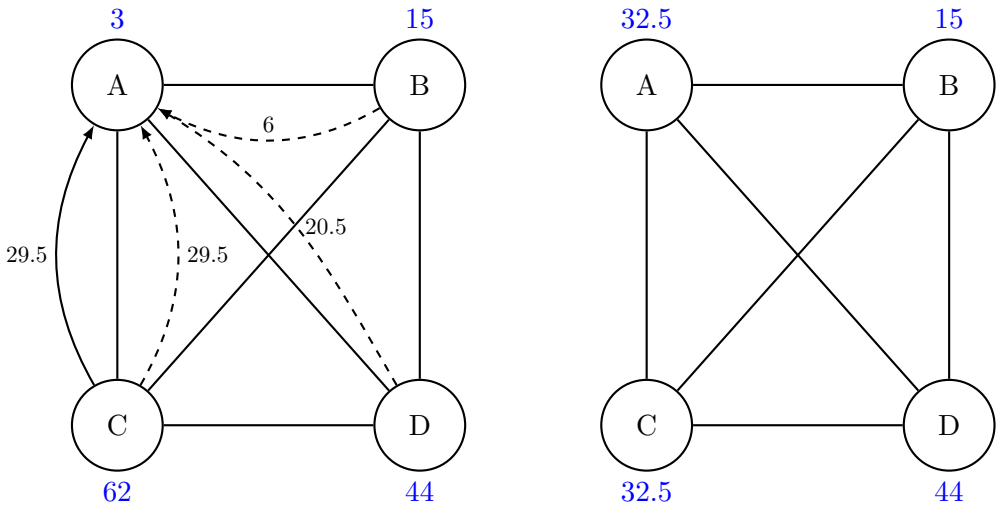


Figure 4: Left: Initial Setup; Right: Result

A does not send an transfer proposal to node B. 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 each transfer proposal and accepts node C's transfer proposal, since node C proposes the maximal 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 depicts the setting after round 1. Node A and C each have a load of 32.5, the loads of nodes B and D are unchanged. The mean squared in the beginning of round 1 is at 542.5, in the end of round 1 the mean squared error decreases to a value of 107.375, which is close to a fifth of the initial mean squared error.

3.4 Adaptive Threshold Push-Pull Sum Algorithm

The Adaptive Threhsold Push-Pull Sum algorithm is composed of different ideas and elements of the two former algorithms, extended by the idea of adaptive thresholding. The Adaptive Threshold Push-Pull Sum consits of different procedures. In the *Check-ThresholdsRequestData* each node i chooses a subset RN of $\log_2(|neighborhood|)$ random neighbors. This is to enhance the chance of a good load transfer to happen. If we only choose one neighbor the chance is lower to find an optimal or good neighbor to execute a load transfer. Then the load difference between the node i and each node in RN is computed and checked if the load difference is bigger than some threshold θ. The threshold is computed in the *CalculateThresholds*-procedure. The threshold θ is computed as $k * \sqrt{MSE_r - 1}$ where k is some factor to adjust the sensitivity of the threshold. A larger k means the threshold is more sensitive meaning that less nodes are eligiable for load transfer and respectively a lower k means that the boundary is less strict, thus more nodes are eligiable for load transfer. This condition prevents load transfers with low effect, and guarantees a selection of nodes with effect above the given threshold. The first eligiable node receives the sum and the weight in form of a push action ($\frac{s_i,r}{2}$) for the sum and ($\frac{w_i,r}{2}$) for the weight. The *ResponseData*, and the *Aggregate*-procedure are analoge to the classic Push-Pull Sum algorithm. The

push and the pull mechanism are directly taken from the Push-Pull Sum algorithm, and are combined with a adaptive threshold mechanism, thus the name Adaptive Threshold Push-Pull Sum algorithm. Two ideas are derived from the Single-Proposal Deal-Agreement-Based algorithm. The idea of the conditional load transfer is analog to the Single-Proposal Deal-Agreement-Based's one, with the only difference that the difference of load between the two negotiating nodes should not be larger than 0, but bigger than a threshold θ. Furthermore, instead of initiaing a load transfer with the maximal loaded 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 avoid that each node looks through the whole set RN and needs to evaluate each proposal, instead the first node proposing a load transfer is accepted as a transfer partner.

Although not shown, the Adaptive Threshold Push-Pull Sum algorithm is expected to hold the mass conservation property, as it converged to the true ground truh for all the experiments, similiar to the Push-Pull Sum algorithm. One indication for this is that the push-pull mechanisms are adapted unchanged, and these operations are the only operations that let nodes transfer or receive node in the algorithm. Also the Adaptive Threhsold Push-Pull Sum algorithm is a stochastic algorithm as it chooses a subset of neighbors randomly.

3.4.1 Example

Figure 5 depicts a similar setting as described in the section 3.2.1. Now, according to the Adaptive Threshold Push-Pull Sum algorithm, each node chooses $\log_2(|neighborhood|)$ neighbors. For this setting, this means that each node chooses 2 neighbors. These neighbors are added to the set $RN_{i,1}$ for each node i. For instance, node A computed $RN_{A,1}$ as $\{B, C\}$. The load differences between node A and nodes B and C are computed respectively. In this example, k is 0.01, thus the threshold θ in this case is $0.01 * \sqrt{542.5} \approx 0.23$. Since both load differences between

