ieeestyle

Middle East Technical University
Department of Computer Engineering
Wireless Systems, Networks and Cybersecurity (WINS) Laboratory

Fisheye State Protocol

CENG513 Wireless Communications and Networking
2024-2025 Spring
Term Project Report

Prepared by
Ahmet Umut AKKAŞ
Student ID: 2448066
umut.akkas@metu.edu.tr
Computer Engineering
20 June 2025

Abstract

I implemented Fisheye State Protocol in Omnet++6.1. I made tests on it to see the weaknesses of my implementations. I then iteratively (using the test results) improved my implementation to reach a final state. Then I made extensive analysis of it for the given input and output variables. We needed at least 5*5=25 tests. I did all of them with different data points and some repetitions. I used residual bootstrap method to obtain %95 confidence bands. I will share my interpretations of the results in the following sections.

Table of Contents

List of Figures

List of Tables

1 Introduction

In mobile ad-hoc networks (MANETs), nodes must dynamically discover and maintain routing paths without relying on fixed infrastructure or centralized coordination. Traditional link-state routing protocols, while providing optimal path selection through complete network topology knowledge, generate significant control overhead as they require every node to maintain and disseminate complete network state information. This overhead becomes particularly problematic in large-scale mobile networks where frequent topology changes due to node mobility, limited bandwidth, and energy constraints make the continuous exchange of complete topology information both inefficient and potentially destabilizing. The fundamental challenge lies in balancing the need for accurate routing information with the imperative to minimize control overhead in resource-constrained mobile environments.

Efficient routing in MANETs is critical for enabling numerous applications including emergency response networks, military communications, vehicular networks, and IoT deployments where infrastructure-based networking is unavailable or impractical. Solving the routing overhead problem enables these networks to scale to larger sizes while maintaining acceptable performance levels, directly impacting the feasibility of deploying MANETs in real-world scenarios. When this problem remains unsolved, networks suffer from excessive bandwidth consumption by routing protocols, reduced battery life due to frequent control message processing, increased packet loss due to network congestion from control traffic, and poor scalability that limits network size and practical applicability. The importance extends beyond technical metrics to real-world impact: inefficient routing can render emergency communication networks unreliable during critical situations, limit the effectiveness of military tactical networks, and prevent the deployment of large-scale sensor networks for environmental monitoring or smart city applications.

The routing overhead problem in MANETs is inherently difficult due to several conflicting requirements and technical challenges. Naive approaches that simply reduce the frequency of topology updates lead to stale routing information, resulting in suboptimal paths, routing loops, and packet delivery failures. Conversely, maintaining highly accurate topology information requires frequent updates that can overwhelm the network with control traffic, especially as network size increases quadratically with the number of nodes in link-state protocols. The mobility of nodes exacerbates this challenge by causing frequent topology changes that invalidate cached routing information unpredictably. Traditional solutions fail because they treat all routing information with equal priority: distance vector protocols like AODV suffer from slow convergence and counting-to-infinity problems, while pure link-state approaches like OLSR generate excessive overhead in large networks. The challenge is further complicated by the need to maintain loop-free routing while operating with partial or outdated topology information, the requirement to adapt quickly to network changes without generating control message storms, and the necessity to balance local routing accuracy (for immediate neighbors) with global reachability (for distant nodes) under severe resource constraints.

My FSR implementation addresses the routing overhead problem through a multi-layered fisheye approach with adaptive -to the mobility of the nodes- update frequencies (neighbor discovery) and custom realistic wireless channel modeling. The key components include: (1) Hierarchical Update Scheduling - implementing five different update timers (pubtim0-pubtim3, pubtimres) with exponentially increasing intervals based on the published network scope; (2) Dijkstra with Unknowns - providing a connection state: UNKNOWN and using those unknown connections to find a unreliable path if a reliable path cannot be found; (3)

Enhanced Neighbor Discovery - implementing a robust HELLO mechanism with counters that tracks neighbor reliability and automatically removes stale neighbors after 8 missed communications; (4) Collision-Aware Wireless Model - incorporating realistic physical layer effects including path loss, shadowing, capture effect, and receiver collision detection to accurately simulate wireless channel behavior; (5) Multi-path Routing with Reliability Tracking - using Dijkstra's algorithm enhanced with unknown link handling and random path selection among equally-good routes to improve load balancing and fault tolerance.

Several limitations constrain the approach: the simplified mobility model may not reflect real-world movement patterns and the fisheye scope parameters require manual tuning for optimal performance in different network densities and mobility scenarios. Additionally, the current implementation lacks adaptive scope adjustment mechanisms and does not handle network partitioning scenarios gracefully, potentially leading to connectivity issues in highly dynamic environments.

1.1 Background and Related Work

Since we just implemented a existing algorithm and the homework did not expect us to do a academical research, this section is not related at all. I -as most of the students- did not do a background and related work research. I just implemented FSR with my own contributions and made extensive tests on that implementation.

2 Main Contributions

2.1 Transmission Power of Control vs. Data Packets

At first, I was not using a advanced model to simulate the network. However, after I switched to a model which uses transmission power etc., I realized that control packets were dominating the data packets in the network. I decided to send the data packets twice as powerful as control packets. This way, they are more important and capture the receiver when they arrive. This decision is made by observation of low delivery ratios when the receiver capacity is limited.

2.2 Adaptive Neighbor Discovery Period

While doing iterative development and testing, I realized that mobility really affects how frequent the neighbor discovery packets should be broadcasted. So I changed and used this algorithm: $T_{neighbor\ discovery} = 1/(log(maximum\_speed + 2))$ seconds. In the other implementations, usually the neighbor discovery period was fixed to a number around 1-5 seconds.

