University of Thessaly
Greece, October 2020

Robust energy-aware routing in multilayer wireless ad hoc networks

Εύρωστη ενεργειακά-αποδοτική δρομολόγηση σε πολυεπίπεδα ασύρματα ad hoc δίκτυα

Georgios Tziokas

Supervisor: Dimitrios Katsaros

Committee Members: Athanasios Korakis, Eleni Tousidou

Diploma Thesis

Department of Electrical and Computer Engineering

University of Thessaly

Volos, Greece

This Thesis was written as part of the requirements for the Diploma of Electrical and Computer Engineering at University of Thessaly.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

«Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής».

Περίληψη

Τα Ad-hoc δίκτυα αποτελούνται από συσκευές οι οποίες είναι αυτόνομα αυτοοργανώμενες μέσα στο δίκτυο. Οι περιπτώσεις χρήσης περιλαμβάνουν, χωρίς να περιορίζονται σε αυτές: Ad-hoc δίκτυα για αυτοκινούμενα οχήματα (**VANETs**), Ad-hoc δίκτυα για κινητά τηλέφωνα (**SPANs**), Ad-hoc στρατιωτικά δίκτυα (**MANETs**).

Υπάρχουν δύο βασικά προβλήματα τα οποία πρέπει να μελετηθούν όσον αφορά τα Ad-hoc δίκτυα. Το πρώτο από αυτά τα προβλήματα είναι ότι στερούνται προϋπάρχουσας υποδομής π.χ. απουσία προκαθορισμένων κανόνων δρομολόγησης, ενώ το δεύτερο έχει να κάνει με την περιορισμένη διάρκεια ζωής της μπαταρίας των συσκευών που απαρτίζουν το δίκτυο. Η βιβλιογραφία περιέχει σημαντική προσπάθεια στα προαναφερθέντα θέματα και αυτή η διπλωματική θα προσπαθήσει να προσθέσει περιεχόμενο στον τομέα αυτό.

Στο δικό μας πρόβλημα, δουλεύουμε με πολυεπίπεδα Ad-hoc δίκτυα με σκοπό να προτείνουμε δύο διαφορετικούς αλγορίθμους για να δημιουργήσουμε μία υποδομή κορμού (backbone). Αυτή η δομή θα εφοδιάζει το δίκτυο με ένα βελτιωμένο σχήμα επικοινωνίας, το οποίο θα αντιμετωπίζει άμεσα θέματα που σχετίζονται με την κατανάλωση της μπαταρίας, τις περιττές μεταδόσεις καθώς και με την εκφόρτιση της κίνησης μέσα στο δίκτυο, δηλαδή λιγότερες συγκρούσεις πακέτων μεταξύ των κόμβων του δικτύου. Πραγματοποιούνται επίσης πειραματισμοί και αξιολόγηση διάφορων εφαρμοζόμενων προσεγγίσεων, αναλύοντας και συγκρίνοντας τις συμπεριφορές τους.

Abstract

Ad-hoc networks consist of devices that are autonomously self-organized into networks. Use cases include but are not limited to: Vehicular Ad-hoc NETworks (**VANETs**), Smartphone Ad-hoc NETworks (**SPANs**), Military Ad-hoc NETworks (**MANETs**).

There are two main problems to consider when working with networks of this type. The first is that they lack a pre-existing infastructure i.e. absence of predefined routing rules , while the second involves the limited battery-life of the participating network components/devices. Literature contains a significant amount of effort on the aforementioned topics and this thesis will attempt to add to this field with additional content.

On to our problem, we work with Multi-layer Ad-hoc Networks with the aim of proposing two different algorithms for creating a backbone infrastructure for our networks. This structure will provide the network with an improved communication scheme, directly tackling issues of battery consumption, unnecessary transmissions and client traffic offloading to the backbone sub-network. A number of approaches were implemented and employed in this thesis, while experimentation and evaluation of their behavior is also carried out.

Acknowledgments

First of all, i would like to express my appreciation for my supervisor, Dimitrios Katsaros, whose quidance and support has been very helpfull throughout this thesis. I am extremely thankful for our friendly chats, your personal support and for all advice during this thesis.

Secondly, I am just as grateful, to professor Thanasis Korakis for being also my thesis supervisor. Last but not least, I want to thank Christos Nanis and Thodoris Deligianidis for providing feedback throughout this thesis.

In addition, I should thank my family for their unlimited love. You are there always for me when I need you. Finally I am thankful to all my friends who helped and supported me all these years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Ad Hoc Networks

Ad Hoc networks' popularity in usage owes mainly to their main ability to temporarily formulate a set of connections amongst a set of given nodes. This is explained by their infrastructure-distributed nature, unlike the case of router networks. The way that any node in an Ad-Hoc Network communicates with another, is by routing its packets in a distributed way. That means that collisions are common and care must be taken in order to impose a non-battery draining profile for nodes in Ad-Hoc networks. Emerging concepts build around that, the most popular one being the IoT (Internet of Things) concept.

In contrast to networks with permanent infrastructure, ad hoc networks can easily and rapidly be deployed, additionally providing a reliable communication in numerous situations. Further, ad hoc networks introduce a low operating cost and are highly robust (Figure from [1]).

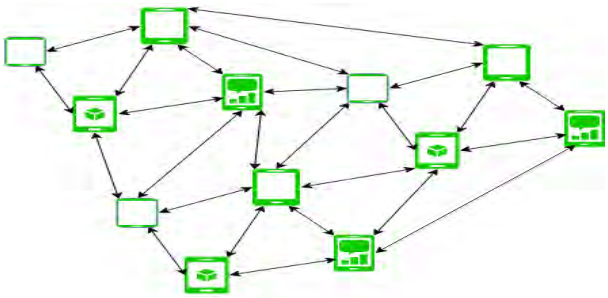


Figure 1.1: Ad hoc network representation

1.2 Virtual Backbone in Ad Hoc Networks

Ad hoc networks do not have a permanent infrastructure as mentioned before, so the way to communicate nodes in a network is to form a virtual backbone.

A Virtual Backbone is a subset of nodes in ad hoc network which relays all packets between nodes in the network. Hence if there is a way to minimize the routing paths in the network you can reduce the search time and the routing time of packets. A real popular way to do that are the Dominating Sets (DS).

Dominating set is a set where every node in the network is either in the set or have at least one neighbor inside the set. Nodes that belong to DS called dominators and all other nodes in the network called dominates.

In the same way a Connected Dominating Set (CDS) is a DS where dominators are connected. The CDS has become one of the most popular way to construct a virtual backbone to ad hoc networks. An additional problem is to keep virtual backbone as small as possible. The objective is to minimize the connected dominating set. There are many different algorithms to construct a MCDS. Some of the proposed pruning algorithms for this purpose is descripted at [2].

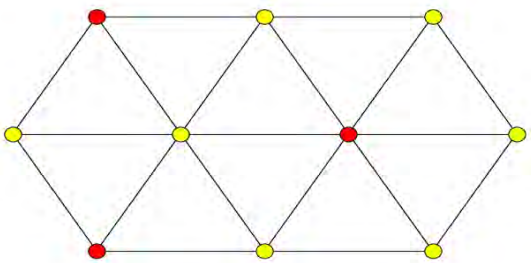


Figure 1.2: Dominating Set (DS) example

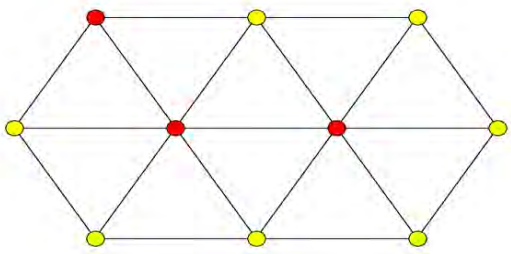


Figure 1.3: Connected Dominating Set (CDS) example

1.3 Network Model

Assume an ad hoc network as a unit disk graph G(V,E) where V is the set of all vertexes in graph and E is the set of all "edges" in the graph. The edges or links between two nodes which are within communication range share a bidirectional link which represents the edge between them.

In Unit Disk Graphs (UDG) each node has an equal communication range. Two different nodes in graph are neighbors if and only if the first are within communication range of second vice versa. So 2-hop neighbors are the nodes which cannot reach one each other directly but have in common one 1-hop neighbor.

This thesis deals only with the topology of network and simulate the construction of a virtual backbone.

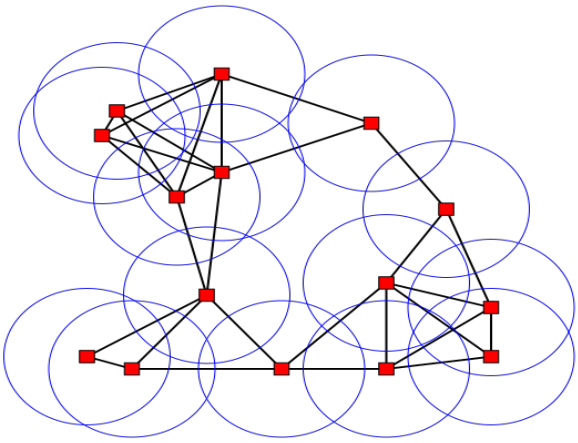


Figure 1.4: Unit Disk Graph (UDG) (Figure from [3])

1.4 Thesis Structure

The remainder of the thesis is organized as follows:

- In chapter 2, we introduce all the different types of PCI metrics used in this thesis and an additional PCI metric created.
- In chapter 3, we present the implementation of the three algorithms, MILCOM, BCA and ROBUST.
- In chapter 4, we present a server client tool which is created for the purpose of this thesis.
- In chapter 5, we present the results of algorithms for all PCIs.
- Finally, in chapter 6, we provide a conclusion to this thesis and some notes for future work that can extend our methodology.

Chapter 2

Definition of PCIs

2.1 Power Community Index

Power Community index or simply **PCI** is a metric which measure the importance of every node in network. It measures the importance of each node in its community, by checking the connectivity of it's neighbors. In this thesis we use 8 different **PCI** metrics and 7 of them are used for multi-layer networks. Below we describe these metrics.

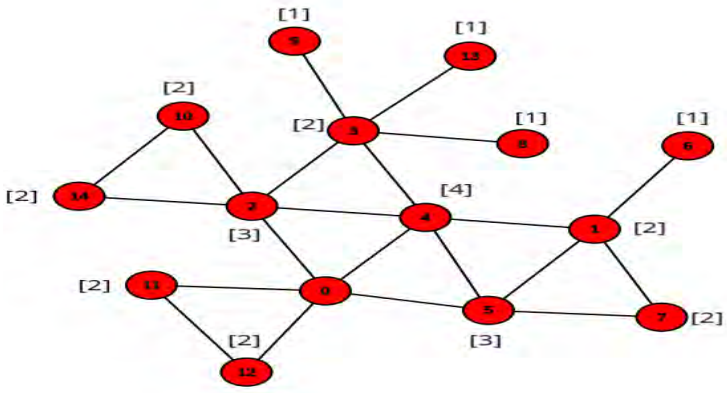


Figure 2.1: Single Layer PCI of each node for a single network

Definition 2.1.1. Single-Layer PCI or simply slPCI metric is utilized as an index for a node u in a network that equals to k , when there are up to k nodes in its 1-hop vicinity with a degree higher or equal to k. The rest of the nodes in that 1-hop vicinity have a degree less or equal to k.

2.2 Multi-Layer PCIs Definitions

Definition 2.2.1. Layer-Agnostic PCI or simply laPCI of a node in network is equal to k, if it has k 1-hop relationships with other nodes , which in turn can have k or greater, inter- or intra-layer 1-hop relationships. laPCI gives greater weight to nodes with the most connections with nodes in different layers , while handling all nodes equally , despite the underlying generating distribution of the layers in the network [4].