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

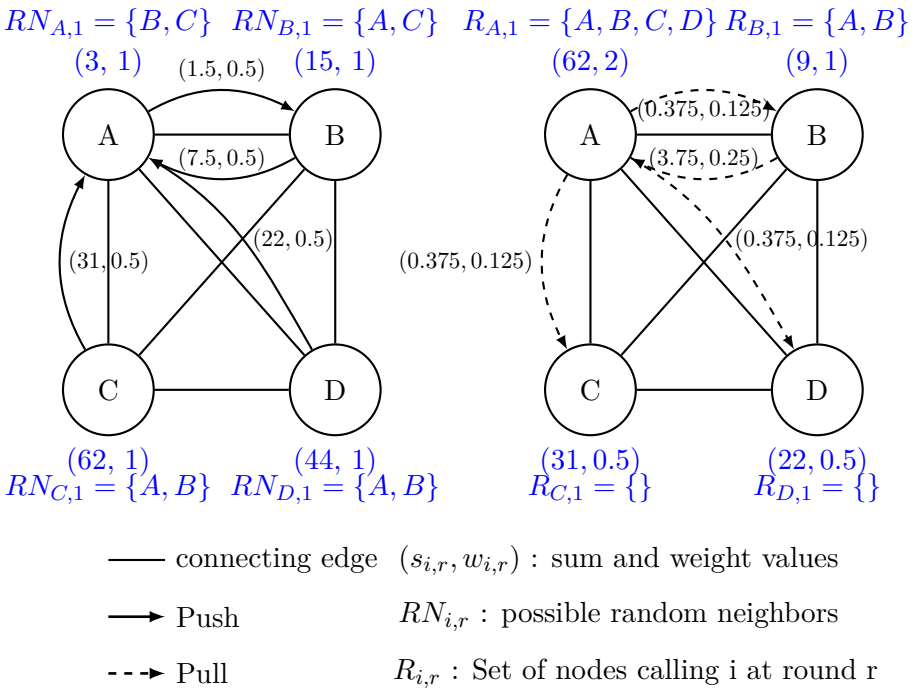


Figure 5: Left: Push actions; Right: Pull actions

nodes A and B, and nodes A and C pass this threshold, both nodes are eligiable to propose to transfer load with node A. In this scenario node A chooses node B as a transfer partner and pushes half of its sum 1.5 and weight 0.5 to node B and itself. Accordingly, for each node respectively. Similar to the example in section 3.2.1 the nodes execute the pull operation. The result is depicted in figure 6. The mean squared error dropped from 542.5 initially, to 251.86.

3.4.2 Aspired Outcome

Thinking of the simulation results presented in [6] the discrepancy between the MSE reduction per round for the different topologies show that the algorithms perform either very well or medicore to bad. The main idea and perspective designing this algorithm was to find a compromise solution to that, to provide a better adaptability for different network topologies.

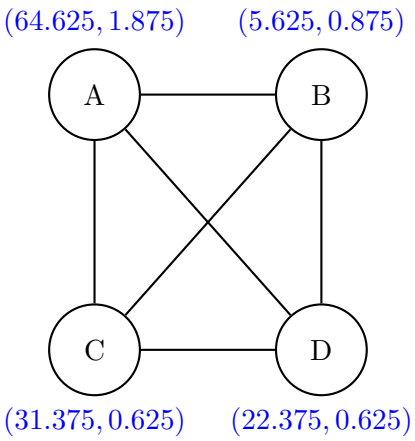


Figure 6: Setting after round 1

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

Table 1: Overview over example outcomes

The final results of the first load balancing step can be taken from table 1. The Push-Pull Sum algorithm shows the lowest mean squared error after round 1, followed by the Deal-Agreement-Based algorithm and the Adaptive Threshold Push-Pull Sum algorithm. 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 threshold adjusts.

4 Topologies

The simulations were conducted for six distinct network topologies. Each network has 2^{10} (1024) nodes. The behaviour 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 the *Complete graph*, *Torus Grid Graph*, *Ring Graph*, *Star Graph*, *Lollipop Graph* and *Ring of Cliques*. In the following the topologies including there 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 or also refered to as 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 (the maximum number of edges per node for undirected graphs is N, when self-loops are allowed), thus is 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 same degree and connectivity, algorithms can treat all nodes uniformly. Complete graph topology is used in financial trading systems and military communications, where high reliability and low latency are critical. It ensures optimal routing paths between nodes, minimizing delays [8].

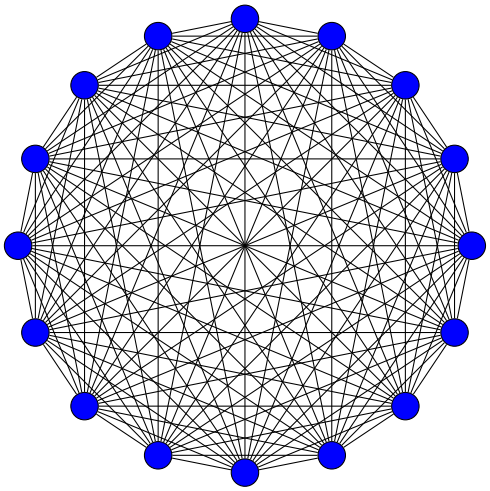


Figure 7: Complete graph: network size 16

4.2 Torus Grid Graph

The two-dimensional Torus Grid graph or also refered to as $k \times m$-Torus graph or $T_{k,m}$ is a two-dimensional mesh with wrap around edges as depicted in figure 8 [9]. k denotes the height and m denotes the width of the grid. The Torus Grid graph has $2 \times k \times m$ edges (if $k, m > 2$) and $k \times m$ nodes and is a regular graph, where each node has a degree of 4. The diameter of a Torus Grid graph is $\frac{min(k,m)}{2}$. Tori are scalable, as the number of connections per node is constant, regardless of the graph size. In the previous research "Comparative Analysis of Load Balancing Algorithms in General Graphs" [6], the simulation results showed to not vary over different network sizes. Torus topology is prevalent 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. Torus topology is commonly found in Fiber Distributed Data Interface (FDDI) and Synchronous Optical Networking (SONET) for local and metropolitan area networks. [8].

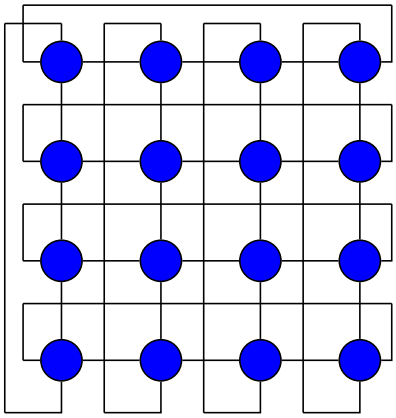


Figure 8: Torus grid graph: network size 16

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 which is closed, meaning that the first and the last nodes of the Path graph 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 $\lfloor \frac{N}{2} \rfloor$. The uniform structure ensures no single node has an advantage, simplifying algorithm design and guaranteeing fairness in load balancing. Most of the Ring structures use a token-based communication model. The node that holds the taken may transmit data. Most of current high speed LANs have a Ring topology [10]. The advantage of an Ring graph is that when faults occur the troubleshooting is relatively easy. However, a single outage of a node may disrupt the whole network activity.

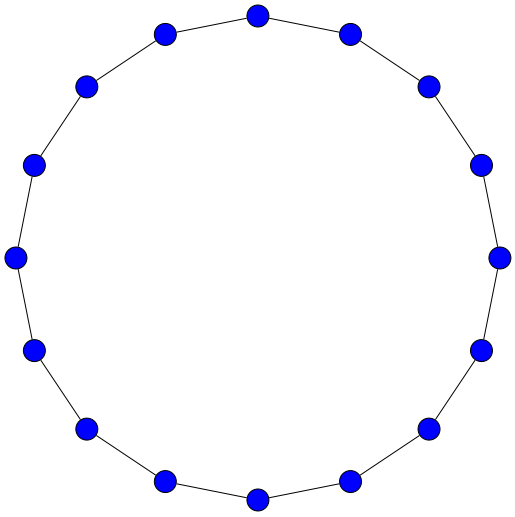


Figure 9: Ring graph: network size 16

4.4 Star Graph

A Star graph S_N as depicted in figure 10 is a bipartite graph [11], structured like a tree graph with one internal node and k-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 (path from one leave through internal node to another leave). Regarding load balancing 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. However, 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]. The challenge 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.