2.3 New Way to Detect Connection Loss

At first, I was using a regular timer and if I did not hear from a node in that time period, I'd assume that we were disconnected. However, during my experiments (to plot data) -especially after switching to a realistic network simulation- I realized that whenever the network is congested, the neighbor discovery packets can take much longer times to successfully arrive. This causes the nodes to assume the connection is broken even if it is not.

My solution to this problem was introducing a new way of connection loss detection: I was keeping a vector of 1-byte counters in each node in the network. The counter keeps track of number of neighbor discovery (HELLO) packets received from that node. Whenever we obtain N discovery packets from a node, we check every counter for other nodes. If we have not heard from that node in that duration, we assume that we are not neighbors anymore. We then clear all counters and start again until we obtain N packets... This way, we can adapt to congestions better. In my case, N=8.

2.4 Residual Bootstrap (Adaptive Residual-Bootstrap Confidence Band for a Neural-Network Regressor)

In the experiments we need to compute confidence intervals. The confidence intervals are usually computed over a single dimensional statistic as we have learnt in the statistics course. However, the nature of the plots we need to present are not 1-dimensional. They represent 2 dimensional mappings.

We repeat a experiment with exact same inputs to calculate N outputs. We normally do this for every distinct input to obtain lots of data points to put into the plot. We obtain the confidence intervals for the mean of each distinct input's output. This way of calculating confidence intervals inherently *assumes* that the abstract ideal model's (simulation, real life environment dynamics) outputs for 2 different inputs are not correlated at all. However, in nearly any case, this is not true. For example, connectivity is a continuous input. The true means we try to estimate with error are also continuous by nature. So if we have 30 data points whose inputs are around 5 to 5,1 connectivity, this means that they are highly correlated with each other.

I start by training a light-weight neural-network regressor on our data **after first scaling both the inputs and the outputs** so the model learns smoothly. Once the network has produced its fitted values, I look at the left-over "errors" (the gaps between the real outputs and the network's guesses). To gauge the model's uncertainty, I take the leftover errors, scramble them, and add one shuffled batch back onto the model's own predictions—creating a slightly altered dataset each time, retrain a fresh copy of the network, and record its predictions. Repeating this many times gives us a whole cloud of possible prediction curves. Finally, for all desired x (sampled from the plotting range) I simply take the 2.5 % and 97.5 % points of that cloud to draw a **95 % confidence band** around the average prediction. Because the shuffled errors can be both positive and negative, the band can rise above or dip below the original data.

This hybrid technique unifies **classical error quantification** (bootstrap) with the **flexibility of neural nets**.

3 Results and Discussion

3.1 Methodology

For the connectivity, every node prints its neighbor count to a file regularly. I then average all the values to see how connected the overall network was.

Apart from that, my methodology to implement the Fisheye State routing protocol is as follows:

3.1.1 Simulation Environment and Tools

The FSR protocol was implemented as a custom module (‘WirelessNode‘) extending OMNeT++’s ‘cSimpleModule‘ class. The implementation consists of approximately 1,200 lines of C++ code, incorporating the complete FSR protocol stack including neighbor discovery, topology dissemination with fisheye scoping, routing table computation using Dijkstra’s algorithm with unknown link handling, and data forwarding mechanisms. The wireless channel model implements realistic path loss with log-normal shadowing, collision detection with capture effects, and SNR-based packet loss calculation to ensure accurate representation of wireless communication characteristics.

3.1.2 Network Configuration

The wireless transmission parameters are configured as follows: transmission power of 20mW, path loss exponent of 2.0, shadowing standard deviation of 4 dB, noise floor of -100 dBm, and interference level of -90 dBm.

3.1.3 FSR protocol parameters

The FSR protocol parameters implement the multi-scope fisheye mechanism with three defined scopes: scope 1 (immediate neighbors) updated every T seconds, scope 2 (2-hop neighbors) updated every 2T seconds, scope 3 (3-hop neighbors) updated every 4T seconds, and remaining nodes updated every 8T seconds, where T is dynamically calculated based on network mobility. Information aging increments are handled every T/2 seconds.

3.1.4 Traffic Generation and Mobility Model

Data traffic generation follows a Poisson process with exponentially distributed inter-arrival times. Data packets have a fixed payload size of 512 bytes plus protocol headers. Destinations are randomly selected using uniform distribution (each node generates data packets independently), excluding self-addressed packets.

Node mobility is implemented using a simple model, where nodes randomly select new directions (they may keep the direction with high probability for each tick) within the simulation field and move at speeds uniformly distributed between 0 and 20 m/s (maximum configurable speed). Direction changes occur with 10% probability at each mobility update interval (0.1 second), with boundary reflection implemented to maintain nodes within the simulation area.

3.1.5 Data Collection and Measurement Techniques

Performance metrics are collected using a comprehensive statistics gathering system implemented within each node. Every node records timestamped measurements at regular intervals and event-triggered recording for packet-level metrics. Key performance indicators measured are already explained in the homework text.

Network topology information is maintained in a NxN adjacency matrix representing connection states (CONNECTED, DISCONNECTED, UNKNOWN) with a age number (how old is that info). Connectivity statistics track the number of active neighbors per node over time.

3.1.6 Experimental Design and Randomization

The simulation employs multiple independent random number generators seeded with node-specific values plus random offsets to ensure statistical independence between nodes and simulation runs. Each experimental scenario is executed with different random seeds to generate statistically significant results across multiple runs. Protocol timing (timing of topology and neighbor discovery packets) includes random jitter (±10% of base intervals) to prevent synchronization artifacts common in discrete event simulations.

The experimental methodology includes systematic parameter sweeps. Each parameter combination is evaluated through multiple simulation runs with different random seeds, ensuring statistical validity of results.