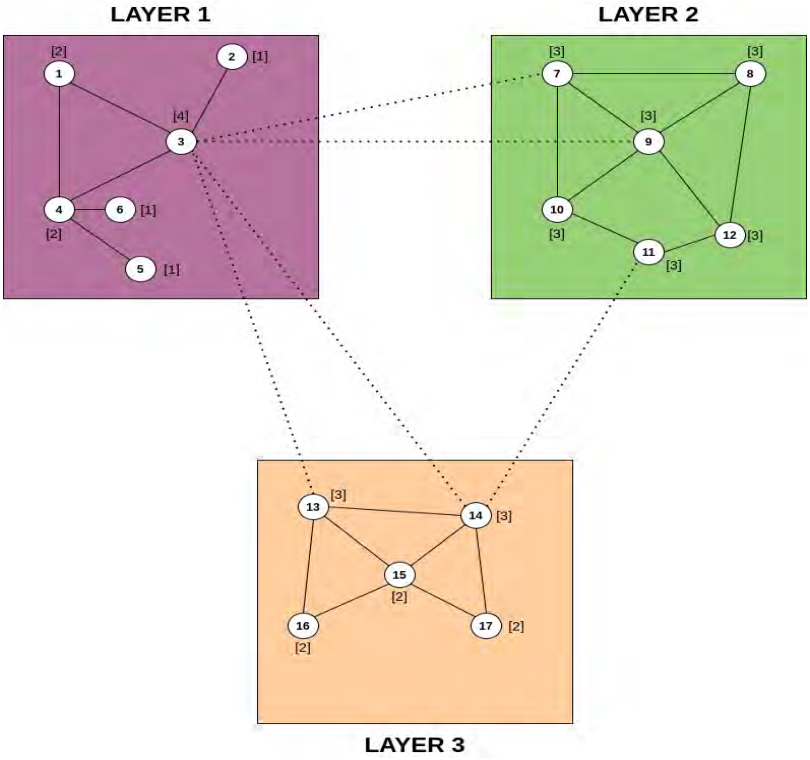


Figure 2.2: Layer-Agnostic PCI of each node for a single multi-layer network

Definition 2.2.2. Minimal-Layer PCI or simply mlPCI of a node in network is equal to k, if it has k neighbors in its 1-hop vicinity who ,in turn, have at least n>=k inter-layer connections. Within the metric that is called mlPCI, a node is characterized as good node if it is well connected in many layers. As it is stated in the above definitions, we can conclude, that the original PCI ignores the connectivity of nodes that do not participate in its definition. So we have to take into account the ignored nodes. (Equation from [4])

$$mlPCI(v) = \sum_{i=1}^{\#layers} mlPCI_i(v)$$

Definition 2.2.3. Exhaustive PCI or simply xPCI is the next PCI definition. In order to obtain the xPCI value for a given layer, we add the PCI index of a node with the PCI value of the remaining nodes. We do this for all layers and add all the values together. Finally we have obtained our xPCI value. The xPCI metric evaluation method cannot be considered a suitable ranking metric because it creates a lot of ties [4].

Definition 2.2.4. Cross-Layer PCI of simply clPCI is an extend of xPCI metric that gives us better ranking results compared to xPCI metric. To obtain the clPCI value we calculate the number of unique links between the nodes that participate in the index of xPCI . In order to get reasonable numbers even for large networks, we multiply the $\log_2(uniqueLinks)$ with each xPCI value.

The above PCI metrics are used for multi-layer networks. So we need to define what a multi-layer network is. A multi-layer network can be constructed by different categories that represent and characterize some of the nodes, that are called **layers**. A multi-layer network is a network that has more than one layer in which nodes are connected within the same layer, called **intra-layer**, or between different layers, **inter-layer**.

So for a multi-layer of n layers we have a pair of $(\mathrm{G}^{ML}, E^{ML})$ *where* $G^{ML} = \{G^i, i = 1, ..., n\}$ *is a set of networks* (G_i, E_i) *and a set of interlayer links* $E^{ML} = \{E_{i,j} \subseteq G_i \times G_j; i, j \in \{1, ..., n\}, i \neq j\}$ [4]

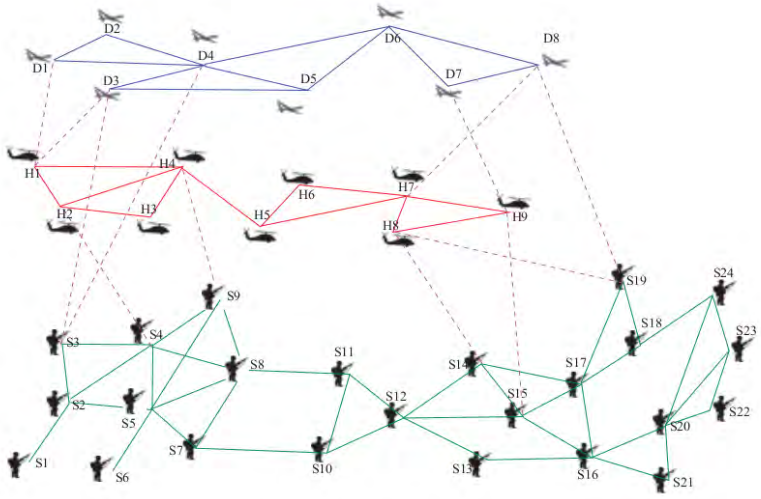


Figure 2.3: Military multi-layer Network (Figure from [4])

2.3 Experimental Settings

Continuing to operate in a similar manner to the approaches presented before, we developed a new metric for the purpose of this thesis. This metric effectively combines a node's PCI value with the PCI values of its one-neighborhood, applying weights to each. Experimentation on the assignment of these values showed that it can be beneficial for the network if the node's PCI is more heavily weighted e.g. 0.8, while the one-hop neighbors can be assigned the remaining 0.2. In this way, high scoring nodes can still retain their importance, while allowing diversity that is introduced by the node's neighborhood to be accomodated in our approach.

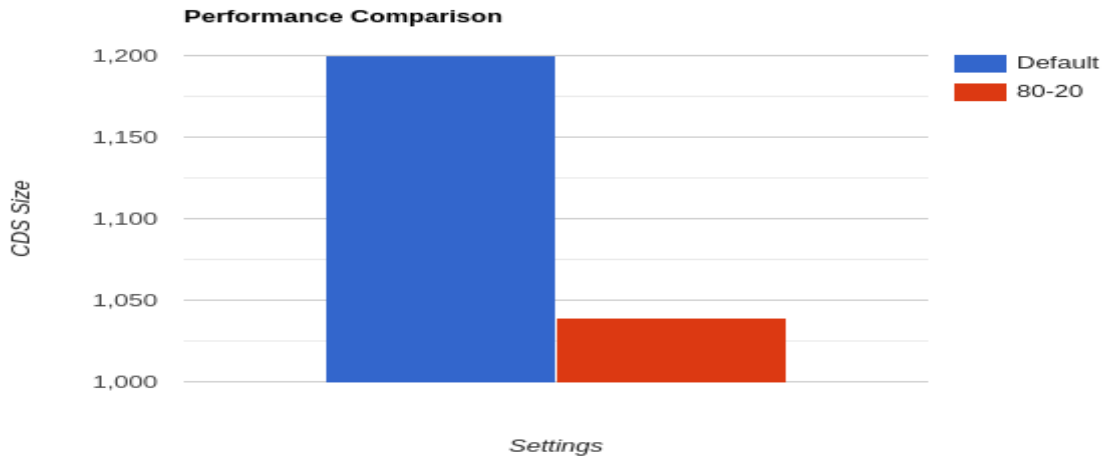


Figure 2.4: Comparison between default and 80-20 settings

Chapter 3

Implementation

3.1 Dataset

As of the input to our algorithms, we used a **MATLAB** generator that creates multi-layer network topologies, providing high flexibility for post-experiment inference from its results. Each generated layer consists of a number of wireless nodes on the 2-D plane, with a respective maximum transmission range R. Each pair of network nodes with an in-between Euclidean distance equal or less than R are considered as connected or, alternatively, form a **Unit Disk Graph** (UDG). In this manner, upon connectivity inference, the actual location of the nodes is incorporated in the procedure. In addition, to efficiently tackle cases of obstructed direct communication of adjacent nodes, non-uniform intra-layer models are utilized, in order to distribute evenly the nodes on the aforementioned two-dimensional plane. The construction of our multi-layer network is affected by the link density of each layer, measured by means of:

- Average Degree (D) of a node
- Per-layer number of nodes i.e. layer size
- Number of layers in out multi-layer network(L)

In order to create the inter-layer connections, two parameters are of particular interest:

- The number of inter-layer links of a given node.
- Distribution of intra-layer connections.

In cases where a specific layer has a higher degree of utilization due to the underlying purpose of the generated links e.g. in the case of a drone layer creating inter-layer connections with the soldiers network. Having successfully considered the above experiment hyperparameters, the **Zipfian** distribution is applied as the interconnectivity generator. Resulted skewness is managed by the s parameter, which ranges in (0,1). We then utilize 3 different Zipfian laws for each hyperparameter. First, we impose the restriction of Sdegree ranging in (0,1), in order to generate the frequencies of appearance of highly interconnected nodes. Secondly, we impose (0,1) range restriction to Slayer , to control how frequently a specific layer is selected. Last, we impose analogously for Snode, to evaluate the frequency of node-specific selection for a given layer.

In order to create the required conditions for the experiment, the input data have been divided into four major categories:

- Degree
- Diameter
- Nodes per Layer
- Number of Layers

First of all, in order to determine the effect degree imposes on the algorithms' performance input files that refer to degree are being altered in the density of a single layer while the remaining layers' degree stays close to initial values. Next we allow the diameter of each layer to vary while at the same time the other variables stay the same. This permits us to find the correlation between the size of the diameter and that of the constructed **CDS** for each method respectively. Later, we examine the relation of number and size of layers with respect to **Connected Dominating Set**. We alter the number of layers and check its effect on the **CDS**. Lastly, we vary the size of layers by sorting them in increasing order, starting at 500 nodes in top layer and increasing by a percentage of 500, then we compare the differences in the resulting **CDS**. [5]

3.2 Library Description

As per this thesis, we developed a library in Python for backbone construction either for a single-layer networks either for multi-layer networks. The library consists of three different algorithms. The first is the **MILCOM** algorithm [4] which constructs a **CDS** as backbone of the network. The rest of the algorithms are the **Backbone Construction Algorithm** (BCA) and the **ROBUST**. BCA algorithm is an extension of MILCOM algorithm with an extra step during the CDS construction while the ROBUST algorithm constructs a **kmCDS** as backbone. These three algorithms use as topology metrics the PCIs described in chapter 2. Table below shows all the possibilities that library offers.

Arguments	**Description**
–help	Print man page in stdout
-fp <filepath>	The path where input file is stored
-p <pci name>	The name of the pci metric
-a <algorithm>	The ID for the algorithm (1: MILCOM, 2: NEW, 3: ROBUST)
-k <integer>	Physical number refers to connectivity of nodes belongs to backbone
-m <integer>	Physical number refers to connectivity having external nodes to internal nodes in backbone
-tol <tolerance>	Quotient between PCI metric and centrality
–centrality	Add node centrality as an extra metric
–cds	Creates a Connected Dominating Set as backbone
–mcds	Creates a Minimum Connected Dominating Set as backbone
–rmcds	Creates a Robust Minimum Connected Dominating Set as backbone
–plot	Plot initial and final multi-layer network
–clock	Print duration of each step
–log	Print log messages to stdout
-lv <log level>	Level of logging (Debug, Info, Warning, Error)
–store_log	Write log messages to file
-lf <log file>	Name of file where store log messages
–testing	This argument used for testing and in Server Client tool

3.3 MILCOM Algorithm

The Milcom algorithm starts its execution by discovering every 1 and 2-hop neighbor for each node. Next, when each node has figured out its neighborhood of 1 and 2 hops, the algorithm calculates the PCI metric, e.g. slPCI, clPCI etc, which the user has chosen, by giving it as input for execution, and then transmits the result value to all the 1-hop neighbors. After that, each node respectively, sorts the results transmitted by its neighboring nodes, in decreasing order. Then each node selects and sets, if it exists, as dominator a neighbor which had already been selected from other nodes. The procedure extends with the 2-hop neighbors of each node. This means that nodes check the 2-hop neighborhood and if there is at least one neighbor without a dominator, they set as dominator the neighbor in the 1-hop neighborhood with the highest PCI score.

Algorithm 1 MILCOM Algorithm

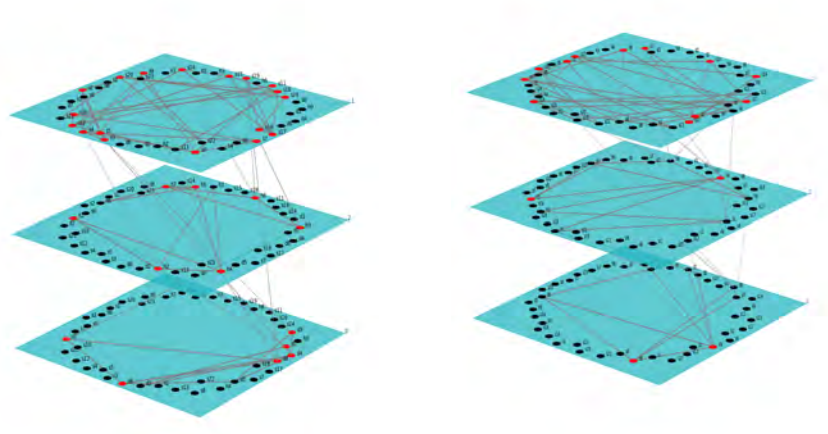
```
1: for node = 1, 2, ..., N do
2:     if pci = cl then
3:         Find unique links between nodes
4:         Calculate clPCI
5:     end if
6:     Get PCI of all neighbors
7:     Pick the neighbor with biggest PCI as dominator
8: end for
9: for node = 1, 2, ..., N do
10:    Add node to CDS
11: end for
12: if MCDS = True then
13:    Minimize CDS
14: end if
```

3.4 BCA Algorithm

In this algorithm we use an extra step in process of backbone construction. In this extra step we use shortest paths between all pairs of nodes in backbone to calculate a prefferedBy value for the nodes. So in this point need to define what sortest path is.