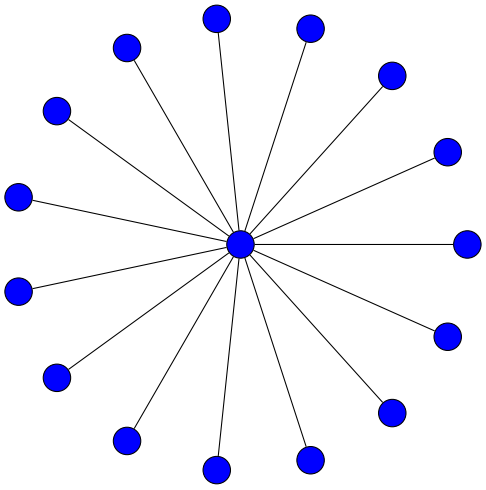


Figure 10: Star graph: network size 16

4.5 Lollipop Graph

A (k, m)-Lollipop graph $L_{k,m}$ is a graph that consists of a Complete graph and a Path graph as depicted in figure 11. The Complete graph and the Path graph are connected by a bridge node, thus a single edge. The (k, m)-Lollipop graph is composed of a Complete graph 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].

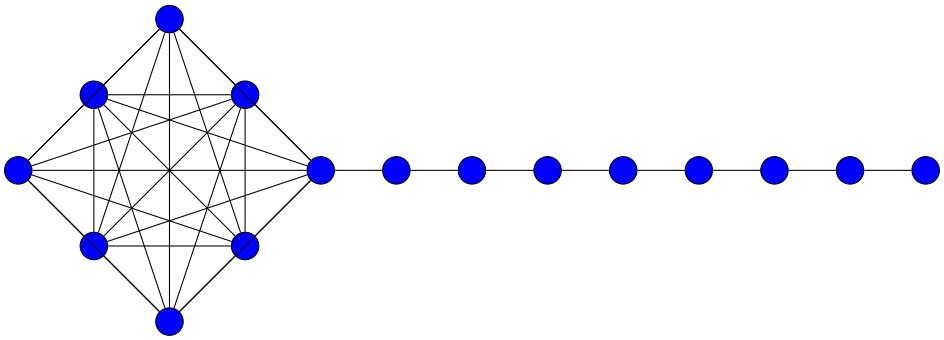


Figure 11: Lollipop graph: network size 16

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.

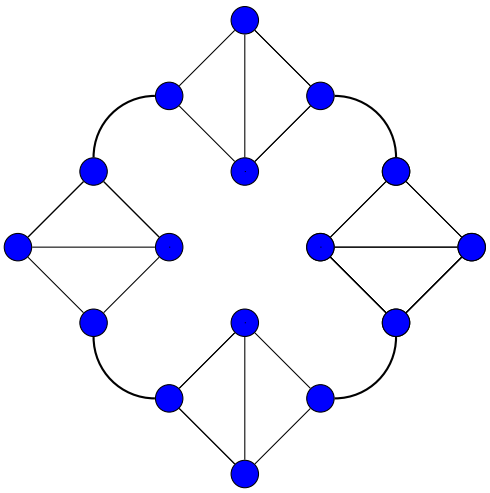


Figure 12: Ring of Cliques: network size 16

4.7 Expected Outcome

Many load balancing algorithms perform well on 2D Tori due to the focus on local redistribution, thanks to the wrap around edges eliminiating boundary effects, ensuring balanced redistribution across entire grid.

For the Ring graphs, where unlike complete or torus graphs, the sequential nature of data transfer makes it slower for loads to propagate across the entire Ring. Load movement in the Ring graph is sequential, making it slower and less flexible than the Complete graph.

Star: For the Push-Pull mechanism the star graph is an advantage, since each push action contains a pull action. All leaves request data from the internal node, so the internal node acts here as an point of redistribution. However for the Deal-Agreement-Based protocol some mechanisms like "if condition" act as a contraproductive way of load balancing.

Lollipop: Algorithms must account for the highly connected complete graph (dense communication) and the linear path graph (sequential communication), which introduces a mixed topology challenge. The node linking the complete graph and the path graph often becomes a bottleneck since it bridges two vastly different connectivity regions. In the complete graph, load balancing is efficient due to the high connectivity, enabling rapid redistribution among nodes. Load movement in the path graph is sequential, making it slower and less flexible than the complete graph. Interesting because dense head and sparse tail. For example, diffusion-based approaches work well in the head but may require additional iterations in the tail. We saw that in "Comparative Analysis of Load Balancing algorithms in General Graphs" [6], once we changed the path size in relation to same complete graph, the DAB was faster in reducing error for the first 50 rounds. **(TODO: Do the simulations over again with different path and complete graph size and compare.)**

Ring of cliques: Within each clique, balancing is fast due to the dense connectivity. Load redistributes efficiently among the clique's nodes. Balancing between cliques depends on the single shared node linking each pair of cliques. This structure ensures sequential redistribution across the ring. The shared nodes act as gateways, controlling the flow of load between cliques. Overloading these nodes can cause bottlenecks.

5 Implementation and Technology Stack

Different technologies were used to achieve the underlying results. Simulations are conducted using *PeerSim* a simulation framework for *Java*. As an artefact of a simulation a output file containing different simulation parameters, settings and metrics is generated. The data analysis part of this project is mostly 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 concuct simulations and post simulation analysis. For plotting purposes *matplolib v.3.9.1* is used and for handling datas and data analysis *scipy v.1.14.1* is used.

