A Performance Study of RPL with Trickle Algorithm Variants

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Computer Science
By

Daniel Benson

Date: 4/28/2016
Project Advisor:

Professor Robert Kinicki, Advisor

ACKNOWLEDGEMENTS

I would like to thank Zhouxiao Wu, who was originally my partner on this project and contributed greatly to the coding and testing aspects of this project. Due to graduation and time requirements, Zhouxiao has submitted separately but was still and integral part of my results.

I would also like to thank Professor Robert Kinicki for help and guidance on this project and his tireless reviewing of this paper.

ABSTRACT

The Internet of Things (IoT) is a worldwide technological revolution, bringing about new challenges in networking and data collection. The Routing Protocol for Low Power and Lossy Networks (RPL) is the industry standard for IoT Wireless Sensor Networks. This project studied the performance of RPL, focusing on evaluating the performance of four variations of the Trickle timer algorithm. The team performed all tests on a simulated environment using the Cooja network simulator. Overall, our newest proposed algorithm, ME-Trickle produced better results than the other algorithms under conditions of less dense networks. All four Trickle variants demonstrate sensitivity to changes in network configuration and density.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

Currently, the world of networking is turning its attention to a new idea called the Internet of Things (IoT). Conceptually, the Internet of Things is a network of many physical objects such as vehicles, buildings, appliances and other devices with embedded microprocessors, software, sensors, and network connectivity. This network enables the objects in it to collect and exchange data, creating a direct integration of computer systems and the physical world. The Internet of Things encompasses many technologies such as smart grids, smart homes, smart cities, and transportation. Each of these network technologies aim to improve energy efficiency, accuracy, and cost.

The Internet of Things (IoT) combines Wireless Sensor Networks (WSNs) with the ability to keep data in the cloud, because each device includes at least one sensor to collect information about an object or area. AWSN consists of many devices called nodes that need to be as energy efficient as possible because they are usually not connected to power, and may run for months or years on small, cost-effective batteries. The nodes do not communicate with the Internet individually, but instead do so via a single central point in the network called a border router. Because the size of a WSN can sometimes be very large, there is a need for messages to travel between nodes before reaching the central point. All of the communications between the nodes and the border router must be done wirelessly. These complications on the network introduce very important problems that engineers must solve in order for the network to be effective and reliable for a long period of time.

In networking, there is a stack of protocols which control how information moves within a network and how each of the members or nodes within the network handle the delivery of packets. The layer of the stack that this research focuses on is the network layer, which controls

how packets are routed. Each node in the network must make decisions on the route a packet must take in order for it to reach its destination. These decisions are made by the routing protocol. Some wireless sensor networks use the MicroIP (μIP) stack, an open source stack implementation of TCP/IP for 8 and 16 bit microcontrollers[4]. The MicroIP stack supports IPv6 with 6LoWPAN protocols, including the Routing Protocol for Low Power and Lossy Networks (RPL). RPL is a proactive routing solution designed specifically for WSN's. In order for a routing protocol to be successful, each node in the network must help to maintain the best route to a given destination. This is accomplished by periodically updating the neighbor nodes with the information they need to determine the best route, through packets sent wirelessly to their neighbor nodes. These packets create routing overhead in the WSN.

This project evaluates the performance of RPL under several different network topologies and densities. The project team varied RPL parameters and the Trickle algorithm that control how the protocol functions. The project focuses specifically on the Trickle timer algorithm, which controls the volume of routing packets transmitted at each node in the network [6]. While researchers have proposed many different versions of the Trickle algorithm, the project team chose to evaluate four specific Trickle algorithms introduced in chapter 2. The issues that are important to this research are the energy efficiency of the sensor nodes, the latency and reliability of messages traveling through the network, and the overhead introduced by the routing protocol packets. We also looked into RPL's performance under different network topologies, studying the effects of node density and topology differences on each of the Trickle algorithm variations.

In order to evaluate the four different Trickle algorithms, this investigation used the Cooja simulator inside of the Contiki operating system. Cooja is a network simulator made for IoT purposes. This project was done entirely in simulation due to the time constraints on the

project. The methodology chapter discusses our testing suite and the simulations that we ran to evaluate the algorithms' performance.

The goal of this project was to conduct a performance evaluation study on the effects of four variations of the Trickle algorithm in RPL within Wireless Sensor Networks. This work attempts to identify the best solution for different network topologies, traffic conditions, and application scenarios. and simulations show ME Trickle generally performs the best under sparser network topologies. ME-Trickle provides the best performance across all of the metrics for configurations that are less dense than our original 80 node test. In addition, the results demonstrate that the four variations of the algorithm are sensitive to changes in the configuration and density of the network. Hence, this investigation shows that there is not a clear winning Trickle algorithm among the four variants for every single network scenario. The final chapter of this report suggests several potential research directions for future work on Trickle performance.

2. BACKGROUND

This chapter introduces information needed to understand the scope and purpose of this project, the tools and systems we used for our simulation tests, and finally the four studied variations of the Trickle algorithm.

2.1. Contiki and Cooja Simulation

This investigation required a test environment that would run implementations of RPL to evaluate the performance of four distinct versions of Trickle. Since the physical implementations on wireless motes would be too difficult for a two-term Major Qualifying Project (MQP), the team decided to utilize a simulator to assess Trickle performance running on the Contiki Operating System. Contiki is an open source operating system built specifically for the application of the Internet of Things [5]. Built on the Linux kernel using a version of the Ubuntu distribution, Contiki interfaces to the MicroIP protocol stack to connect low cost and low power microcontroller devices to the Internet.

This research selected Contiki as the operating system for many reasons. The main reason is that Contiki supports the popular microIP (µIP) networking stack which includes full support for the IPv6 standard. Contiki also includes the Cooja network simulator, because Contiki often runs on a large number of the nodes in a wireless sensor network. While developing and debugging software written for these types of networks can be very challenging, Cooja provides a convenient simulation environment for a variety of fully emulated hardware devices which enables observations on large scale networks and applications with extreme detail. It also gives control over a wide range of different devices and network topologies. This study used the Cooja

simulator to implement the changes to the Trickle algorithm as described in Section 2.5, vary the parameters of RPL, and analyze the data of its performance throughout the simulated network. The Cooja simulator provides tools to output data from each test very easily in a readable format. This made it very simple to produce graphs from the output by using Python scripts to filter the resultant data into desirable performance metrics.

Another reason the team chose Cooja is that it provides a large amount of example code that can be used as a base for this project's implementation code. This sample code featured the MicroIP stack which was very helpful. Additionally, Cisco has developed the Contiki IPv6 stack and the stack is fully certified under the IPv6 Ready Logo program.

2.2. RPL

Researchers developed the Routing Protocol for Low Power and Lossy Networks (RPL) specifically for Wireless Sensor Networks (WSN) in the Internet of Things.

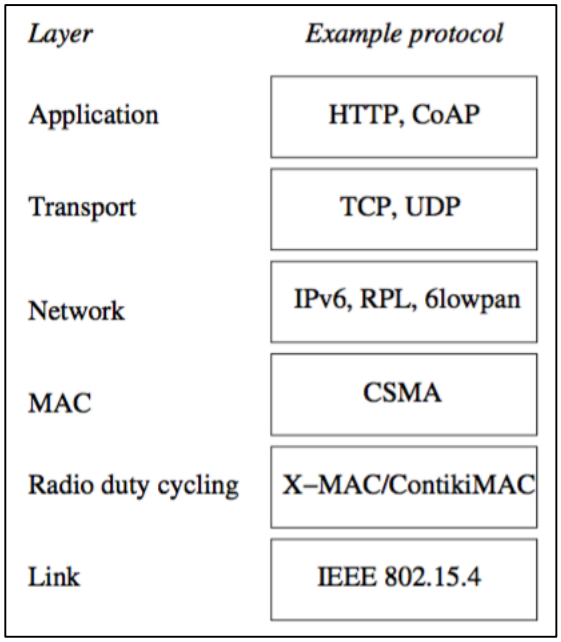


Figure 1: MicroIP Protocol Stack [7]

RPL operates on the layer above the IEEE IPv6 6LoWPAN above IEEE 802.15.4 wireless networking standard [7](see Figure 1). The protocol is divided into a forwarding engine, which uses a routing table to decide which neighbor node is the next hop for a given packet, and the routing protocol, which populates the routing table at each node. RPL works by assigning each node a rank, such that a node's rank increases the farther the node is from the border router. This creates a graph of the paths of communication through the network called a Destination Oriented Directed Acyclic Graph (DODAG) [1].

The border router is the node in the network which does not have the same constraints as all the sensor nodes in the WSN. It does not consider power usage because it will normally have available power and it will not be collecting its own data. The role of the border router is as an interface to the Internet and to aggregate the information that it receives from the WSN nodes. The border router can also control the sending of updates to the nodes. An example of this is a code patch that needs to be inserted at each node. The border router would then broadcast this patch.

RPL works in two directions from the border router: upward routing transmitting packets towards the border router from any of the sensor nodes, and downward routing sending packets from the border router to any node. There are three types of messages used in the creation and optimization of the routing table in RPL. DODAG Information Objects (DIO's) form the DODAG and maintain the best communication routes at each node. DODAG Information Solicitations (DIS) solicit DIOs from other neighbor nodes. Finally, DODAG Destination Advertisement Objects (DAO) support downward paths to parent or ancestor nodes.

2.3. Trickle

Inside of RPL, there is a timer which uses the Trickle Timer algorithm to control the updating and construction of the DODAG. The DODAG contains the information for forwarding the packets every node receives. The Trickle algorithm controls the amount of routing traffic in the form of DIO's that enter the network. It also controls the amount of time that a node listena for new information and how often it senda out its current information to its neighbor nodes. This project evaluates four algorithms for the Trickle timer. Section 2.5 discusses these algorithm variations in detail.

2.4 Objective Functions

An objective function defines the strategy that a RPL node uses to select its transmission path and then optimize its routes based on the information objects available at that node. An objective function uses a metric to evaluate the paths of communication for each of the wireless links. This metric could be throughput of the connection, latency, or qualities about the node such as how it draws its power. The objective function looks at the network quantitatively using these metrics to optimize routes through the network and fulfill the network constraints. RPL encodes the metric into the objective function logic that it uses during its operation to decide the best transmission routes. There are two objective functions implemented inside of Contiki's RPL, OFo and ETX. OFo uses hop count as a routing metric while ETX uses the Expected Transmission Count (ETX) as a metric for selecting the optimal path. Expected Transmission Count is the expected number of transmissions needed for a packet to successfully reach its destination.

By using different objective functions RPL and Trickle strive to satisfy many optimization criteria for a wide range of devices, network topologies and applications.

2.5. Literature Review and Algorithm Comparisons

Trickle is a propagation scheduling mechanism initially developed for reprogramming algorithms to efficiently disseminate code updates through a Wireless Sensor Network. However, researchers have found Trickle to be a robust technique usable over a broad scope of applications, including service discovery, control traffic scheduling, and multicast propagation [3]. Due to its reliability, scalability, and low maintenance cost, the Trickle algorithm is very popular and it is the focus of many recent research papers on the Internet of Things. Some researchers claim that because the Trickle algorithm is already concise and efficient, it is not worth the cost to tweak its behavior in most cases [7]. However, through the study of the algorithm in multiple scenarios and network topologies, researchers have gradually introduced new possible variations. There is initial evidence that these new variations may out-perform the original Trickle Algorithm in certain cases and situations [3] [4].

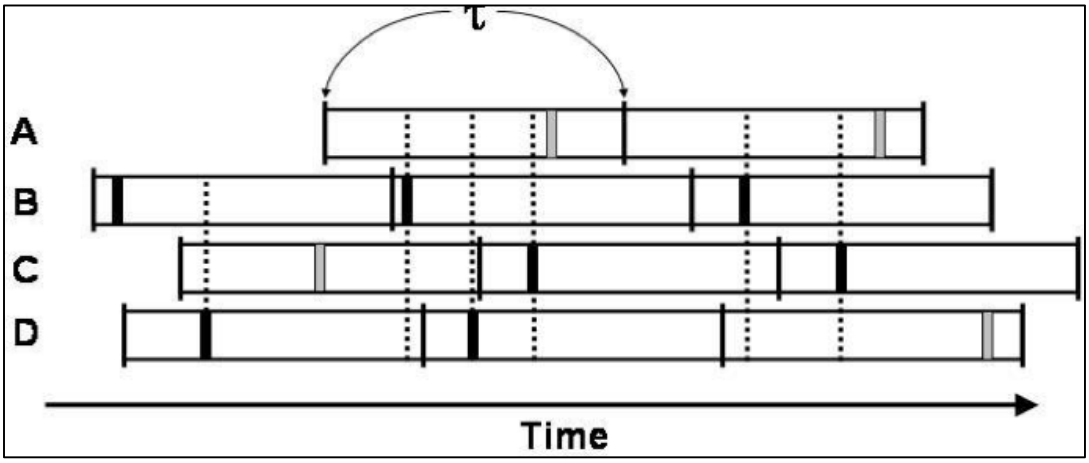


Figure 2: The Short Listen Problem for Motes A, B, C, and D

The original Trickle algorithm, documented as an Internet standard in RFC 6206 [1], introduces a listen-only period to solve the short-listen problem. This algorithm chooses the transmission time *t* uniform randomly between half of an interval to one interval. Hence, as

Trickle never causes a transmission in the first half of a interval, this becomes the listen-only period.

The short-listen problem is the one major difficulty that the Trickle algorithm encounters [7]. There would not be an issue if Trickle synchronized every node but since this is not the case, redundant messages are sent. This is because certain node's messages will not reach every node in the network. Therefore, those nodes that always multicast first will use too much energy. Greater energy consumption is very detrimental in Wireless Sensor Networks because of the limited power resources available to the device. Figure 2 demonstrates a sample of the short listen problem. B transmits soon after the start of all intervals shown in the graph, while A suppresses every time. The following is a detailed explanation of short-listen problem:

> "Trickle can suffer from the *short-listen* problem. Some subset of motes gossip soon after the beginning of their interval, listening for only a short time, before anyone else has a chance to speak up. If all of the intervals are synchronized, the a mote's interval begins just after the broadcast, and it too has chosen a short listening period. This results in redundant transmissions" [9].

The second algorithm that we introduce attempts to solve the issues of the short listen problem of the original Trickle algorithm.

Ghaleb came up with an efficient method for studying three different Trickle algorithm variants. This project adapted their solution to compare four versions of Trickle. The following six steps recap the operation of the basic Trickle algorithm [4]:

1) Trickle starts its first interval by setting I to a value from the range $[I_{min}, I_{max}]$, usually it sets the first interval to a value of I_{min}.

2) When an interval starts, Trickle resets the consistency counter c to 0, and assigns a randomly selected value in the interval to the variable t, chosen from the range [I/2, I). This helps to reduce effects of the listen only period.

3) Upon receiving a consistent message, Trickle increments its counter, *c*, by a value of 1.

4) At the randomly chosen time *t*, if the counter *c* is greater than or equal to the redundancy constant, *k*, Trickle suppresses its scheduled message. Otherwise the node sends its RPL message.

5) When the interval *I* expires, Trickle doubles the size of the interval. If the size of the new interval would exceed the maximum interval length I_{max}. Trickle sets the interval size I to I_{max} and re-executes the steps from step 2.

6) If Trickle detects an inconsistent message, Trickle sets I to I_{min}, if it was not already set to I_{min} and starts a new interval as in step 2. An inconsistent message is a DIO which has information that is different from the current best route information.

The following pseudo-code presents the original Trickle algorithm that is inside of RPL. The different steps are described above as the algorithm works through different cases of receiving information objects from its neighbors.

Original Trickle

```
I. Initialization
        I ← I_min
II. Start New Interval (infinite loop)
        I ← I ×2
        c ← 0
        if I_max ≤ I then
                I ← I_max
        end if
        t ← random[I/2, I)
III. Received Consistent Transmission
        c ← c +1
IV. Received Inconsistent Transmission
        I ← I_min
V. Random Timer Expires
        if c < k then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO
        end if
```

Figure 3: Original Trickle pseudo-code adapted from [4]

The listen-only period makes Trickle scale logarithmically with the density of nodes in a network, however, it increased delays. According to the authors of *E-Trickle: Enhanced Trickle Algorithm for Low-Power and Lossy Networks*, "this I_{min}-dependent delay gets accumulated at every hop in multi-hop networks, which results in a considerable latency for a packet travelling long distances in terms of hops" [4]. Hence, Ghaleb proposed an optimized version of Trickle (referred to as opt-Trickle in the rest of this report), which, when a new interval begins, chooses *t* values based on the current state. If the interval is reset, it chooses *t* from 0 to I_{min}, and if it was

newly configured or started from an expired interval, it chooses *t* from half of interval to the whole interval.

Figure 3 provides the pseudo-code for optimized Trickle. The plus signs indicate the difference between optimized algorithm and the Original Trickle by adding new pieces of code.

Optimized Trickle

```
I. Initialization
        I ← I_min
II. Start New Interval (infinite loop)
        I ← I ×2
        c ← 0
        if I_max ≤ I then
                I ← I_max
        end if
+       if from step V
+               t ← random[0, I_min)
+       else
                t ← random[I/2, I)
+       end if
III. Received Consistent Transmission
        c ← c +1
IV. Received Inconsistent Transmission
        I ← I_min
V. Random Timer Expires
          if c < kn then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO

        end if
```

Figure 4: Optimized Trickle pseudo-code

However, researchers criticized the opt-Trickle's assumption of a MAC protocol with 100% duty-cycle, which is neither reasonable nor realistic [8]. Additionally, opt-Trickle still has the listen-only period, and would lead to increased latency, especially in a lossy network. Thus a Levin [8] presented a new algorithm called E-Trickle which does not have a listen-only period. Instead of resetting *c*, the consistency counter, at the beginning of the interval, it resets *c* at a randomly chosen time to eliminate the cumulative effect of the short-listen problem. However, this yields unequal intervals for some of the nodes. With unequal intervals, some nodes have a much higher likelihood to transmit. In other words, some nodes might transmit much more frequently, which is the exact result of short-listen problem. To solve this problem, Ghaleb [4] came up with a mechanism to stretch *k*, the redundancy constant, accordingly in order to make every node to have roughly the equal chance to transmit. In their solution, they introduced a new variable named I_{nz}, which is the time difference between two transmission times. The value of k was readjusted using the following formula:

$$\text{new } k = (k \times (2 \times I_{nz} - I)) / I$$

Figure 5: Formula to calculate new k

Although Ghaleb [4] did not explicitly state how they came up with this formula, it can be inferred that as I_{nz} becomes larger, the value of k grows as well. In other words, if a node has two transmissions that are far apart, the k value will increase so that the redundancy counter c will be more likely smaller than k, hence it is more likely that the node will transmit.

The following pseudo code figure presents the E-Trickle algorithm. The plus signs in the diagram represent the adding of code to the original Trickle Algorithm. The minus signs represent code that has been removed from the original Trickle algorithm.

E-Trickle

```
I. Initialization
        I ← I_min
+       c ← 0
II. Start New Interval (infinite loop)
        I ← I ×2
−       c ← 0
        if I_max ≤ I then
                I ← I_max
        end if
+       t ← random(0 , I)
−       t ← random[I/2, I)
III. Received Consistent Transmission
        c ← c +1
IV. Received Inconsistent Transmission
        I ← I_min
+       c ← 0
V. Random Timer Expires
+       if I_nz > I
+               kn ← ( k × ( 2 × Inz – I )) / I

+       else
+               kn←k
+       end if
          if c < kn then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO
```

```
            end if
+           c ← 0
```

Figure 6: E-Trickle pseudo-code adapted from [4]

Finally, E-Trickle's authors pointed out a possible modification for their Trickle version which is promising. They observed that in most of scenarios, all the nodes were able to resolve the inconsistency within two intervals. Hence, it might be more efficient for all the nodes to jump to the maximum interval length instead of doubling the interval multiple times when there is no inconsistency detected. Hence, the node would send fewer RPL packets, which would conserve more energy, while still fixing the problems with inconsistency. This report refers to this variation of the E-Trickle algorithm as the Modified E-Trickle (ME-Trickle). The "#" symbol in Figure 6 indicates the difference between ME-Trickle and E-Trickle. The difference is jumping to I_{max} immediately rather than doubling the interval. The normal plus signs are still the new lines of code relative to the original Trickle algorithm.

ME-Trickle

```
I. Initialization
            I ← I_min
+           c ← 0
II. Start New Interval
#−          I ← I ×2
#+          I ← I_max
−           c ← 0
            if I_max ≤ I then
                        I ← I_max
            end if
+           t ← random(0 , I)
```

```
−       t ← random[I/2, I)
III. Received Consistent Transmission
        c ← c +1
IV. Received Inconsistent Transmission
        I ← I_min
+       c ← 0
V. Random Timer Expires
+       if I_nz > I
+               k_n ← ( k × ( 2 × I_nz – I )) / I

+       else
+               k_n←k
+       end if
          if c < k_n then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO
        end if
+       c ←  0
```

Figure 7: ME-Trickle pseudo-code

The source code for the four Trickle algorithms are available online.

3. METHODOLOGY

This chapter introduces the experimental methodology the team used to evaluate the performance of the four Trickle variants. The chapter discusses the important testing parameters, network topologies and densities, and data collection and visualization involved in this research.

3.1 Testing Configuration and Network Topologies

3.1.1 Parameters

This investigation varied the some of the parameters in Table 1 (denote with a *) to provide a thorough examination of RPL performance within the Contiki OS. The team chose these parameters based on the research described in the Background chapter, as well as, the sanity check described in Section 3.2. They consist of network stack parameters and Cooja simulator parameters. The team performed a series of simulations in which they isolated a single varied parameter on each of the four algorithms and then studied the effects of the parameter on the algorithms' performance.

The starred parameters that we varied to study the performance of the Trickle algorithms are the DIO minimum interval, DIO doubling, radio duty cycling check rate, and frequency of application messages [1]. DIO minimum interval is the smallest interval possible send interval as described in the previous section as I_{min}. The DIO doubling is maximum number of times that I_{max} can double before hitting its max value. In the case of ME Trickle, this means that it is increasing to the maximum interval immediately rather than doubling. Radio duty cycling check rate is the number of times that the radio is checking the medium for a possible message at a rate per second. Finally, the frequency of application messages is what we will call send interval in the results section. This is the amount of time between the "hello" messages sent from the application layer.

Table 1 RPL Parameters and Locations in Contiki and Cooja

Parameters	**Names in Contiki**	**Locations**	**Default Values**
RPL Objective Function	RPL_OF	rpl-conf.h	ETX
DIO Min*	RPL_DIO_INTERVAL_MIN	rpl-conf.h	12
DIO Doubling*	RPL_DIO_INTERVAL_DOUBLINGS	rpl-conf.h	8
RDC Channel Check Rate*	NETSTACK_RDC_CHANNEL_CHECK_RATE	netstack.h	16
Trickle Redundancy counter, *k*	REDUNDANCY_COUNTER	Rpl-conf.h	10
Send Interval*	SEND_INTERVAL	udp-client.c	4
Reception Ratio	RX Ratio	Cooja	70%
Transmission Ratio	TX Ratio	Cooja	100%
Transmission Range	TX Range	Cooja	50 meters
Interference Range	INT Range	Cooja	55 meters
Start Delay	Mote startup delay	Cooja	1 second

The testing focused on isolating each one of the parameters to find their effects on the behavior and performance of the four algorithms. A series of tests would be run to vary a single

parameter over a range of values, while all of the other parameters remained constant. The team then simulated each of the four algorithms over the range of the chosen parameter.

There were also a number of parameter choices which did not change throughout all of the testing: the objective function, the receive rate, the transmission range, the interference range, the size of the simulated testing area, and the total simulation time. Initially, our simulations matched the parameter settings in Ali's master's thesis [1] to provide a sanity check for our implementations of Trickle. The team decided to keep these six parameter settings for the remainder of the tests for consistency, and because they represented reasonable and realistic networks.

As the ETX objective function is generally accepted as the industry standard, all simulations employed ETX for the objective function of RPL. Ali's master's thesis [1] provided the values used in the simulations for receive rate, transmission range, and interference range. Table 2 provides these values.

Table 2 Simulation Parameter Values

Objective Function	ETX
Receive rate	70%
Transmission range	50 meters
Interference range	55 meters
Testing Area	100 meters by 100 meters
Simulation time	15 minutes

The team performed extensive tests to determine how long to run the simulations such that the obtained values had small variance and to be sure that the setup time of the network was not significantly affecting the results. With two or three minutes of simulated time, the results were very inconsistent. The team increased the simulated time in five minute intervals from five minutes to 20 minutes, and analyzed the change in the observed performance results. Since the tests that were run for 15 minutes produced results similar to those run for 20 minutes, we chose 15 minutes of simulated time as the standard duration for all our Cooja simulations.

3.1.2. Network Topologies

The team evaluated three distinct network topologies over varying node densities. The simulation testing began with a dense layout of 80 client nodes in a 100m x 100m area. The 81st node in the upper left corner represents the border router. The border router is placed in the corner so that there would be nodes with multiple hop transmission distances from the border router. Figure 8 depicts the original layout with the 80 nodes randomly placed in the simulated area.

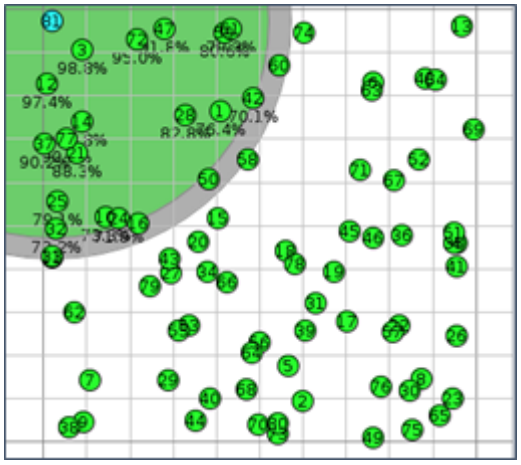


Figure 8: Random topologies with 80 client nodes

The majority of the testing used the above layout to facilitate comparisons to Ali's master's thesis. Section 3.2 describes the method for this comparison.

The next testing stage involved testing different node densities. The team analyzed at Trickle performance at 40 nodes and 10 nodes in the same 100m x 100m area. Figure 8 depicts the random topology for the 80 node test, while Figures 9 and 10 show the 40 node and 10 node randomly chosen topologies. Again, for each of these sets of simulation tests, the team placed the border router (BR) node in the upper left corner of the available area. If the BR was in the middle of the area, it would not produce interesting results as it could send to nearly every node in the topology in one hop and the routing algorithm would not matter.