Definition 3.4.1. Shortest Path is a problem in graph theory, of finding a path between two vertices. Adjacent vertices called that they have a common edge. A path in graph subscripted as a sequence of vertices P = (u_1,u_2,..,u_n) $\in V, (G = (V, E))$.Assume that $e_{i,j}$ is an edge that indicent to both u_i,u_j then the sortest path between vertices u, u' where u =u_1 and u'=u_n is the path over all n paths that minimizes the sum $\sum_{n=1}^{n-1} f(e_{i,i+1})$.

This algorithm works in two phases. In the first phase, it works exactly like the Milcom algorithm in that it follows the process of creating the first Connected Dominating Set. In the second phase, each dominator node traces the shortest paths to every other dominator in backbone and increases a counter, called preferred by, of every node that belongs to the shortest path. If one node has zero preferred by value and is not a fundamental node of the backbone it is being ignored and will not be included as dominator in the final backbone. On the other hand, if node does not belong in backbone already and it has high preferred by value then it is being included in the final backbone. This extra step aids us in optimizing and stabilizing the backbone of the network.



(a) CDS as backbone **(b)** MCDS as backbone

Figure 3.1: Red nodes are nodes of backbone

The above figure shows two type of backbones given the same network topology as input.

The first figure graphs a Connected Dominating Set while the second shows the resulting minimized set after the pruning process which was applied on the initial. The nodes shown red are dominators while the black are the dominates.

Algorithm 2 BCA Algorithm

```
Input: listOfNodes
for Every node in network do
    Discover 1-hop and 2-hop neighbors
end for
for Every node do
    Calculate input PCI metric for every 1-hop neighbor
    Sort neighbors by their PCI values
    Transmit the PCI of node having the higher value
end for
for Every node do
    Every node add as dominator a neighbor which has already been
    selected from other node, if it exists
    if not all 2-hop neighbors have dominator then
        Add a neighbor from 1-hop as dominator if it is cover at least one node in 2-hop neigborhood
    end if
end for
for Every node in CDS do
    Find the sortest paths to every other dominator
    Increase a prefferedBy value of every intermediate node in the sortest paths
end for
# A no fundamental node have prefferedBy value equals to zero
for Non fundamental node in backbone do
    # A no fundamental node can be removed from CDS if all
    # dominators are still connected and every other node
    # continues to have at least one relationship with some
    # dominator
    Remove it if it is possible
end for
for Every significant node do
    Try to add node to CDS
end for
```

In this library we have included an extra topological metric, called betweenness centrality.

Definition 3.4.2. Betweenness Centrality measures the centrality of nodes in graph theory based on shortest paths described in 3.4.1. Betweenness centrality for a vertex u is the number of shortest paths those which include vertex u.

$$g(u) = \sum_{s \neq u \neq t} \frac{\sigma_{st}(u)}{\sigma_{st}}$$

σ_{st} refers to the total number of shortest paths and $\sigma_{st}(u)$ to those which include vertex u.

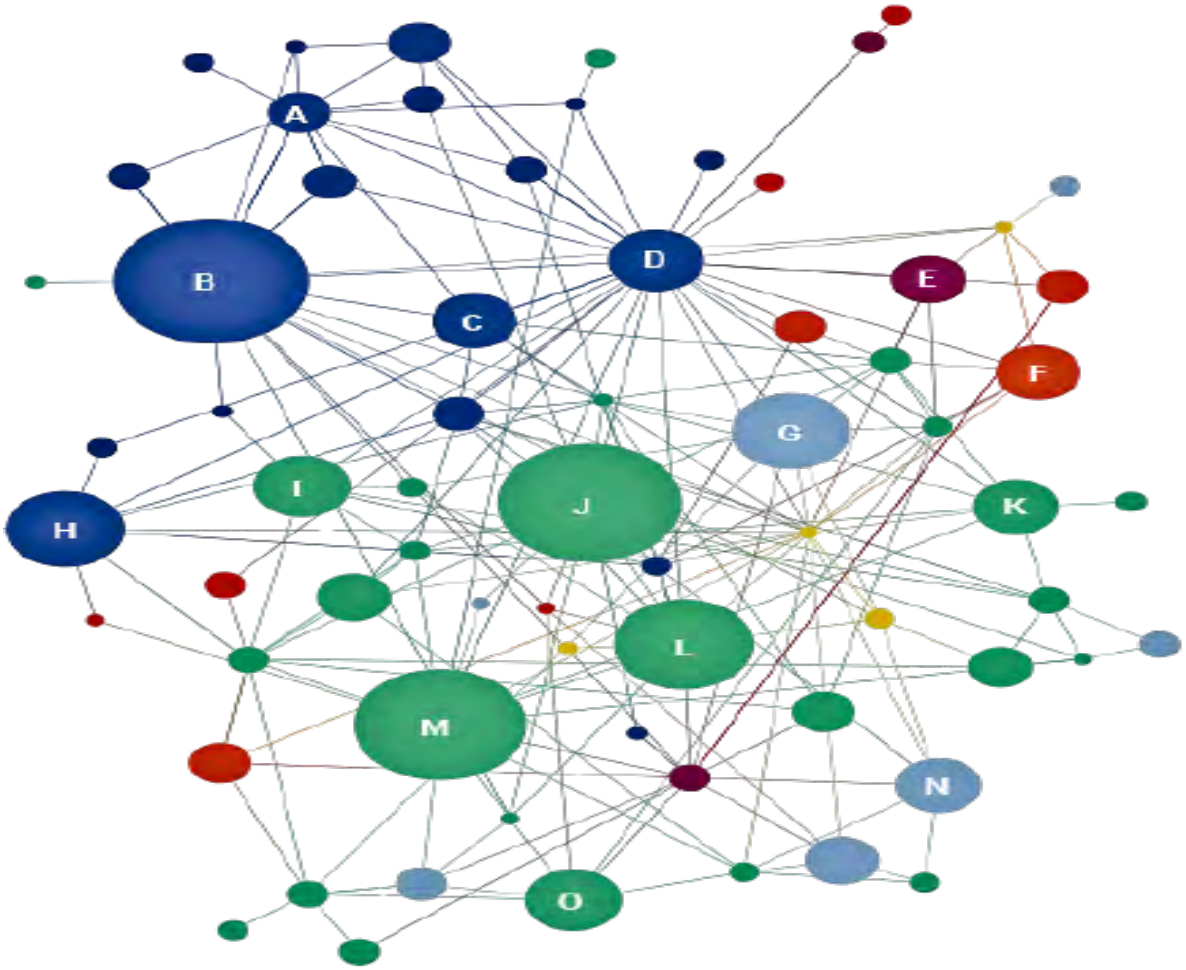
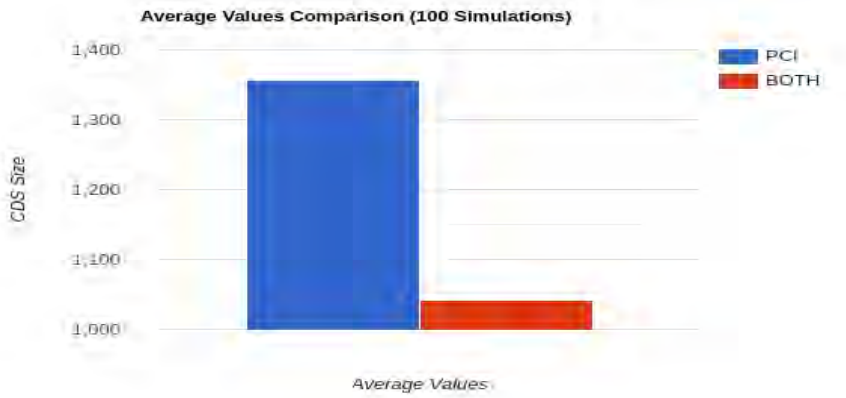
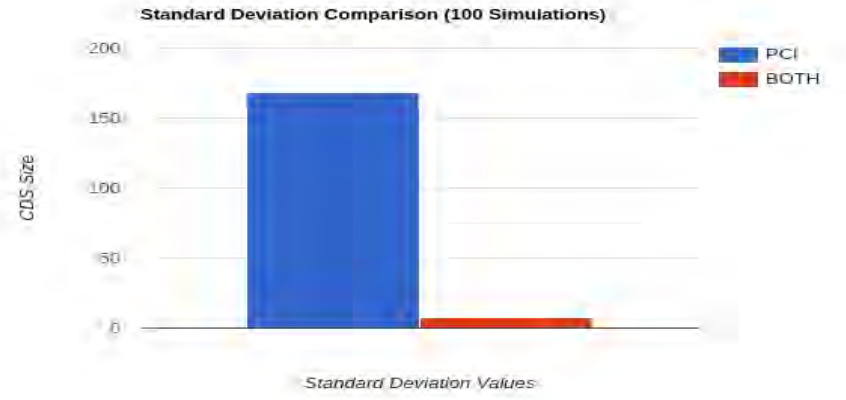


Figure 3.2: Single Layer PCI of each node for a single network (Figure from [6])

3.4.1 Selection Algorithm Description

When betweeness centrality is added as a considered metric, the local (max 2-hops distance) betweeness centrality is calculated for every neighbor of each node. Firstly, we sort the nodes by their respective PCI value. Then in order to have more accurate results, we find the ratio of the two nodes with the greater PCI values and compare it with their centralities' ratio. If the difference is greater than a fixed tolerance value of e.g. 0.2, we swap the two nodes i.e. if the two most significant nodes are characterized by PCI and centrality values of 1100,0.2 and 1000,0.6 respectively,the difference in the ratios is greater than the tolerance value. We thus select the latter node as the most significant and we perform a swap. Below the procedure described above is presented.





As we can see from the two figures above, when we use both PCI and centrality metrics, we get better results of average CDS size.

Algorithm 3 Selection Algorithm

```
1: function CHOOSE_NODE(first_node,second_node)
2:     if first_node.centrality / second_node.centrality <= 1 − tolerance then
3:         return second_node
4:     else
5:         if second_node.pci / first_node.pci > 1 − first_node.centrality / second_node.centrality then
6:             return first_node
7:         else
8:             return second_node
9:         end if
10:    end if
11: end function
12: if first_node.pci > 0 then
13:    if second_node.pci / first_node.pci >= 1 − tolerance then
14:
15:        if second_node.centrality > first_node.centrality then
16:            return CHOOSE_NODE(first_node, second_node)
17:        else
18:            if first_node.centrality > 0 then
19:                return CHOOSE_NODE(second_node, first_node)
20:            else
21:                return first_node
22:            end if
23:        end if
24:    end if
25: else
26:    if second_node.centrality > first_node.centrality then
27:        return second_node
28:    else
29:        return first_node
30:    end if
31: end if
```

3.5 Robust Algorithm

This algorithm works by giving as input two natural numbers k and m. At the end of this procedure we will have created a k-m-Connected Dominating Set or **kmCDS**. A kmCDS is a CDS where the dominators are k connected and all dominates have at least m dominators as 1-hop neighbors. This algorithm consists of 4 phases.

The first phase of the algorithm is a combination of the Milcom algorithm and an extra node decision method. The extra node decision method starts with each node finding the number of links to dominators in their 1-hop neighborhood. If the number of dominators is smaller than a given natural number m, it checks if there are any neighbors which belong to the CDS and set them as additional dominators. After this, it checks again the number of dominators in its 1-hop neighborhood and if it is higher than or equal to m, the node becomes a dominatee. If the number of dominators is smaller than m then the node checks if it has links to at least k dominators in its 1-hop neighborhood and sets itself as a dominator. This is the end of phase 1 of the Robust algorithm.