3.1.7 Statistical Analysis and Data Processing

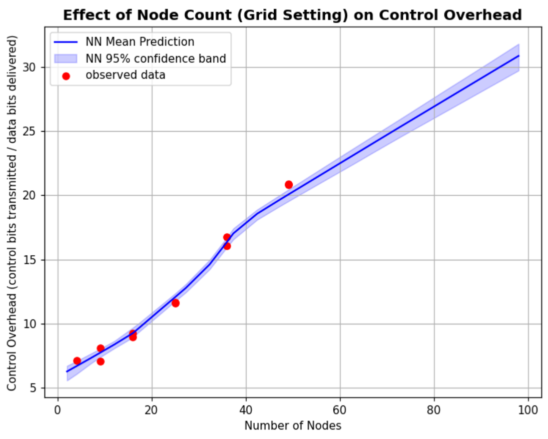
These are already explained in the contributions section. No need to repeat the homework text here.

All measurement data is exported to structured text files with timestamps and metric values, enabling post-processing analysis using external statistical tools (custom Python scripts).

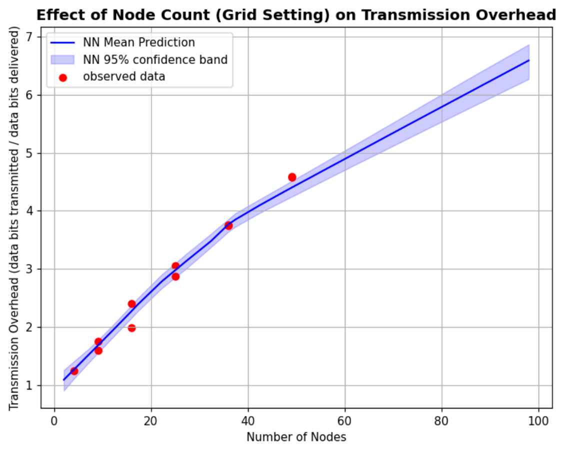
3.2 Results

3.2.1 Network Size

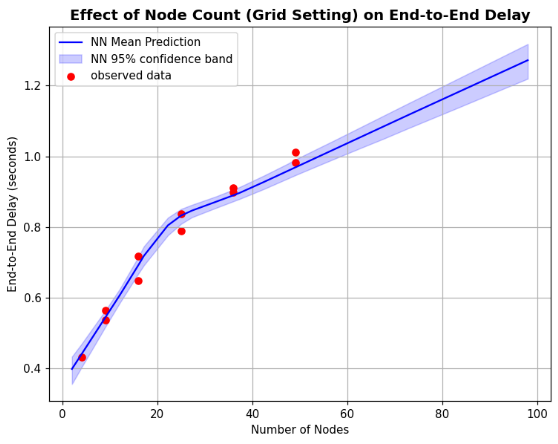
In the all experiments here, the topology is a grid. Mobility is enabled with a maximum speed of 2m/s. Receivers' (Nodes') reception capacity is 32kbps. The traffic generation rate of a single node is 0,08 pps.



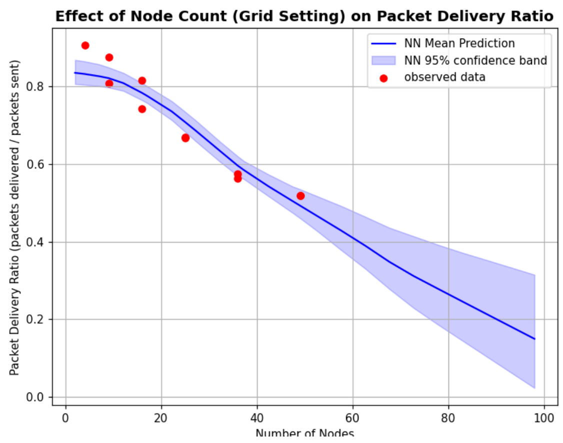
Here we see that the node count is directly related to the control overhead. This is probably because of my algorithm: Every node -as the last level of the topology publishings- sends all its more than 3 hop *neighbor*s at once periodically. I thought like this should either not exist, or be done like sending a single row of the corresponding rows of the topology matrix to reduce the control overhead. And I tried this approach. However, the results were actually worse.



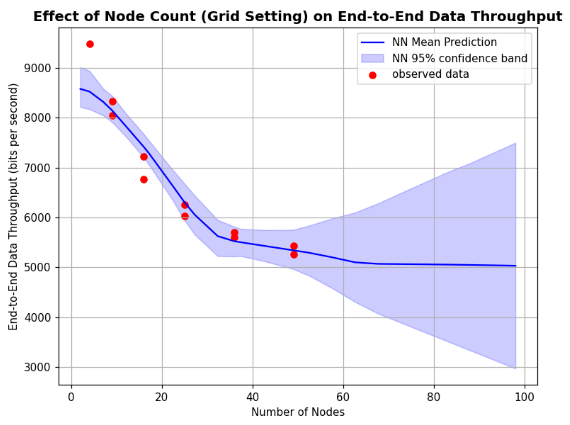
Here we also see the transmission overhead linearly increases with the node count. This is normal, since the more the nodes, the more the random destination's distance (hop count).



Here we see that up to a certain node count, delays increase linearly. Then, the increase rate drops. The reason for that is that when the size of the network increases, the information rate needed to keep a topological representation of that network also increases. However, the nodes have a limited reception capacity and the periods of published information does not depend on network size.



As you can see, the delivery ratio gets lower as the node count increases. This is because of the similar reason as above. The nodes need increasing time and bandwidth to have ideal representation of the network's topology.