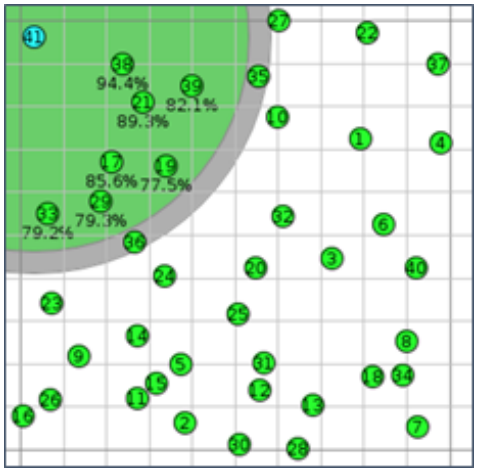


Figure 9: Random topology with 40 client nodes

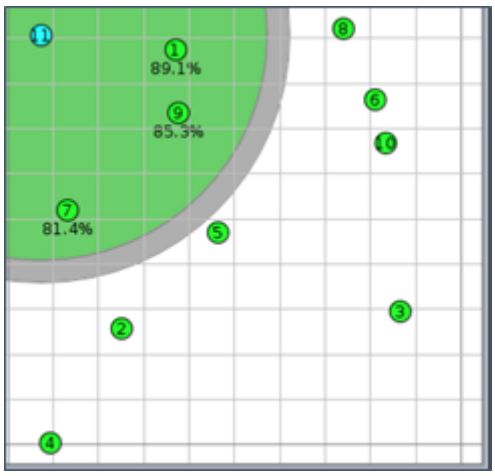


Figure 10: Random topology with 10 client nodes

After running the set of simulation tests with random topologies, the next testing stage was to test a grid configuration. The team decided to simulate the same number of client nodes in order to simplify comparison with the data previously collected. Figure 11, 12, and 13 illustrate the set up for the grid layouts. As previously done, all configurations placed the border router node in the corner of available area to create more interesting results. Since the

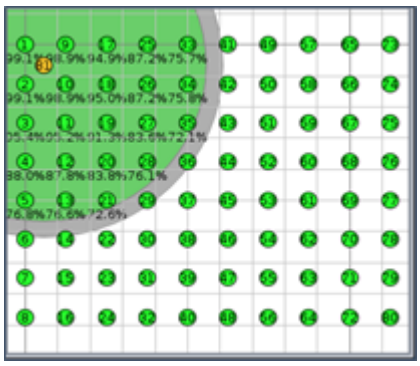


Figure 11: Grid topology with 80 client nodes

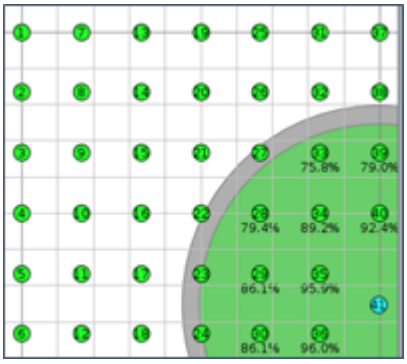


Figure 12: Grid topology with 40 client nodes

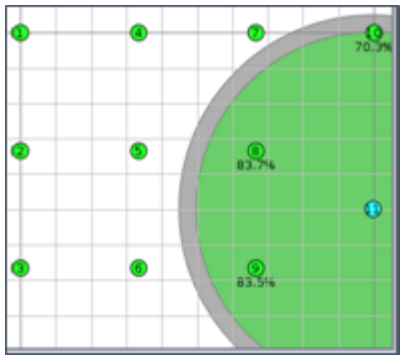


Figure 13: Grid topology with 10 client nodes

Finally, to study a topology which requires a larger number of hops, the team simulated a linear layout of 10 client nodes (see Figure 14). The area in this scenario is different than in previous simulation such that the nodes can only talk to their adjacent neighbors. Thus the configuration places the nodes evenly spaced 40 meters apart with the border router node in the middle.

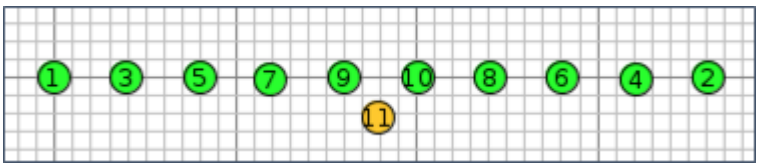


Figure 14: Linear topology with 10 client nodes

3.1.3. Application Level

To run the Cooja simulated configuration tests with an application layer, the team used a sample UDP Contiki application called “Hello World!”. This simple application sends a “hello” message at an interval the team defines. The team used the simulator to load each client node with the application and such that each client node send a “hello” message to the border router at the defined application layer message sending interval for the duration of the 15-minute test. The interval is defined in the udp-client.c file as stated in Table 1. The border router used udp-server.c and all the sensor nodes used udp-client.c.

3.2 Initial Simulations to Check Trickle Code

Before the team conducted the main set of simulation runs, it was important to perform a sanity check to make sure that the code for the four Trickle algorithms was correct and that the process of data collection within Cooja was accurate. Consequently, the team ran preliminary tests with nearly the same configurations as Ali’s master’s thesis [1]. The only difference between our simulations and Ali’s simulations was the randomly generated locations of the nodes. All of these preliminary simulations yielded performance results that followed similar trends to that of the master’s thesis. This provided confidence for the validity of the simulations studied in this research.

The team modified the RPL source code in Contiki to write the four Trickle algorithms as described in the background chapter. The changes include modifications to the rpl-timers.c file

(see Appendix A). Each time the team investigated the performance of one of the four Trickle variants, the simulation setup involved replacing this file with the appropriate version of the Trickle algorithm.

3.3 Data Collection

For each set of the simulated runs, the team collected a series of statistics on the performance of the algorithm at each of the nodes (see Table 3). Each of the sets of simulations used the same random seed for consistency.

Table 3: Description of Performance Metrics

Name of Statistic	**Description**
Network Convergence Time	Set up time for all client nodes to join the network
Total Packets Sent	Total number of application and routing packets sent throughout the test
Delivery Ratio	Percentage of application messages successfully delivered to the BR during the test
Radio on Time	Percentage of radio on time, averaged over all of the nodes in the simulation

Network Latency	Average latency of all application packets sent over the duration of the simulation

The network convergence time of the network is the amount of time needed for all of the client nodes to initially join the network, and then begin sending application packets. This means identifying itself to the server node and its client neighbor nodes. The server then acknowledges and adds the node to the DAG, which it maintains. The team derived this statistic using a script to determine when all nodes had printed to the simulator output that they joined the DAG.

The total number of packets is the combined number of application layer messages sent from the client nodes to the BR and the RPL routing packets. Since the number of application level packets sent in a test is nearly identical for each simulation run, for comparative analysis it was not necessary to remove them from the overall total. This would simply shift the RPL packet routing overhead down by a constant value. Cooja tracks all of these statistics.

The packet delivery ratio is the number of successful “hello” application messages sent from each of the client nodes to the sink. Cooja calculates this metric by computing the average over each of the nodes for the entire test; the program totaled the number of received outputs and divided it by the number of send outputs to obtain this average. Cooja found this number for each of the nodes and then determined the average for all of the delivered percentages per client node.

Radio-on time is the percentage of the simulated 15-minute test in which the node’s radio is on, for either sending or receiving. The team took the average of this value over all of the client nodes with the value of the border router removed. This value is tracked by the Powertrace

tool [5]. Powertrace is a built-in tool of the Cooja simulator, which tracks the radio on percentage.

Latency is the amount of time that it takes for a packet to travel from a client node to the server (border router). Average latency, the final statistic collected from each test, computes the average latency over each of the sensor nodes because latency will be very small near the border router and longer when there are more network hops between source and destination. The team determined this metric by running a script over the Cooja text output file.

3.4 Visualizing the Data

To visualize the performance results, the team created a series of graphs using Microsoft Excel from each of the test stages to show the changes in the metrics and to make comparisons between the algorithms over variations in key parameters. The team created graphs to analyze and compare each of the performance metrics the team collected data on, as discussed in the previous section. The following results chapter contains the most interesting results and graphs. The full set of performance graphs can be found in the Appendix A.

4. RESULTS

This chapter discusses and analyzes results for the four Trickle timer algorithms simulating RPL with the Cooja simulator. The chapter utilizes the performance metrics detailed in the prior chapter to provide graphs that demonstrate the performance differences in the four simulated Trickle algorithms. The first two section introduces results from the variation of the DIO Minimum interval and DIO Doubling parameters. This is then followed by the results from the different network topologies and densities comparison against the four Trickle algorithm variants operating at optimal parameter settings.

4.1 DIO Minimum Interval

The first important RPL parameter is the DIO Minimum interval. This is the minimum possible interval value in the Trickle algorithms (namely, I_{min} in Trickle pseudo code). The first set of comparison simulations involved varying the value of DIO Minimum value from six to sixteen. This value determines the minimum interval length using $2^{\mathbf{x}}$ milliseconds where **x** is the chosen DIO Minimum value. As DIO Minimum values between two and five provided weak performance results, this analysis does not include these specific simulated results.

Figures 15-18 provide graphs that compare performance metric results for the four Trickle variants (Original Trickle, Opt Trickle, E Trickle and ME Trickle) where the x-axis varies the DIO Minimum value from six to sixteen. All of the other RPL and Trickle parameters were set to default values as listed in the methodology chapter in Table 1. The network topology for all of these tests is the 80 node random configuration.

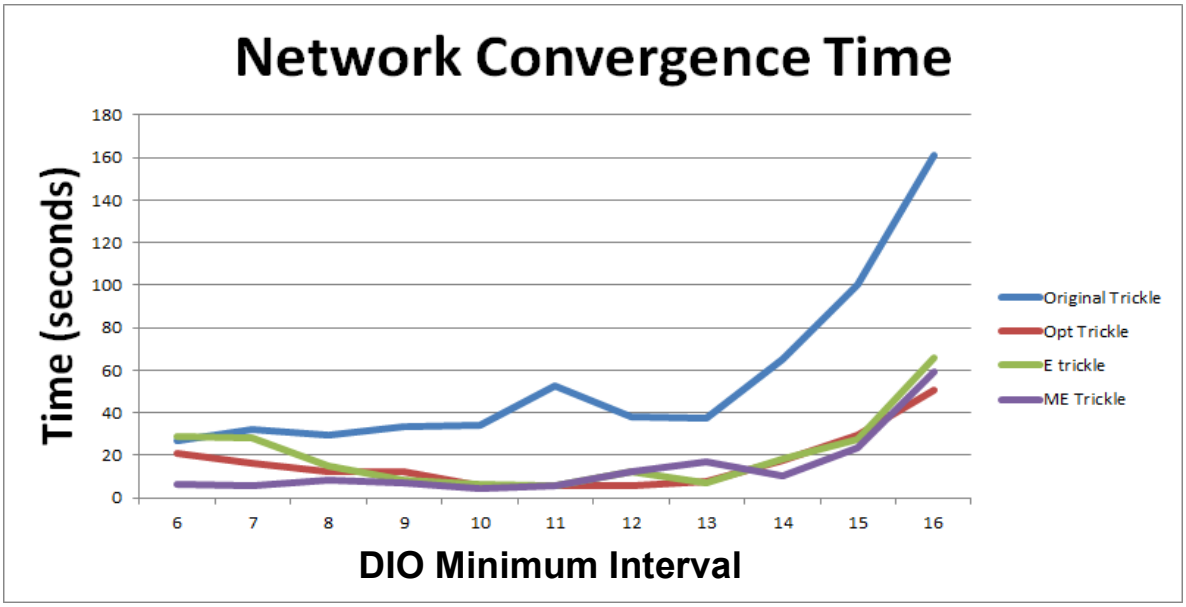


Figure 15: Network Convergence Time versus DIO Minimum Interval

Figure 15 demonstrates that the three new algorithms yield better convergence times than the original Trickle algorithm. The best DIO values are in the range from nine to twelve. For all four Trickle variants the convergence times grow quickly after values above 14. This is because as the value of the minimum DIO interval increases Trickle waits longer to resend if a message fails. Network convergence time does not affect Trickle performance significantly after the first few minutes, but it does have a negative effect on the radio-on power usage.

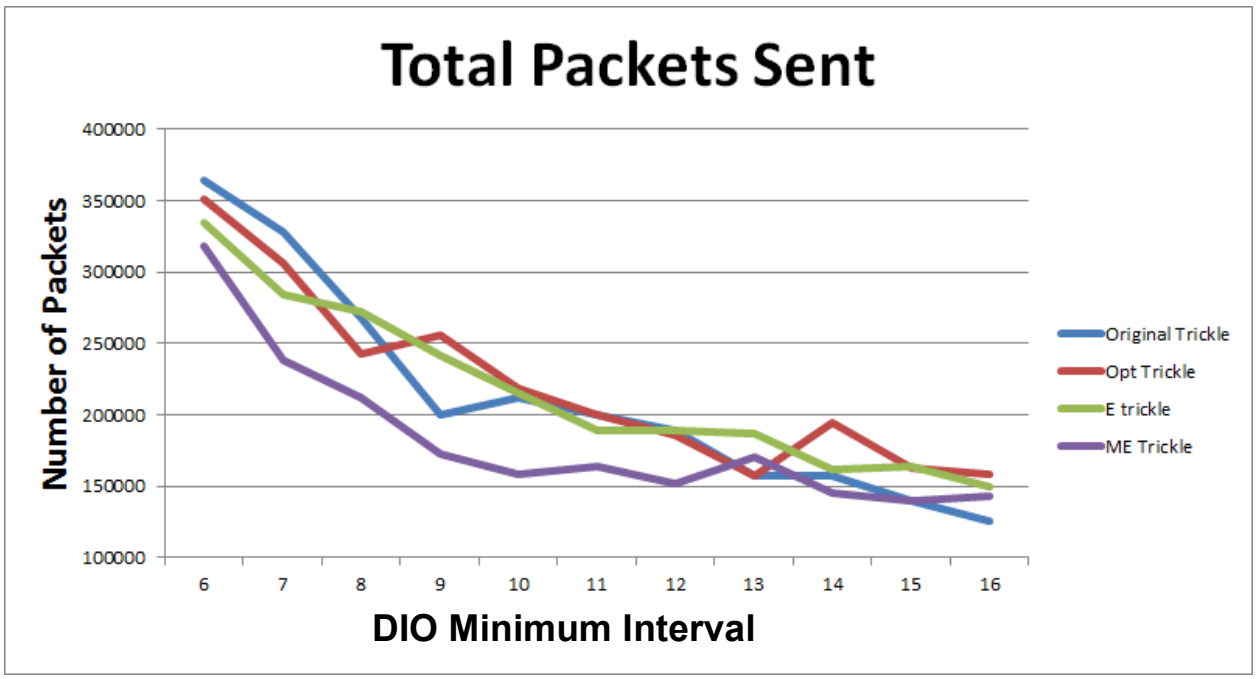


Figure 16: Total packets sent versus DIO Minimum Interval

Figure 16 compares the total number of packets sent by the four Trickle variants over the 15-minute simulation as the DIO Minimum value varies. The packet totals include both the application level packets and the routing packets. As expected, a shorter DIO Minimum interval produces a much greater number of packets being sent than the larger interval. When an RPL sensor node sends a larger number of packets, there are also larger send queues at each of the sensor nodes causing many more packet collisions and packet drops due to transmission interference. This causes many nodes to issue multiple resends before yielding a successful reception. This has a negative effect on the number of packets sent, as well as many of the other performance metrics measured.

Over most of the DIO minimum values, ME-Trickle continues to maintain the best performance in terms of the least number of packets sent. This is due to the maximization of the interval after a single consistent interval.

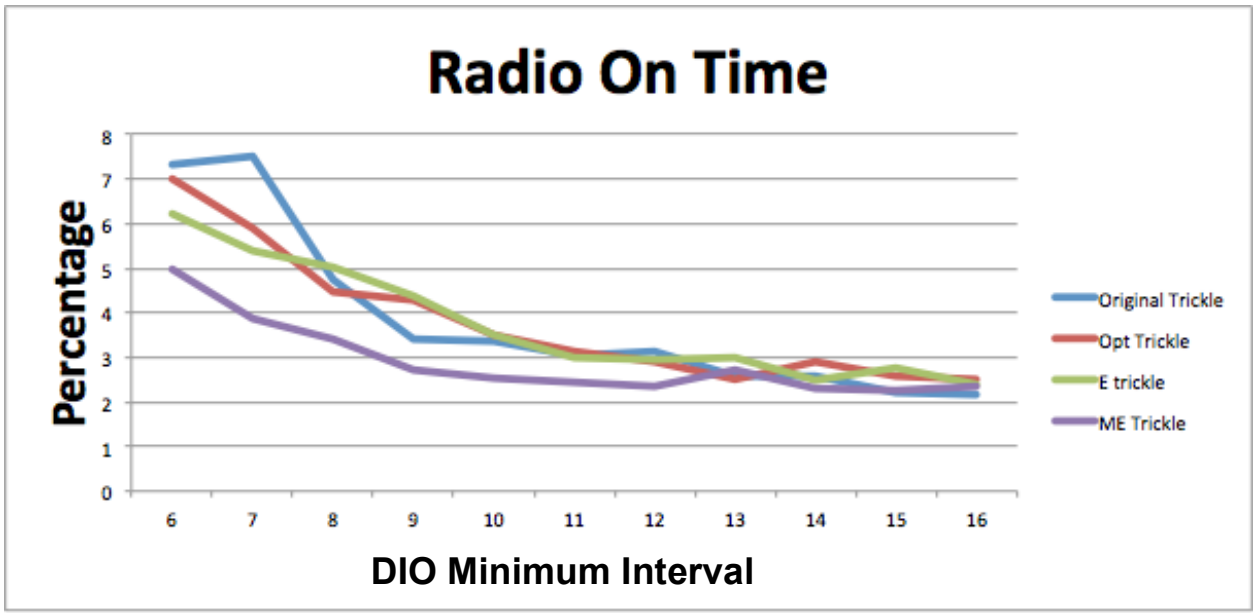


Figure 17: Radio on time versus DIO Minimum Interval

As shown in Figure 17, the average percentage of radio on time correlates somewhat to the number of packets sent. Due to the large number of packets sent for the lower DIO Minimum intervals, there is a large increase in the percentage of radio on time as one would expect. While the radio on time percentage is still rather low for a DIO Minimum interval of six, it is large in comparison, at nearly twice the value, to the larger DIO Minimum interval values. As with the previous performance metric, ME-Trickle consistently performs better than the other algorithms for DIO Minimum values 6 through 12. The other three algorithms perform very similarly in this metric for DIO Minimum values 13 and higher.

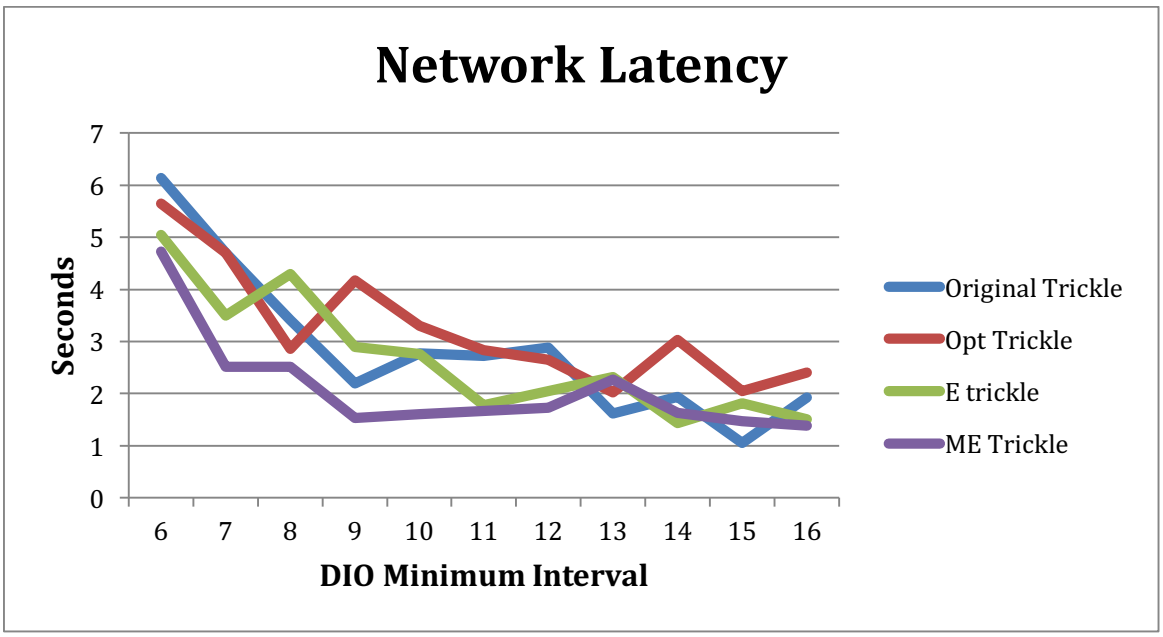


Figure 18: Network latency versus DIO Minimum Interval

Figure 18 gives the network latency, averaged over all of the 80 client nodes for all of the packets being sent during an entire simulation test, versus the variation of the DIO Minimum interval. This is another performance metric which has a similar trend to the number of packets being sent through the network. The value for latency shown is the average for the 15-minute test for all packets, both routing and application. As one would expect, when there are a greater number of packets in the network there are larger latency values. Again, this is likely due to larger numbers of collisions, packet resends, and larger send queues at each node in the WSN. The ME-Trickle algorithm again performs the best across most of the lower to mid-range interval values due to its lower number of routing packets. It also performs similarly to the other algorithms with higher interval values.

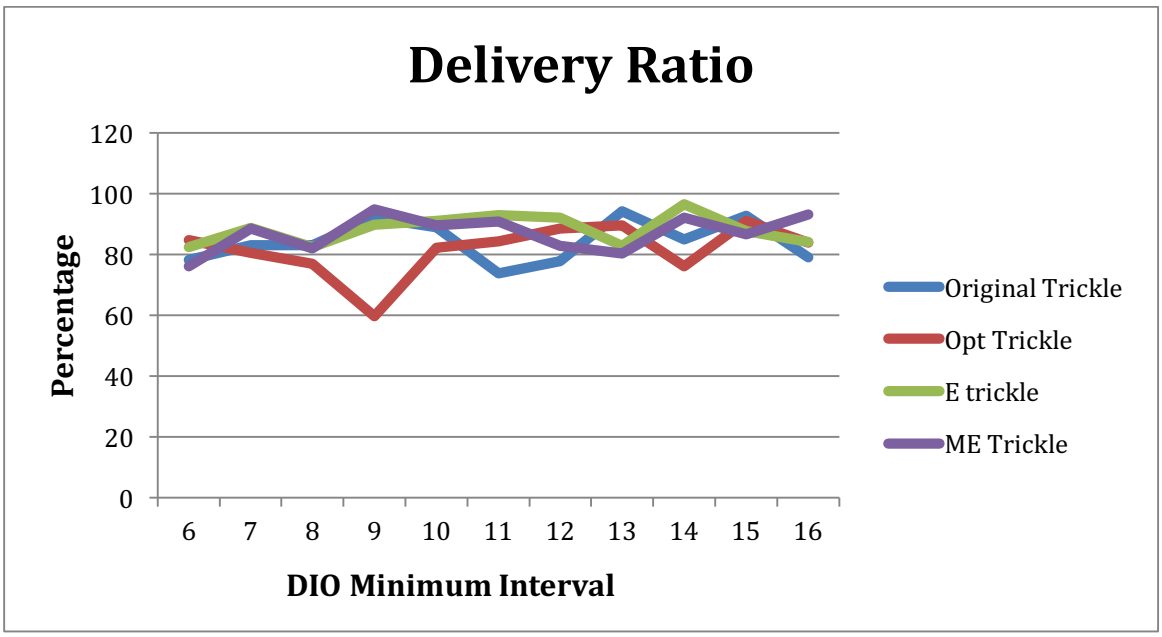


Figure 19: Delivery Ratio versus DIO Minimum Interval

Figure 19 lays out the average delivery ratio of application packets from the nodes to the border router from each of the nodes over the 15-minute simulation. This shows that the DIO Minimum interval does not seem to have a large or clear effect on the delivery of application packets for any of the algorithms. They are all relatively similar in performance with variation in the data mostly due to the randomness of the simulator.

The above graphs indicate that the ME-Trickle algorithm performs the best for lower DIO Minimum interval values. But it does not have a significant increase in performance over the other algorithms values closer to the DIO Minimum default value of 12. ME-Trickle at this point does not represent a great performance enhancement for this relatively dense network.

4.2 DIO Interval Doubling

The next parameter that the team considered was the DIO interval doubling which is the maximum number of times an algorithm doubles the DIO interval before reaching I_{max} as previously defined. We ran this test with values from six to sixteen. Minimum values between

two and five provided very weak performance results. Thus, this analysis does not include these specific simulated results. There are not results from 13 to 16 for the ME Trickle algorithm because the interval became too large for the simulator to manage, causing the time for the simulation to complete to become very excessive. Thus we decided not to include these simulations for ME Trickle.

The following figures provide graphs that compare performance metric results for the four Trickle variants (Original Trickle, Opt Trickle, E Trickle and ME Trickle) where the x-axis varies the DIO Interval Doubling values from six to sixteen. All of the other RPL and Trickle parameters were set to the default values as described above in Table 1 of Section 3.1.1. The network topology remained as the random 80 node topology used in section 4.1.

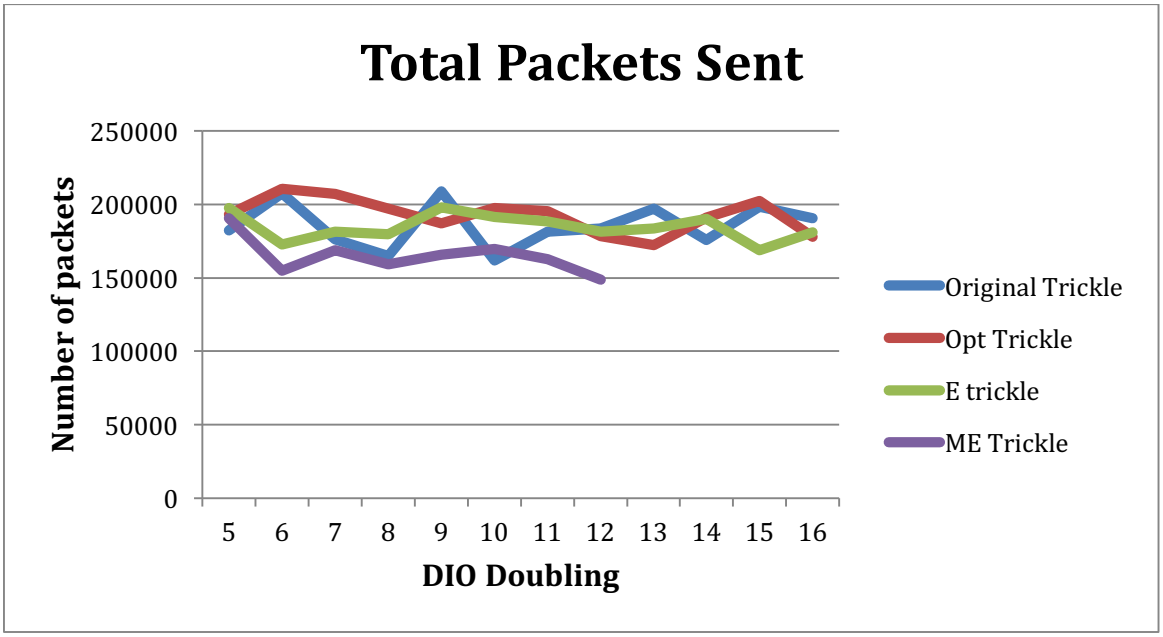


Figure 20: Total packets sent versus DIO Doubling

Figure 20 exhibits the total number of packets sent through the network. The DIO doubling parameter did not have as clear of an effect on the four different algorithms as the variation of the DIO Minimum interval did. The slope of the number of packets that were sent is

similar for all the algorithms. There is randomness in the data which could be the cause of the lack of a clear winning algorithm. In addition, the number of packets remained nearly the same despite the changing of the values. We suspect that this is because the network is not reaching the maximum interval value due to the high level of dropped packets causing inconsistencies in the network. The ME Trickle algorithm still sent the fewest number of packets in almost all cases.