Algorithm 4 Phase 1 of Robust Algorithm

```
if node.num_of_dominators < m then
    for neighbor ∈ N(u) do
        if neighbor ∉ node.dominators then
            Add neighbor to dominators
        end if
    end for
    if node.num_of_dominators < m then
        if node ∉ backbone then
            if node.dominators >= k then
                Add node to backbone
            end if
        end if
    else
        Add node to dominates
    end if
else
    Add node to dominates
end if
```

In phase 2 of the algorithm we need to check a constraint that has been analyzed in [7] This constraint tells us that every dominator needs to have at least k dominators in its 1-hop neighborhood and all dominates need to have at least m dominators in their 1-hop neighborhood respectively. Before initiating phase 2 of the algorithm, we assign a large value to K. In phase 2 of the algorithm a node checks for dominators in its neighborhood. If this node is a dominator then if it has less than k dominators then it becomes a non-connected dominator. On the other hand, if the node is a dominate then if it has less than m dominators as neighbors, it becomes a non-connected dominatee. When phase 2 ends, a set of dominators which added to dominators graph is returned. Assume a binary decision variable x_i:

$$x_i = \begin{cases} 1, if\ vertex\ is\ chosen\ to\ be\ a\ dominator \\ 0, if\ vertex\ is\ chosen\ to\ be\ a\ dominate \end{cases}$$

We thus can describe our second phase using the equation below:

$$\sum_{j \in N(i)} >= kx_i + m(1 - x_i), \forall i \in V$$

If a node is close to be a dominator, $x_i = 1$, then it should have at least k dominators in one-hop neighborhood. On the other hand, if a node is close to be dominatee, $x_i = 0$, then it should have at least m dominators in its one-hop neighborhood.

In this moment starts the phase 3 of the algorithm. In phase 3 there are two subphases, the first executes only the first iteration and the second is executed repeatedly. In the first subphase, as of the nodes that selected the previous phase's dominators, they will start searching for all minimum vertex cuts with all other dominators in backbone and if a minimum vertex cut size is smaller than k then all dominators that belong in the minimum vertex cut will be removed from CDS. In the second subphase, we will use the nodes which selected the previous phase's dominators and if they have k one hop neighbors

which are dominators, then we will set this node as a dominator. The algorithm stops when it finds a number K which is higher than or equal to natural number k and moreover every dominator is k-connected with every dominate having at least m dominators in their one hop neighborhood, consequently returning a kmCDS.

$$\sum_{i \in c} x_i >= min(m, k), \forall c \in C$$

Algorithm 5 Phase 3 of Robust Algorithm

```
1: function NODES_REMOVAL(new_nodes)
2:     for u ∈ new_nodes do
3:         for dominator ∈ CDS do
4:             if dominator ∉ N(u) then
5:                 c = minimum_vertex_cut()
6:             end if
7:             if size_c < k then
8:                 for vertex ∈ c do
9:                     Remove vertex of CDS
10:                end for
11:            end if
12:        end for
13:    end for
14: end function
15: if first_time then
16:    NODES_REMOVAL(new_nodes)
17: else
18:    for u ∈ new_nodes do
19:        if dominators(u) >= k then
20:            Add u to CDS
21:        end if
22:    end for
23: end if
```

3.6 Pruning Phases

We have two different implementations for the pruning phase. MILCOM and our BCA algorithm share a common pruning phase in contrast to the Robust algorithm which has its own pruning phase. The pruning phase for the first two algorithms begins with the removal of a node from the backbone. Next each node should have at least one neighbor in its 1-hop neighborhood which belongs to the Connected Dominating Set. Furthermore all nodes which belong to the CDS should remain connected after the removal of a node. If the two restrictions above are satisfied, then the node removed permanently from the CDS, proceeding with the next dominator in the set. Otherwise, we add the node back to the CDS. When this phase ends, we have create a MCDS for backbone of the network

Algorithm 6 Prunning Phase

```
1: Input: CDS
2: while CDS size > 1 do
3:     Mark node for removal from CDS
4:     _to_remove = 1
5:     for u ∈ V do
6:         if dominators(u) == 0 then
7:             Add node again to CDS
8:             _to_remove = 0
9:         end if
10:    end for
11:    if _to_remove == 1 then
12:        Remove dominator from CDS
13:    end if
14: end while
```

This algorithm has a slightly different pruning phase than the above presented algorithms. The main difference is that it needs to take into consideration the natural numbers k and m. Pruning phase begins in the same way like the pruning phase of the other two algorithms, by removing the first dominator of the final CDS. After the removal of the node, we check if all other dominators in the backbone are k connected with each other and if all dominates of the whole network stay m connected with other dominators. If the two restrictions above are satisfied, then the node is removed permanently from the CDS and we proceed with the next dominator, otherwise we add the node back to CDS.On this phase's end, we have created a kmMCDS as backbone of our network.

Algorithm 7 Prunning Phase

```
1: Input: CDS,k,m
2: while CDS size > 1 do
3:     Mark node for removal from CDS
4:     _to_remove = 1
5:     for u ∈ V do
6:         if u ∈ Dominatees then
7:             if dominators(u) < m then
8:                 Add node again to CDS
9:                 _to_remove = 0
10:            end if
11:        end if
12:        if u ∈ Dominators then
13:            if dominators(u) < k then
14:                Add node again to CDS
15:                _to_remove = 0
16:            end if
17:            if Dominators not k-connected then
18:                Add node again to CDS
19:                _to_remove = 0
20:            end if
21:        end if
22:    end for
23:    if _to_remove == 1 then
24:        Remove dominator from CDS
25:    end if
26: end while
```

3.7 Libraries used in implementation

In this thesis, we use 2 different libraries for network topologies simulation, networkx and multilayer-networks-library (Pymnet). Both are open-source libraries in Python. The first is used for constructing all the network related structures while the second is used for plotting the result networks in a multi-layer way. Both are very flexible and easy to use with many functionalities as shown in the examples below, concerning the installation procedure for the two libraries [8], [9].

networkx installation:

```
$ pip install networkx
```

multilayer-networks-lybrary installation:

```
$ hg clone https://bitbucket.org/bolozna/multilayer-networks-library
$ python setup.py install
```

Simple networkx example:

```
>> import networkx as nx # Import library
>>
>> G = nx.Graph() # Creates an instance of network
>> edges = [(1,2),(1,3),(2,3)] # List of edges (nodes: [1,2,3])
>> G.add_edges_from(edges) # Create a network representation
                           # based on above connectivity
```

Simple multilayer-networks-library example:

```
>> from pymnet import * # Import library
>> edges = [(,),...,(,)] # A list of tuples (node_name,node_layer)
>> for edge in edges:
>>       # Add edges between neighbor nodes in any layer
>>       mnet[node1_name,node1_layer][node2_name,node2_layer()] = 1
>> draw(mnet,show=False)  # Visualize network
```

Chapter 4

Server Client Tool

4.1 Tool Description

Server-Client Tool is merely an additional feature of our framework, instrumenting the data distribution amongst the network client nodes. It is developed and maintained in Python, starting its execution by initiating the server side. Next, the server side awaits for client connections.By sending an identification packet, the client informs the server on how many cores can be allocated in this scope. When the server side accepts a client connection, it creates an instance of this client and sends it back as a chunk of files that need to be analyzed and a unique ID, in order to be informed on the ordering of that specific client among all client connections. When clients receive input data chunks, they deploy multiple processes, equal in number to the number of cores that are able to be allocated. The processes are then started and are waited on until all processes have finished. The last process is responsible for merging all results and returning them to the client. Analogously, the client sends its results back to the server. This process extends for all clients that are connected to server. When the server receives results from a client, it writes the results to a file and send the next available chunk of files to the client. This continues for all input files that the server has. The communication between server and clients uses TCP/IP connections. The way they communicate is with one purpose protocol which is created for the sole purpose of this thesis. Below I will elaborate on the subject of packet creation. There are two types of packets. The first is the packet which sends the client to server and the second is the packet which sends the server to client, named

client_packet and server_packet respectively. The client packet has 5 different message IDs while the client packet has 4. The two tables below shown the format of the two packets:

4.2 Packet Structures

Table 4.1: Client Packets Format

PacketID	Packet Headers	Packet Payload	Description
1	packetID sizeOfPayload	numberOfCores	Send number of cores to server
2	packetID	-	Request from server the next chunk of input files
3	packetID fileID sizeOfPayload	resultString	Send results to server
4	packetID	lastFileID offset	Close session with specific client because closed unexpectedly
5	messageID	lastFileID offset	Tells server to resend last chunk of input files

Table 4.2: Server Packets Format

PacketID	Packet Headers	Packet Payloas	Description
1	packetID	chunkStart chunkEnd fileID	Send to client the next chunk of input files
2	packetID	-	Close session with client
3	packetID fileID sizeOfPayload	listOfFiles	Send to client a list with all regular paths of input files
4	messageID	chunkStart chunkEnd fileID	Resend last chunk of input files to client