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

5.2 Simulation Framwork

PeerSim is a simulation tool developed for the Java programming language, which is composed of two engines. The cycle-driven engine and the event-driven engine. We

chose PeerSim for our simulations, since PeerSim is suited for large scale Peer-to-Peer simulations. In previous projects, we were able to conduct simulations up to 2^{14} nodes and could probably push the boundaries by scaling to even higher dimentsions. PeerSim is designed to use pluggable components, implemented in interfaces, which are intuitive to use generally. A simulation in PeerSim can be realized following four simple steps. First, in order to probably set up a simulation a **configuration file** is needed. Here, simulation parameters may be defined, like the network size, the load balancing protocols that are beeing simulated etc.

```
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 parts of an configuration file used in our project. This configuration sets up a simulation of the Push-Pull Sum algorithm for a network with 1024 nodes (**lines 2 and 3** in Listing 5.1) for 100 rounds (**lines 6 and 7** in Listing 5.1). From **lines 10 to 12** the Push-Pull Sum protocol is loaded. The way we implemented the algorithms, one algorithm contains of at least two Java classes, the *<ProtocolName>Protocol.java* and the *<ProtocolName>Observer* and a shared *loadBalancingParameters*-class containing parameters like the cycle or topology specific parameters as depicted in figure **??**. Finally at **line 15** the controll class is declared and used at lines **18 and 19**, with the parameters of type **protocol** and **numberOfCycles**. And so step 2 and three of creating a simulation is also handled by **selecting the protocols to simulate** and **select control objects**. Following that, the last step is to **invoke the simulator class**, which is *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 variety of classes and interfaces for simulation. For 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.

In PeerSim, nodes are implemented as containers that hold various protocols. Each node is uniquely identified by an ID and interacts with the Linkable interface, which provides access to neighboring protocols. 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.

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 **??** depicts a process model, modelling the methodic chosen to get from the creation of experiments to the plots and data analysis part. 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. These configuration files are read with a Java script and each node is assigned a initial load value. Then the simulations are conducted. Each simulation outputs a file containing the simulation results, mainly the mean squared error per round, the loads per round and the configuration of the network (e.g. which topology chosen, network size...). The simulation results are averaged per round and then analyzed. Out of the simulation results plots are generated, showing the mean squared error reduction per round in log-log or log-linear graphs. Also the technique of model fitting is applied.

6 Simulation Outcomes

When analyzing load balancing algorithms we consider the stability and efficiency of the algorithms. The aspect of stability is expressing how well a algoithm averages any initial load of an network [4]. In the following this is tested 1. by conducting 30 different experiments for each topology with different initial error values in the network 2. by choosing six different topologies (some with different symmetry) to simulate in order to test the adaptability. The efficiency shows how fast a 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.

In the following the load balancing algorithms are abbreviated. The Single-Proposal Deal-Agreement-Based algorithm is abbreviated as DAB, the Push-Pull Sum algorithm is abbreviated as PPS and the Adaptive Threshold Push-Pull-Sum algorithm is abbreviated as ATPPS. The simulation outcomes are presented in plots, mostly log-log or log-linear graphs (logs for the MSE-data are base 10), since the MSE values vary a lot (across different magnitudes) over the 100 rounds of simulation. Figures **??**, <slopes images referencing> depict the slopes for three distinct regions (of the x-axis) and the general slope over all the 100 rounds. For each algorithm the function that distributes the MSE data over the rounds is model fitted with the models of linear regression, polynomial regression, exponential decay or logarithmic regression, depending on the best-fit.

Linear Regression: The linear regression fits $MSE_r = m * r + b$ to the data. Where

m is the slope: $\frac{\Delta MSE}{\Delta r}$ ($\frac{MSE_{r_2} - MSE_{r_1}}{r_2 - r_1}$). In the case of load balancing the slope is mostly negative, meaning that for each round, the MSE_r decreases in comparison to the $MSE_r - 1$, the mean squared error of the previous round. b is the mean squared error, in round 0, so in that round where the network is initialized in our experiments. In round 0 the mean squared error is calculated with the unbalanced network, so MSE_0 is the initial mean squared error, where no load balancing is applied yet.

Polynomial Regression: The polynomial model $MSE_r = a_0 + a_1 * r + a_2 * r^2 + a_3 * r^3 + ... + a_n * r^n$ is a model where the mean squared error 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 fits $MSE_r = a * e^{-br}$ to the data. Exponential models capture the initially steep drop in mean squared error at early rounds, followed by slower reductions in later rounds. The exponential decay model is also an non-linear model. The initial value of the mean squared error is captured by a. In the initializing round where $r = 0$, $MSE_r = a * e^{b*0}$ evaluates to $MSE_r = a$. Larger a values indicate larger initial error in the network. The decay rate is captured by b. The b value determines how quickly the MSE decreases per round r. For the exponential decay model b is negative. A larger negative value for b indicates a faster error reduction in the network, while values closer to 0 indicate slower error reduction.

Logarithmic Regression: The logarithmic regression models data as $MSE_r = a + b * \log(r)$. a is the initial mean squared error, when $\log(r) = 0$. b is the rate of reduction per unit of logarithmic increase in rounds. A large positive b decreases the error faster.

(TODO: The polynomials before chapter ROC are in the wrong order constant term is the last term and so on. Correct that and the analysis of that.)

6.1 Complete Graph

Figures **??** show the mean squared error reduction over the rounds for the three load balancing algorithms simulated on the complete graph on a log-log graph.

Deterministic Agreement-Based Algorithm (DAB): One can observe that the black line shows constant mean squared error reduction, indicating that this algorithm does not improve the load balance by much over time. The DAB performs poorly due to the rigid determinism; DAB uses a fixed, deterministic load redistribution rule, where one node looks for the minimal loaded neighbor and proposes to that node. Since for the complete graph each node are interconnected, the neighbors with the minimal loads are the same for every node. Thus, the deterministic strategy proposing to the minimal neighbor does not distribute load in suboptimal ways. The number of load transfers in this scenario per round is very limited, resulting in poor mean squared error reduction. The main observation is that deterministic methods can struggle in high-connectivity environments because they fail to exploit randomness or adaptivity. Figure **??** shows the exponential regression fit for the MSE data when the DAB algorithm as a load balancing algorithm is applied to the network. The fitted curve is expressed by the equation $MSE = 844.63 * e^{-0.01*r}$. 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. In summary, the exponential regression model captures the slow and steady MSE reduction of DAB over rounds, highlighting its slow but consistent convergence behavior.