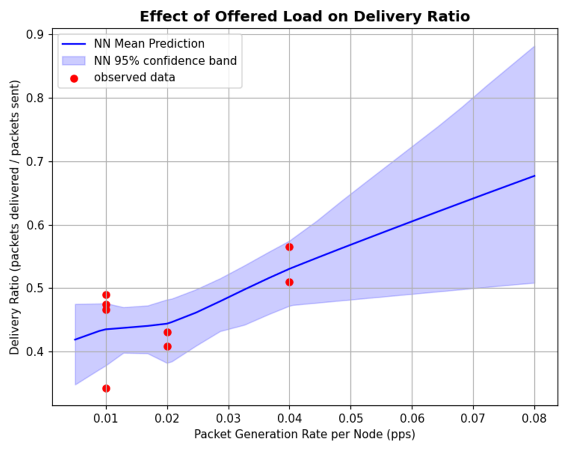


The interesting fact is the sudden change in the slope. It is because of the unreachable distant nodes causing a bias in throughput statistics (no arrival means no contribution). However, the dropped packets could be thought as 0 throughput values, this is my fault.

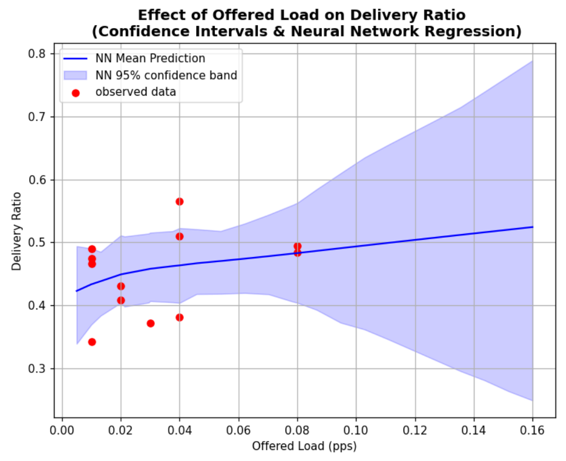
3.2.2 Offered Load

I did not do a unified testing for this. So I will explain all fixed parameters for every plot below.

Here, topology type is a circle. The field is 300*300 m. Mobility is enabled and the max. speed of the nodes is 2 m/s. The reception capacity of the nodes is 16kbps. And there are 16 nodes.

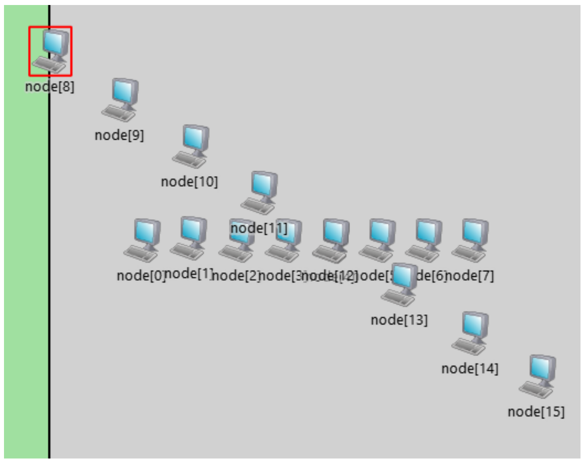


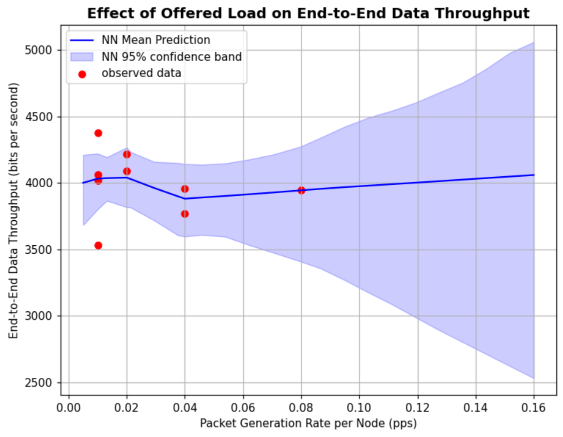
The reason of increasing delivery of the packets cannot be properly explained, this is why I conducted more experiments and had this result:



As you can see, the confidence intervals show that might be no change at all. So the load -intuitively too- does not affect delivery ratio.

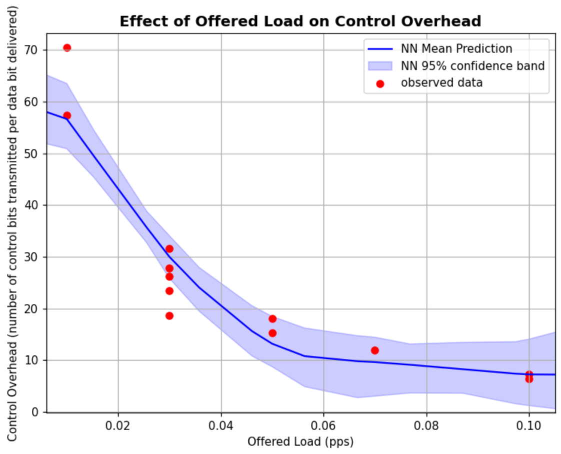
Here, I changed topology type to cross:





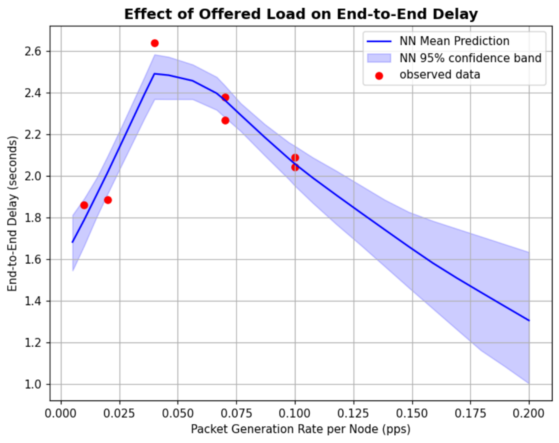
We also see that the offered load do not determine the throughput. The reason for that is the throughput is calculated per delivery. So as long as the packets are delivered, the throughput does not change much.

This is the older setting where the reception capacity is 8kbps and the topology type is a circle in a 200*200 field. The node count is 8 and maximum mobility is 4 m/s.