4.3 Server Client Flow Chart

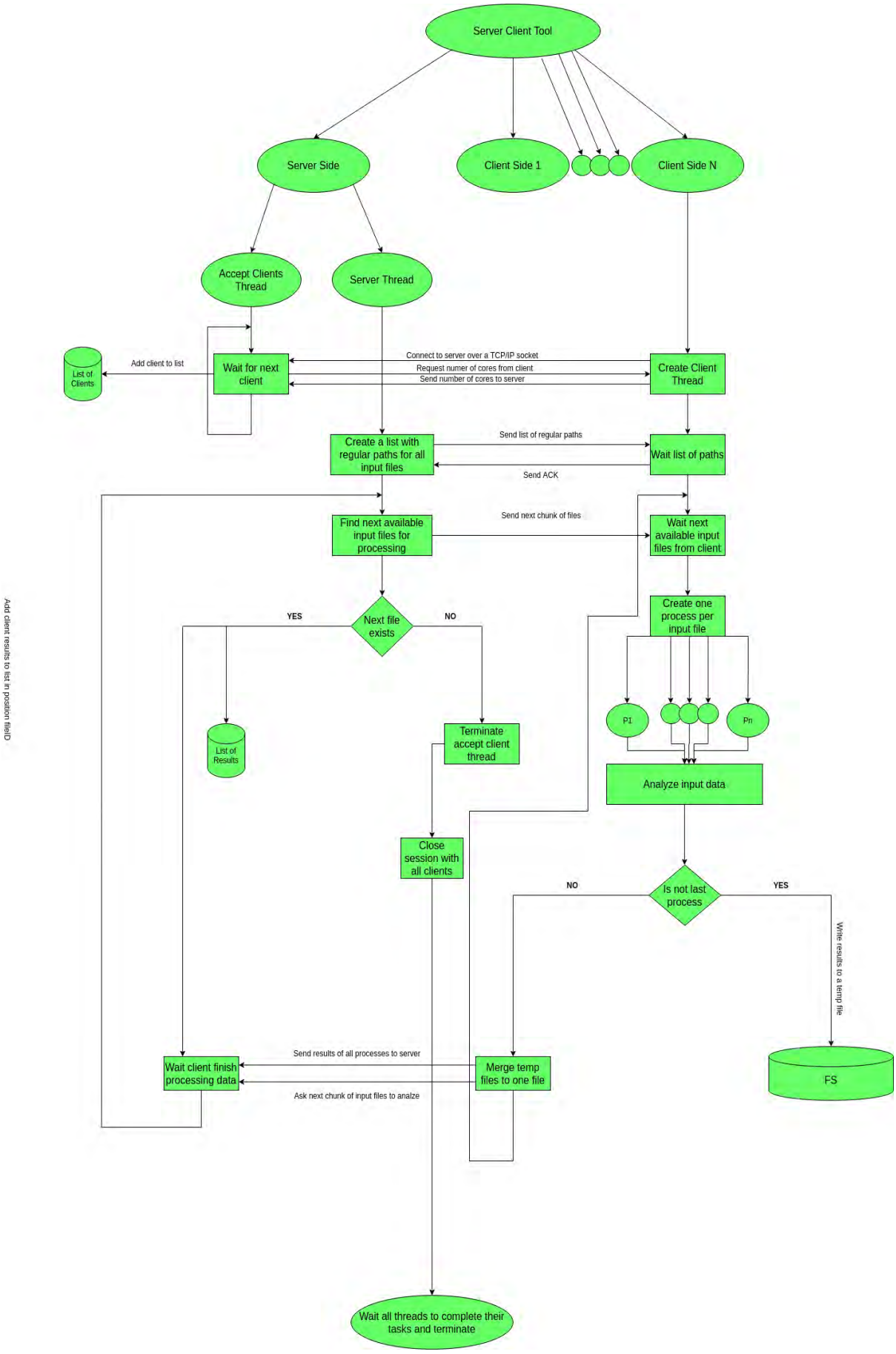


Figure 4.1: Server Client Tool Flow Chart

Chapter 5

Results

5.1 First Algorithm Results

5.1.1 CDS per PCI per Layer Plots

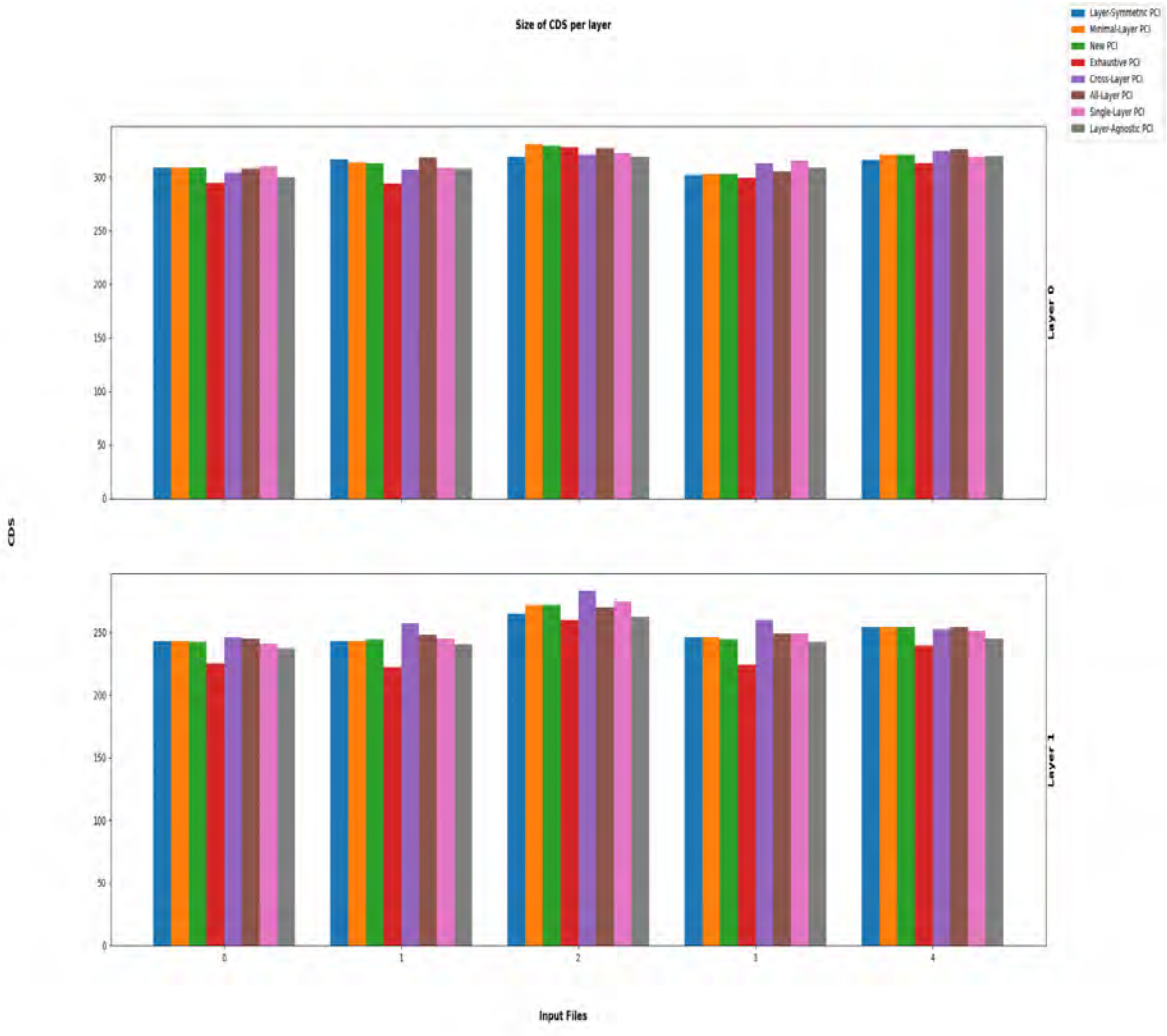


Figure 5.1: Two Layers Plot

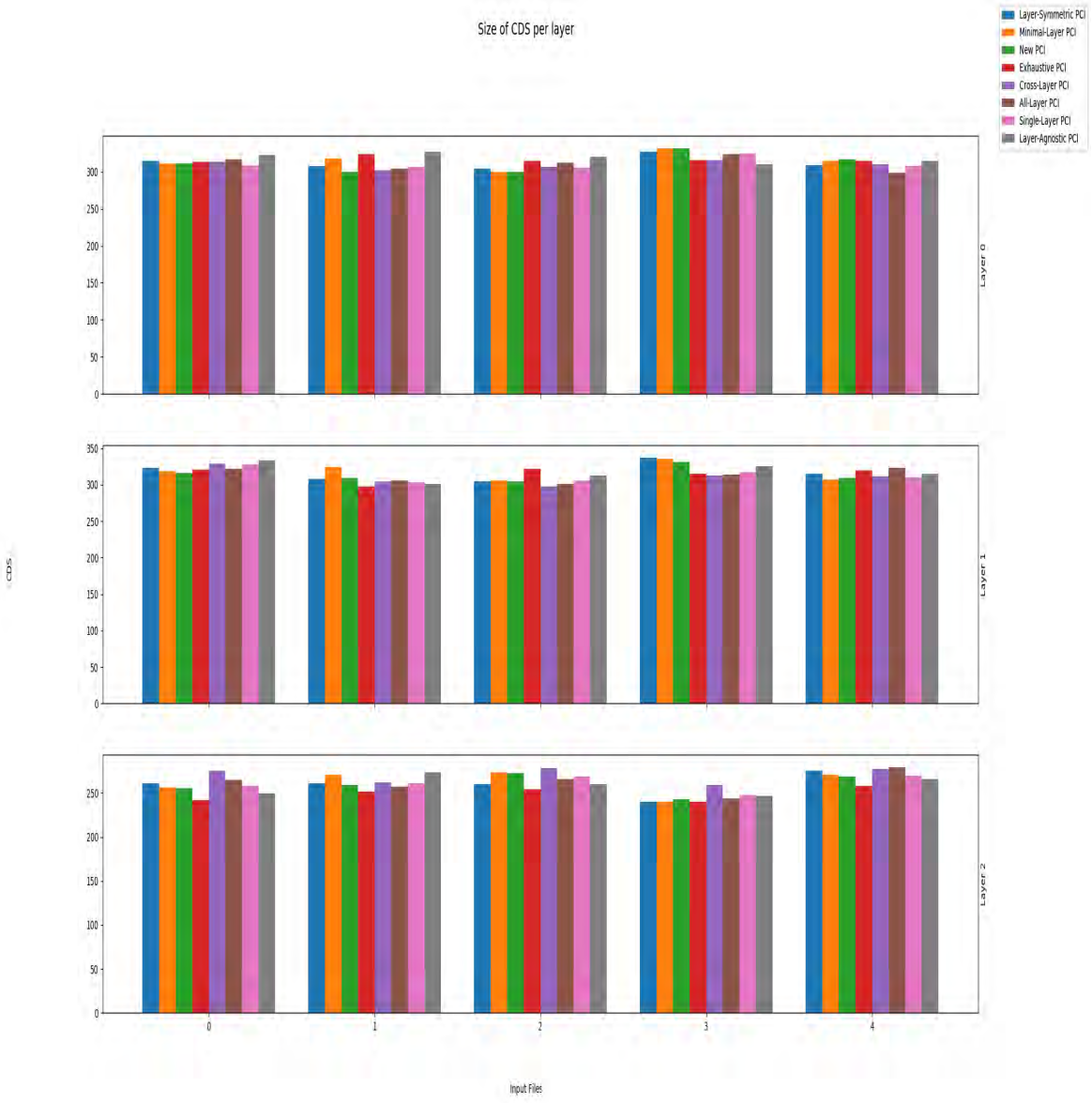


Figure 5.2: Three Layers Plot

In our experiments we use 4 different types of network topologies, as they are described in 3.1. Every type of input network topologies have various number of layers. More specifically, each input topology has 2,3,4,5 or 7 number of layers with varying number of nodes in every layer. In the above figures, we represent the results for every layer. Each layer has a set of input files in which every PCI metric is represented by a different color. Show that the relative performance is similar for all PCI metrics. Some of the resulted CDS related with input files as its shown in figure 5.3 have more dominators per layer. This happens because in these networks there is smaller number of linked nodes. This extends for every layer.