Push-Pull Sum algorithm (PPS): The PPS-curve decreases gradually in mean

squared error, showing significant error reduction over time. Since all nodes are equally connected, no structural constraints slow down the process of pushing and pulling load from neighbors. Over time, repeated randomized exchanges smooth out load imbalances, leading to a exponential decay in mean squared error. Classic PPS does not differentiate between "highly imbalanced" and "near-balanced" nodes. Load transfers happen uniformly, even when the system is close to equilibrium, which slows convergence in later rounds. Figure **??** visualizes the MSE data of the PPS load balancing algorithm, plotted for the rounds 10 to 80 and fitted with the exponential regression model. The best-fit follows the equation $MSE = 2530.41 * e^{-0.9*r}$. The graph uses a logarithmic scale for the mean squared error. The fitted curve aligns closely with the MSE data, so the exponential model accurately describes the behaviour of the algorithm for this region. The decay rate of -0.9 indicates a steep decrease in MSE. This suggests that the PPS load balancing algorithm reduces the MSE in a exponential way.

Adaptive Push-Pull Algorithm (ATPPS): The ATPPS-curve decreases faster than the red curve, reaching lower MSE values earlier. Adaptive Thresholding: Nodes adaptively decide when and how much load to exchange based on thresholds. For example: Nodes with significantly higher load push more load to neighbors. Nodes closer to balance reduce their interactions. Adaptive strategies reduce redundant exchanges when the system is near equilibrium, focusing resources on nodes with the largest imbalance. In a complete graph, adaptive decisions propagate quickly because every node can directly communicate with others. The adaptive mechanism leverages the symmetry and high connectivity of the complete graph to reduce MSE more quickly and efficiently. In Figure **??** the exponential regression fit is visualized for the MSE data of the ATPPS load balancing algorithm graph in the complete graph for the rounds 10 to 65. The fitted curve is expressed by the equation $MSE = 4309.94 * e^{-1.06*r}$. 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.

Figure **??** 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 mean squared error decreases over rounds. PPS and ATPPS have steep negative slopes in the start region, namely the 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 errror reduction in later rounds (round ∼80-100).

6.2 Star Graph

For the star graph the push-pull mechanism-based protocols seem to perform well in MSE reduction, while the DAB algorithm seem to struggle with in this topology as depicted in figure **??**

Deal-Agreement-Based Algorithm: DAB shows a much slower MSE reduction compared to the push-pull 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 topology, likely due to its deterministic nature and lack of dynamic adaptability. In the star topology all the leaves have the central node as a partner, and thus each node chooses the same node as an transfer partner. 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). As seen in figure **??** 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 load balancing algorithm to balance the network within 100 rounds to a satisfying magnitude.

Push-Pull Sum Algorithm and Adaptive Threshold Push-Pull Sum Algorithm: Both push-pull-based protocols 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 (as of round 45) before the graph plateaus. Figures **??** and **??** show the fitted curves for the MSE data of the PPS and ATPPS load balancing algorithms respectively. The best-fit model for the MSE data of the PPS load balancing algorithm for the rounds 10 to 45 follows the equation $MSE = 29794.60 * e^{-1.39*r}$. 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 very fast error reduction in the network, especially if compared to the DAB load balancing algorithms ability to reduce the error for star graph. The ATPPS load balancing algorithms 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 may not sufficiently impact performance under these conditions.

Overall the scenario is similiar to the complete graph, where the DAB reduces the error exponential with a very slow rate, while the push-pull based algorithms perform a very fast error reduction. This behaviour is captured in the heat map of figure **??**. While the DAB load balancing algorithm reduces the error in average per region -3.75 ± 0.55, the push-pull 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 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 load balancing algorithms deterministic nature and lack of adaptability lead to slower convergence and less effective load balancing, as reflected in its consistently shallow slopes.

6.3 Ring Graph

The DAB curve in figure **??** 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 PPS-based algorithm as depicted in figure **??**. 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 topology, where load balancing is already well-facilitated by regular communication patterns. 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, local differences might not vary significantly enough to make the threshold mechanism advantageous for each neighborhood. In the classic 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. The DAB algorithm 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 10 to 60 follow the equation: $MSE_r = 1.72 \times 10^{-05} r^4 - 2.30 \times 10^{-3} r^3 + 0.19r^2 - 5.99r + 114.83$ (figure **??**). The intercept (a_0 term) is very close to zero, meaning the starting MSE value is almost entirely determined by higher-order terms. The small negative coefficient of the a_1-term indicates a slight initial linear decrease in MSE as rounds progress, contributing a modest downward slope. The positive quadratic term (a_2 term) suggests a curving trend. This term provides more flexibility to account for changes in the decay rate of the MSE early in the rounds. The cubic term (a_3 term) adds further curvature but with a decreasing influence with proceeding rounds. It could model an inflection point in the MSE trend. The large positive coefficient of the cubic term ($a - 4$ term) reflects the steep decay behavior of MSE over a significant portion of the range. A similiar behaviour is shown for the PPS and ATPPS curves which are fitted for the model: $MSE_r = 2.99 \times 10^{-05} r^4 - 0.5 \times 10^{-2} r^3 + 0.32r^2 - 9.68 + 166.30$ (figure **??**)for the PPS and the model: $MSE_r = 3.04 \times 10^{-05} r^4 - 0.5 \times 10^{-2} r^3 + 0.32r^2 - 9.64r + 161.86$ (figure **??**)for the ATPPS.

6.4 Torus Grid Graph

Figure **??** shows the MSE reduction over rounds on a Torus Grid Graph, plotted on a log-log scale. At the beginning all three algorithms start with similar MSE values around 1000. The DAB algorithms curve appears to have a slightly slower initial reduction compared to PPS algorithms curve and ATPPS 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 PPS based algorithms (figure **??**). PPS's curve and ATPPS's curve maintain nearly identical performances during the middle phase so the rounds 8 to 40, reducing MSE at a similar rate, with a slope of -1 for the PPS curve and -0.83 for the ATPPS curve. DAB shows a noticeable improvement over 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 PPS-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 introduced in this paper achieves a compromise solution in this scenario. It seems to perform better than the classic 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 impactful.