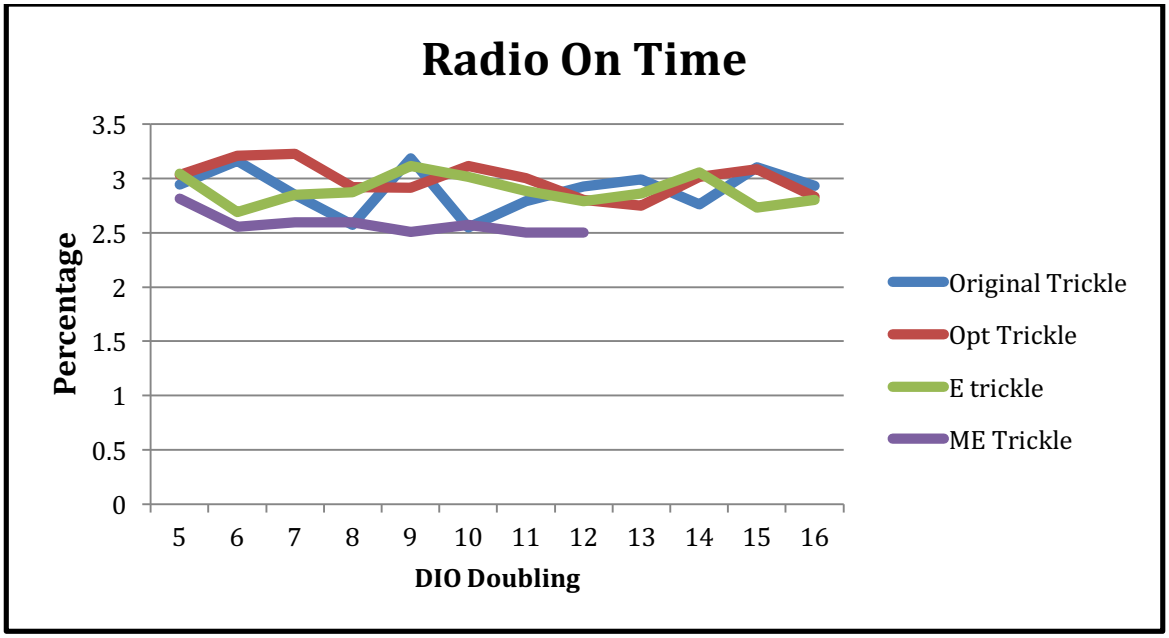


Figure 21: Radio on time versus DIO Doubling

Figure 21 produces the performance of the four algorithms with respect to the radio on time percentage as DIO doubling varies. As with the previous test, ME Trickle still performs the best because it sends the fewest number of packet, although the parameter does not seem to have a large effect on this performance metric for ME Trickle. The performance improvement is small compared to the other algorithms. All of the algorithms have a very good perfromance with less than 4% of the radio on time. This is a much better result than was produced in the majority of

the tests involving the DIO minimum variation because it uses 12, the default value of the DIO minimum, which yield similar performance across the four Trickle variants.

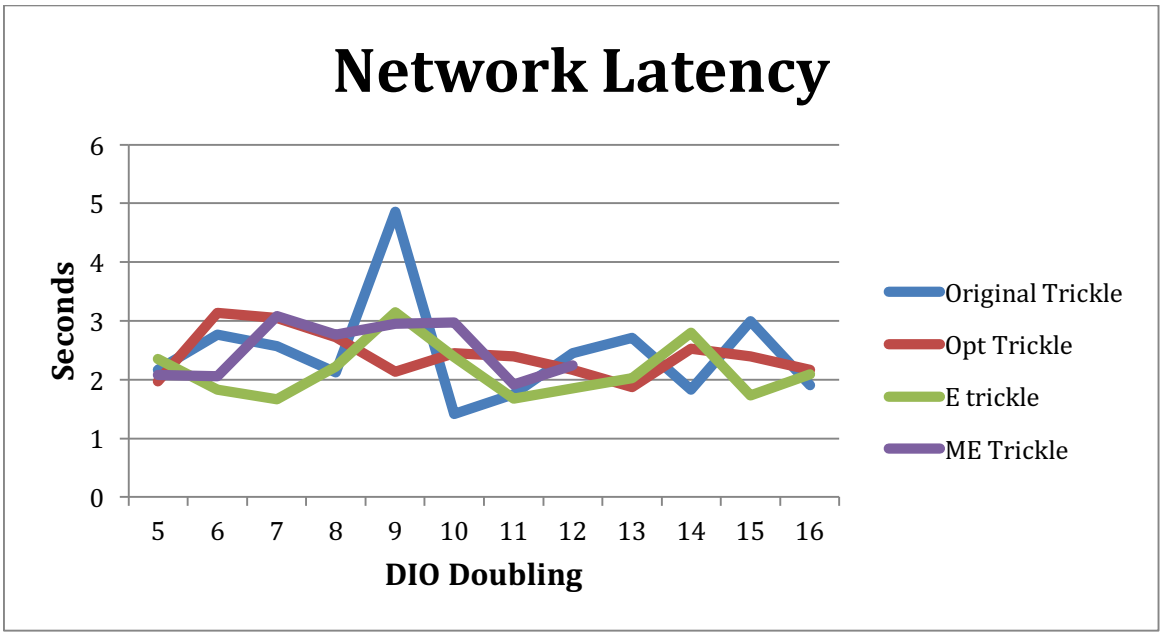


Figure 22: Network Latency versus DIO Doubling

The first thing to notice about Figure 22 is that the network latency for the original Trickle algorithm at a DIO doubling value of 9 is an anomaly. It can be seen in the other graphs for DIO doubling but it can be seen most clearly here. The team believes that this point is inordinately high compared to the other values. Figure 22 demonstrates that there is not a clear effect of DIO doubling variation on the latency of packets through the network. The randomness of the network and the random number generator within the simulator are likely contributors to the oscillation and randomness of the result.

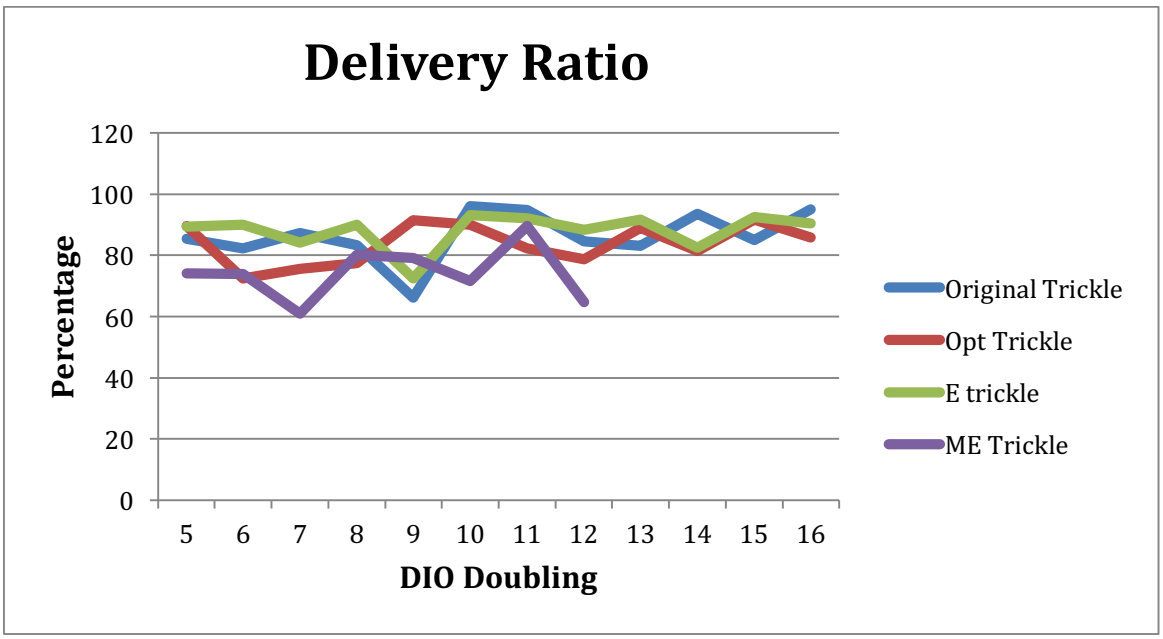


Figure 23: Delivery ratio versus DIO Doubling

Figure 23 reveals the average delivery ratio of application packets from the nodes to the border router from each of the nodes over the 15-minute simulation. This graph illustrates that the DIO doubling does not have a large or clear effect of the performance of the algorithms. ME-Trickle seems to have the worst performance overall here. This follows the trend that ME-Trickle does not yield high delivery ratios for relatively dense networks.

4.3 Other Parameter Variation Tests

In addition to testing the effects of varying the value of the DIO Minimum interval and the DIO doubling against each of the algorithms for an 80 node random configuration, the team also tested variation of two other parameters. These parameters were the RDC channel check rate and the send interval of the application messages.

The RDC channel check rate is the number of times that the radio turns on each second to test the medium to see if there is a message for it to receive. In our preliminary simulations, this parameter did not have a large or very clear effect on any of the four Trickle algorithms so the team has decided not to include these graphs in the results chapter for the sake of brevity.

Additionally, the team tested the variation the send interval of the application level packets. These results are not analyzed here for two reasons. The first is that this parameter is actually not part of the network layer but the application layer and is out of the control of RPL or Trickle. Thus is it not fair to show the results as a performance factor for the algorithms. Secondly, this set of results are completely uninteresting, all of algorithms simply follow the same, expected curve of performance for all metrics.

All of the graphs for these two set of simulation tests can be found in the Appendix A.

4.4 80 Node Simulations with Varied Configurations

The team conducted the next set of simulations to study the performance of the Trickle variants across different sensor node configurations and densities. All the simulations reviewed up to this point used the random configuration, so we decided to simulate the 80 nodes in a grid like configuration (see Figure 11). For this set of tests, we decided to choose the optimal values for the parameters for each specific Trickle variant based on the prior random configuration simulation results. Table 4 presents the optimal performance values that we chose.

Table 4: Optimal Parameter Values

Trickle Algorithm	DIO Minimum Interval	DIO Doubling	RDC Check Rate	Send Interval
Original	14	10	64	4
Opt	15	13	64	4

E-Trickle	11	11	2	4
ME-Trickle	11	10	16	4

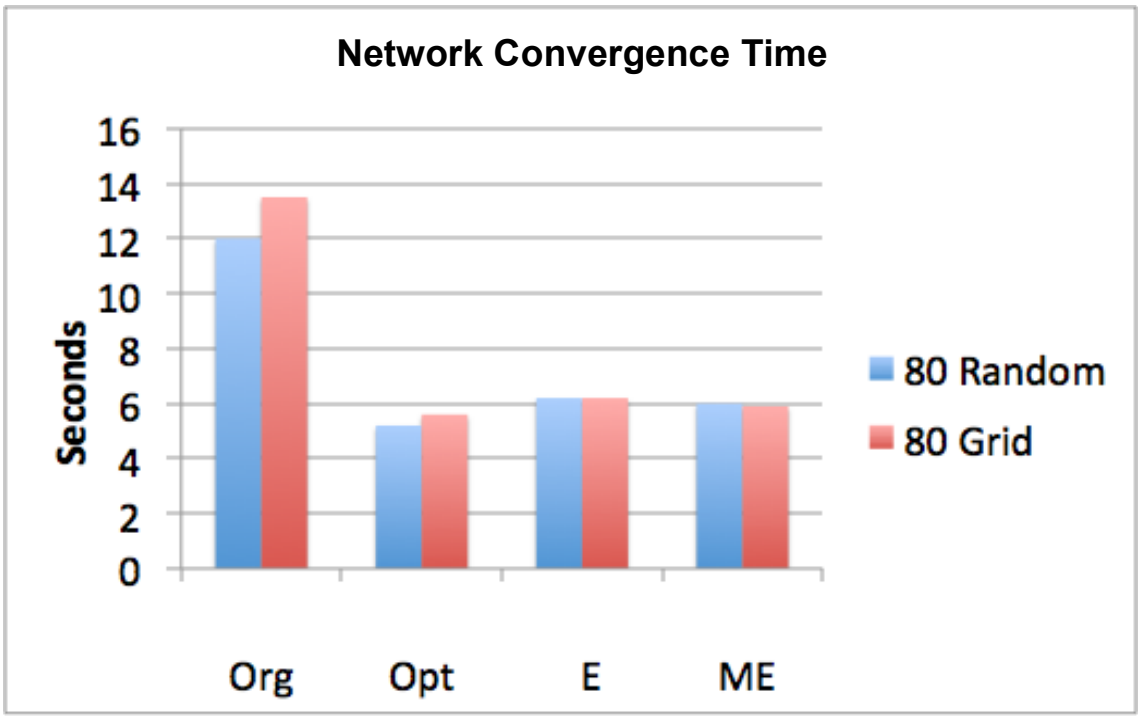


Figure 24: Network convergence time for 80 node configurations

Figure 24 illustrates the differences in the convergence time of the network for both 80 node configurations over the optimized four Trickle variants. The x-axis represents the four variations of the algorithms. The original Trickle algorithm is shown here as "Org." As shown previously, the original Trickle algorithm performs very poorly compared to the other three algorithms using this performance metric. The other three all perform roughly the same with Opt being the best choice for both configurations. Overall, the node configuration does not have a significant effect on convergence time in this relatively dense situation.

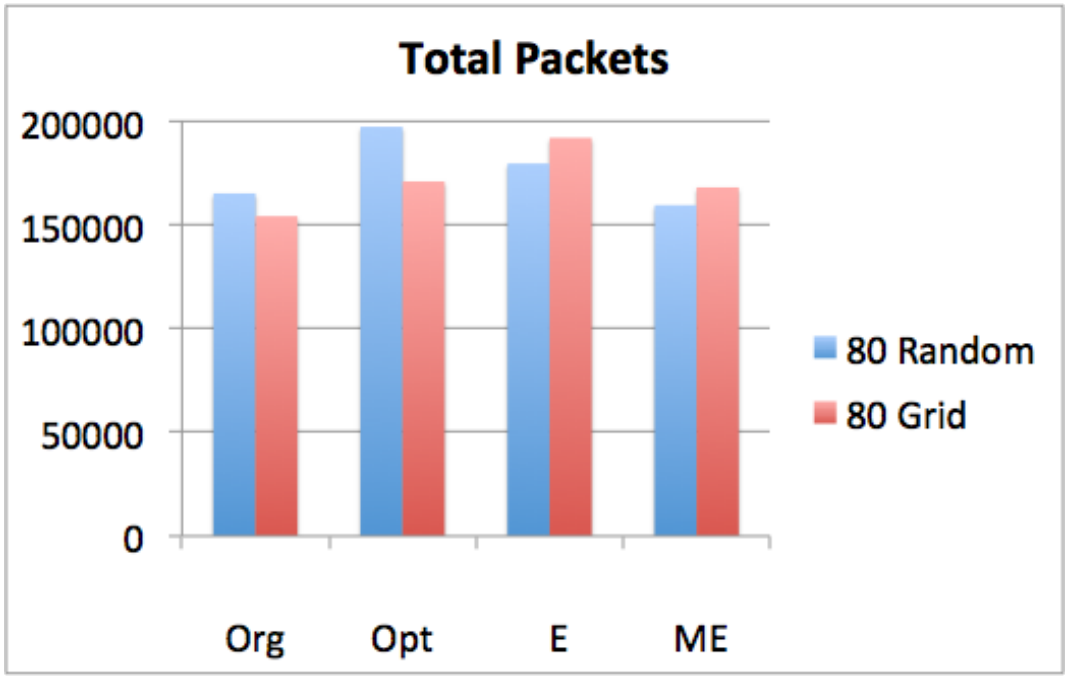


Figure 25: Total packets sent for 80 node configurations

Figure 25 displays the total number of packets sent through both of the networks over the 15-minute testing period. The original Trickle algorithm performs the best for the grid layout, while the ME-Trickle algorithm performs the best for the random layout. Interestingly, the number of packets is larger in the random layout than in the grid layout for the original and Opt Trickle algorithms, while the opposite is true for the E-Trickle and ME-Trickle algorithms.

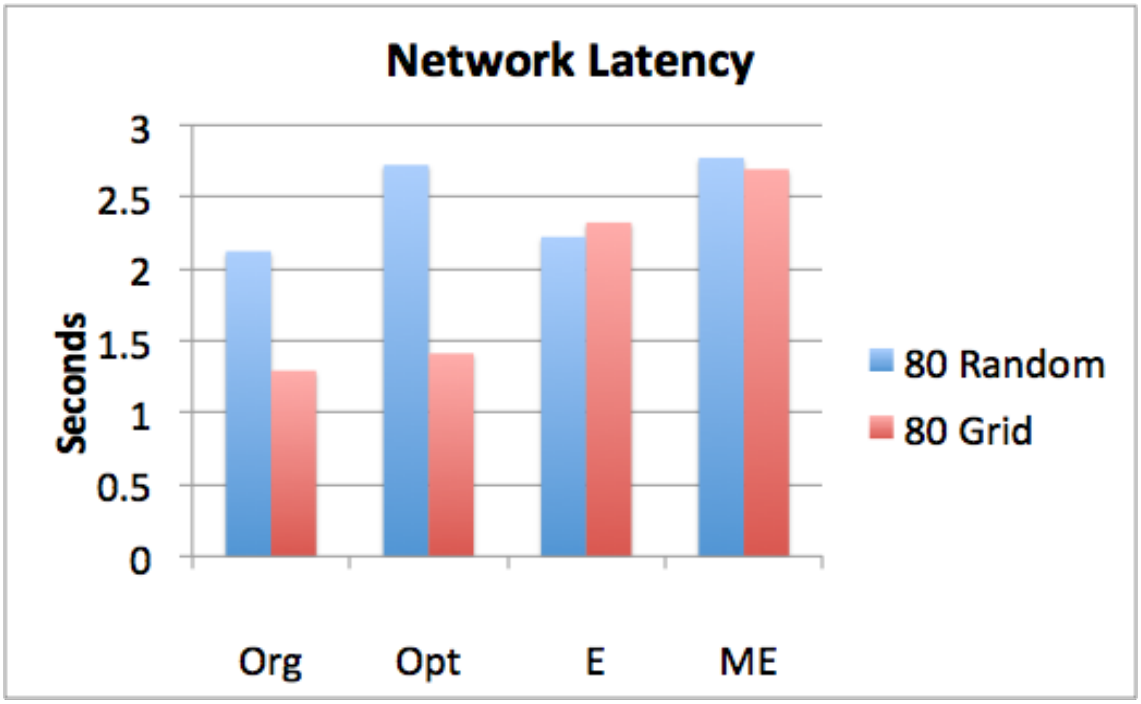


Figure 26: Network latency for 80 node configurations

Figure 26 graphs the average network latency from all of the nodes for the two configurations. The result demonstrate that the grid layout led to better or even performance in most cases. The more consistent layout allowed for fewer packet hops from the farthest nodes to the border router. In the case of the original and opt Trickle algorithms, the latency was nearly twice the time in the random layout of the grid layout. The optimal configuration of parameters allowed for better performance from original, opt, and E-Trickle than these algorithms produced in the two previous sections. ME-Trickle continues to have a worse performance in networks with a higher density.

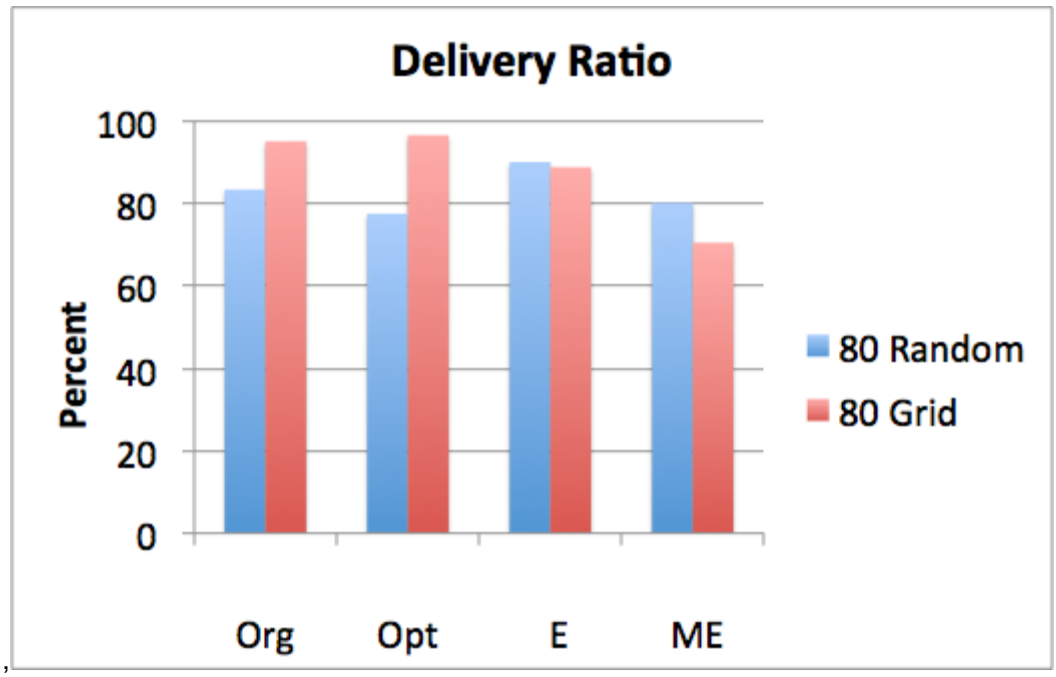


Figure 27: Delivery Ratio for 80 node configurations

Figure 27 expresses the average deliver ratio for all of the nodes. This data exhibits a similar pattern to the latency. Namely, the original and opt Trickle algorithms produce the best results in the grid layout. The ME-Trickle algorithm performs much worse in the grid layout, nearly 25% lower than original and opt algorithms. It performs similarly to the other algorithms but overall it continues to not be as good for larger densities.

The overall performance trends also continue with the original and E-Trickle algorithms having the best overall performance for both the grid and random layouts. Opt Trickle performs the best for the grid layout but the worst for the random layout. Additionally, opt Trickle performs the best in most of the other metrics for the grid layout. The takeaway that is emerging is that the performance of the four Trickle variants is affected by the specific topology of the WSN.

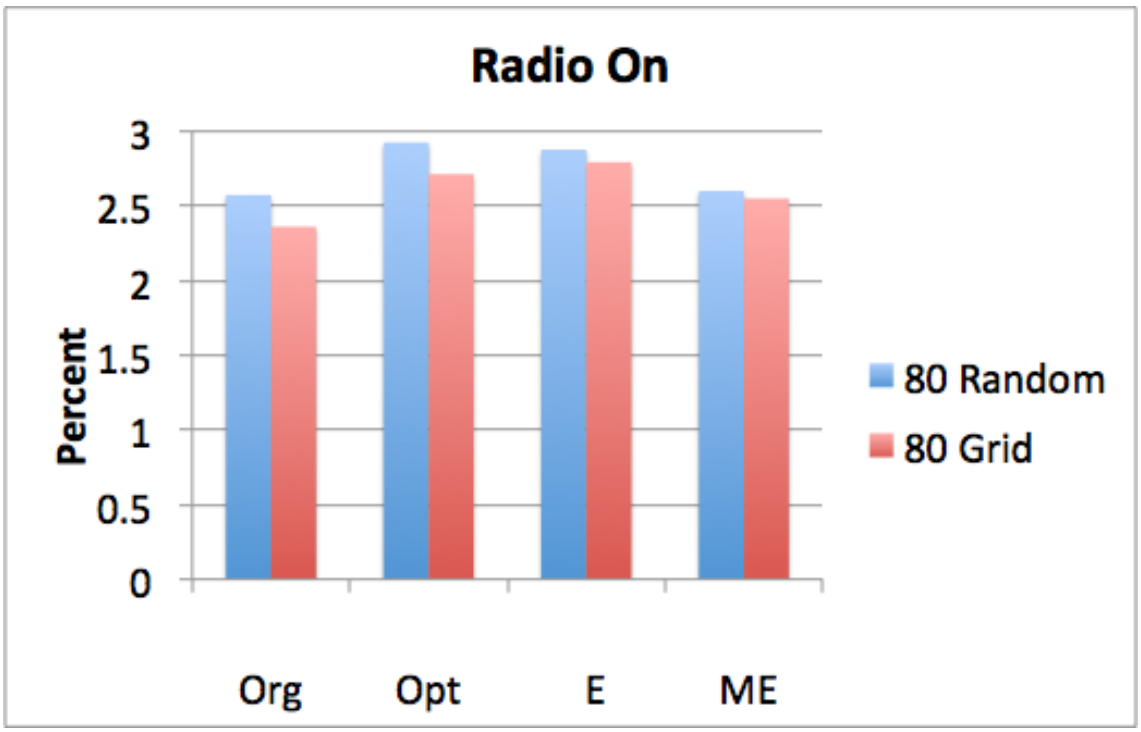


Figure 28: Radio on time for 80 node configurations

Finally, Figure 28 demonstrates the average radio on time over all of the client nodes when using optimal parameter settings. Logically, this metric is related to the number of packets that is sent over the course of the test. Thus, it is not surprising that the original Trickle and ME-Trickle algorithms have the best performance for this metric. The difference in performance between the different algorithms is very small at less than .5 % in most cases.

4.5 40 Node Configurations

The next variation the team investigated was the algorithms' performance against different network densities of the client nodes. The next set of simulation tests were the 40 node configurations as described in the methodology chapter. Since the area of the testing environment is the same and the number of nodes is cut in half, the density of the network is much less. For these simulation tests, all of the parameters are the default values as they were

defined in Table 1 in section 3.1.1. of the methodology chapter except for the values of DIO minimum, DIO doubling, and RDC check rate which are the same optimal values as defined in Table 4. In testing this sparser network, we also considered varying the value of the redundancy counter *k* which is defined in the background chapter and in Table 1 in the methodology chapter. We tried multiple lower values for this counter. However, these tests did not produce any conclusive results so we do not include them in this report and provide results for only the k default value of 10.

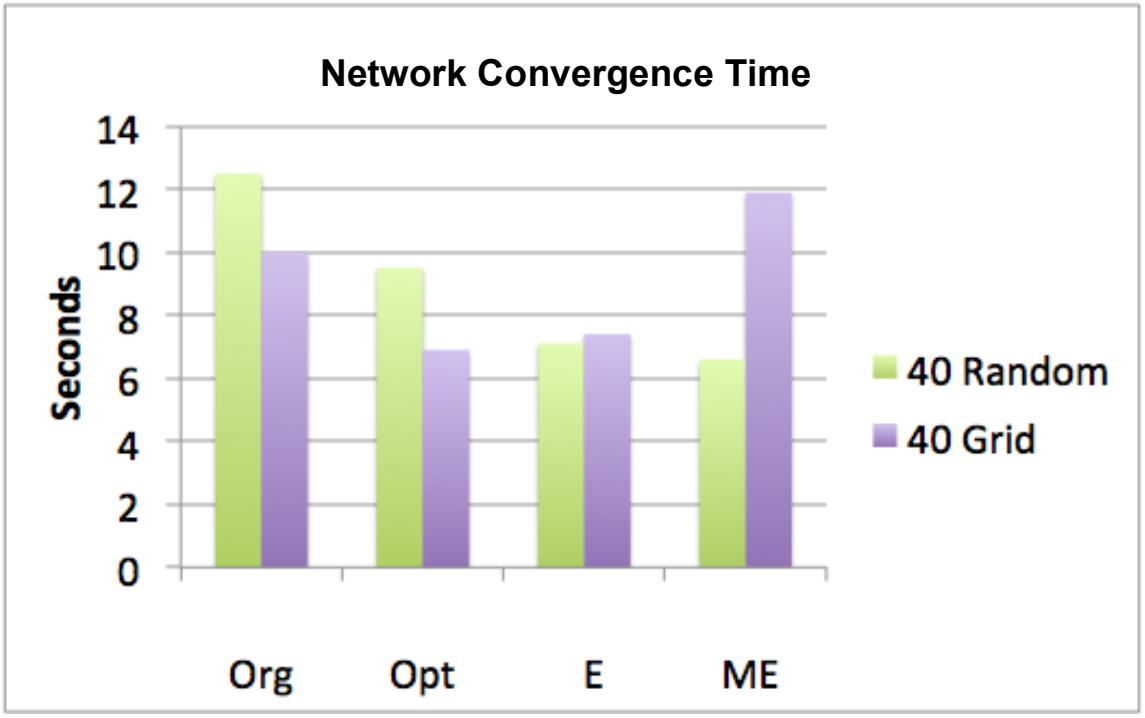


Figure 29: Network convergence time for 40 node configurations

Figure 29 illustrates the convergence time of the four different algorithms for the two different 40 node network configurations. First, we believe that the convergence time for ME-Trickle for the 40 client node in a grid layout data point is an unexplained anomaly in the data based on the previous results and other data points we gathered for this graph. Similar to the 80 node tests, the original Trickle algorithm performs the worst for both layouts if the anomaly is

ignored. Although the difference is much smaller with a lower network density. The other three algorithms all perform similarly, but we believe that the ME-Trickle should have the best performance in this metric as it has from previous simulations.