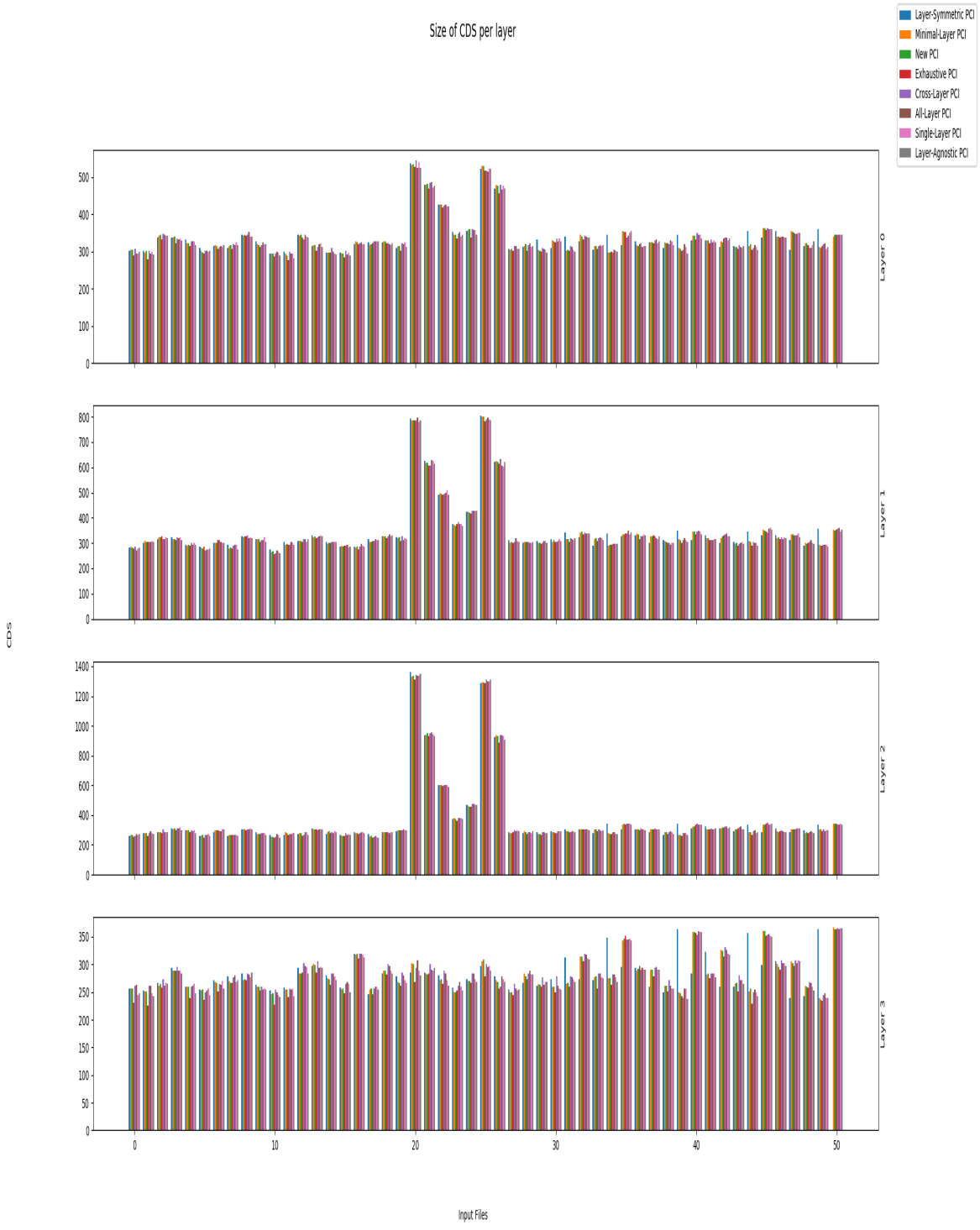


Figure 5.3: Four Layers Plot

5.2 Robust Algorithm Results

5.2.1 CDS for every k-m combination per Layer for a particular PCI (cross-layer) Plots

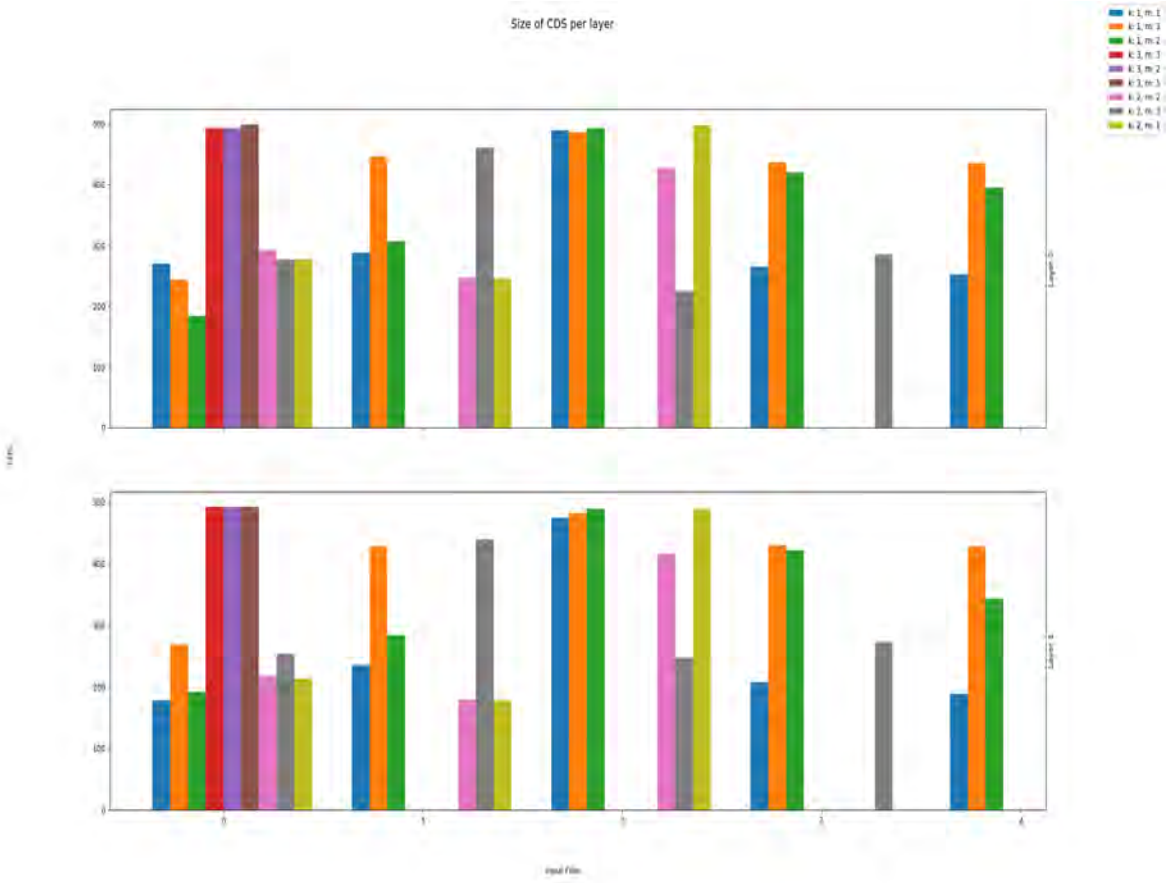


Figure 5.4: Two Layers Plot

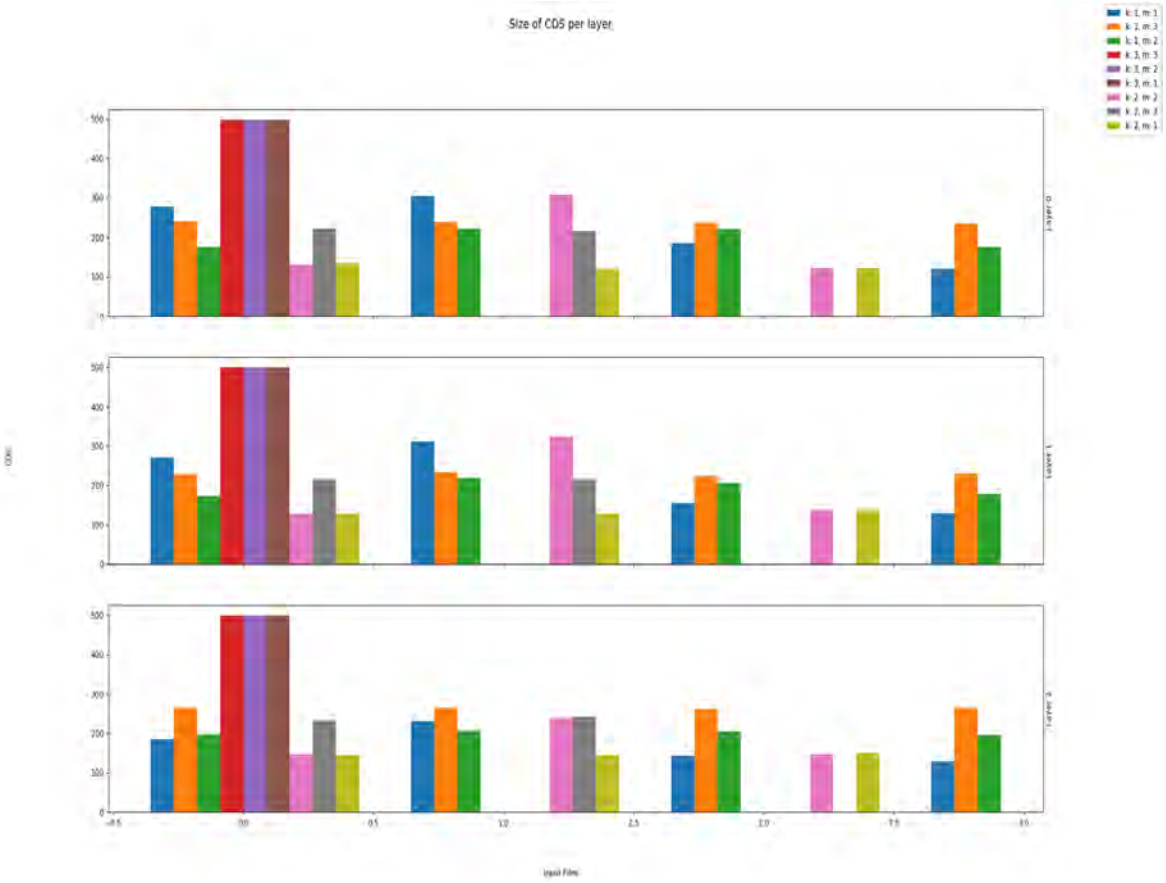


Figure 5.5: Three Layers Plot

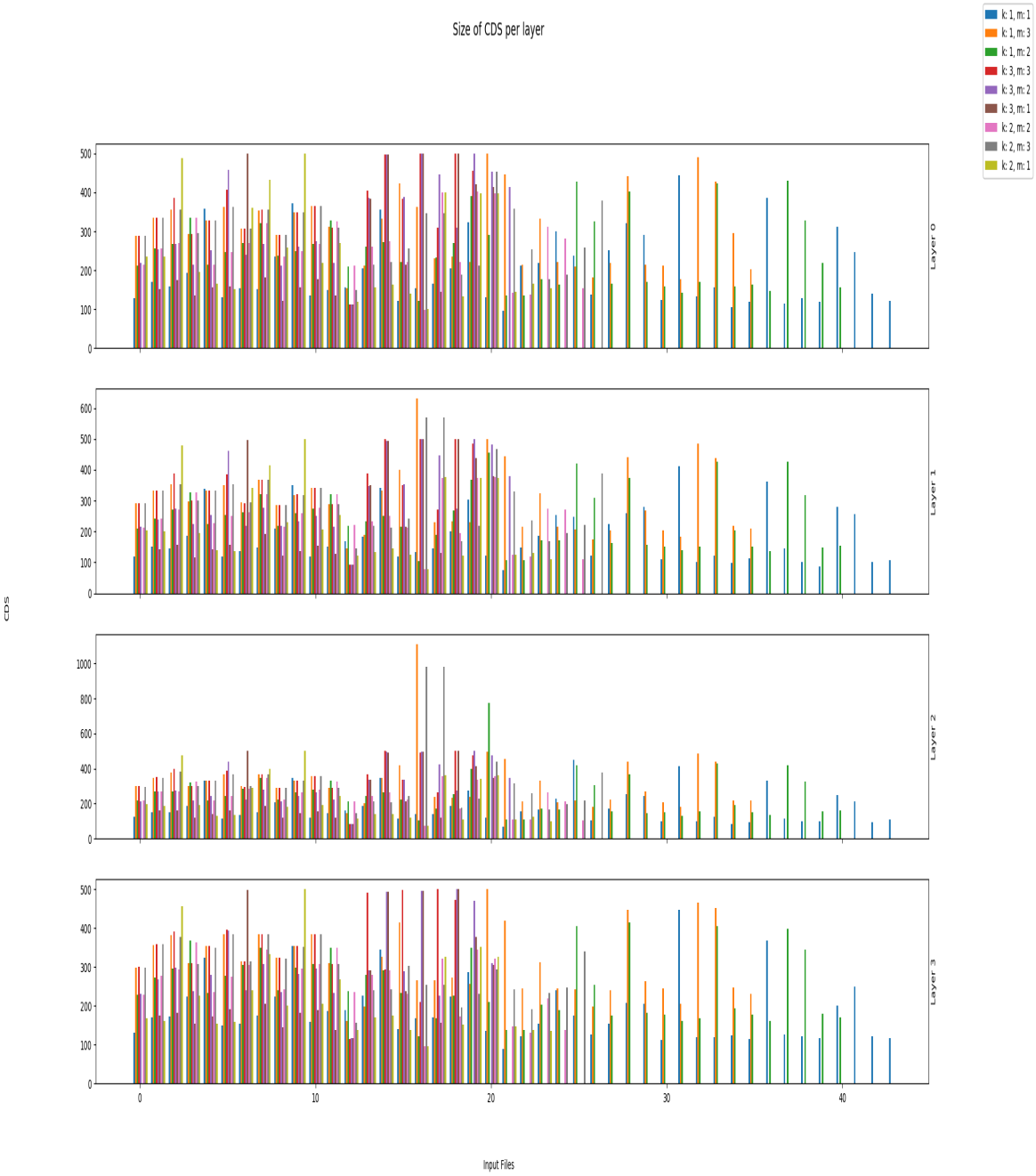


Figure 5.6: Four Layers Plot

We can derive from the plots above that although there are 9 different combinations of physical numbers k and m we see that not all those are graphed. This happens because, as k increases it becomes more difficult to construct k-CDS. On the other hand, as m increases there is no need for dominatees to be m-connected but they have to be linked to m dominators. Furthermore, as we can see on the figures above, the most of the input network topologies have 4 number of layers. Finally, we see that as both number increase

so does the number of nodes that belong to CDS.