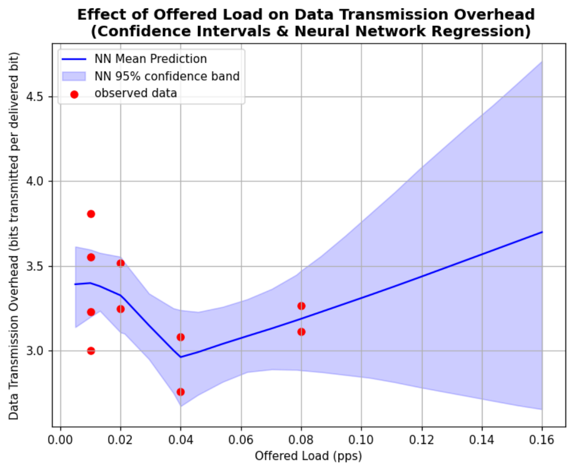
We see a lower control overhead as the load increases. Because the control packet frequencies do not change. But we also see a settling plot. My comment on this is that it could be due to the reception capacity.

Parameters are the same as the previous one here.



The reason of it starting from less delay, topping around a number and starting to decrease is as follows: The low packet generation rates such as 0.01 pps means that a node generates -in avg.- 1 packets per 100 seconds. The simulation takes 300 seconds so this is really low. Since data rate is not too much, the packet either gets lost becuse of [inference / busy channel], or it is delivered to a 1-hop destination (much less likely to get lost then.). The optimum represents a kind of optimal packet generation rate what is most suitable for the current setting (reception capacity). The decrease of delay (Seems like it gets better somehow, but it actually means the packets cannot reach to multi-hop destinations.) is then caused by highly loaded links where nodes creating lots of data packets to random destinations are interfering with each other. This means that most of the packets that are to be reached to a multi-hop destination gets lost, so is not received in the receiver to have their end to end delay recorded as a statistic datum.

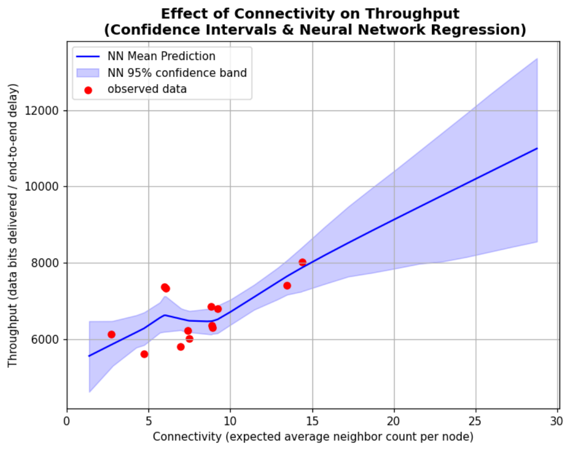
In this case, the number of nodes is 16 and field is back to 300*300. Also, the maximum speed is 2m/s and reception capacity is 16kbps.



We see no notable effect in transmission overhead.

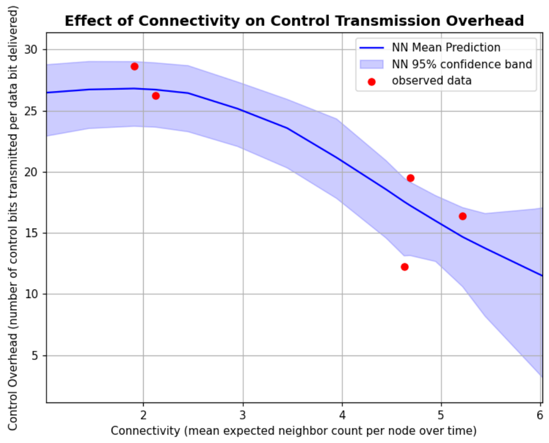
3.2.3 Connectivity

Here, data rate is 32kbps, max. speed is 2 m/s. Number of nodes is fixed at 25. The packet generation rate (offered load) is 0.08 pps. The connectivity is controlled by applying different topology types.



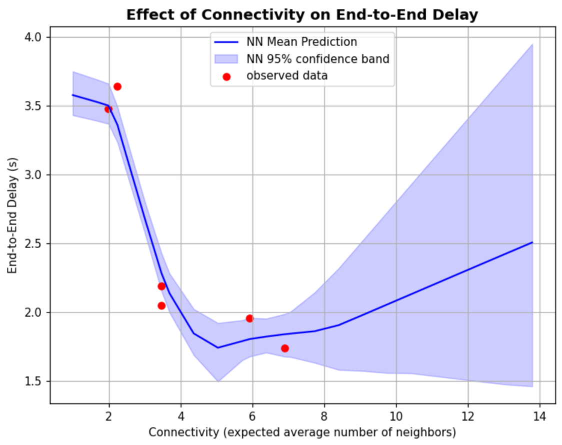
We see that increased connectivity results in a increased throughput. This is because less hops is needed to reach destination in more connected networks.

Here, data rate is smaller: 8kbps. The topology is circle. Field is 200*200 m. Max. speed is 4 m/s. Traffic generation rate is 0.04pps.



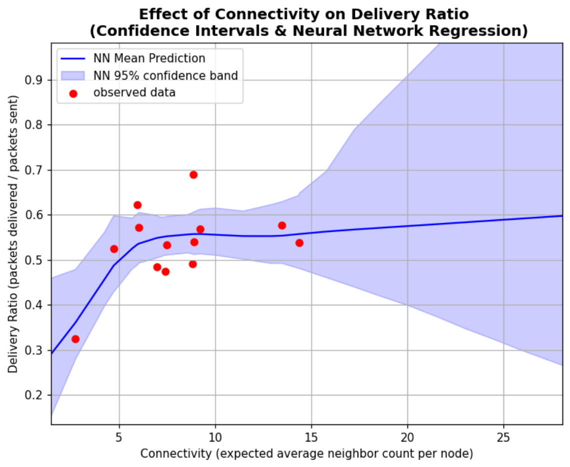
We see the expected result of reduced control overhead in more connected networks.