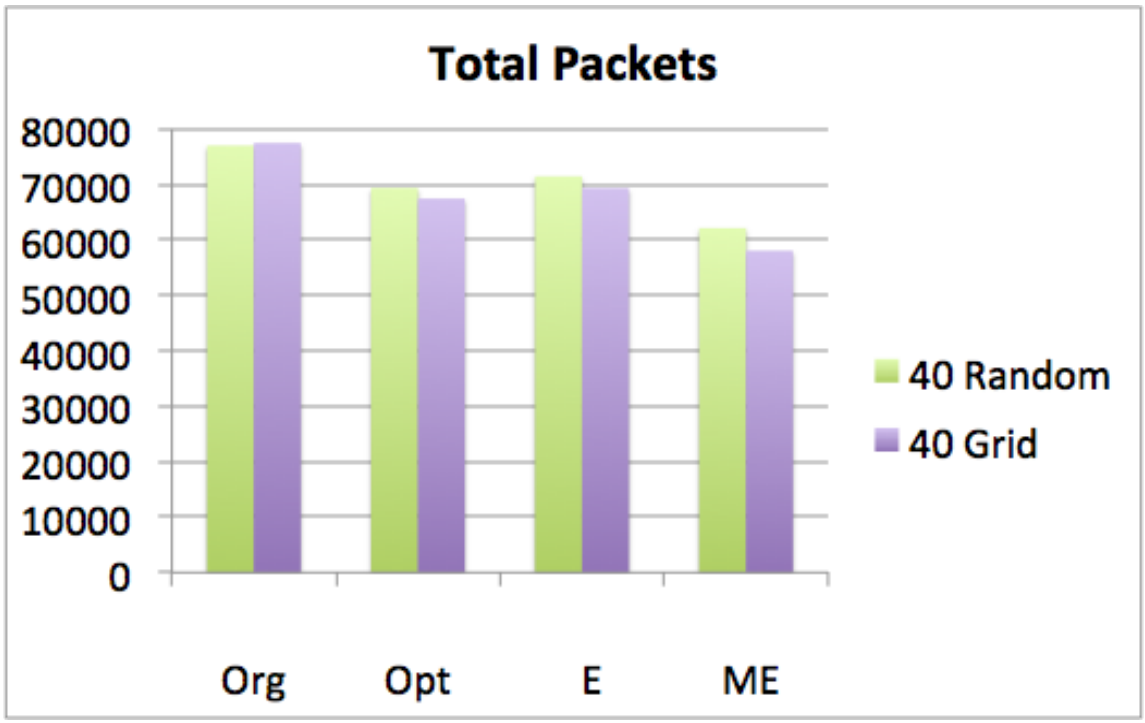


Figure 30: Total packets sent for 40 client node configurations

Figure 30 exhibits the total number of packets sent through the network over the 15-minute test. Note, there are far fewer packets sent when the number of sensor nodes is half. Having fewer nodes results in fewer collisions and fewer lost packets, causing the total number of packets sent to be much less. This data leads to a different trend than the 80 node configuration performance results. Previously, the original Trickle algorithm performed very well for this metric, here it is clearly the worst by nearly 10000 packets. ME Trickle does the

best for this metric. The opt Trickle algorithm has also done much better for the 40 node configuration than it did with the 80 node configurations.

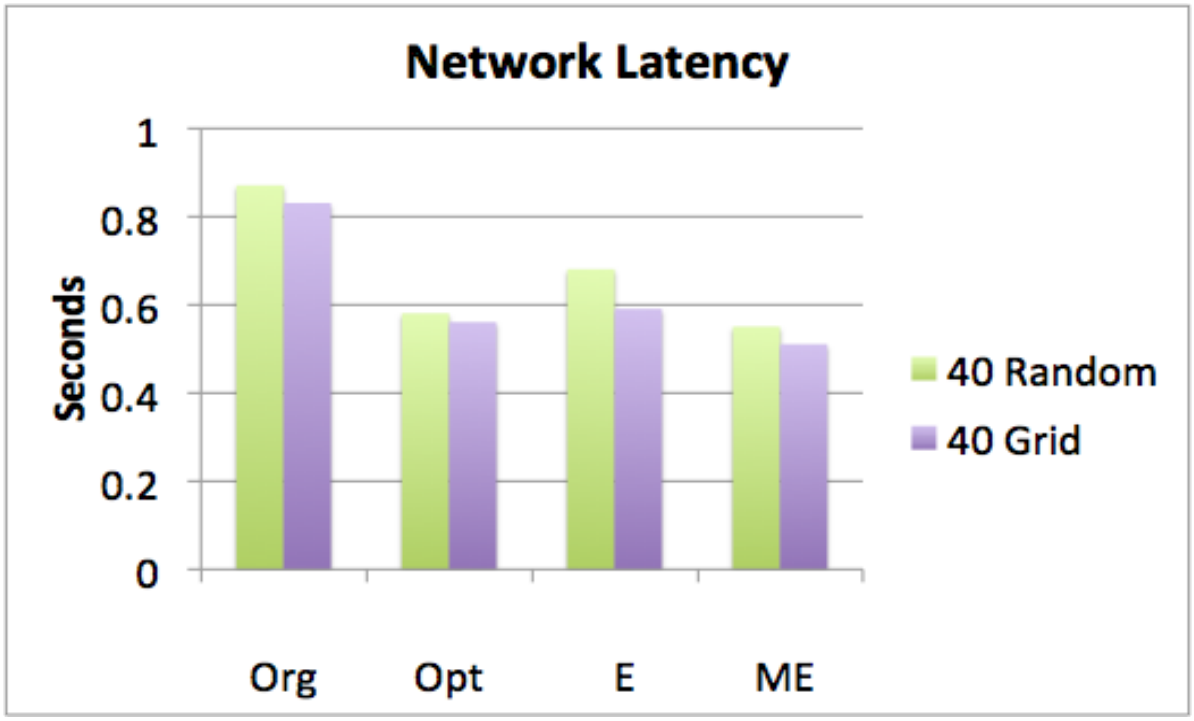


Figure 31: Network latency for 40 client node configurations

Figure 31 displays the average network latency from all of the nodes. Note, the latency for each of the algorithms is much lower in this network density versus the 80 node configuration. These performance trends track well the number of total packets performance. Original Trickle does worse than the other algorithms comparatively. The scale here is important as the difference is small except between original and ME-Trickle which is about 300 milliseconds. Additionally, ME-Trickle continues to have the best overall performance for both the random and grid layouts for networks with lower densities versus its poorer performance with 80 nodes.

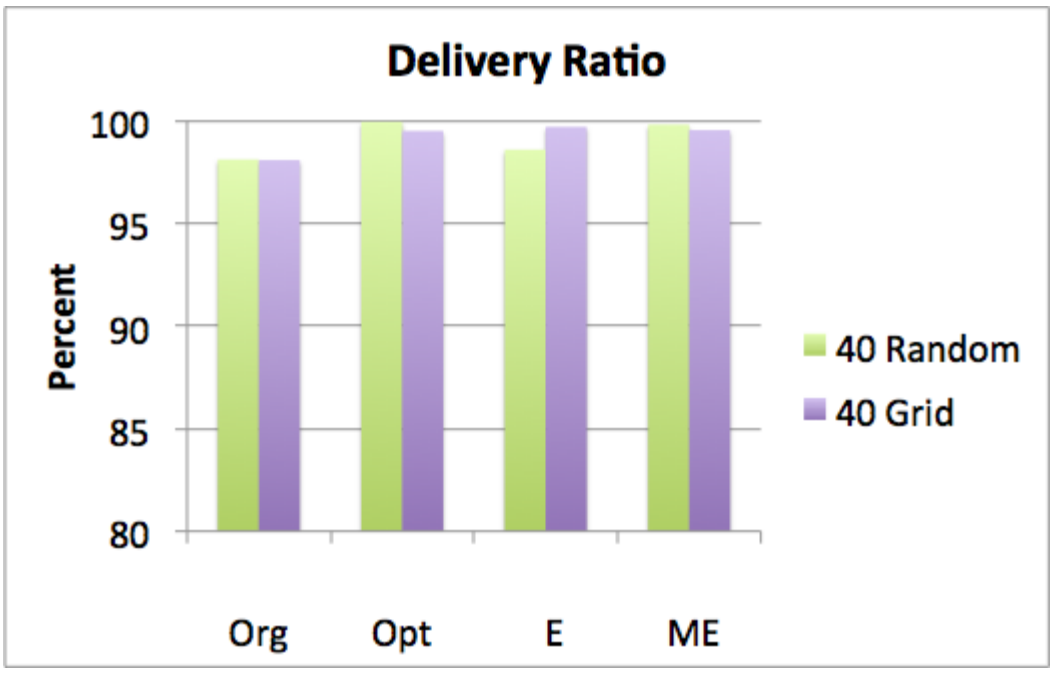


Figure 32: Delivery Ratio for 40 client node configurations

Figure 32 indicates the delivery ratio averaged over all of the 40 client nodes. Note the difference in the y-axis scale in Figure 32 and that most of the delivery ratios are higher than the results from the 80 node configuration, except the original Trickle results which are nearly the same. This is most likely due to there being less packets sent and fewer packet hops to the destination, resulting in there being less of a change of a dropped packet. While the original Trickle algorithm continues to perform the worst for both of the network configurations, the other three algorithms all perform very well for both layouts.

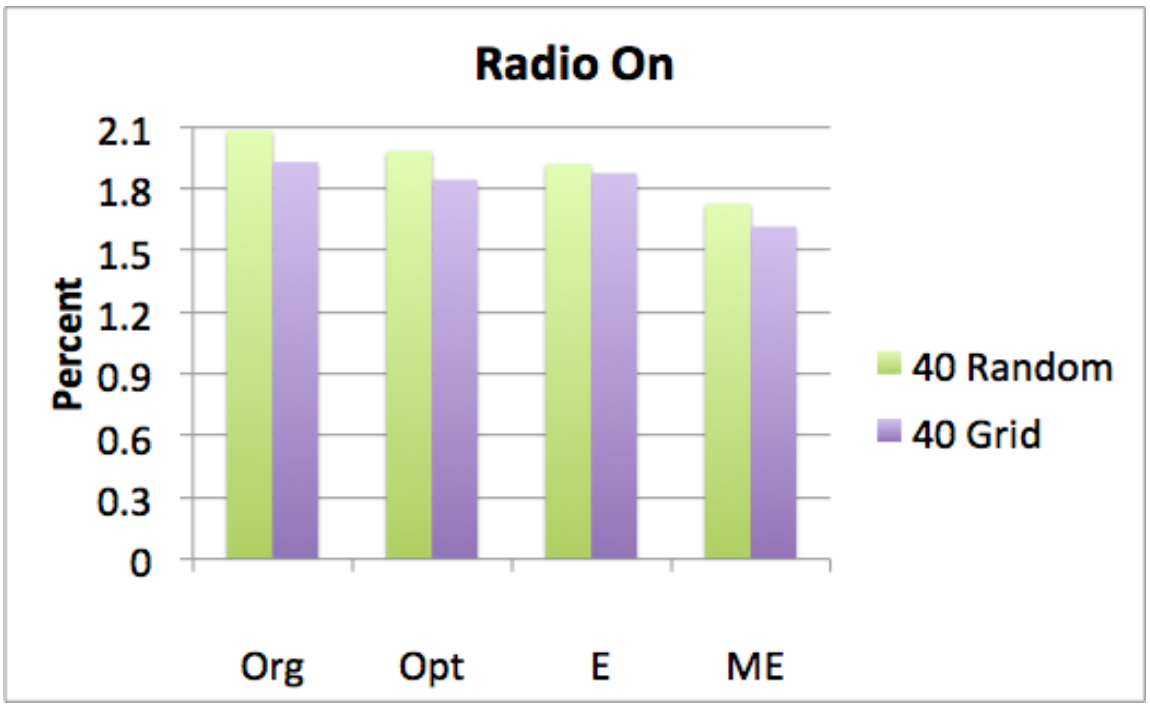


Figure 33: Radio on time for 40 client node configurations

Finally, Figure 33 displays the radio on time percentage averaged over all of the 40 sensor nodes. This metric continues to relate the number of packets sent through the network. Thus, the performance trends from the past figures continue to show that the original Trickle algorithm has the worst performance for both of the 40 node configurations. ME-Trickle has the best performance for this metric, as it still sends the fewest number of packets. The differences between all four variants is very small at less than .4%.

The simulation results from this section indicate the ME-Trickle algorithm performs the best overall for this network density. It has the best performance on all of the performance metric for both the random and grid layout. Original Trickle yields the worst performance for this network density.

4.6 10 Node Configurations

The final set of simulation testing that the team performed was on 10 node configurations as described in the methodology chapter. The team simulated these configurations to study the performance of the four algorithms against a sparse network. We decided to simulate three different layouts as described above: the line, grid, and random configurations (see Figures 10, 13 and 14 respectively). Since the area of the simulated environment is the same and the number of nodes is much smaller the density of the network is much lower. For these tests, all of the parameters were the default values as defined in Table 1 except for the values of DIO minimum, DIO doubling, and RDC check rate which are the same optimal values as defined in Table 4 for the 80 node and 40 node configuration tests.

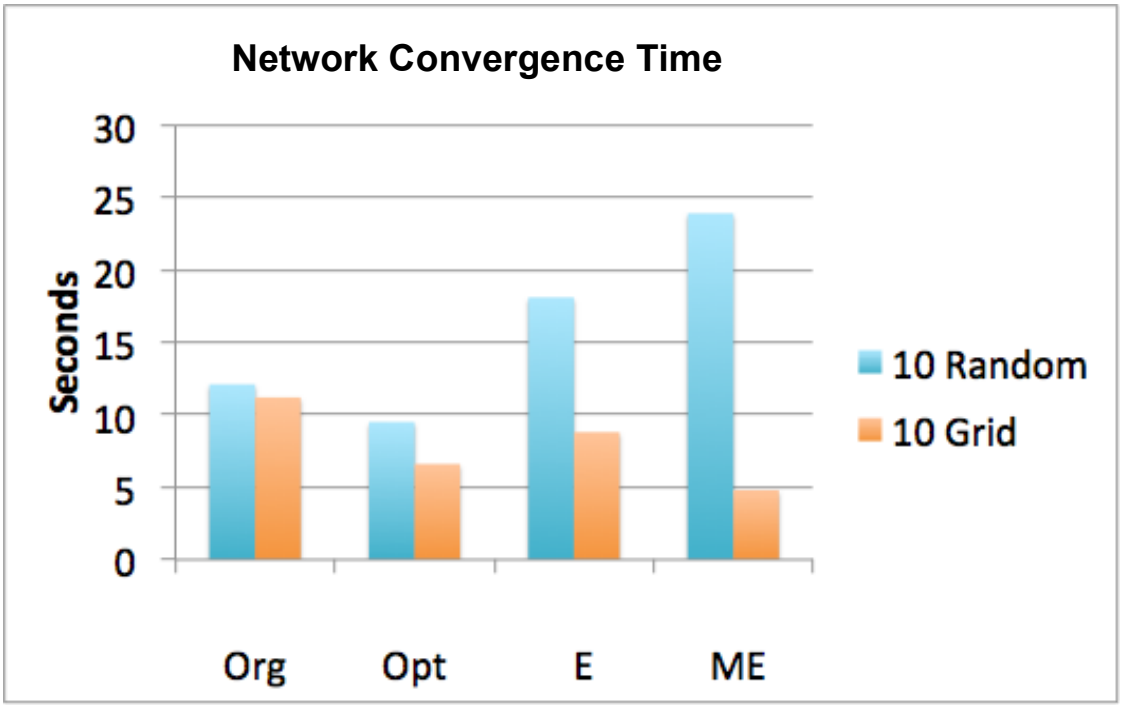


Figure 34: Network onvergence time for 10 client node configurations

Figure 34 demonstrates the network convergence time for both configurations. First, the team believes there are problems with results from ME-Trickle and E-Trickle for the random

layout based the other values gathered in this test and in the past data from the higher densities. Since we saw problems with the previous network convergence times for ME-Trickle, there may be a bug in the determination of this metric in our data capture within Cooja. All of the algorithms converge quicker on the grid layout. As with the 40 node configuration simulations, ME-Trickle has the best performance for the grid layout.

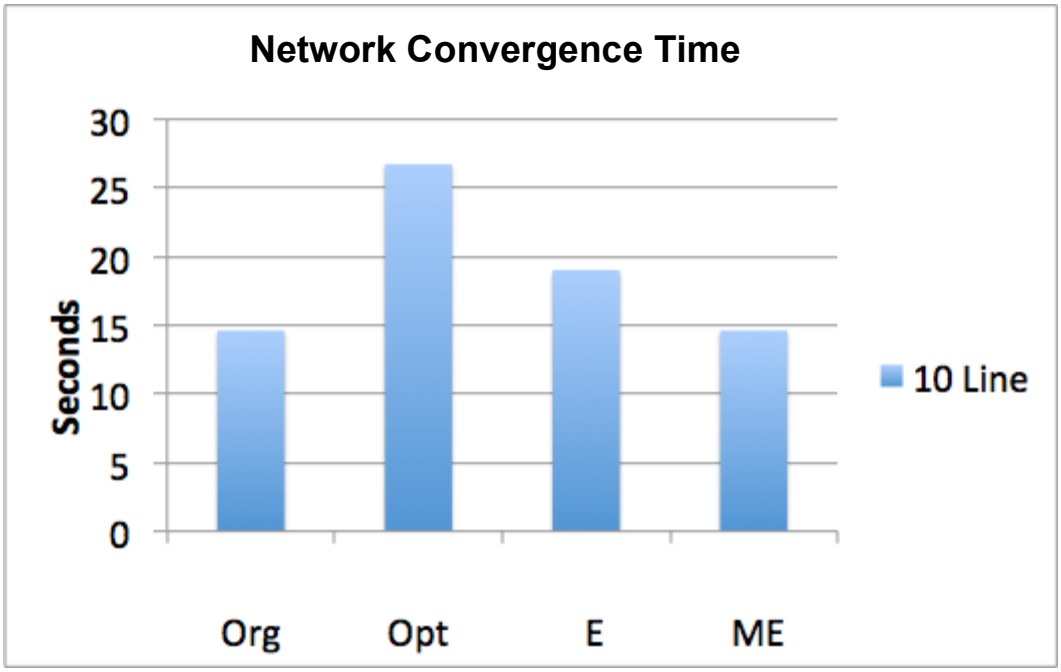


Figure 35: Network convergence time for 10 node line configuration

Figure 35 offers the network convergence time for the line layout. These convergence times are much worse than for the other two layouts. The longer times are likely due to the fact that packets from the outer nodes need to take more hops to reach the border router and general congestion near the border router. Both of these behaviors increase the likelihood of dropped packets along the path. The best result is from ME-Trickle which continues to perform the best for networks with smaller numbers of nodes. Surprisingly, original Trickle produces a nearly identical result to ME-Trickle. The opt Trickle result is very slow compared to all of its other results from the previous tests.

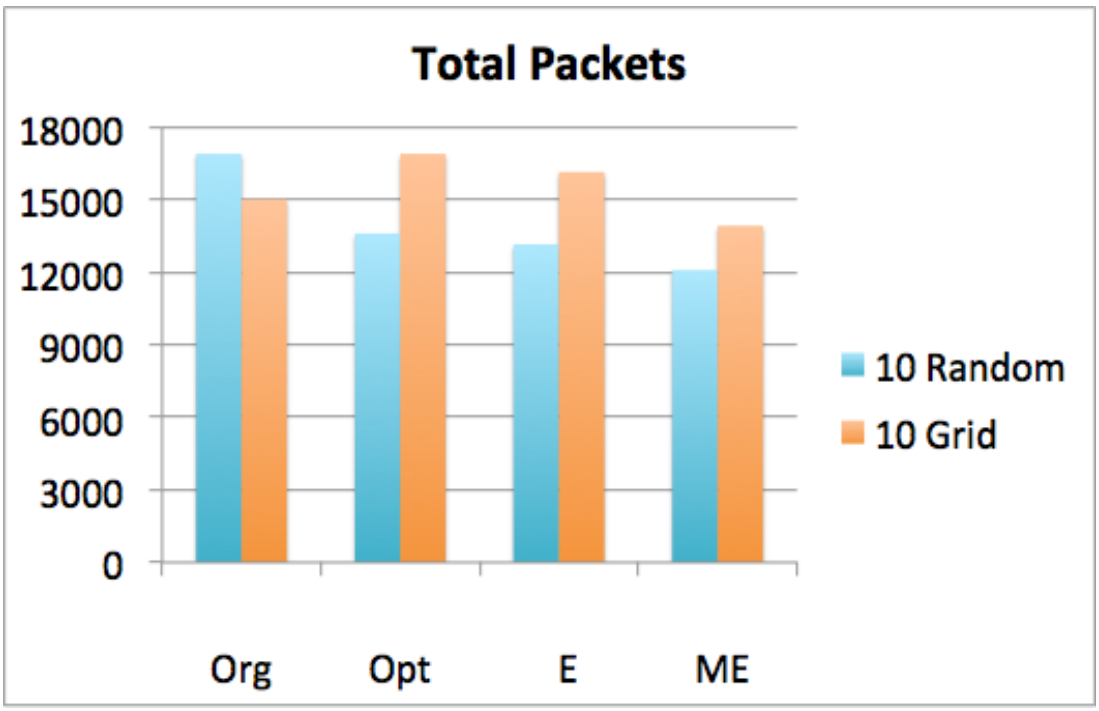


Figure 36: Total packets sent for 10 client node configurations

Figure 36 illustrates the total number of packets sent through both 10 node networks for the entire 15-minute simulation period. Interestingly, the data indicates that the majority of the algorithms send fewer packets in the random configuration. ME-Trickle continues to produce the lowest number of packets sent for both layouts. The original Trickle algorithm in the random configuration produces the worst results for this performance metric.

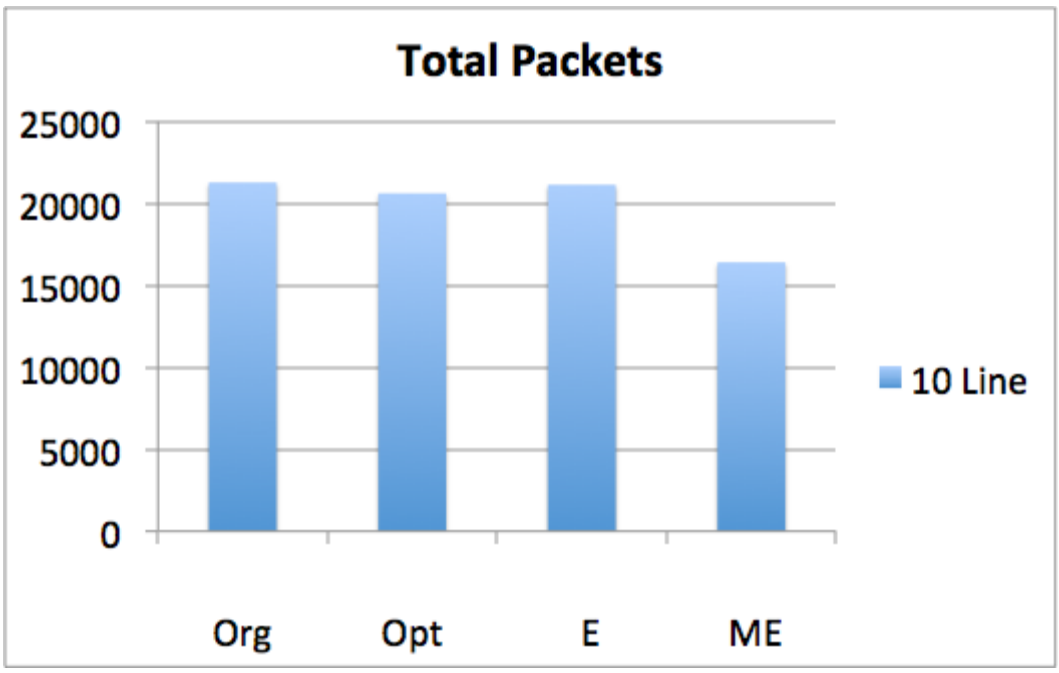


Figure 37: Total packets sent for 10 node line configuration

Figure 37 exhibits the total number of packets sent for the 10 node line simulations. ME-Trickle still produces the lowest number of packets sent through the network. One interesting result is that the number of packets sent in this test under org, opt and E-trickle are much higher than their corresponding total packet data from the other 10 node configurations. The increase in the number of packets sent is approximately about 5000 packets.

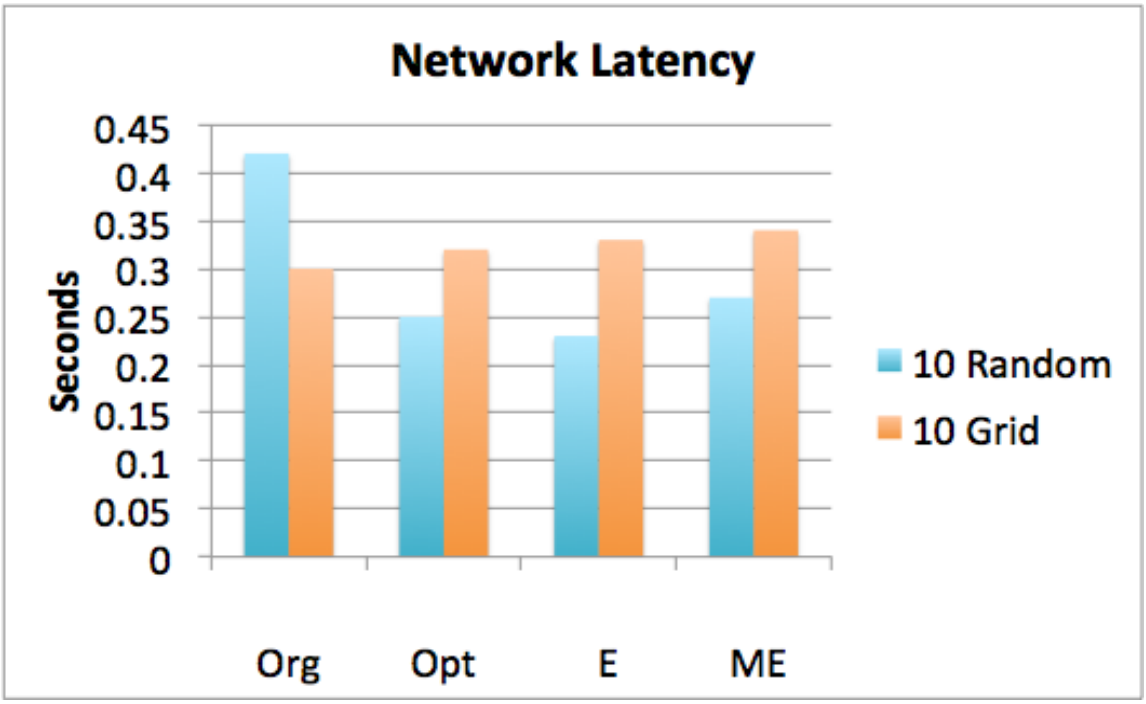


Figure 38: Network latency for 10 client node configurations

Figure 38 offers the average latency from all of the nodes in both configurations over the entire 15-minute simulation duration. The graph indicates a trend similar to the previous metric where all the algorithms produce lower average latencies on the random configuration than the grid layout except in the case of the original Trickle algorithm. Opt, E-Trickle, and ME-Trickle all produce very similar results for this metric with the difference between them being less than a tenth of second except for the original Trickle on the random layout which has a nearly 200 millisecond difference from the next closest result.