The uniform neighborhood structure of a torus ensures that DAB's deterministic decisions (e.g., always choosing the minimal neighbor) are consistently effective across the graph. The protocol doesn't suffer from random noise introduced by probabilistic neighbor choices, making it inherently 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. The PPS-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 fitted polynomial curve of degree 5 matches the MSE data for DAB effectively, capturing the nonlinear dynamics during rounds 10 to 39 following the equation: $MSE_r = -1.35 \times 10^{-6} r^5 + 1.89 \times 10^{-4} r^4 - 0.01 r^3 + 0.30 r^2 - 4.6 r + 34.10$. This suggests that in the early 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 with equation: $MSE_r = -6.01 \times 10^{-06} r^3 + 1.66 \times 10^{-3} r^2 - 0.16r + 6$. The fitted polynomial curve of degree 4 fits the MSE data well in later rounds. 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 behaviour for the PPS and ATPPS are similar to that of the DAB curve expressed for rounds 10 to 40 by the equations: $MSE_r = -5.54 \times 10^{-06} r^5 + 7.65 \times 10^{-4} r^4 - 0.04r^3 + 1.16r^2 - 16.81r + 112.86$ for the PPS 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. The r^5 term ($88.16 * r^5$) dominates the polynomial at higher rounds. This reflects the initial high variance (high MSE) at the start of the process when nodes have not yet balanced their loads effectively.As rounds progress, the lower-degree terms (e.g., r^4 and r^3) become more influential. These represent the gradual stabilization of the MSE as the load balances across the graph. The negative coefficients for r^4 ($-12.52r^4$) and lower powers ($-0.03r^2$) show the decline of MSE over time as the algorithm pushes load differences to zero. The values slightly differ for the other algorithms but the idea is the same, as the fitted models for each algorithm are structured the same. The equations that describe that are fitted to the MSE data for the rounds for the rounds 41 to 100 for the PPS is:$MSE_r = -1.15 \times 10^{-5} r^3 + 3.205 \times 10^{-3} r^2 - 0.33r + 13.72$ and for the ATPPS is: $MSE_r = -9.99 \times 10^{-6} r^3 + 2.8034 \times 10^{-3} r^2 - 0.28r + 11.29$. Here the structure is also similar, however described with less terms (degree 3).

6.5 Ring of Cliques

6.5.1 32x32 Ring of Cliques

In the early rounds, PPS and ATPPS exhibits the fastest MSE reduction showcased by a very steep downwards trend between rounds 1 and 5 in figure **??** a). Both PPS-based approaches start with faster convergence rates compared to DAB. The

initial steep downwards slope of -200 for each PPS-based algorithm as depicted in figure **??** within the first 5 rounds is due to the unbalanced state of each clique. The PPS-based approaches has shown to be outperforming the DAB algorithm for complete graphs. While the Cliques are still unbalanced the PPS-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 seem to be -48 compared to ~ -2 for the PPS-based algorithms. Nodes within cliques tend to converge quickly internally due to their dense interconnections for the PPS-based algorithms. However, load balancing between cliques, especially when the nodes select for random neighbors is slower, creating bottlenecks for global convergence. 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. In the Ring of Cliques, this mechanism prioritizes communication between cliques (where differences in loads are more significant) over redundant communication within cliques (where loads are already close to balanced). This helps speed up inter-clique convergence. And so it acts as a compromise solution between the PPS and the DAB again achieving results close to them of the DAB algorithm.

The polynomial fit for the DAB algorithm is: $MSE_r = 1.04\times10^{-6}r^4 - 2.941\times10^{-4}r^3 + 0.03r^2 - 1.66r + 45.30$. Thus the Mean Squared Error (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: $MSE_r = -0.05r + 24.70$. 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. This is because:

PPS focuses on a push-pull mechanism that is inherently less targeted, 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: $\log(MSE_r) = -7.59 \log(r) + 43.26$. By exponentiating this equation, the relationship between the Mean Squared Error and the number of rounds is: $MSE_r = 10^{43.26} * r - 7.59$. The steep negative slope with 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 rings is increased from 2^5 to 2^7 in subsection 6.5.2 and thus the number of cliquesize of each ring is decreased to 2^3. Subsection 6.5.3 covers the experiment where the cliquesize is increased and the number of rings is decreased in the same magnitude as discussed before.

6.5.2 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 PPS-based algorithms as seen in figure 13. All the load balancing algorithms show an rapid decrease of error in the network, indicated by the steep negative slopes as visualized in figure 17. The PPS-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 13 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. **(TODO: One sentence about the values of 32 32 and the magnitude to see if this structure benefits any algoithm or nah)**.

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

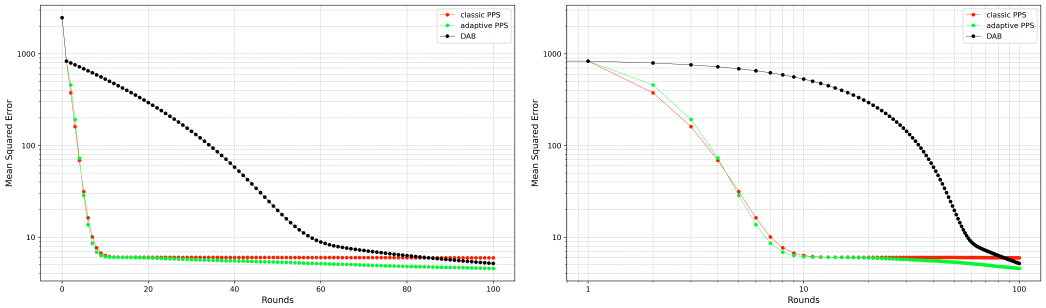
6.5.3 8x128 Ring of Cliques

Increasing the clique size and reducing the number of cliques favors the push-pull based algorithms. The PPS-based algorithms 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 PPS 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 PPS 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 by the slopes in the Middle regions. The PPS-based algorithms reduce the error by ~ -2 while the DAB shows a still very high value of ~ -30. The reason behind this behaviour is due to the impact of the clique sizes. The PPS-based algorithms achieve good performances in reducing error in this scenario compared to the DAB which struggles in the context of dense graphs

as it seems. The steep decrease and the low MSE values after rounds 5 and 10 stems from the fast error reducing in the cliques while still having the change to spread loads from clique to clique through the briding nodes. Compared to the experiment with 128 cliques with size 8 and the in 6.5.2 the error after rounds 100 drops 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 PPS based algorithms achieved a broadly balanaced state of the network after round 10 dropping the error to lower 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 15. 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 $MSE_r = 6.07 \times 10^{-5}r^2 - 0.02r + 6.39$. 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 $MSE_r = -3.33 \times 10^{-3}r^3 + 0.63r^2 - 40.23r + 872.75$ while in later rounds the power relation is a bit complexer, expressed by a fourth degree polynomial following the equation $MSE_r = 2.96 \times 10^{-6}r^4 - 9.97 \times 10^{-3}r^3 + 0.13r^2 - 7.16r + 161.73$.

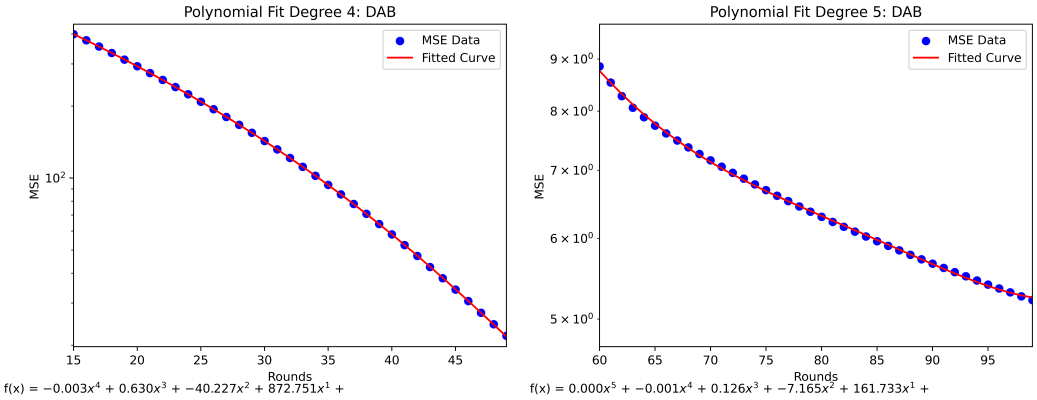
Snippets of the simulation outcomes after the 100th rounds of each load balancing algorithm verify that within the cliques the load is balanced, however the inter clique connection seems to be pending. Listing 6.1 shows two connected cliques of the ATPPS simulation outcomes in round 100. While the first 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 aligh, however the averages do not alighn 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 13: Ring of Cliques: mean squared error per rounds (left: log-linear; right: log-log)



(a) **(b)**

Figure 14: Polynomial Regression Fitting: left: Rounds 20 to 50; right: Rounds 55 to 100

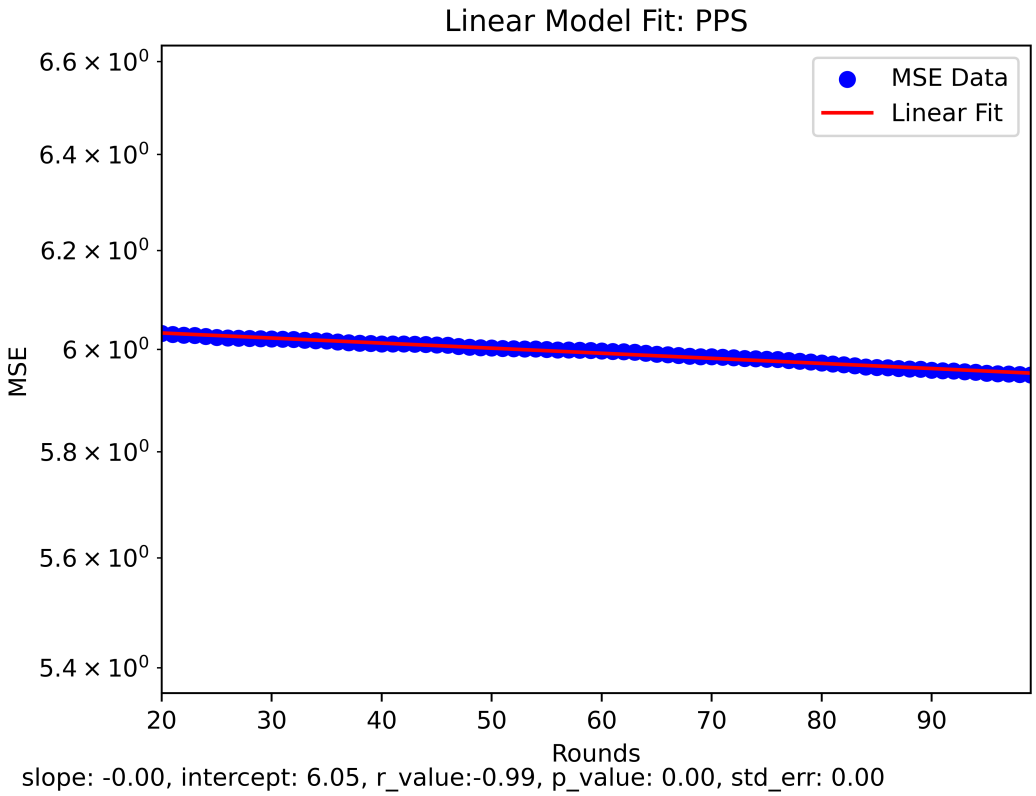


Figure 15: Polynomial Regression Fit: PPS

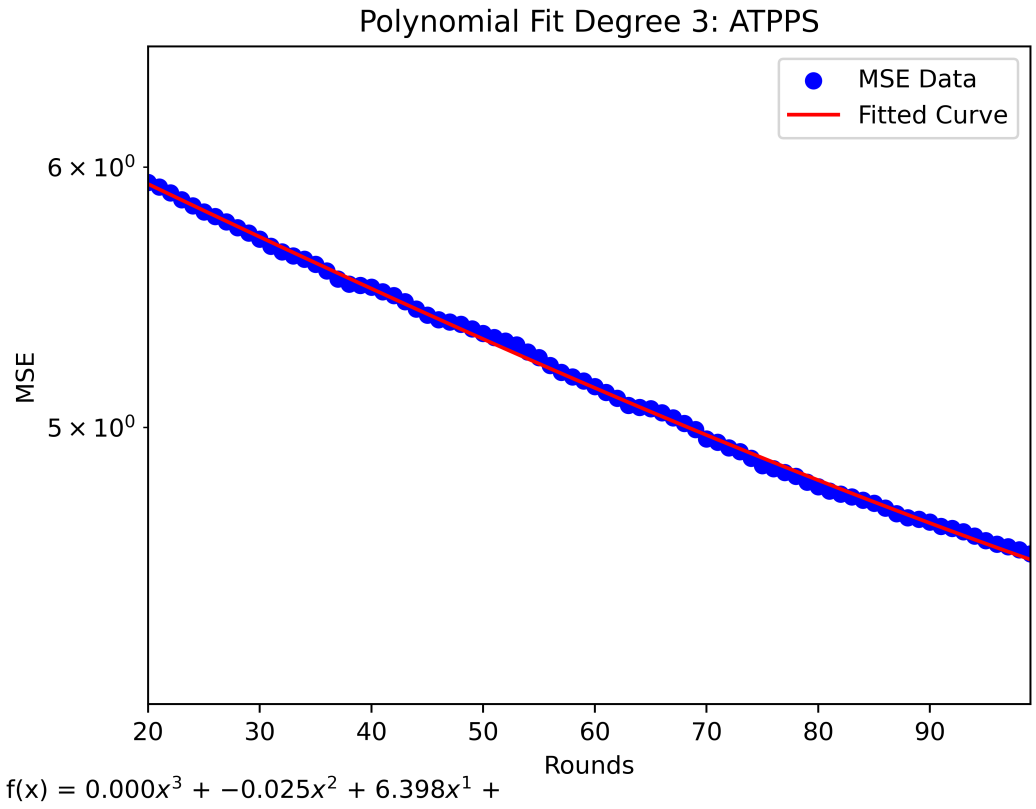


Figure 16: Polynomial Regression Fit: PPS