Here data rate is 8kbps and the node count is 8. Connectivity is controlled by tranmission range and diferent topologies.



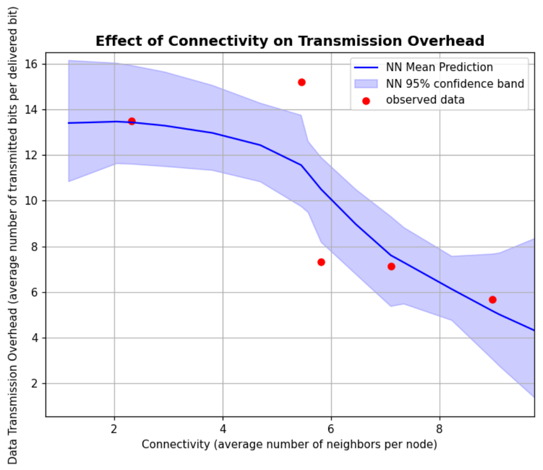
Here we see reduced delay when connectivity is increased. The expected result. This plot is from my earlier experiments. First, I was getting a constant delay value for different connectivity settings. This was due to the nature of multi-hop data packets getting interfered by control packets before they can reach to their destination. So I changed the code so that the data packets are sent using more transmission power than the control packets. This plot is the immediate result of my change.

Here, reception capacity is 32kbps, max. speed is 2 m/s. Number of nodes is fixed at 25. The packet generation rate (offered load) is 0.08 pps. The connectivity is controlled by applying different topology types.



We see that there is a minimum connectivity needed for the network to start functioning better. This seems to be around 4 (degree). The widening of the graph means that we do not have enough info to estimate how the delivery ratio will change when we increase the connectivity.

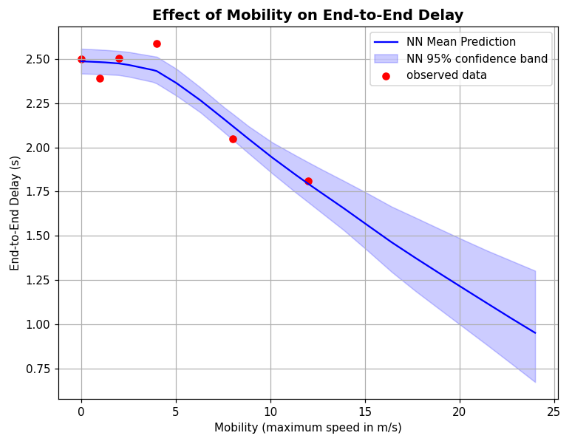
Here, there is a distinct setting. The reception capacity is 16kbps, topology is circle. Node count is 16 and mobility is enabled. However, the maximum speed is very high: 43 m/s. We handle the connectivity control using changing field sizes.



Here we see even in this fast speeds, the transmission overhead decreases when the connectivity is increased. This is meaningful since the maximum distance (hop count) to the destinations also decrease.

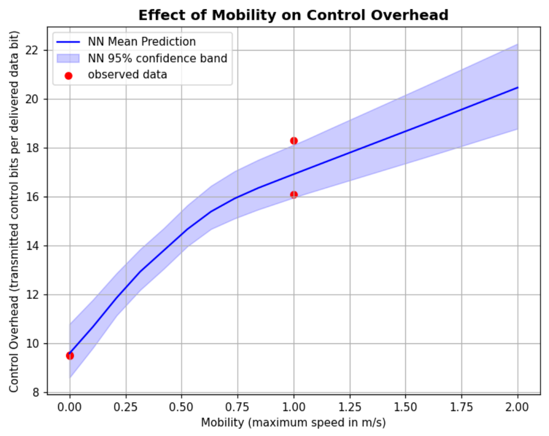
3.2.4 Mobility

Here, data rate is 8kbps. The topology is a ring (circle). The field dimensions are 200*200 m. The node count is 8.



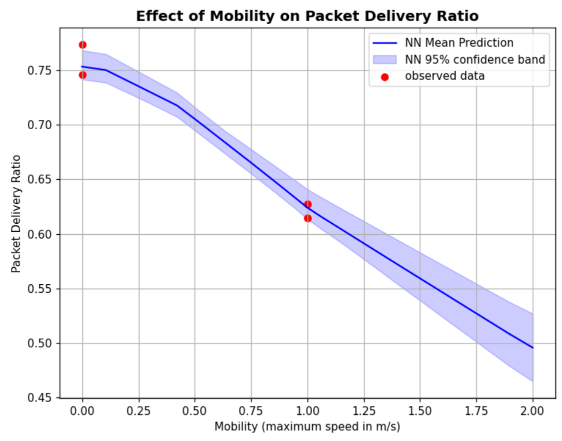
This plot clearly shows the amount of mobility the current protocol can bear. The mobility level the graph decreases is the limit. The decrease happens because multi-hop packets are starting to getting dropped.

Here, data rate is 16kbps and field dimensions are 300*300 meters. There are 16 nodes this time. The traffic generation rate is also 0.04 pps (default is 0.1).