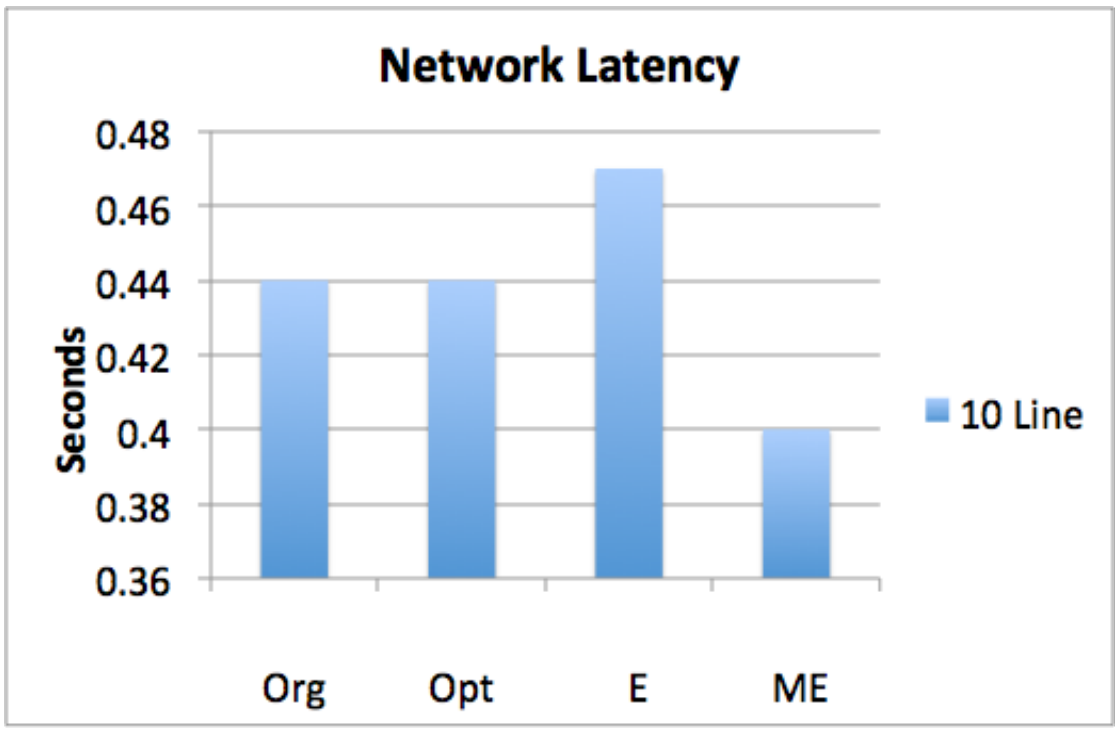


Figure 39: Network latency for 10 node line configuration

Figure 39 presents the network latency averaged over all of the nodes in the line configuration. The interesting result is that again the line configuration creates average latencies which are worse than the other 10 node configurations. These results suggest that the increased latencies may be due to increased number of hops for the packets along the line of nodes. An additional cause could be the congestion at the queues in the middle near the border router. Additionally, ME-Trickle continues to be the best for a smaller network.

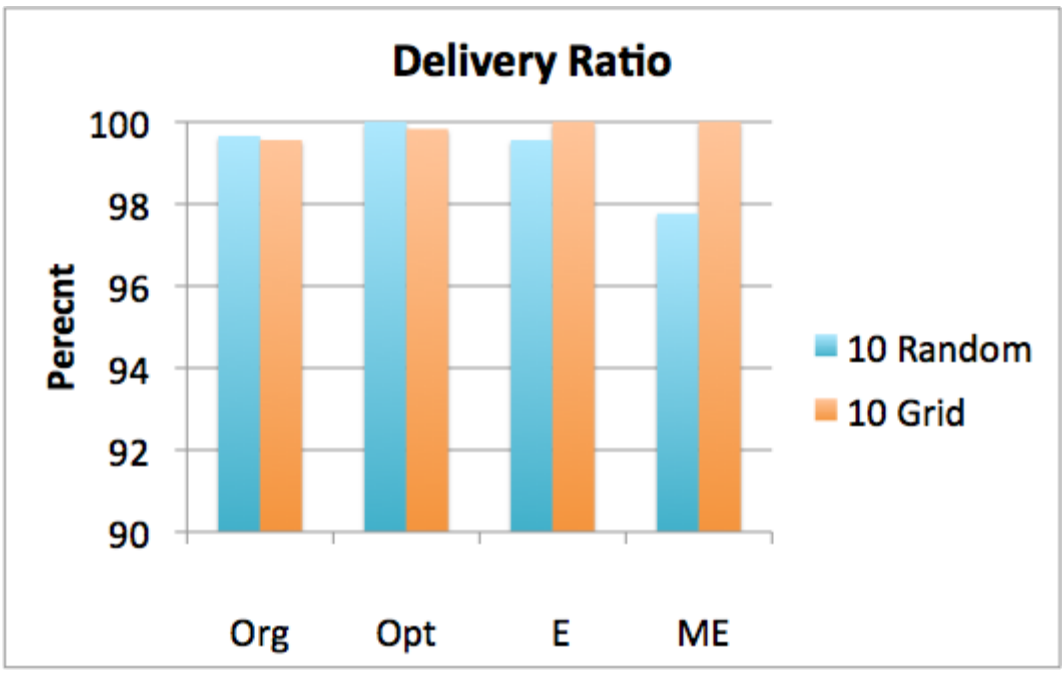


Figure 40: Delivery ratio for 10 node configurations

Figure 40 indicates the average delivery ratio over all 10 client nodes in the simulation. The delivery ratio results from these node configurations are very good for all four Trickle algorithms. The scale is important on this plot as the difference is very small at less than 3% and all have very good performance at about 97%.

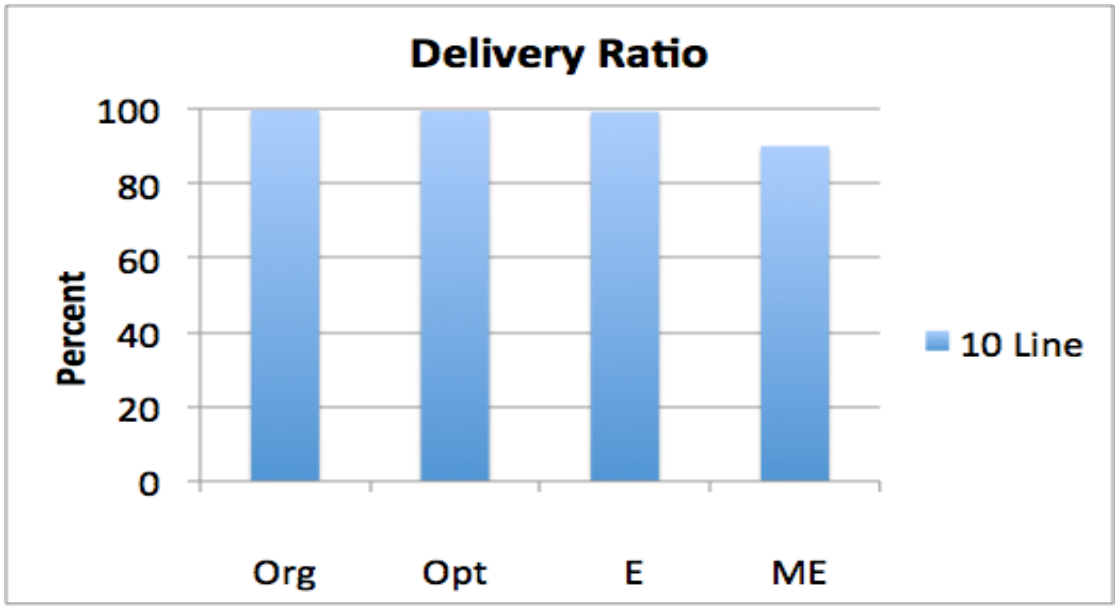


Figure 41: Delivery ratio for 10 node line configuration

Figure 41 expresses the delivery ratio averaged over the 10 client nodes in the line configuration. All four algorithms produce a very high packet delivery ratio for this configuration. Although, still acceptable, ME-Trickle has the worst delivery ratio in this configuration.

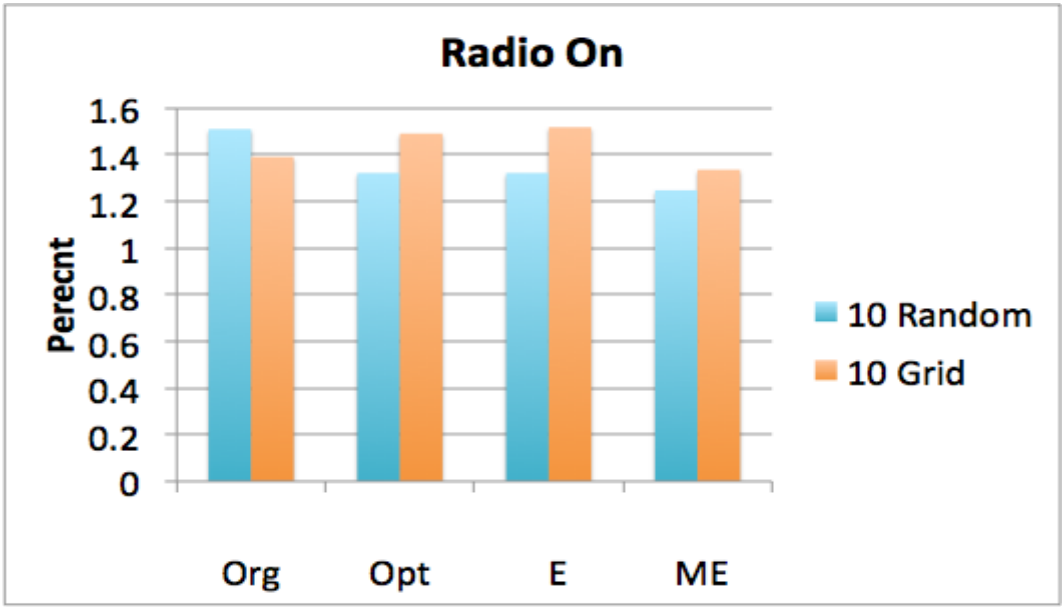


Figure 42: Radio on time 10 client node configurations

Figure 42 explains the radio on time percentage averaged over all of the nodes. The radio on time correlates with the number of packets sent through the 10 node networks. Thus, ME-Trickle continues to have the lowest radio on time under both configurations. The grid layout continues to have worse results than the random layout for all of the algorithms except the original Trickle algorithm. The margin of difference between all of the algorithms is very small for this performance metric.

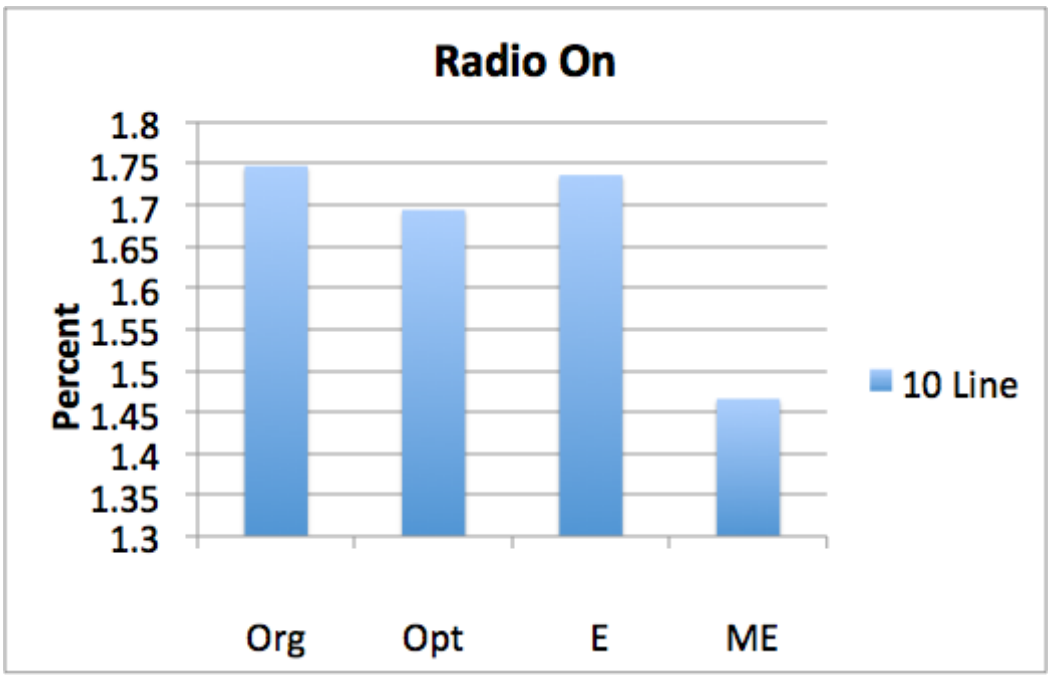


Figure 43: Radio on time 10 node line configuration

Figure 43 depicts the radio on time averaged over all of the 10 client nodes in a line configuration over the 15-minute simulation period. ME-Trickle again produces a much lower radio on times than the other three algorithms. The line configuration produces a much higher radio on time than the other two 10 node configurations for all Trickle variants.

This set of simulation tests indicate that in most cases ME-Trickle provides the best performance among the four Trickle algorithms for a smaller number of nodes in a network. The other observation is that while the three network configurations have the same number of sensor nodes the performance of the Trickle variants differs over the three layouts.

5. CONCLUSIONS

Over the course of the two months of this MQP, the project team ran over 300 distinct simulation tests for nearly 400 hours of simulated time. The performance metrics from these simulations suggest that the ME-Trickle algorithm generally performs the best under network topologies when the node density is not high. ME-Trickle provides the best performance across all of the metrics for 40 nodes configurations, and nearly all metrics for 10 node configurations. Additionally, this simulation study demonstrates that all of the four algorithm variants are sensitive to changes in the configuration of the nodes, as well as the density of the network. Overall, there is not a clear winner among the four Trickle variants for every single network scenario. The following section suggests several possible research directions to improve Trickle performance.

5.1 Future Work

Due to the time constraints on the project, there are a number of simulation tests that the team was not able to run. There are several further directions that could be to further evaluate the performances of the four algorithms. Unfortunately, there was not sufficient time to experimenter with varying some of the other important RPL parameters. For example, all of tests used a fixed wireless receive rate probability of 70%, which is very low. Thus, research should be done to simulate Trickle performance with higher receive probabilities to fully understand how this affects the four algorithms. Furthermore, the team was not able to do simulations of a network with mobile sensor nodes, which could be a very practical use for RPL and the Trickle algorithms in the future.

In addition, future work could delve deeper into the effects of the value of k, the redundancy counter for each of the algorithms. The team began to test this with the original

Trickle algorithm and Opt-Trickle, but the results were not adequately conclusive to draw strong, reasonable conclusions about the effectiveness of different values is various scenarios.

Another possible future work involves considering the impact of the number of network hops that a packet travels on average on Trickle performance. To facilitate future research using Cooja, it would be useful for someone to devise an automated mechanism for starting a series of Cooja simulations which facilitate changing the parameters in the source code in each successive test. This automation would allow future researchers to save testing time spent and avoid doing redundant work. Unfortunately, the project team was not able to quickly address this issue and was forced to manually change the parameters and start each test.

Finally, the team suggests a further optimized version of the ME-Trickle algorithm. The team observed that ME-Trickle, while in a network with a small number of hops to the border router, has the ability to conserve more energy by sending fewer packets. This allows it to maintain a higher delivery ratio and lower setup time than the other algorithms. However, in general, the results suggest ME-Trickle does not perform well in networks where packets travel a large number of hops. Therefore, the team proposes another optimization for the ME-Trickle algorithm.

The idea behind ME-Trickle is that the DIO interval jumps from minimum to maximum if there are no inconsistent transmissions. However, this strategy may not work well larger network with more hops because there could be more inconsistent transmissions that will not be received with a maximum interval after one consistent period. Based on this information, a version of ME-Trickle which doubles the DIO interval one time if there is a consistent transmission period would perform better. After the next consistent interval, the DIO interval would then jump to the maximum possible value. This proposed version of algorithm should

handle high hop counts scenarios better since it has a shorter response time to inconsistent packets, while also maintaining low energy consumption by quickly maximizing the DIO interval.

REFERENCES

[1] Ali, H. (2012). *A Performance Evaluation of RPL in Contiki*. Blekinge: Swedish Institute of Computer Science.

[2] Benson, D. "A Performance Evaluation of RPL with Variations of the Trickle Algorithm," WPI MQP, March 2016.

[3] B. Djamaa and M. Richardson, "Optimizing the Trickle Algorithm," IEEE Communications Letters, vol. 19, no. 5, pp. 819–822, May 2015.

[4] B. Ghaleb, A. Al-Dubai and E. Ekonomou, "E-Trickle:Enhanced Trickle Algorithm for Low-Power and Lossy Networks", School of Computing, Edinburgh Napier University Edinburgh, UK, Sep, 2015

[5] Dunkels, A. (2012). Contiki: The Open Source OS for the Internet of Things. Retrieved October 13, 2015.

[6] Lewis, P., T . Clausen, J. Hui, O. Gnawali, and J. Ko. "RFC 6206 - The Trickle Algorithm." *RFC 6206 - The Trickle Algorithm*. Internet Engineering Task Force, Mar. 2011. Web. 19 Jan. 2016.

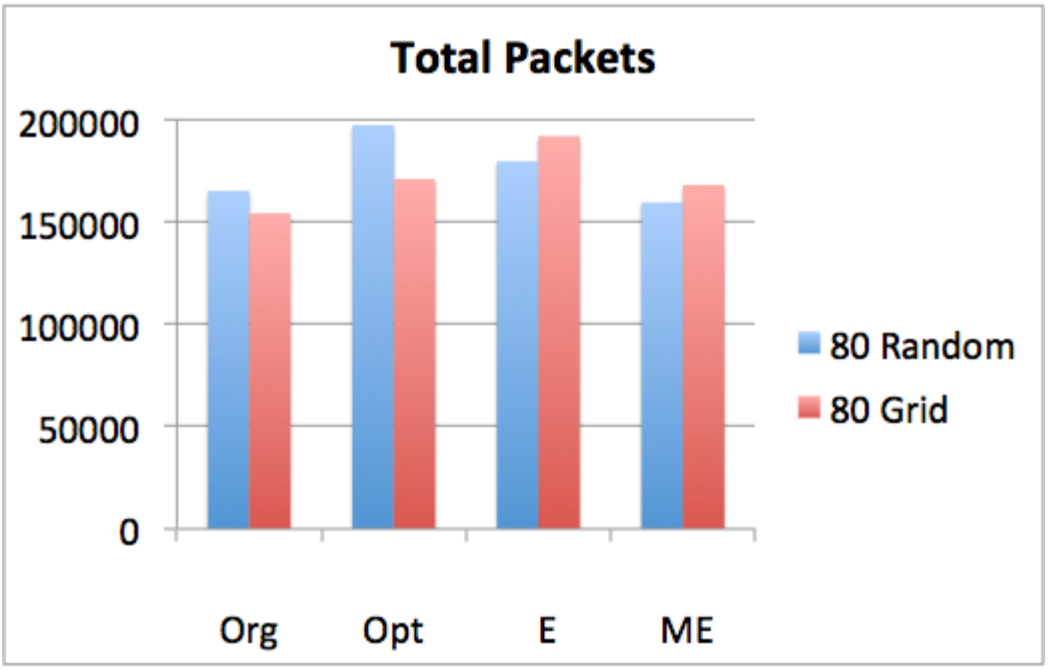
[7] N. Tsiftes, A. Dunkels, and J. Eriksson, "Low-power Interoperability for the IPv6-based Internet of Things", Swedish Institute of Computer Science.

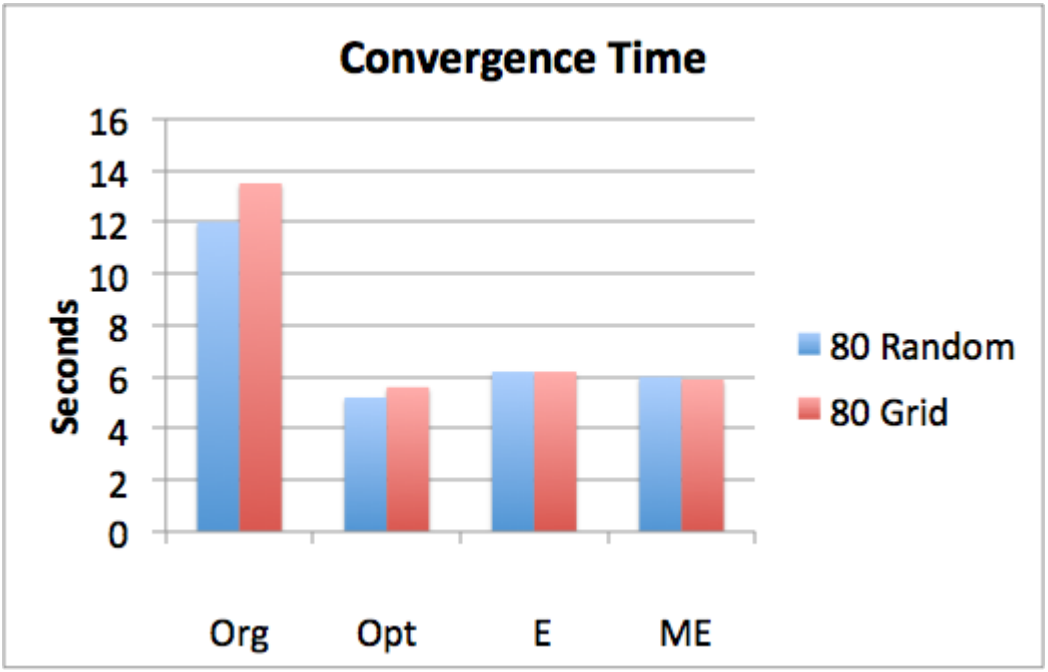
[8] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "RFC 6206: The trickle algorithm," *IETF*, 2011.

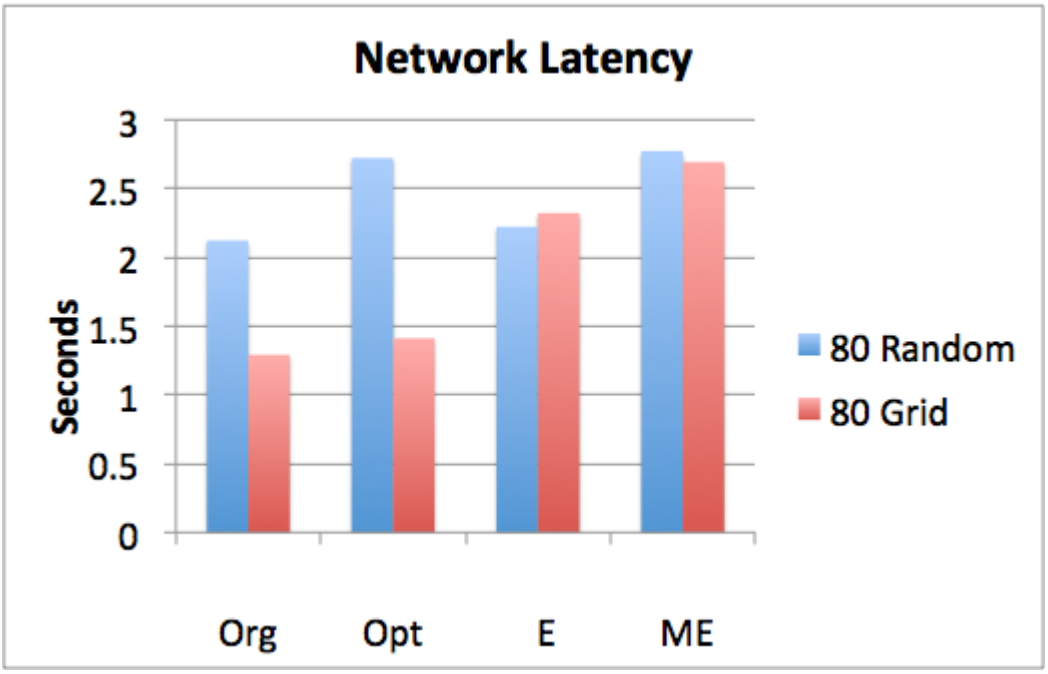
[9] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI*, 2004, pp. 15–28.