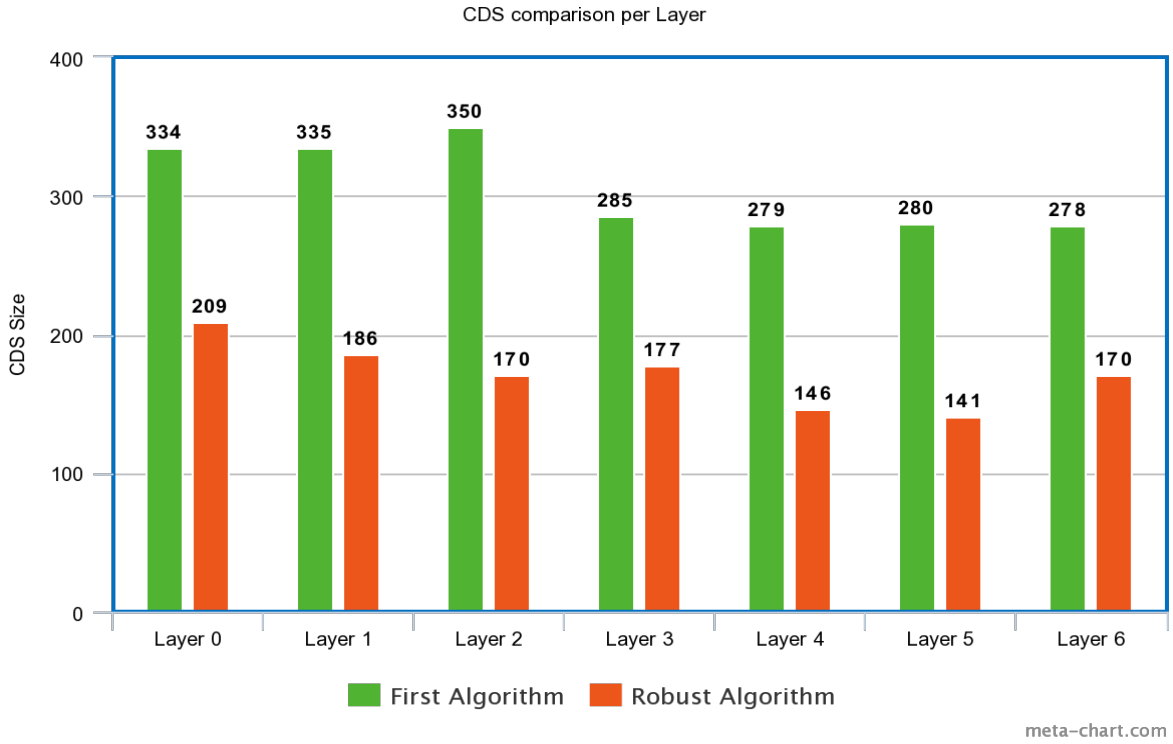


Figure 5.7: Average CDS per layer per algorithm

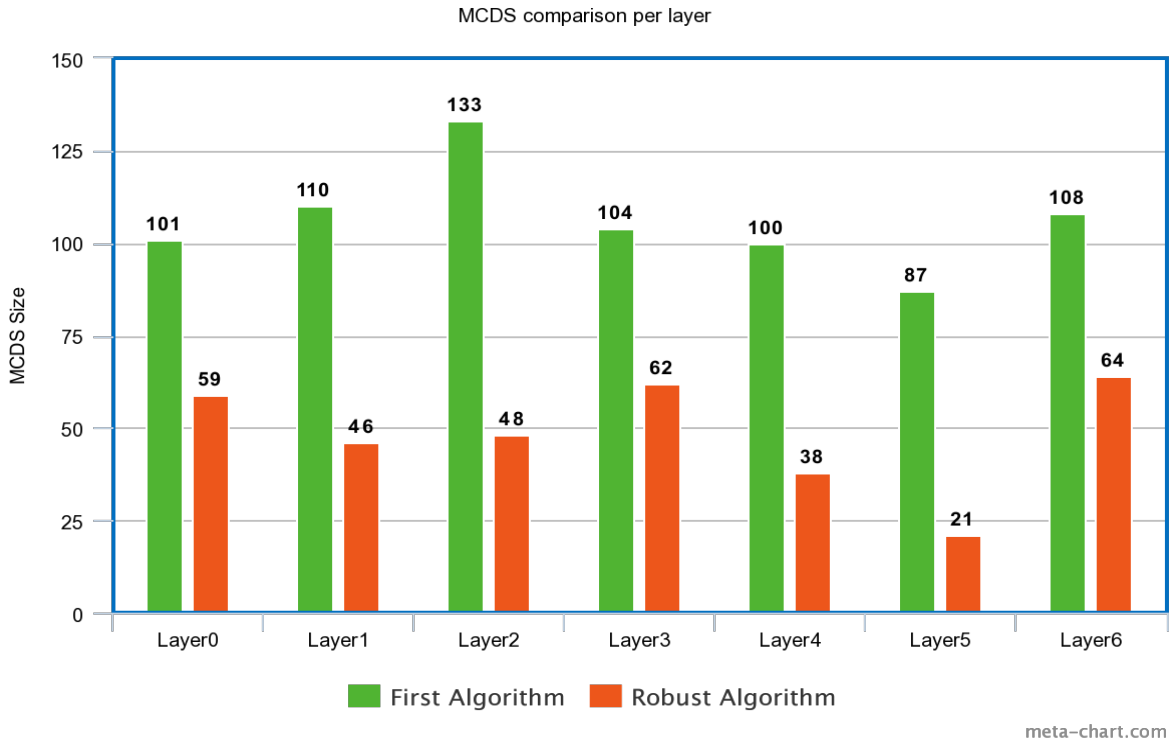


Figure 5.8: Average MCDS per layer per algorithm

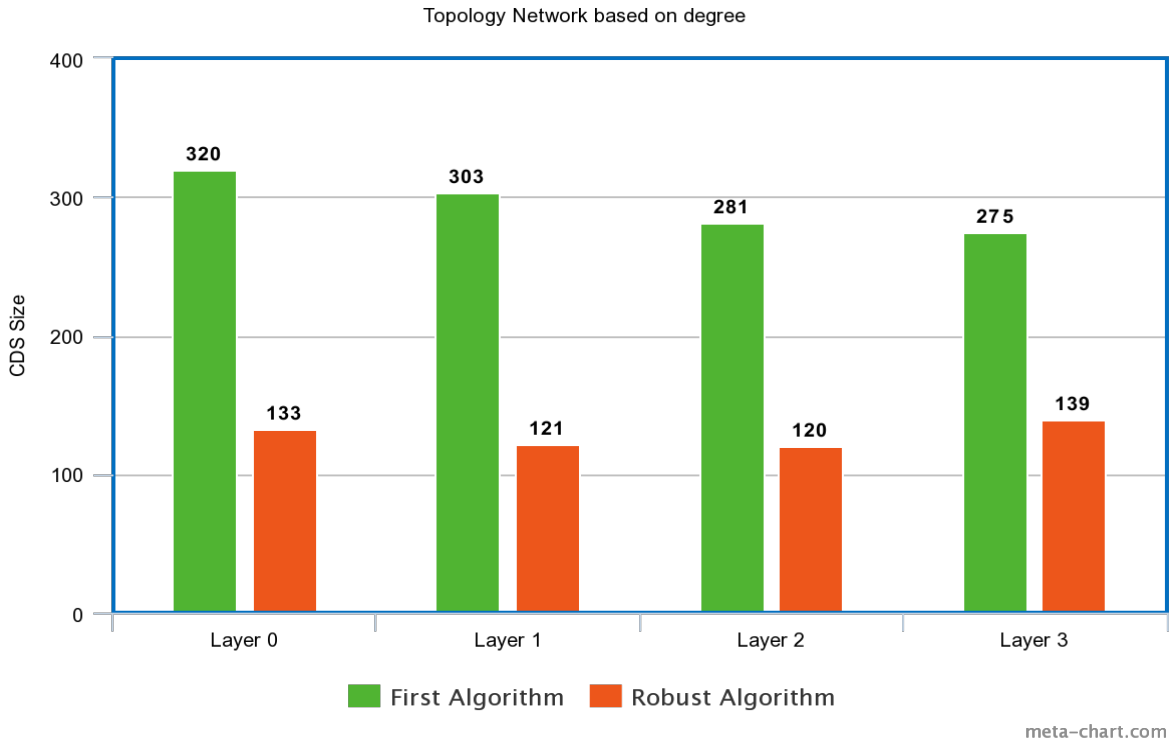


Figure 5.9: Average CDS per layer of topologies based on Degree

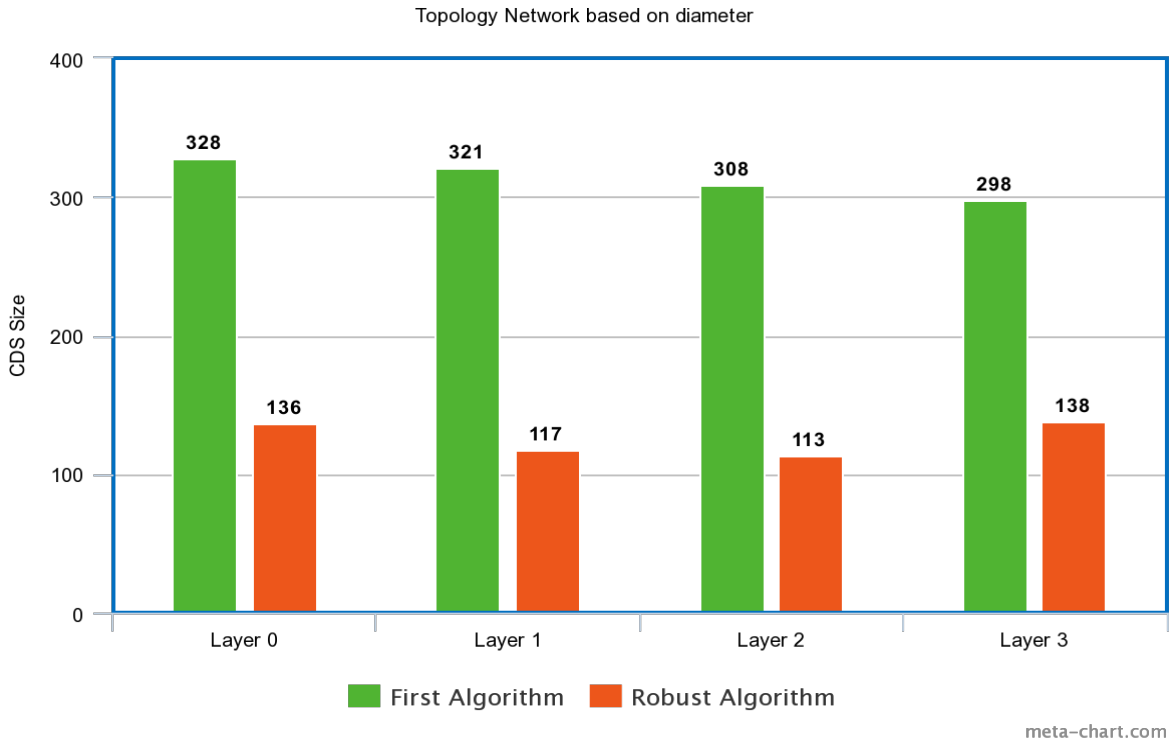


Figure 5.10: Average CDS per layer of topologies based on Diameter

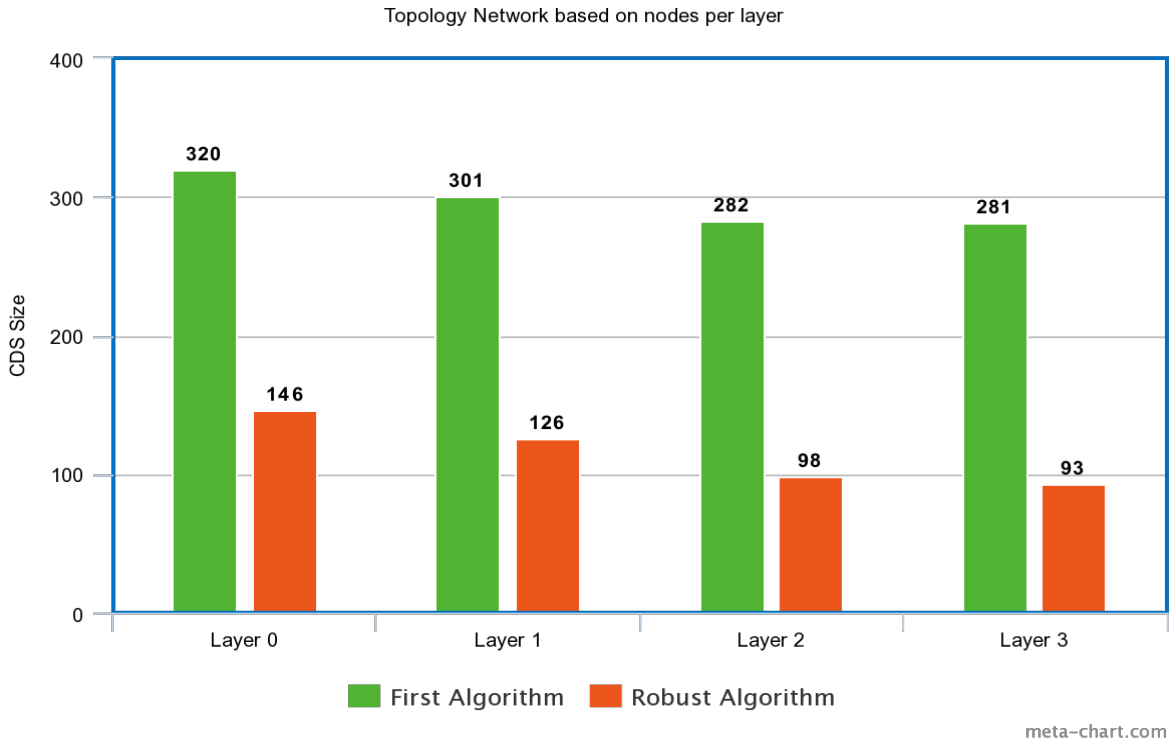


Figure 5.11: Average CDS per layer of topologies based on Layers

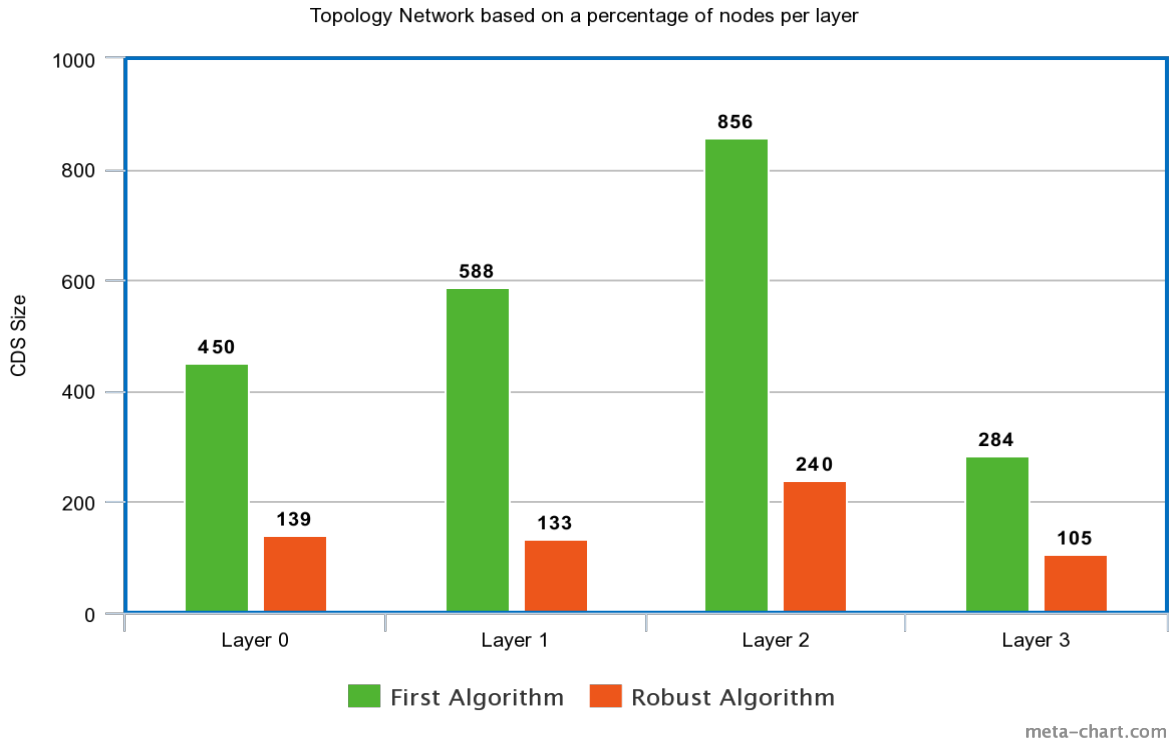


Figure 5.12: Average CDS per layer of topologies based on a percentage of nodes per layer