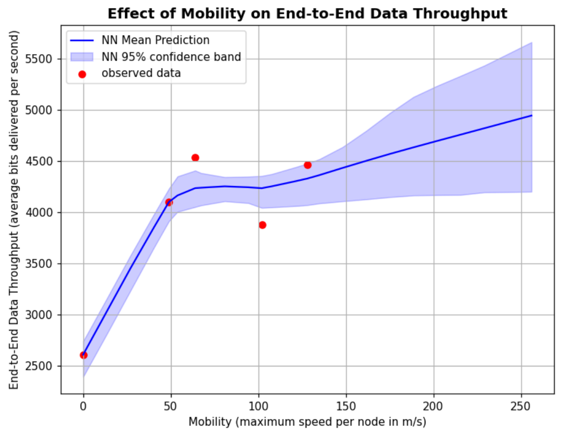
We see that, when the mobility appears, some data will be lost. This is why control overhead increases.

The same setup.



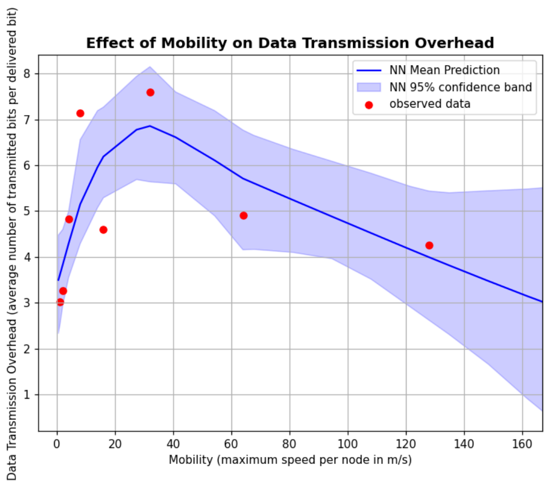
It shows the same thing as the former plot. This only measures the packets which -at the generation time- is thought to have -or which might have- a route to the destination. So if a packet -at the creation time in a node- has no -known- route apriori, it is discarded and this statistic is not updated.

The same setup.



You see that high mobility results in similar throughput. The immobile case here shows that the throughput decreases with mobility due to lost multi-hop packets. So after plotting this; I discovered that the more mobile the nodes, the higher frequency it is needed to discover new neighbors. So, I decided to change neighbor discovery packet period. I now divide it to log(max_speed+2). I also increased the reception capacity from 8 to 16kbps. Because, most of the data is getting lost over busy receivers and inference. Thanks to those, at least very tiny percentage of multi-hop packets can be successfully sent in high mobility (64m/s) in under the 300s simulation time.

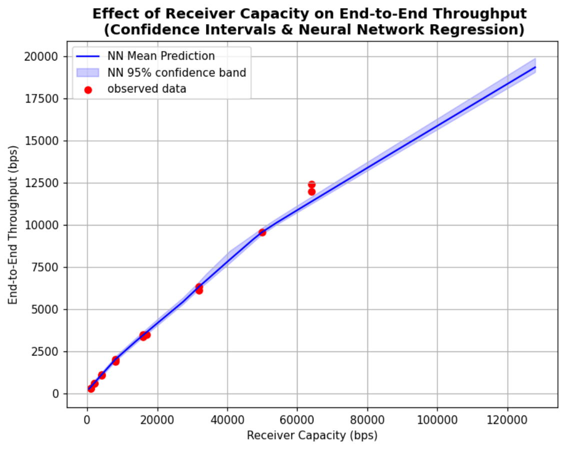
This is a older graph where the reception capacity is 8kbps and the node count is also 8.



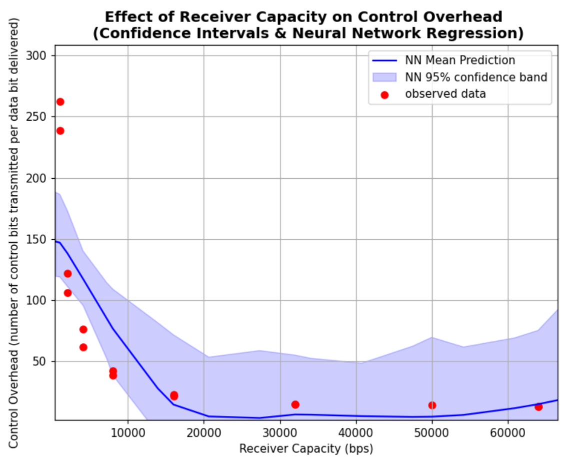
This plot, with seeing the previous plots, clearly shows that; in high mobility, the data packets are getting transmitted a couple of hops before getting discarded due to a unreachable destination. However, the decline in the graph after reaching very high mobilities (after 100 m/s), the data packets whose destinations are multi-hop are getting discarded immediately so the transmission overhead decreases since the packet is not transmitted at all. This is because, at those very high speeds, the nodes probably are not even able to form reliable connections to keep even a erroneous topology table... However, I -as said before- solved this by increasing the neighbor adversitement frequency according to the max. mobility.

3.2.5 Reception Capacity

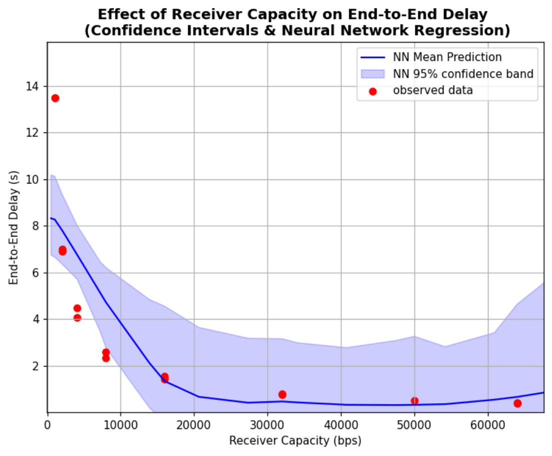
For all experiments below, mobility is enabled. The traffic generation rate per node is 0.08 pps. The node count is 25. Maximum speed of nodes is 3 m/s. The topology type is a pyramid shape where the nodes are arranged in a tree-like manner. The field is 500*1000 m.