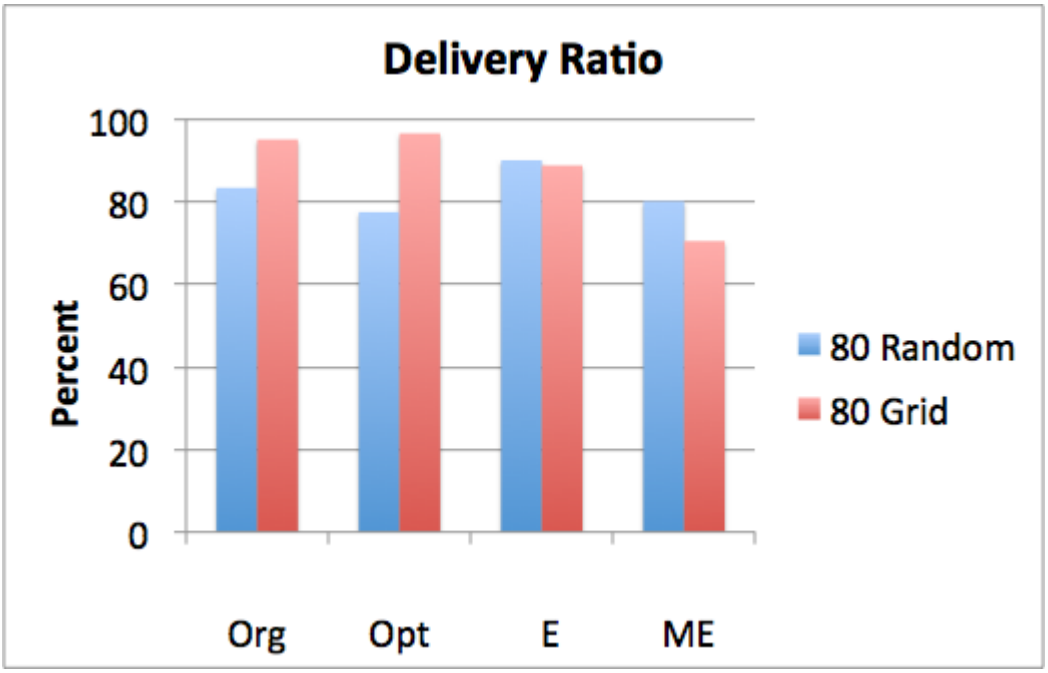
Appendix A

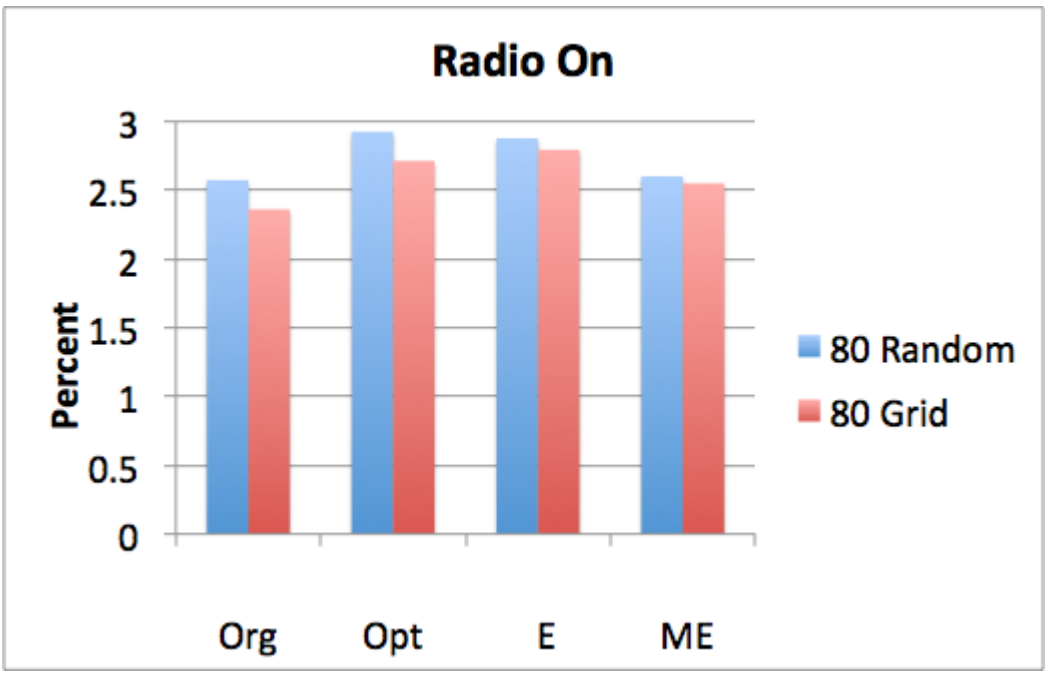
A.1 80 Node Configurations



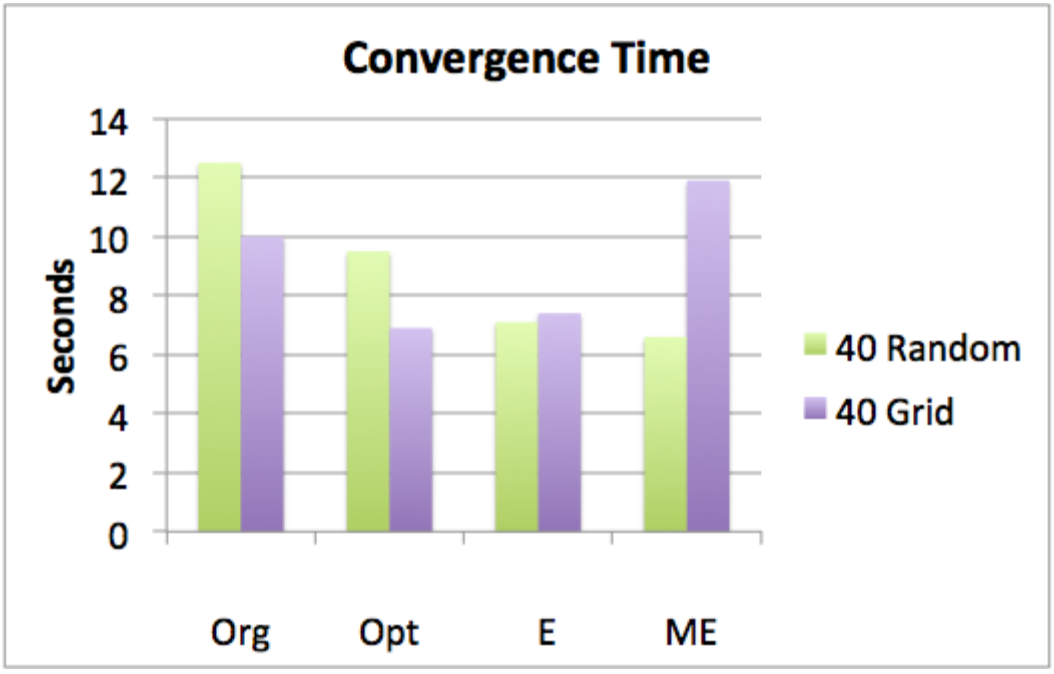


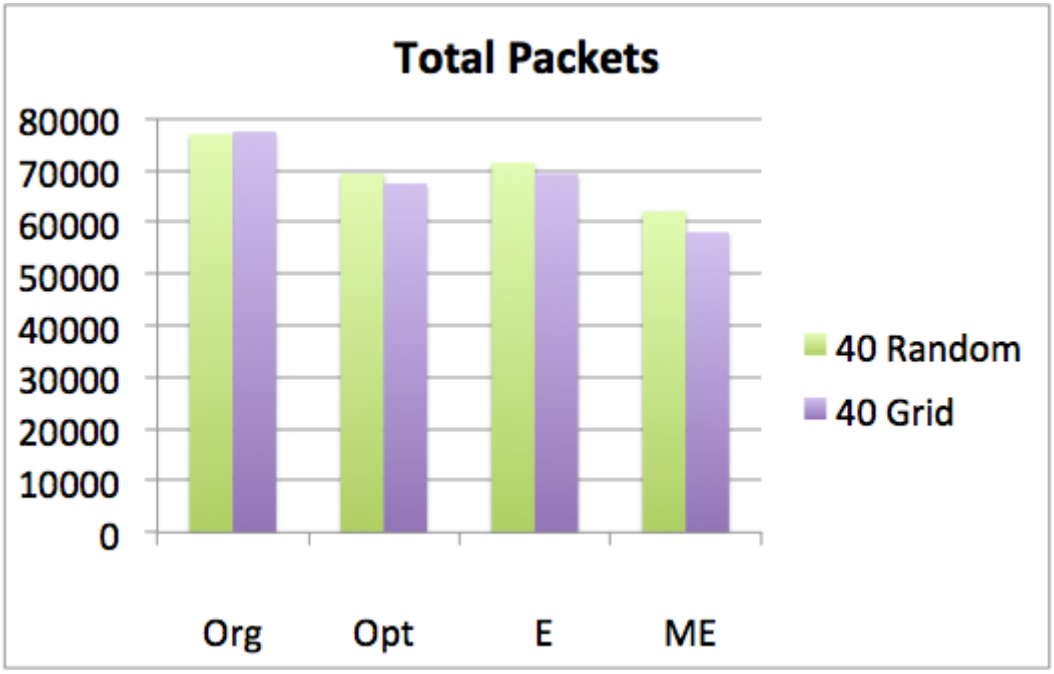

Network Latency
Seconds
3
2.5
2
1.5
1
0.5
0
Org
Opt
E
ME
80 Random
80 Grid

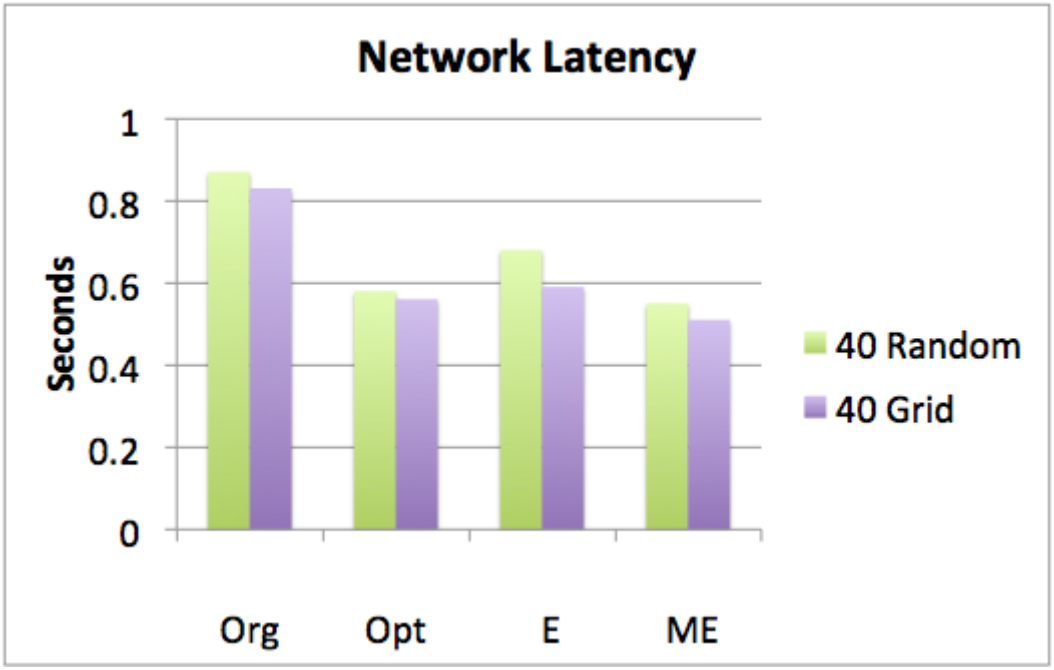

Delivery Ratio
Percent
100
80
60
40
20
0
Org
Opt
E
ME
80 Random
80 Grid

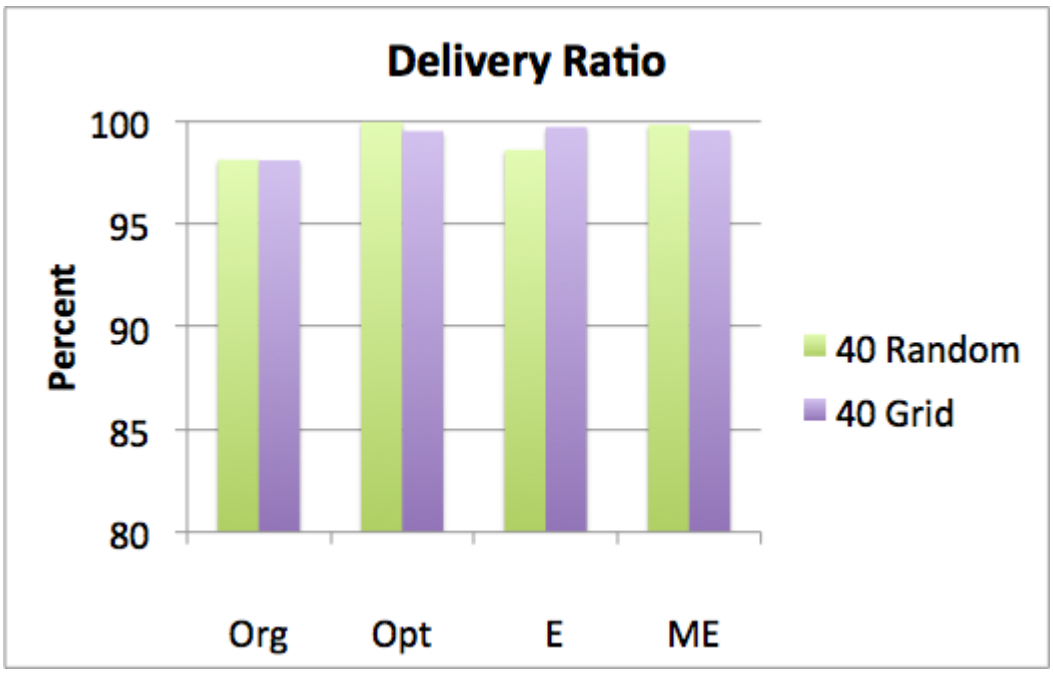


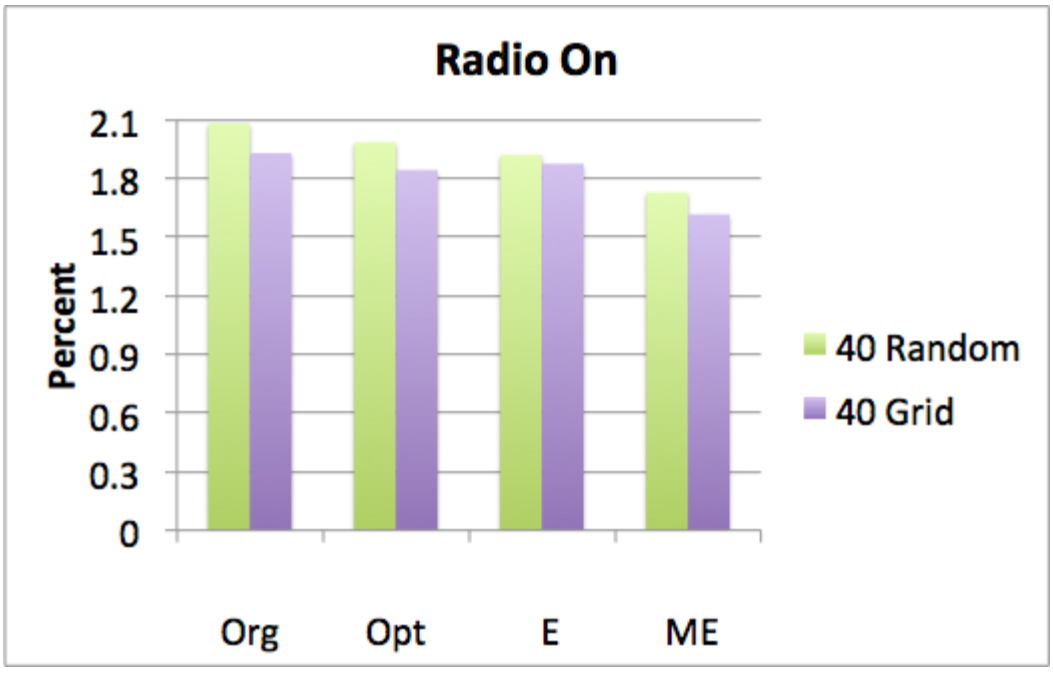
A.2 40 Node Configurations



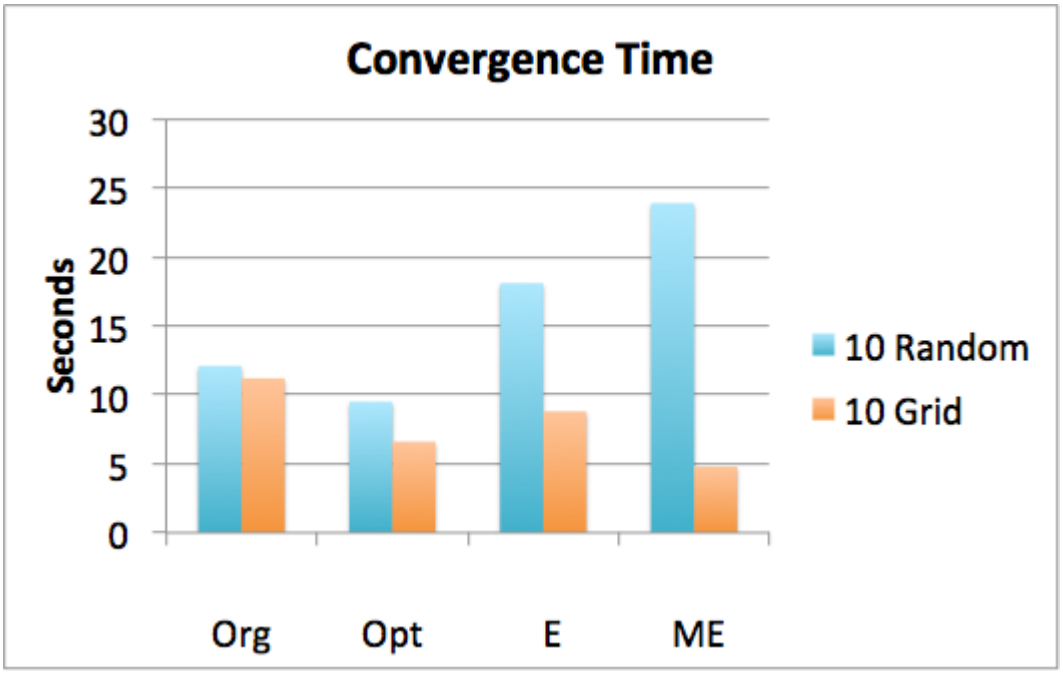

Total Packets
80000
70000
60000
50000
40000
30000
20000
10000
0
Org
Opt
E
ME
40 Random
40 Grid

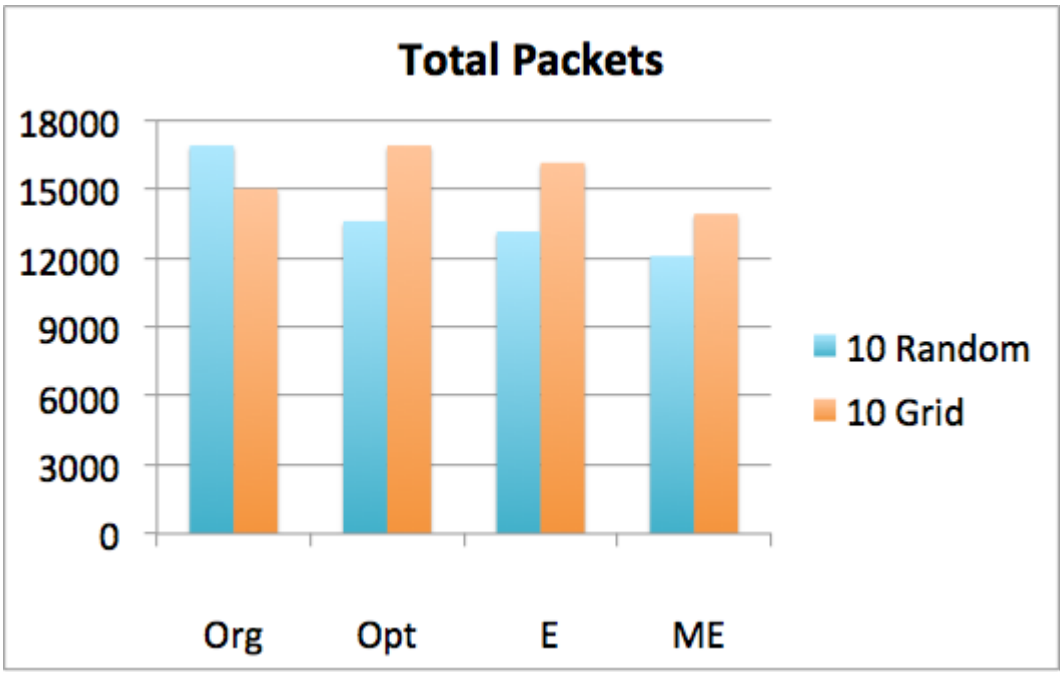

Network Latency
Seconds
1
0.8
0.6
0.4
0.2
0
Org
Opt
E
ME
40 Random
40 Grid

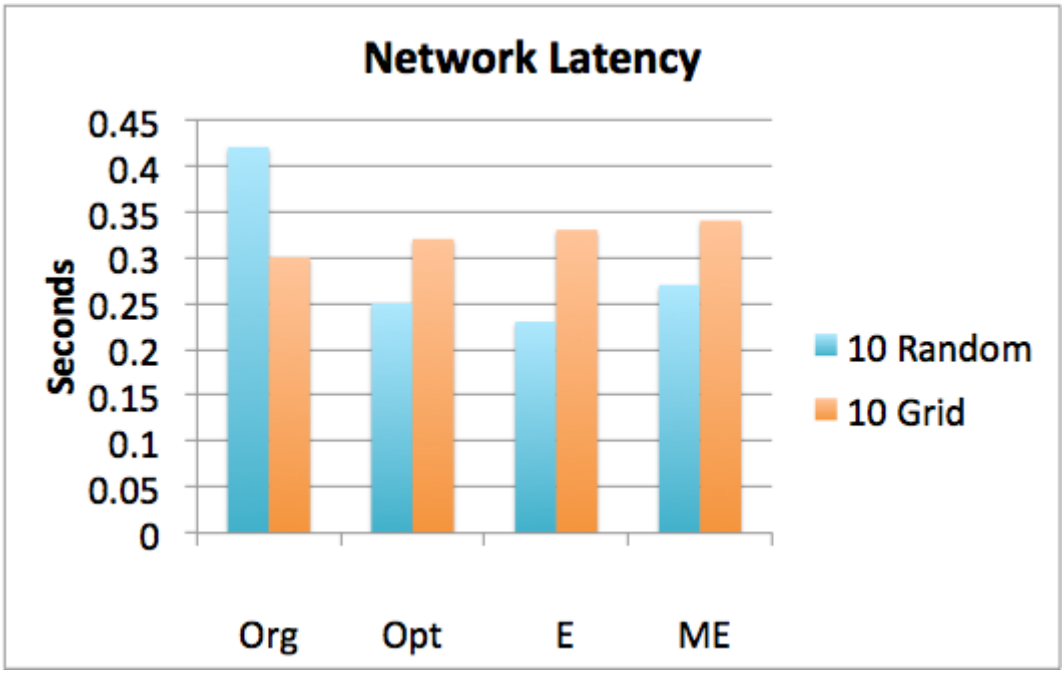

Delivery Ratio
100
95
90
85
80
Percent
Org
Opt
E
ME
40 Random
40 Grid

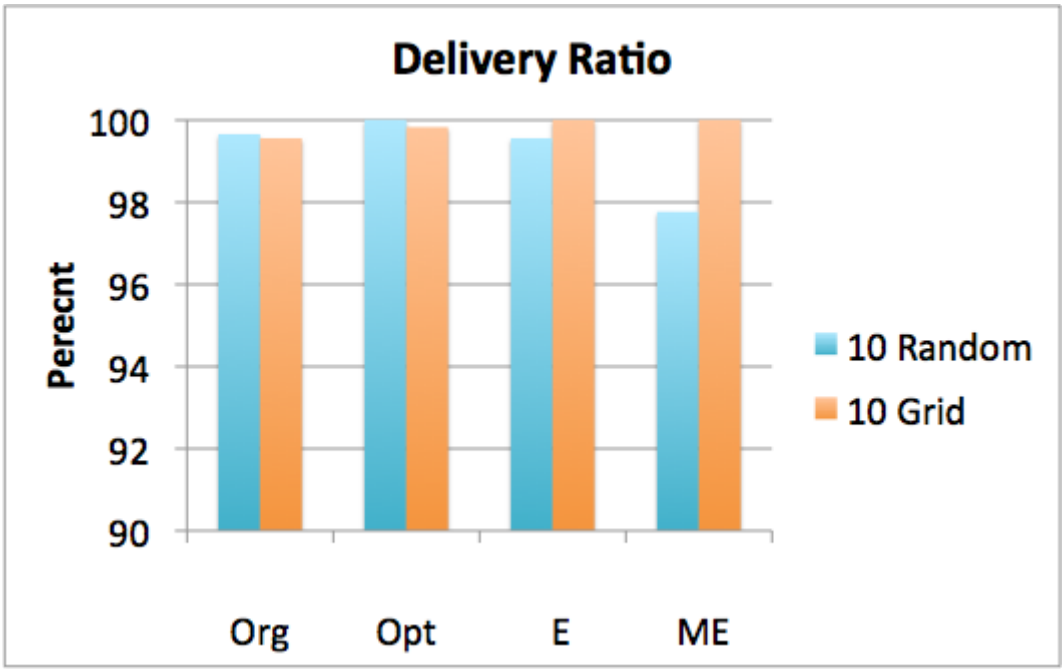

Radio On
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0
Percent
Org
Opt
E
ME
40 Random
40 Grid

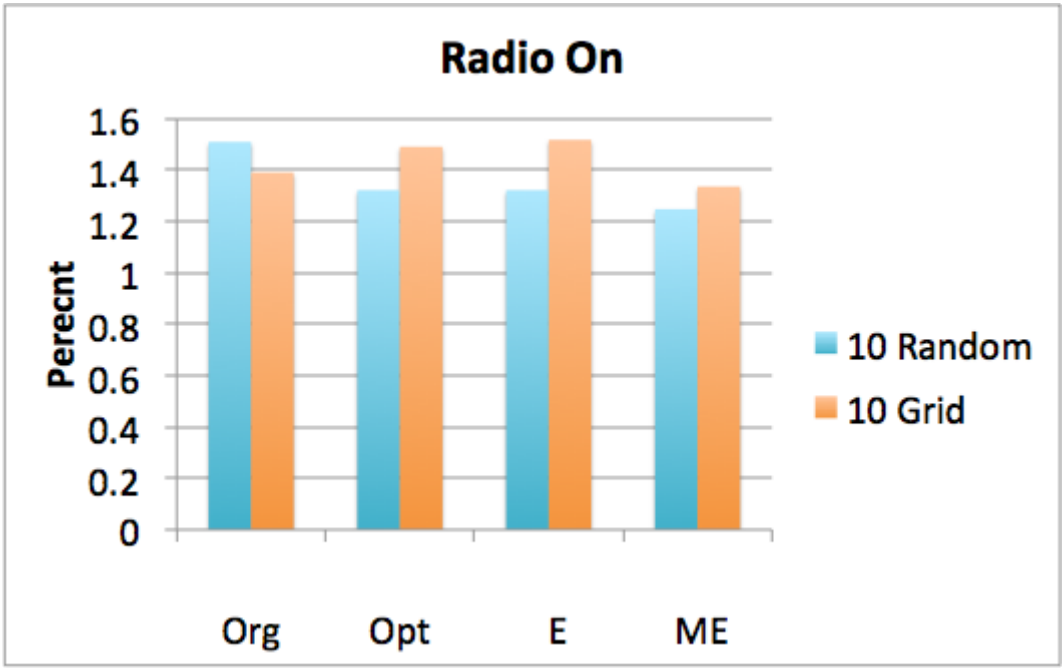
A.3 10 Node Configurations

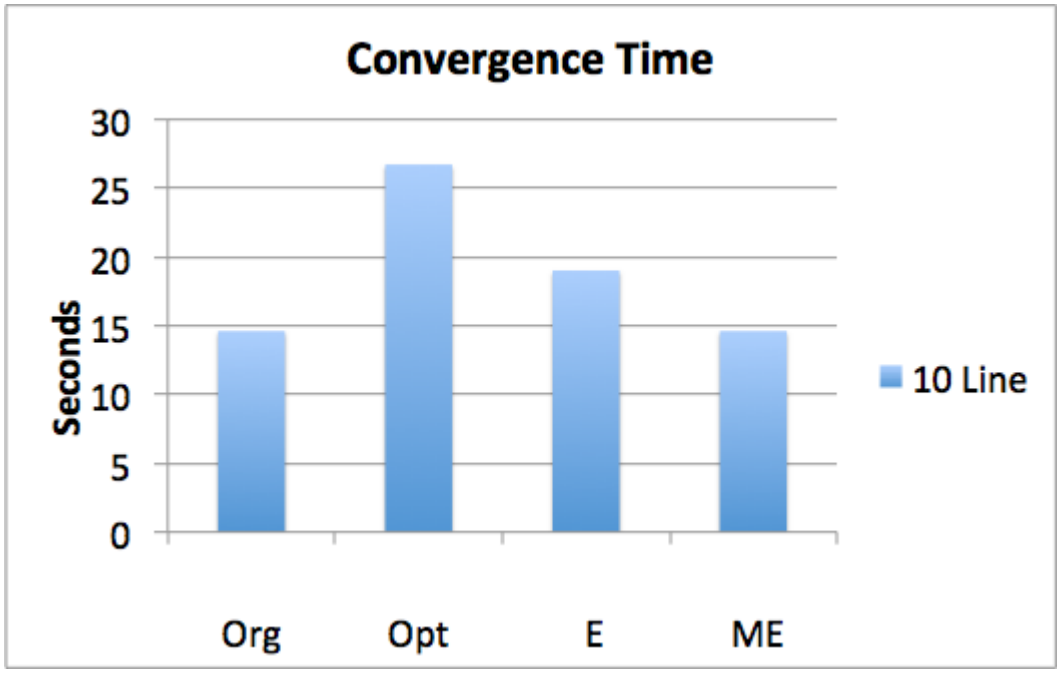


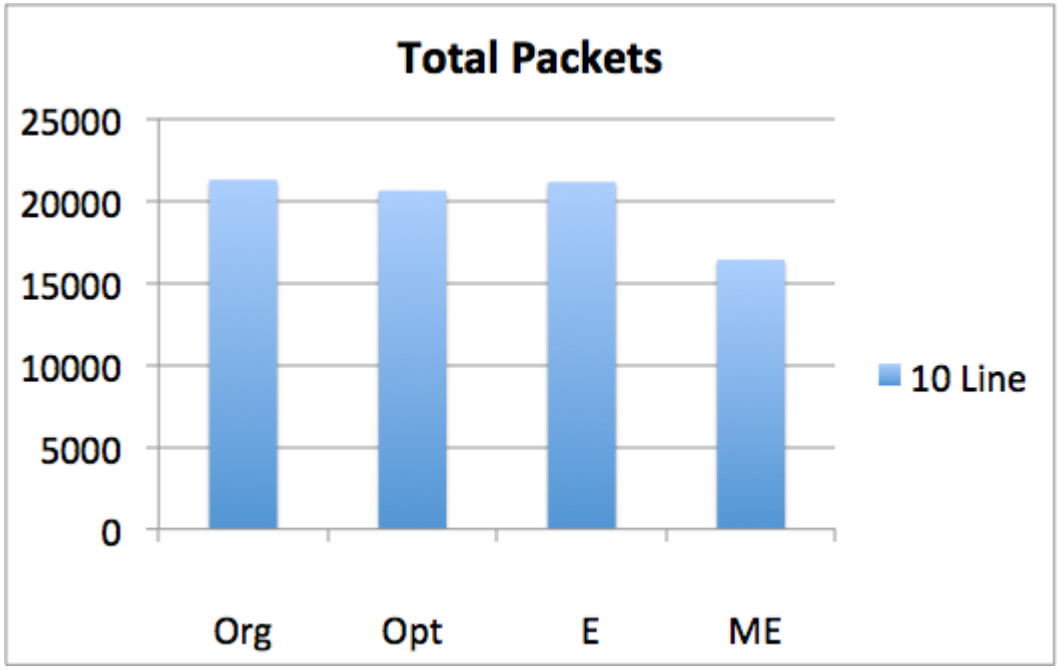


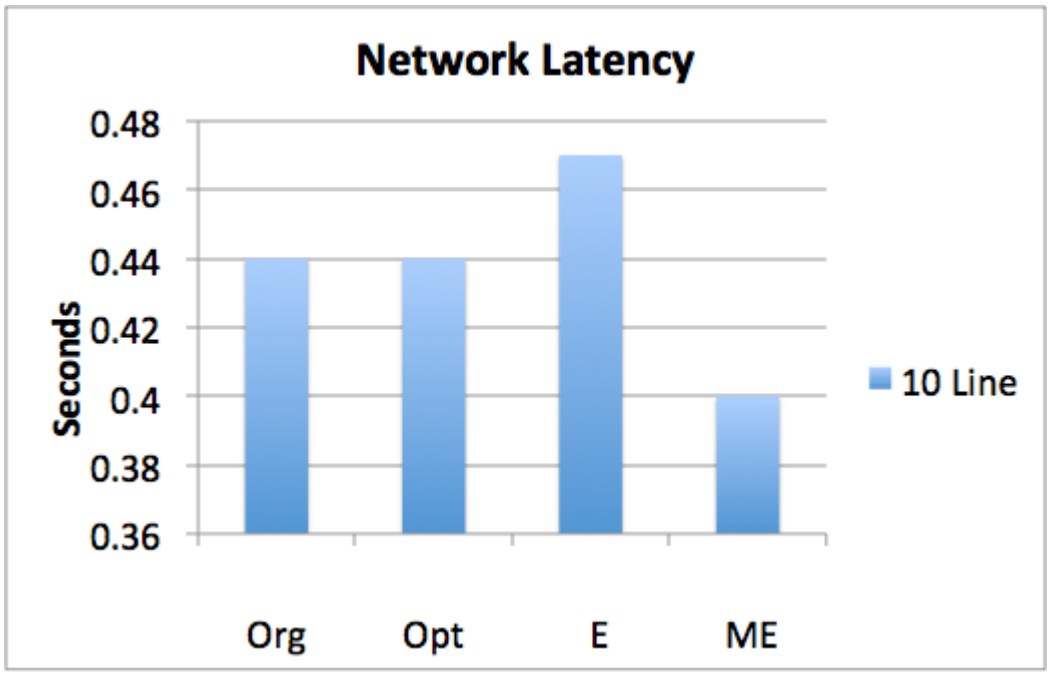

Network Latency
Seconds
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Org
Opt
E
ME
10 Random
10 Grid