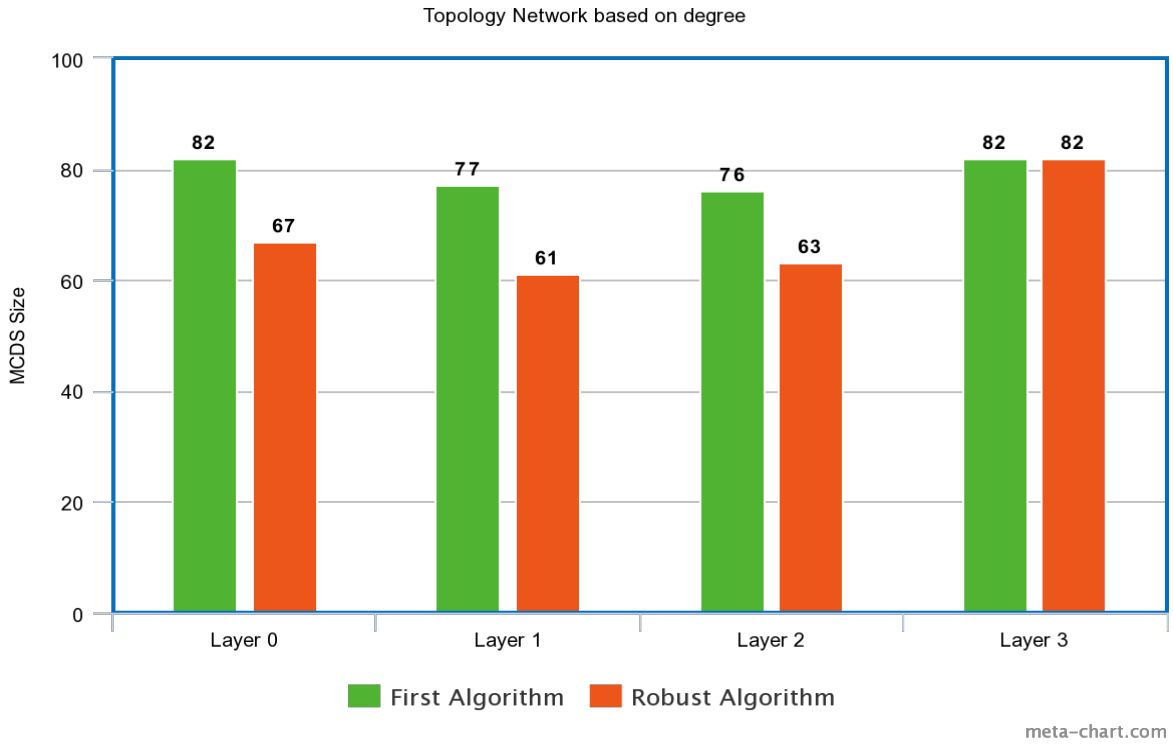


Figure 5.13: Average MCDS per layer of topologies based on Degree

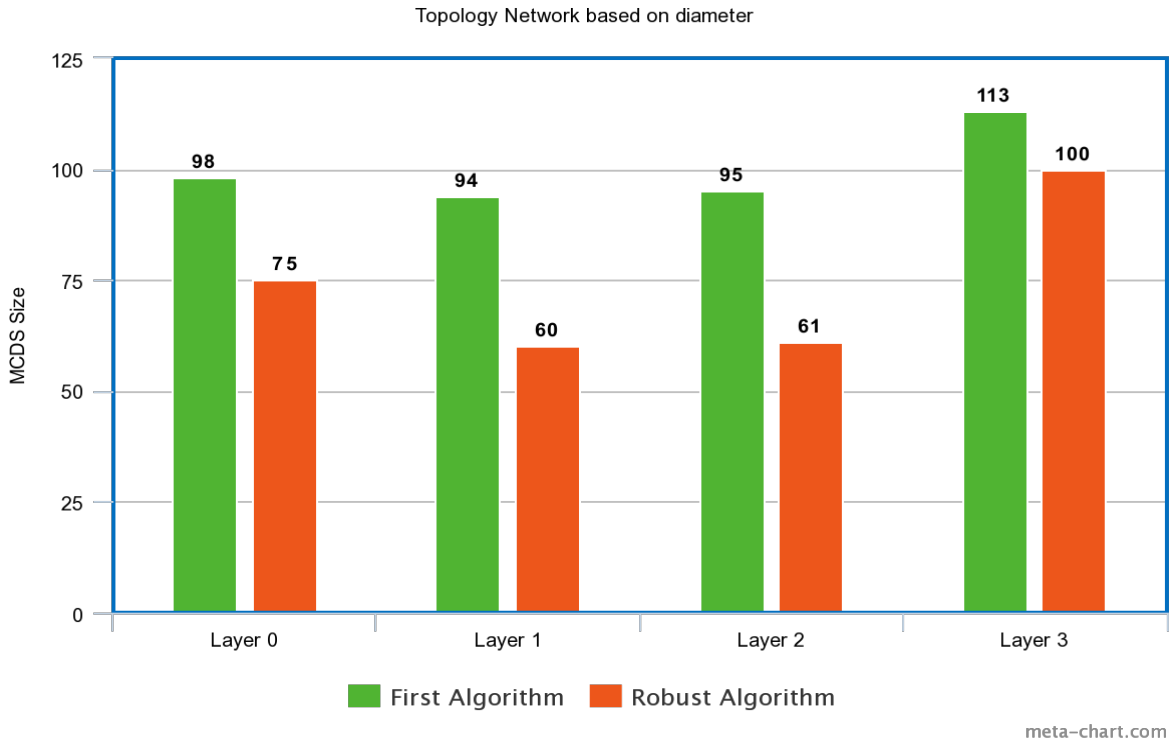


Figure 5.14: Average MCDS per layer of topologies based on Diameter

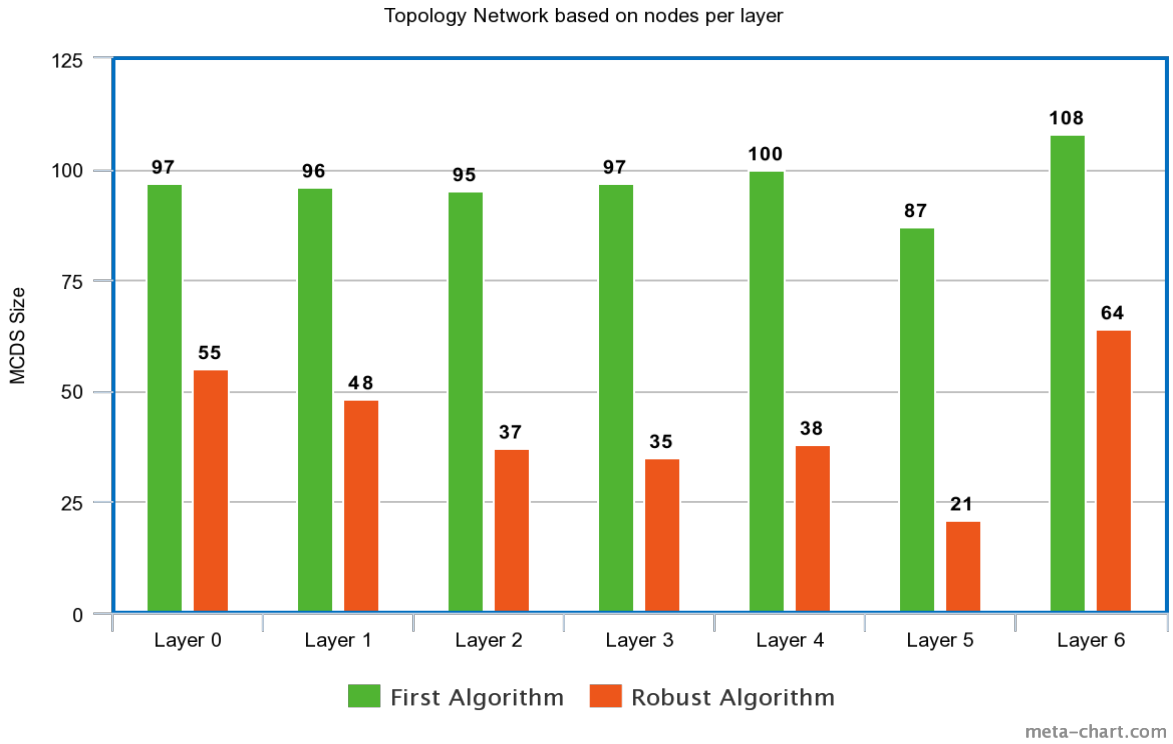


Figure 5.15: Average MCDS per layer of topologies based on Layers

The first two figures shows the average CDS and MCDS sizes or each layer for all network types. The rest of the figures shown the average size of CDSs and MCDSs which the two algorithms construct for all the network types (Network Degree, Network Diameter etc.). As we can see in all figures, robust algorithm construct smaller CDSs of BCA algorithm. This happens because the BCA algorithm costruct an initial CDS and continues by removing nodes from the existing CDS if those nodes are not significant for the others while add other nodes which not already included in CDS but are more important for the other nodes within CDS for the better stability of it. The robust algorithm has the same initial CDS but the extra steps leads to removal of nodes in existing CDS in higher rate than add nodes to.

Chapter 6

Conclusion

In this thesis, we represent a solution for constructing backbones for multi-layer networks. We used two different algorithms, one that creates 1-1-CDS and the other k-m-CDS as backbone. In the second algorithm the highest value of k and m is 3 because for a higher number it is not probable that a CDS will be created, because the complexity increases vastly. These two algorithms use 9 different PCI metrics or a combination of a PCI and the local centrality of each node. We have execute various input topology networks and get resulted backbones and we compare the results of the two algorithms.

Nevertheless, we don't compare these algorithms with some of the existing ones. Furthermore, we don't testing in depth the combination of PCI and centrality metrics as well as the ratio between them for better efficiency. Finally, we can also run some tests for values of k and m above 3.

Bibliography

[1] https://www.geeksforgeeks.org/introduction-of-mobile-ad-hoc-network-manet/,.

[2] N. Al-Nabhan, M. Al-Rodhaan, and A. Al-Dhelaan, "Distributed energy-efficient approaches for connected dominating set construction in wireless sensor networks," *International Journal of Distributed Sensor Networks*, Jun. 2014.

[3] https://en.wikipedia.org/wiki/Unit_disk_graph,.

[4] D. Papakostas, P. Basaras, D. Katsaros, and L. Tassiulas, "Backbone formation in military multi-layer ad hoc networks using complex network concepts," *IEEE Transactions on Network Science and Engineering*, 2016.

[5] D. Papakostas, S. Eshghi, D. Katsaros, and L. Tassiulas, "Energy-aware distributed edge domination of multilayer networks," *International Journal of Distributed Sensor Networks*, Jul. 2019.

[6] https://medium.com/@julien.carbonnell/theoretical-background-stakeholder-engagement-and-networ

[7] N.-S. Ahn and S.-S. Park, "An optimization algorithm for the minimum k-connected m-dominating set problem in wireless sensor networks."

[8] https://networkx.github.io/,.

[9] http://www.mkivela.com/pymnet/,.