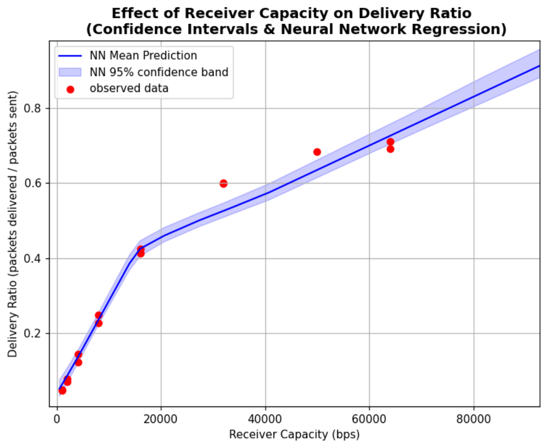
The linear relation is what we expect: Reception rate is increasing...



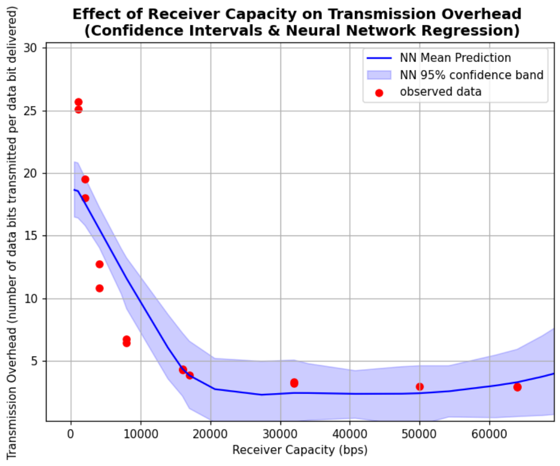
You can see the graph gets to a asymptote and keeps that horizontal line after a capacity. That shows the capacity needed for route information processing in the given setting. So after the capacity is reached, the data packets are getting sent better. The left part of the graph shows that the control packets are getting lost due to collisions etc.. This is a really important graph that shows the needed capacity.



This figure shows the normal decrease of delay. Transmission delay decreases when the reception capacity increases (we assume the transmission capacity is infinite.).



No need to comment. The sudden change of slope hints another bottleneck (the capacity might be stopping to be a bottleneck at that point).



We see a extremely similar plot to the control overhead. With this figure, it is obvious that the needed receiver capacity of our network setup is minimum around 7-8 kbps.

3.3 Discussion

I discussed the results in the previous section.

4 Conclusion

The *Fisheye State routing protocol* **actually just** defines a family of routing protocols. It is not well-defined in the internet and its boundaries are vague. The parameters and implementation details are usually left to the developer who plans to implement this algorithm in their hardware/software. It is a simple perspective towards the giant set of ad-hoc routing algorithms with mobility. It is probably mostly used to keep routing persistent among time to be able to deliver not-so-frequently sent packets quickly. The use case is mostly oriented around reducing the response time of a slightly mobile ad-hoc network without flooding it.

It can be expanded or mixed with other ideas -such as mine- to solve different problems of different needs. A ad-hoc network's features define the hyperparameters of the algorithm. This concludes my work and experience implementing and testing the FSR protocol.

References

Appendix A Assessment (I leave this section as-is for you.)

Your report will be assessed based on the following list of criteria.

Appendix A.1 Style

[15 points] The report states title, author names, affiliations and date. The format follows this style?

1. Structure and Organization: Does the organization of the paper enhance understanding of the material? Is the flow logical with appropriate transitions between sections?
2. Technical Exposition: Is the technical material presented clearly and logically? Is the material presented at the appropriate level of detail?
3. Clarity: Is the writing clear, unambiguous and direct? Is there excessive use of jargon, acronyms or undefined terms?
4. Style: Does the writing adhere to conventional rules of grammar and style? Are the references sufficient and appropriate?
5. Length: Is the length of the paper appropriate to the technical content?
6. Illustrations: Do the figures and tables enhance understanding of the text? Are they well explained? Are they of appropriate number, format and size?

Appendix A.2 Abstract

[10 points] Does the abstract summarize the report? These are the basic components of an abstract in any discipline:

1. Motivation/problem statement: Why do we care about the problem? What practical, scientific, theoretical or artistic gap is your research filling?
2. Methods/procedure/approach: What did you actually do to get your results? (e.g. analyzed 3 novels, completed a series of 5 oil paintings, interviewed 17 students)
3. Results/findings/product: As a result of completing the above procedure, what did you learn/invent/create?
4. Conclusion/implications: What are the larger implications of your findings, especially for the problem/gap identified?

Appendix A.3 The Problem

[15 points] The problem section must be specific. The title of the section must indicate your problem. Do not use generic titles.

1. Is the problem clearly stated?
2. Is the problem practically important?
3. What is the purpose of the study?
4. What is the hypothesis?
5. Are the key terms defined?

Appendix A.4 Background and Related Work

[15 points] Does the report present the background and related work in separate sections.

1. Are the cited sources pertinent to the study?

2. Is the review too broad or too narrow?
3. Are the references recent?
4. Is there any evidence of bias?

Appendix A.5 Design

[15 points] Does the report present the design of the study.

1. What research methodology was used?
2. Was it a replica study or an original study?
3. What measurement tools were used?
4. How were the procedures structured?
5. Was a pilot study conducted?
6. What are the variables?
7. How was sampling performed?

Appendix A.6 Analysis

[15 points] Does the report present the analysis?

1. How was data analyzed?
2. Was data qualitative or quantitative?
3. Did findings support the hypothesis and purpose?
4. Were weaknesses and problems discussed?

Appendix A.7 Conclusion and Future Work

[15 points] Does the report state the conclusion and future work clearly?

1. Are the conclusions of the study related to the original purpose?
2. Were the implications discussed?
3. Whom the results and conclusions will effect?
4. What recommendations were made at the conclusion?