```
# 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
```

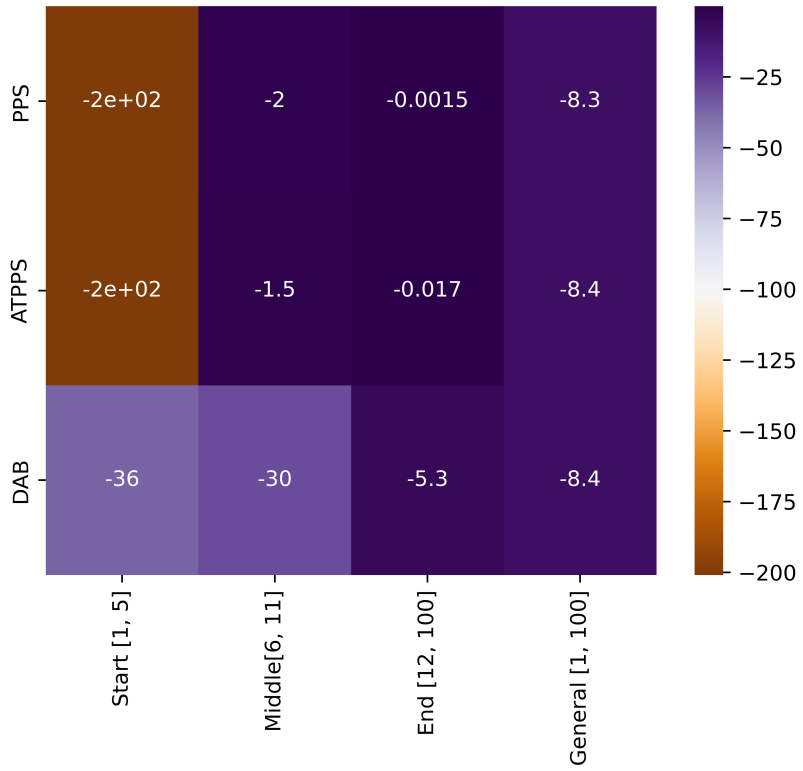


Figure 17: 8x128 Ring of Cliques: heat map of slopes per region