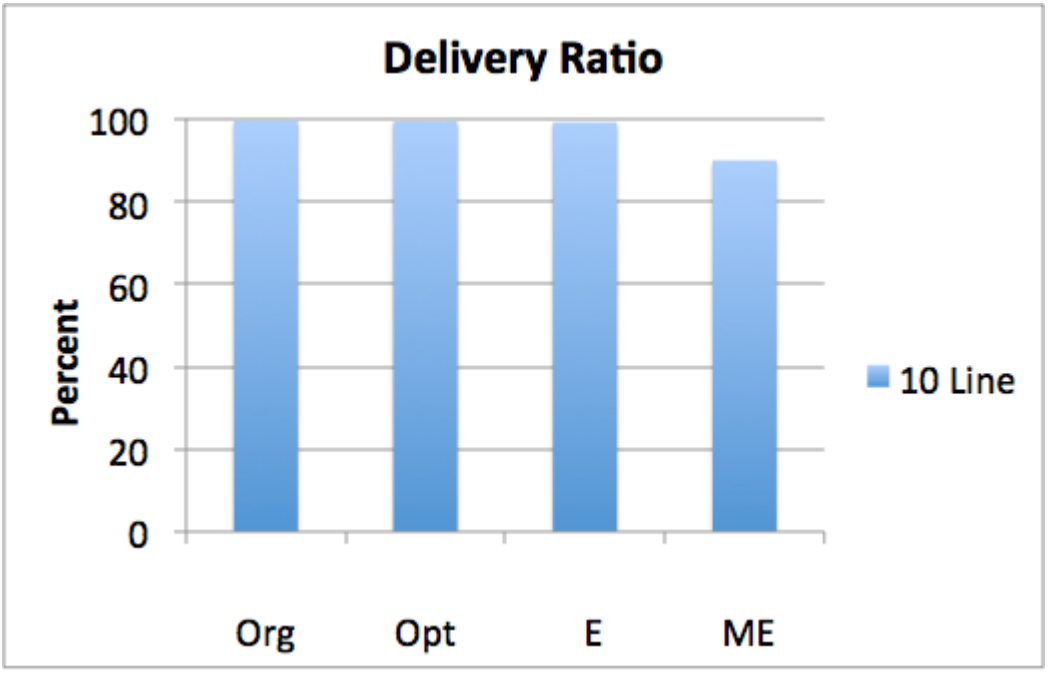

Delivery Ratio
Perecnt
100
98
96
94
92
90
Org
Opt
E
ME
10 Random
10 Grid

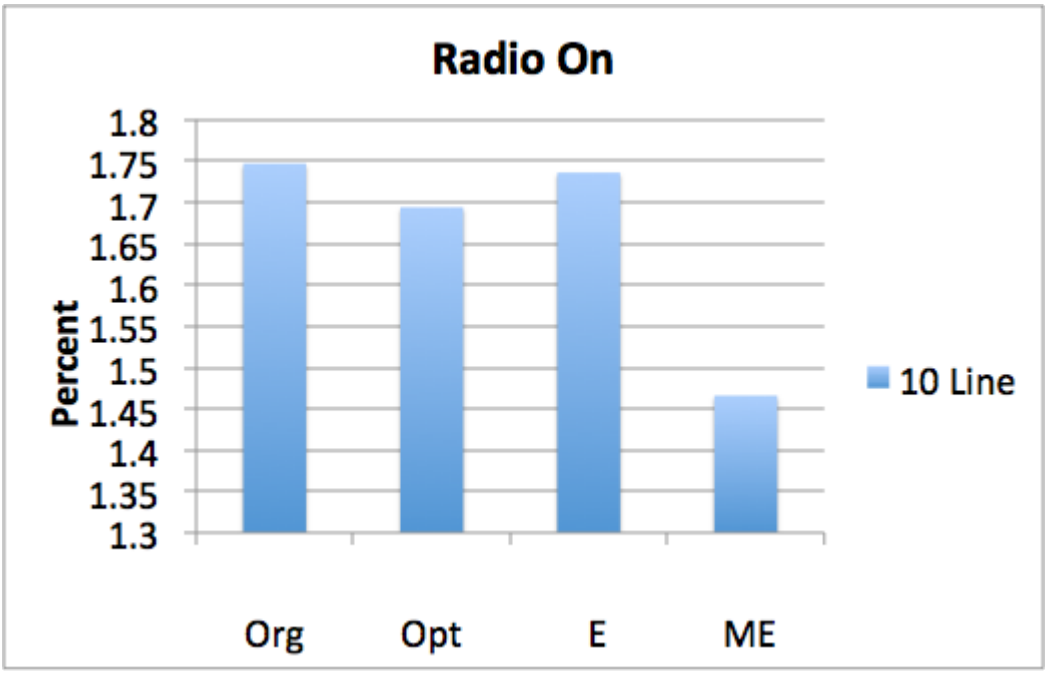

Radio On
Perecnt
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Org
Opt
E
ME
10 Random
10 Grid


Convergence Time
Seconds
30
25
20
15
10
5
0
Org
Opt
E
ME
10 Line

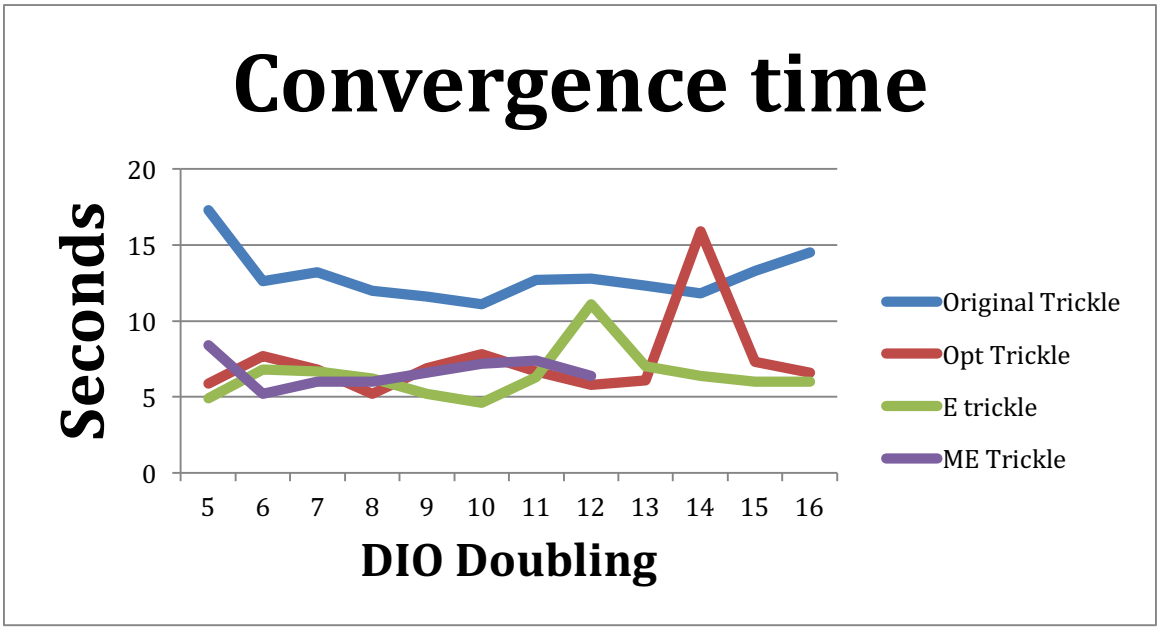

Total Packets
25000
20000
15000
10000
5000
0
Org
Opt
E
ME
10 Line

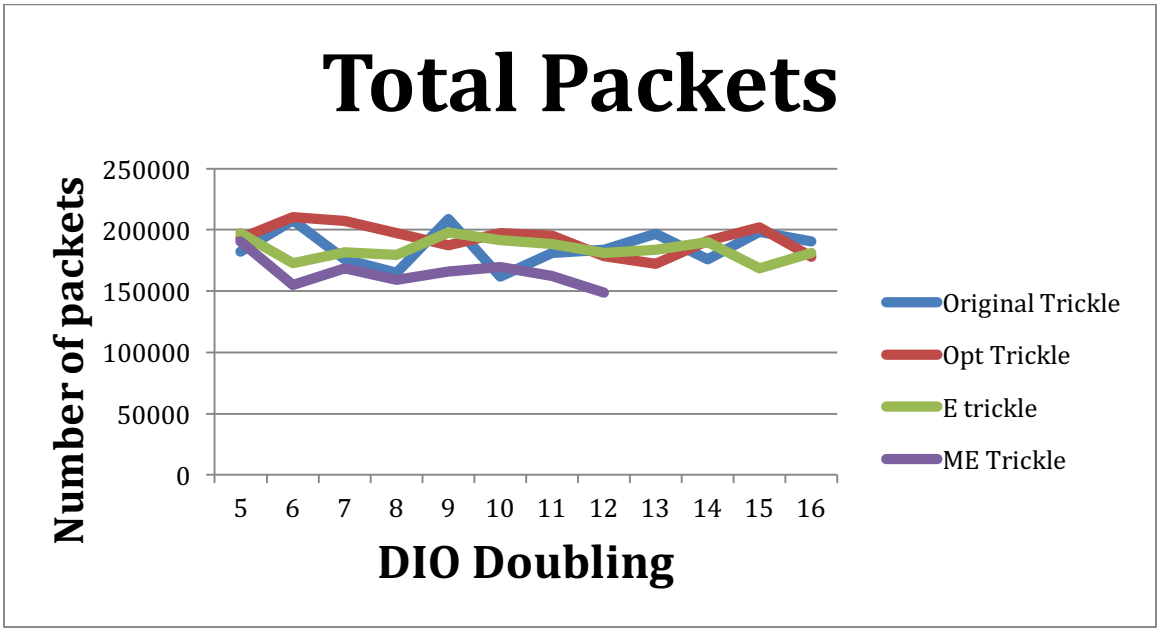

Network Latency
Seconds
0.48
0.46
0.44
0.42
0.4
0.38
0.36
Org
Opt
E
ME
10 Line

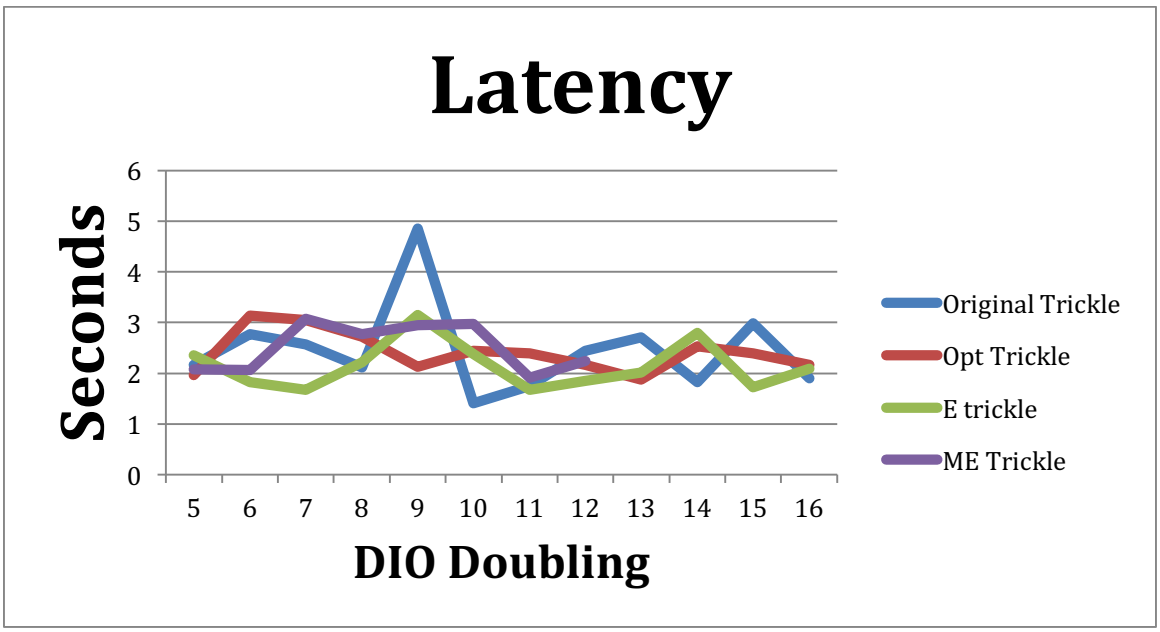

Delivery Ratio
100
80
60
40
20
0
Percent
Org
Opt
E
ME
10 Line

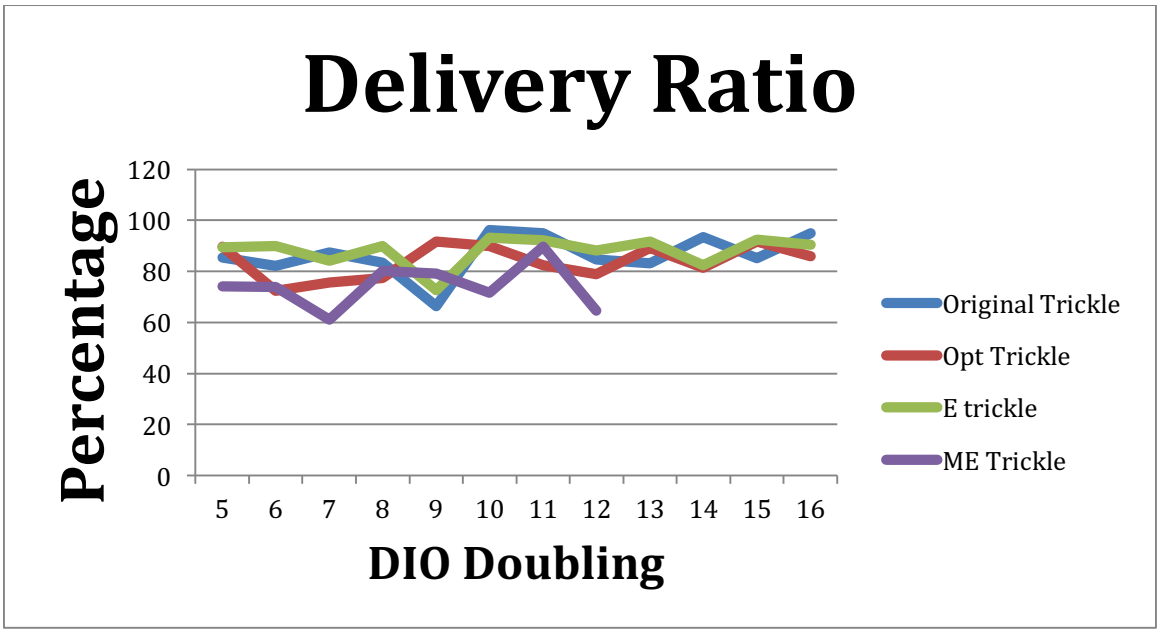

Radio On
1.8
1.75
1.7
1.65
1.6
1.55
1.5
1.45
1.4
1.35
1.3
Percent
Org
Opt
E
ME
10 Line

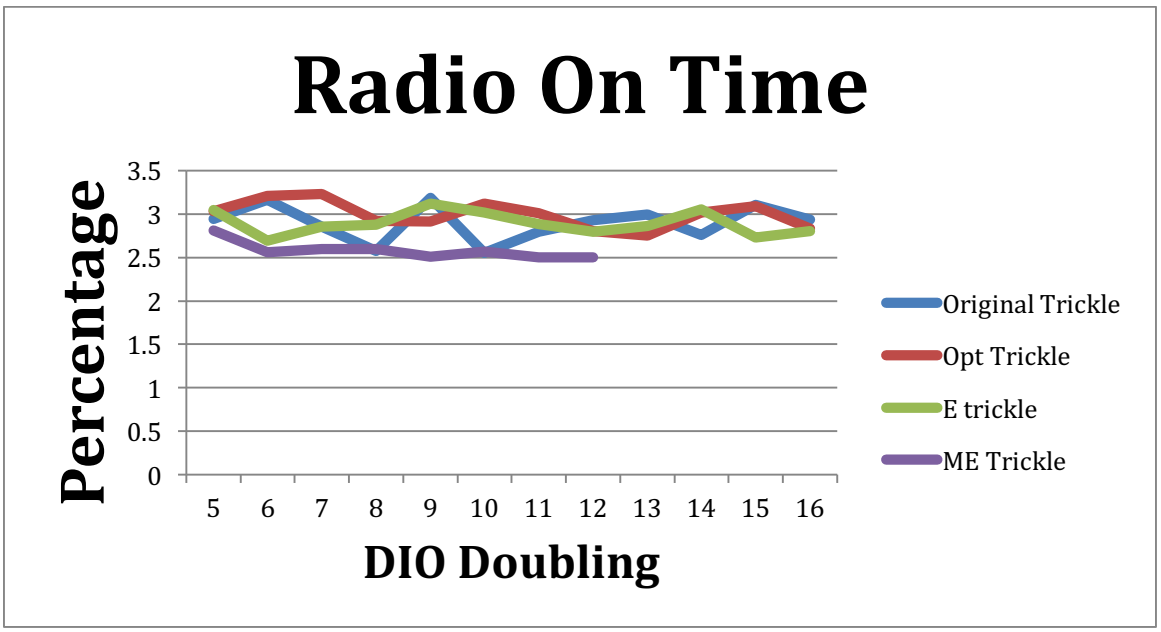
A.4 DIO Doubling Test Data



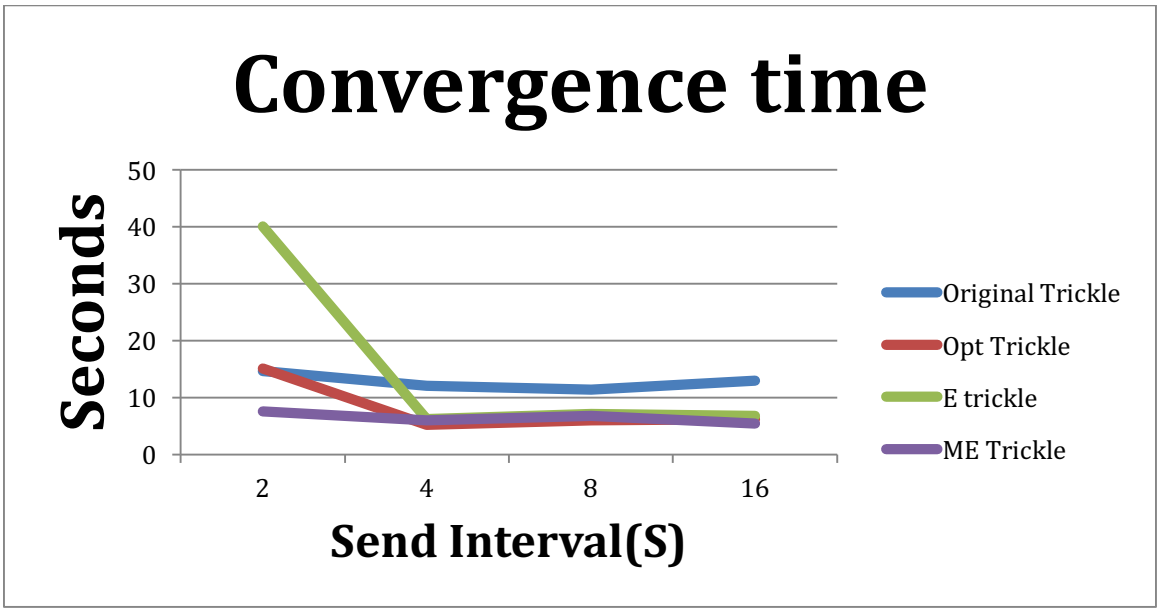


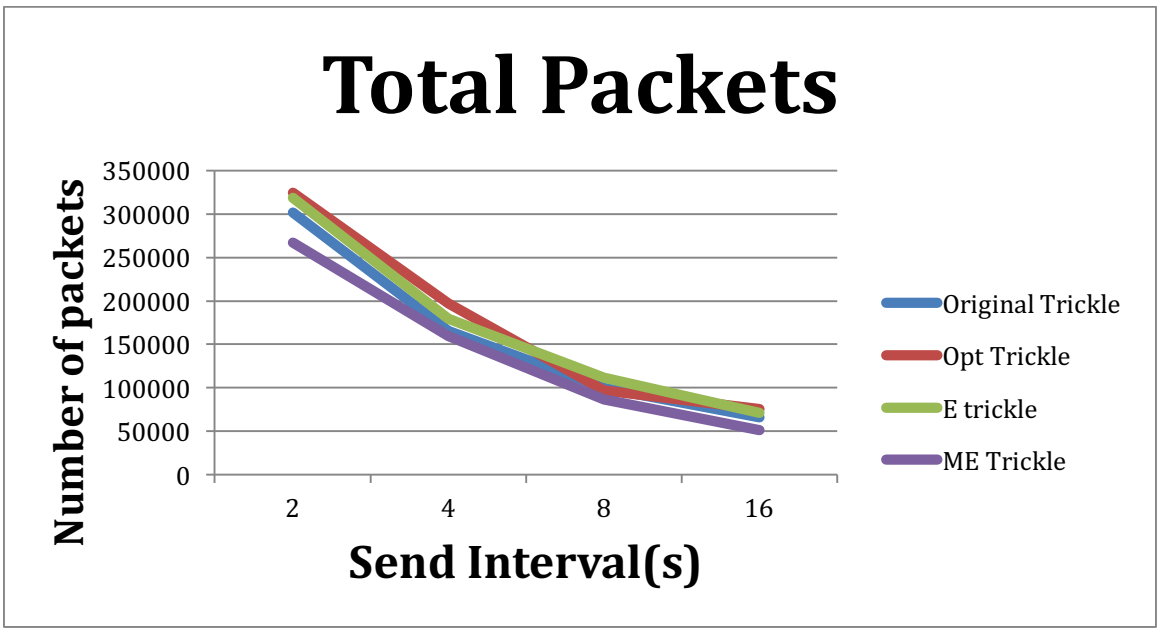

Latency
Seconds
0
1
2
3
4
5
6
5 6 7 8 9 10 11 12 13 14 15 16
DIO Doubling
Original Trickle
Opt Trickle
E trickle
ME Trickle

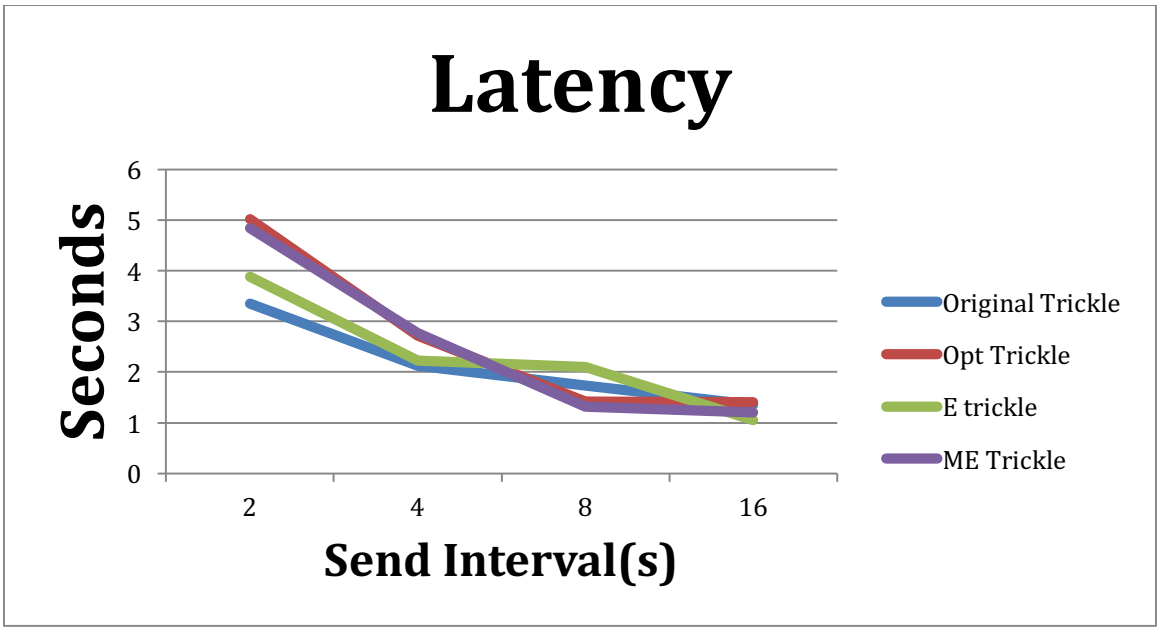

Delivery Ratio
Percentage
0
20
40
60
80
100
120
5 6 7 8 9 10 11 12 13 14 15 16
DIO Doubling
Original Trickle
Opt Trickle
E trickle
ME Trickle