```
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: exp2

6.6 Lollipop Graph

6.6.1 (512, 512) Lollipop Graph

Figure **??** shows the three curves of the load balancing algorithms. The ATPPS curve slightly outperforms the PPS curve. The DAB curve lacks to perform as good as the other two load balancing algorithms in reducing the error. The superior performance

of PPS compared to DAB in the lollipop graph can be explained by the 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 PPS based algorithms. The path region consists of limited connectivity slows down information propagation and balancing for the PPS based algorithms and favors the deterministic algorithm DAB. The initial discrepancy in the first 10 rounds is due to the potentialy fast decrease of error for the PPS in complete graphs. The DAB load balancing algorithm struggles to perform good in this scenario as observed in section 6.1. The 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 complete graph. Load imbalances in the clique region take longer to converge because DAB doesn't exploit the random spreading mechanism of PPS-based algorithms In the initial phase PPS rapidly reduces the MSE, demonstrating strong initial convergence. This quick decrease is due to PPS's proportional load-splitting behavior, which diffuses load across neighbors efficiently in the highly connected clique portion of the lollipop graph. In this region the PPS based algorithms achieve a steep downwards slope with value -130 between rounds 1 to 7, while the DAB reaches nearly half of it with value -67 (figure **??**) In the mid-to-late phase the slope flattens for both of the PPS-based algorithms, indicating slower convergence this is also indicated by the slopes droping to as low as $-0.12to \sim -0.9$. In this phase, the path section of the lollipop graph dominates the residual imbalances. The adaptive PPS achieves nearly identical behavior to the PPS in the early rounds, as it starts with similar proportional strategies. Adaptive PPS 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, 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: $MSE_r = 8.44 \times 10^{-07} r^4 - 2.52 \times 10^{-4} r^3 - 0.03r^2 - 1.64r + 56.68$ and the ATPPS model fit follows the equation: $MSE_r = 8.69 \times 10^{-07} r^4 - 2.56 \times 10^{-4} r^3 + 0.03r^2 - 1.62r + 54.48$.

To see the impact of the pathsize and cliquesize on the simulation results respectively the simulation results, the simulations are conducted with a less nodes assigned to the clique in subsection 6.6.2. 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 6.6.3.

6.6.2 (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 **??**. DAB, on the other hand, has a slightly slower initial convergence compared to PPS showcasing a slope of -110. Compared to the slopes of the previous experiment where the pathsize and the cliquesize 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 moddle 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, and 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 PPS

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.

A polynomial fit has been applied to the MSE data of all three load balancing algorithms. 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 and the best fit polynomial is a polynomial of degree 5. The fitted model of the DAB MSE data follows the equation: $MSE_r = 3.46 \times 10^{-06} r^4 - 1.12 \times 10^{-3} r^3 + 0.14 r^2 - 7.69 r + 190.78$ and the one of the ATPPS follows the equation $MSE_r = 1.59 \times 10^{-06} r^4 - 4.74 \times 10^{-4} r^3 + 0.054 r^2 - 2.94 r + 70.59$. The polynomial fitted to the PPS MSE data follows the equation: $MSE_r = -3.72 \times 10^{-8} r^5 + 1.31 \times 10^{-5} r^4 - 1.85 \times 10^{-3} r^3 + 0.13 r^2 - 4.96 r + 84.91$.

6.6.3 (896, 128) Lollipop Graph

Now that the more nodes are assigned to the clique and taken from the path size compared to the initial experiment where the clique and the path had the same number of nodes, the discrepancy between the PPS-based and the DAB algorithms is more obvious, as shown in figure **??**. The slopes also indicate this. In the Start region the PPS-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 **??**) 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: $MSE_r = -1.936 \times 10^{-4} r^3 + 0.05 r^2 - 5.33 r + 745.95$. The polynomial for the PPS-basde algorithms follow the equation: $MSE_r = 2.00 \times 10^{-7} r^4 - 6.01 \times 10^{-5} r^3 + 6.95 \times 10^{-3} r^2 - 0.39 r + 13.44$ 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.

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

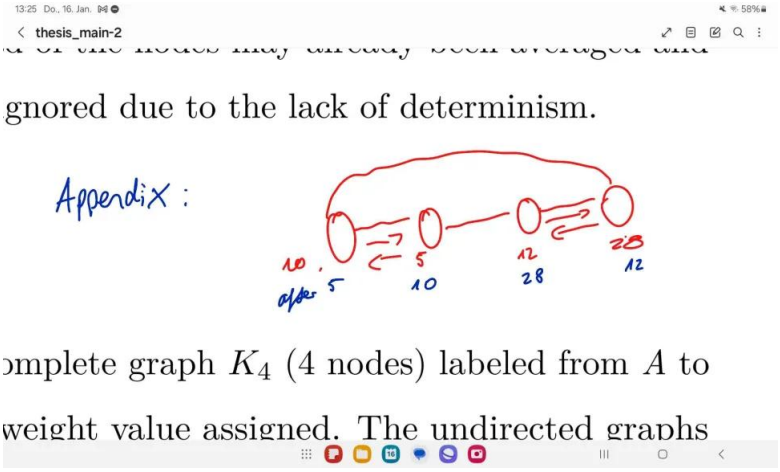


Figure 18: Example Ring Graph

ToDo Counters

To Dos: 3; 1, 2, 3

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.