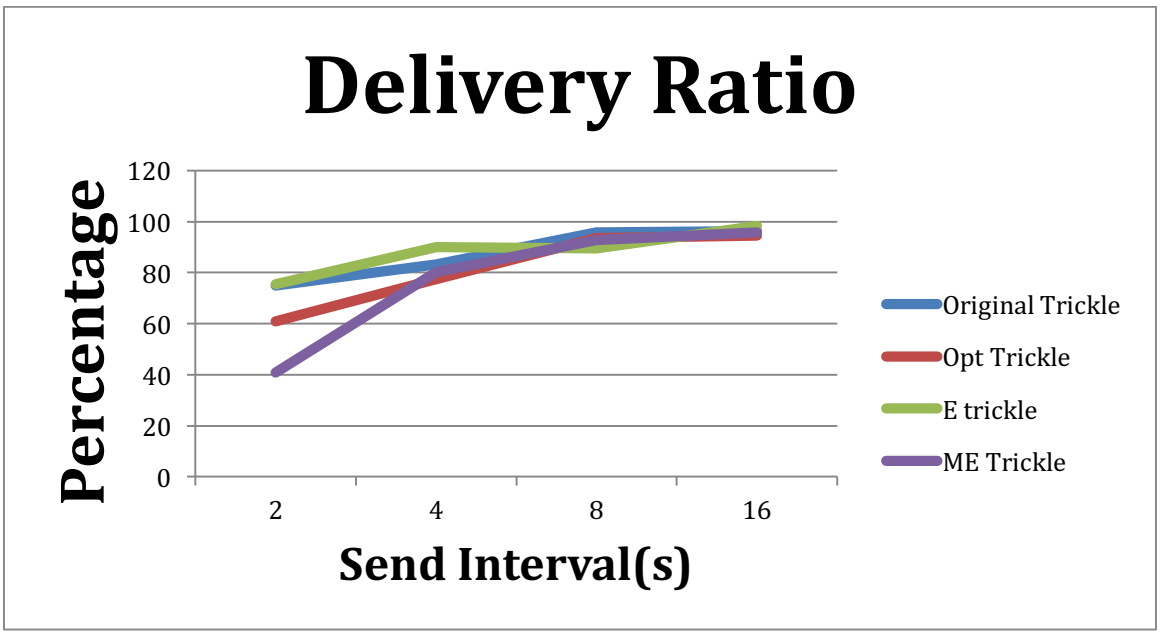


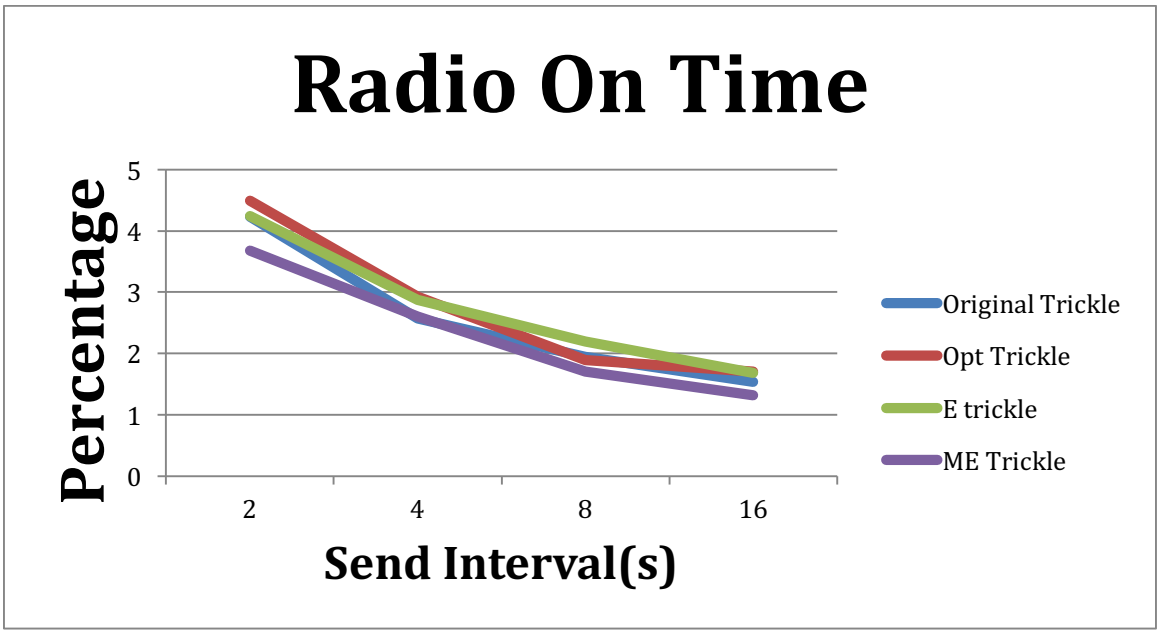
A.5 Send Interval Test Data



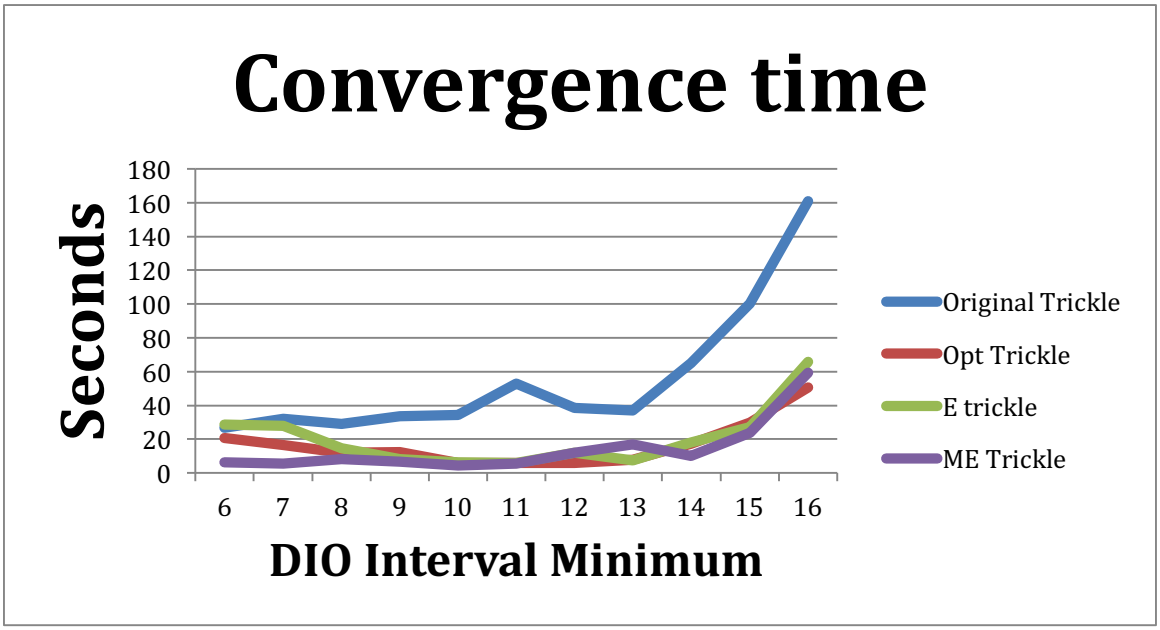

Total Packets
Number of packets
350000
300000
250000
200000
150000
100000
50000
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

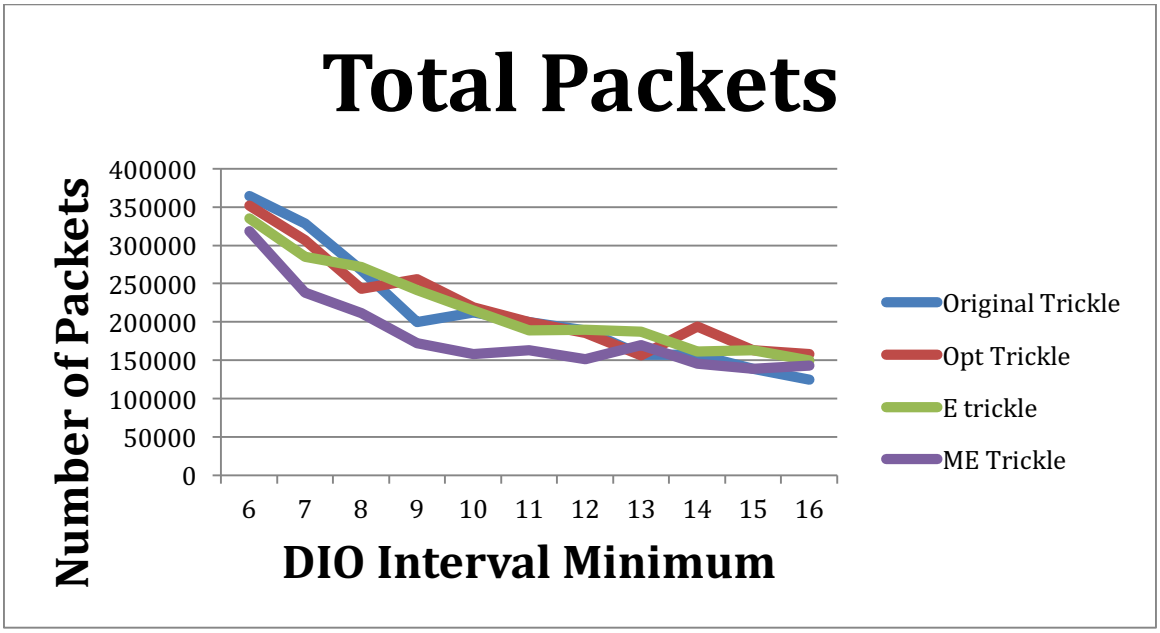

Latency
Seconds
6
5
4
3
2
1
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

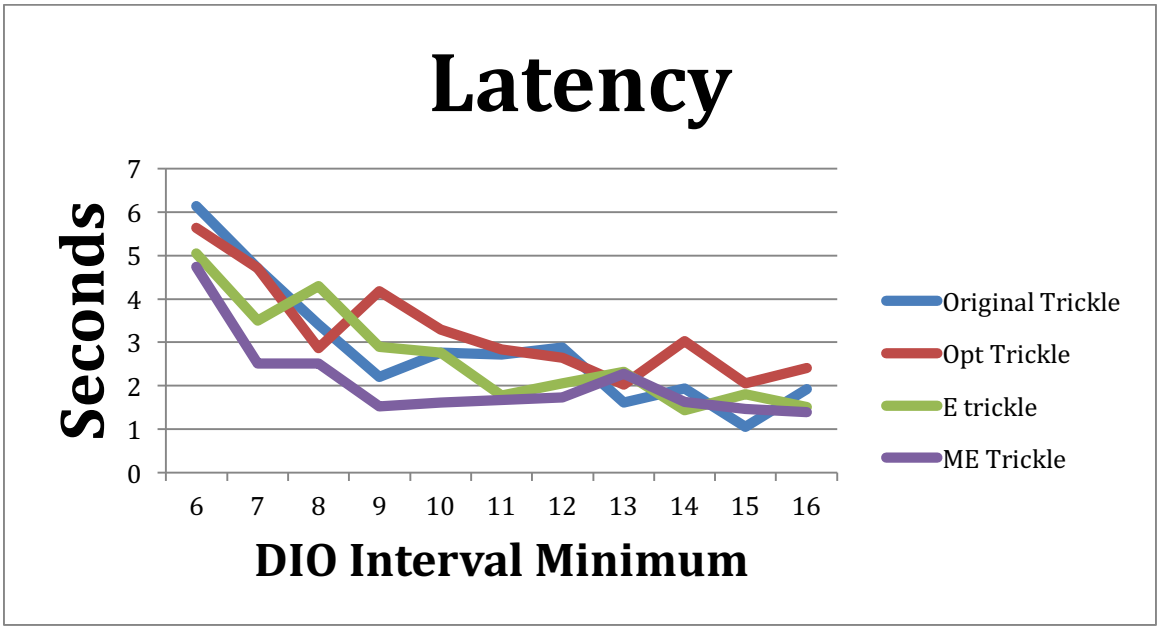

Delivery Ratio
Percentage
120
100
80
60
40
20
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

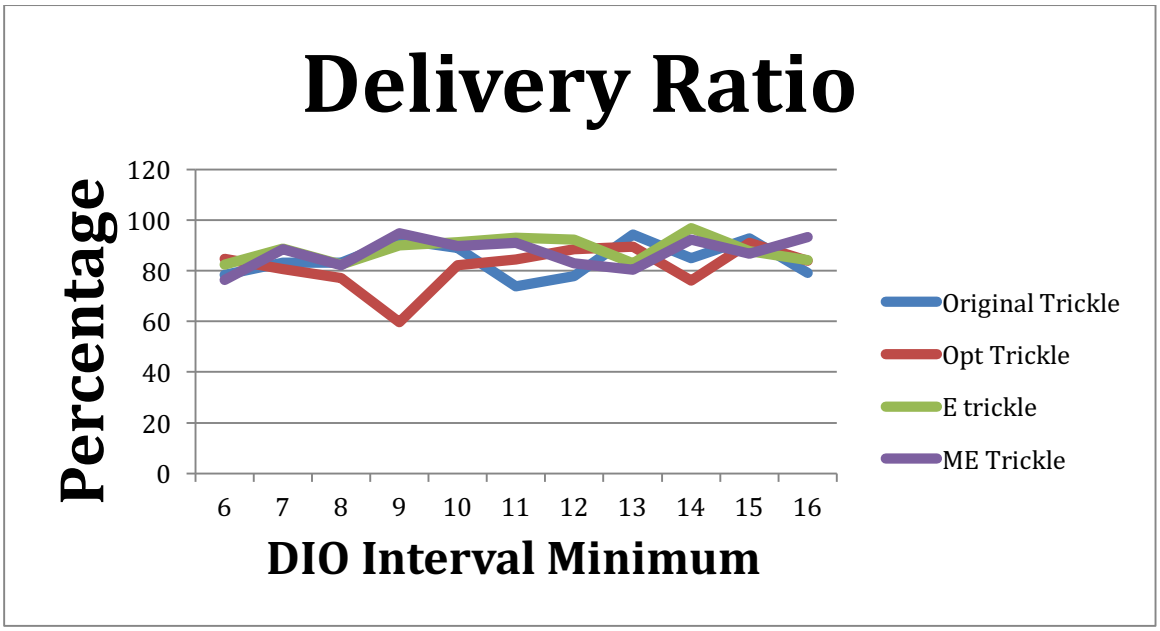

Radio On Time
Percentage
5
4
3
2
1
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

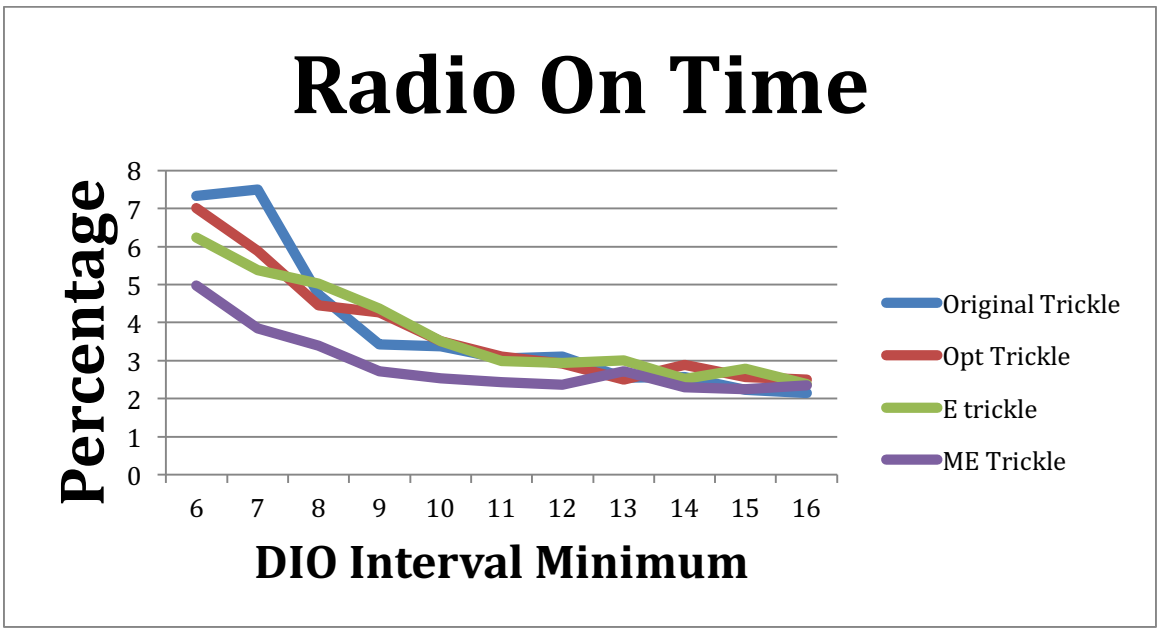
A.6 DIO Interval Minimum Test Data



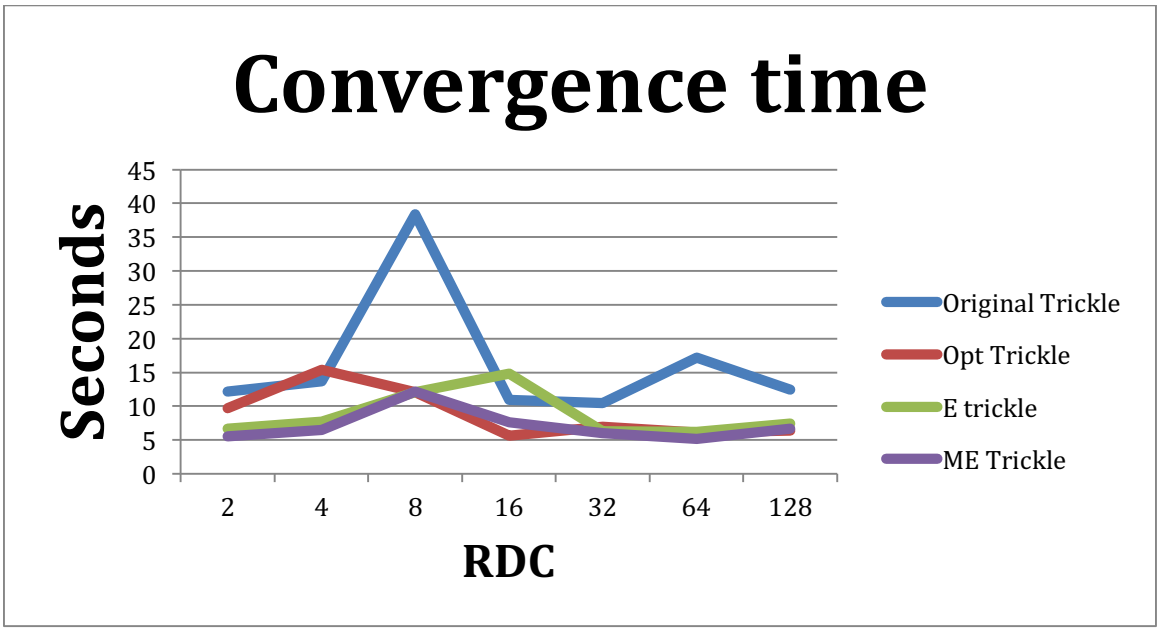


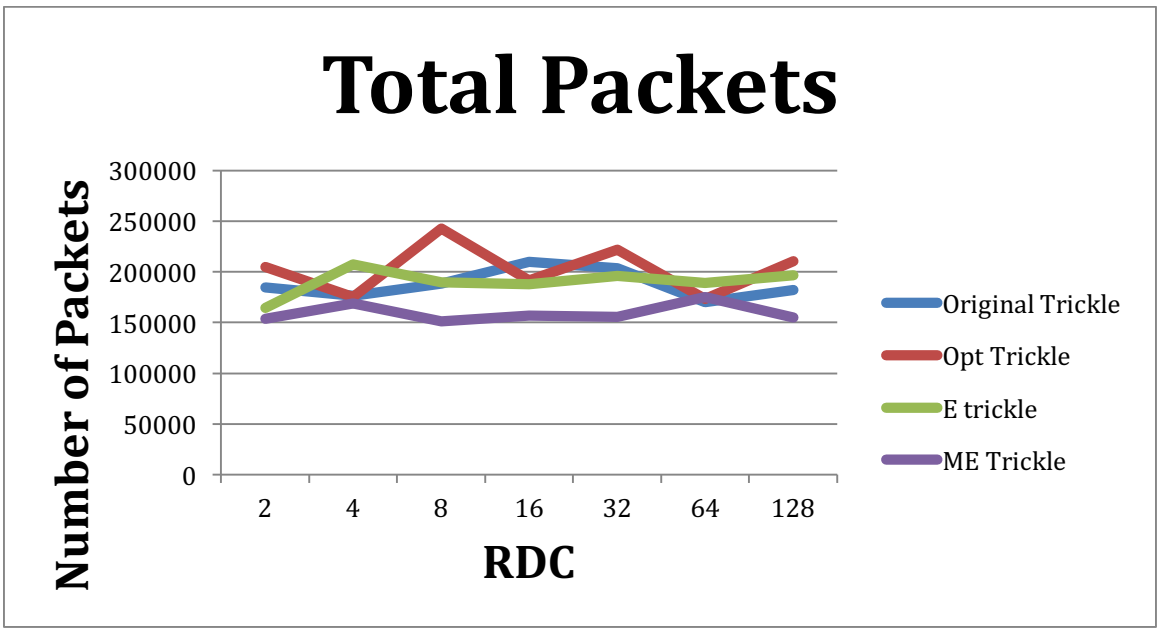

Latency
Seconds
0
1
2
3
4
5
6
7
6
7
8
9
10
11
12
13
14
15
16
DIO Interval Minimum
Original Trickle
Opt Trickle
E trickle
ME Trickle

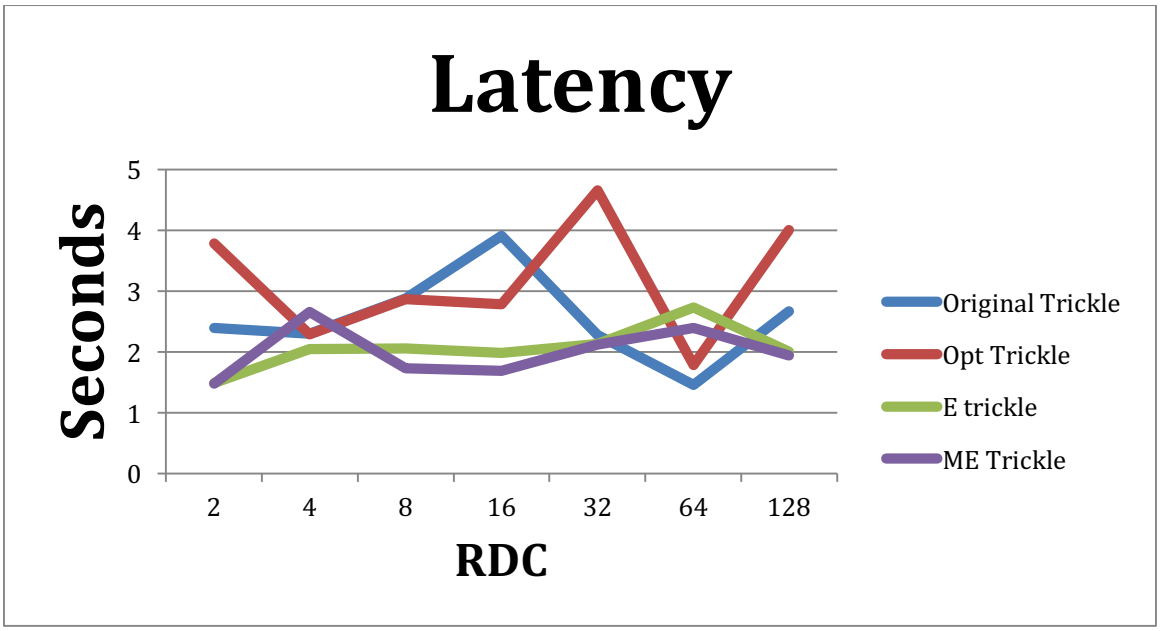

Delivery Ratio
Percentage
0
20
40
60
80
100
120
6
7
8
9
10
11
12
13
14
15
16
DIO Interval Minimum
Original Trickle
Opt Trickle
E trickle
ME Trickle

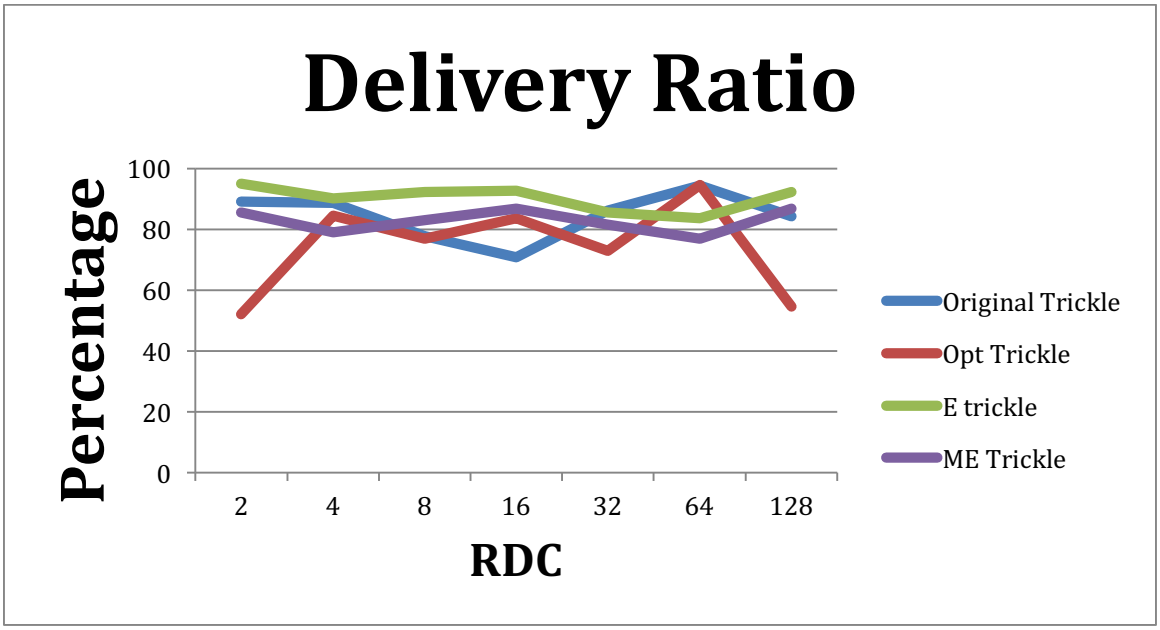


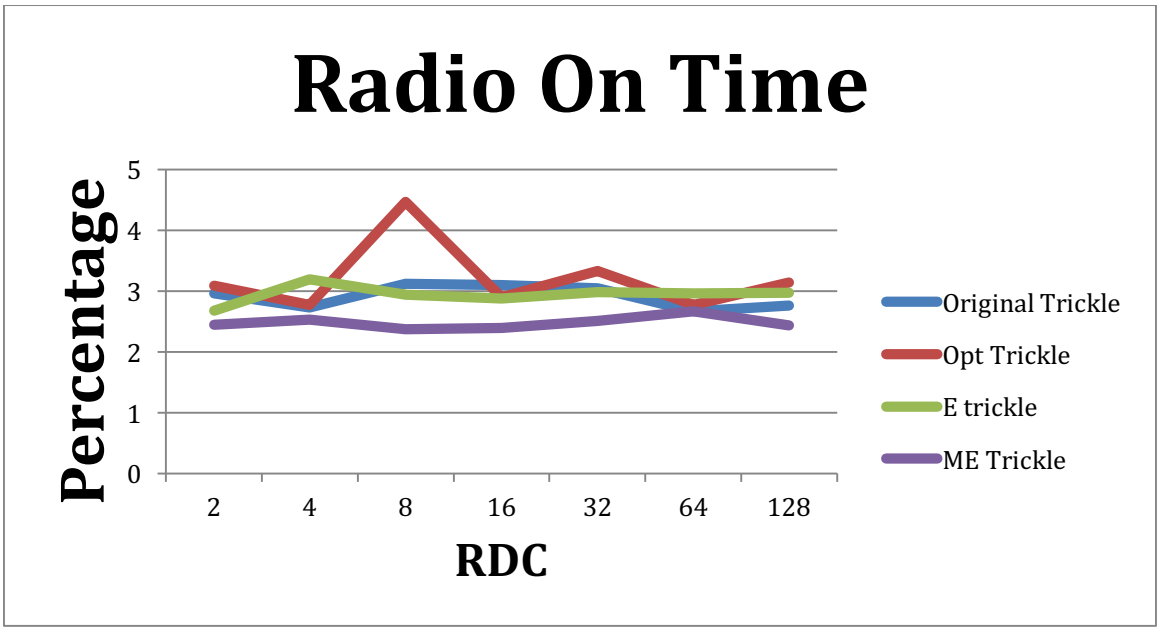
A.7 RDC Check Rate Test Data




Total Packets
Number of Packets
300000
250000
200000
150000
100000
50000
0
2
4
8
16
32
64
128
RDC
Original Trickle
Opt Trickle
E trickle
ME Trickle


Latency
Seconds
5
4
3
2
1
0
2
4
8
16
32
64
128
RDC
Original Trickle
Opt Trickle
E trickle
ME Trickle


Delivery Ratio
Percentage
100
80
60
40
20
0
2
4
8
16
32
64
128
RDC
Original Trickle
Opt Trickle
E trickle
ME Trickle


Radio On Time
Percentage
5
4
3
2
1
0
2
4
8
16
32
64
128
RDC
Original Trickle
Opt Trickle
E trickle
ME Trickle