A Performance Evaluation of RPL with Variations of the Trickle Algorithm

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

Zhouxiao Wu

Date: 1/28/2016
Project Advisor:

Professor Robert Kinicki, Advisor

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 delved into the performance of RPL. It focused on evaluating the performance of four variations of the Trickle timer algorithm, which is a key element in RPL's functionality, and on the performance of RPL based on multiple parameters of Trickle algorithm. The simulations show ME Trickle generally performs the best under the limited scenarios tested.

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 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 efficiency, accuracy and cost.

When many Wireless Sensor Networks (WSNs) are combined with the ability to keep data in the cloud, it is called Internet of Things (IoT), because each device includes at least one sensor to collect information about an object or area. The WSN is made of many devices called nodes. The nodes need to be as energy efficient as possible because they are usually not connected to power, and are intended to run for months or years on small, cost-effective batteries. The nodes do not communicate with the Internet individually, but instead 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 must be solved 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 sent from node to node. 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. 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 periodicallyly updating the neighbor nodes with the best known route, through broadcast messages. These messages create an overhead in the network of routing information packets.

This project focuses on analyzing and evaluating the performance of RPL in different traffic conditions and network topologies. In order to accomplish this, the team varied parameters and algorithms within RPL that control how the protocol functions, and then by varying the environment in which the protocol runs. The project focuses specifically on the Trickle timer algorithm, which controls the volume of routing protocol traffic transmitted at each node in the network [6]. The issues that are important to this research are the energy efficiency of the nodes, the latency and reliability of messages traveling through the network, and the overhead introduced by the routing protocol messages.

The goal of this project was to deliver a performance evaluation analysis 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 the limited test scenarios.

2. BACKGROUND

This chapter introduces the vital information needed to understand the scope and purpose of this project, the tools and systems we used for our testing, 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. Developing and debugging software written for these types of networks can be very challenging. Cooja provides a 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 simple Python scripts to filter the resultant data into desirable performance metrics.

Another reason to use Cooja is that it provides a large amount of example code that can be used as a base for this project's implementation code. Additionally, Cisco has developed the Contiki IPv6 stack and it is fully certified under the IPv6 Ready Logo program.

2.2. RPL

The Routing Protocol for Low Power and Lossy Networks (RPL) is made specifically for Wireless Sensor Networks (WSN) in the Internet of Things. It operates on the layer above the IEEE IPv6 6LoWPAN above802.15.4 wireless networking standard. 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 a routing protocol, which populates a routing table at each node. The routing protocol 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 the all the sensor nodes. It does not consider power usage because it will be connected to 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 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 broadcast this patch and it would fo through the network to target nodes.

RPL works in two directions from the border router: upward routing transmitting packets towards the border router from the leaf 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 will be listening for new information and how often it will be sending out its current information to its neighbor nodes. This project evaluates four algorithms for the Trickle timer. Section 2.5 discusses these algorithms and their 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 is powered. 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, 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 be successfully received at 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 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 found new possible variations. There is initial evidence that these new variations could out-perform the original Trickle Algorithm in certain cases and situations [3] [4].

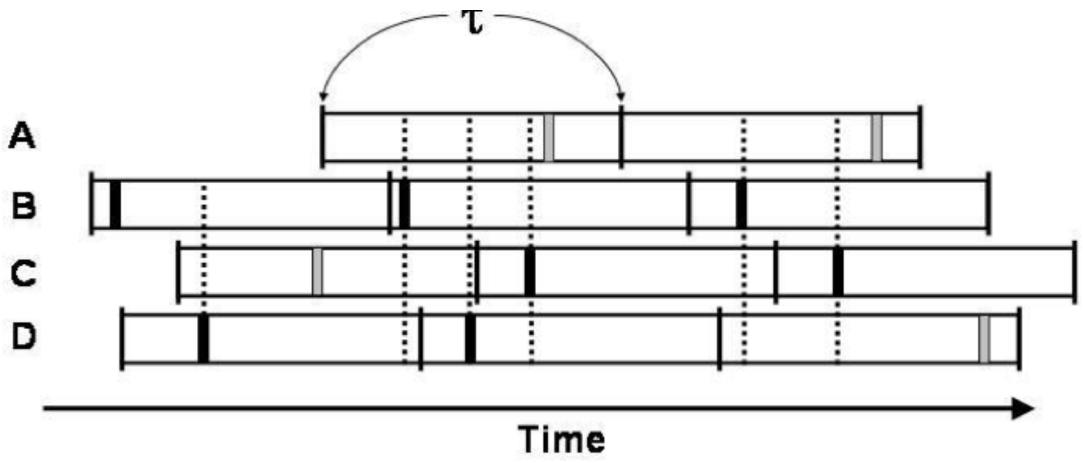


Figure 2: The Short The Short Listen Problem for Motes A, B, C, and D

Dark bars represent transmissions, light bars suppressed transmissions, and dashed lines are receptions. Tick marks indicate interval boundaries. Mote B transmits in all three intervals [7].

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 in half of an interval to one interval. Hence, the first half of a Trickle interval became the listen-only period.

Ghaleb has come up with a very good 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).

3) Upon receiving a consistent message, Trickle increments its counter 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.

Original Trickle

```
I. Initialization
    $I \leftarrow I_{min}$
II. Start New Interval
    $I \leftarrow I \times 2$
    $c \leftarrow 0$
    if $Imax \leq I$ then
        $I \leftarrow Imax$
    end if
    $t \leftarrow random[I/2, I)$
III. Received Consistent Transmission
    $c \leftarrow c + 1$
IV. Received Inconsistent Transmission
    $I \leftarrow I_{min}$
```

```
V. Random Timer Expires
        if $c < k_n$ then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO
        end if
```

Figure 2: Original Trickle pseudo-code Adapted from [4]

The one major difficulty that the Trickle algorithm encounters is the short-listen problem [7]. There would not be an issue if Trickle synchronized every node but since this is not the case, redundant messages will be sent. This is because certain node's messages will not reach every node in the network. Therefore, those that always multicast first will use too much energy. Greater energy consumption is very detrimental in Wireless Sensor Networks because of the limited resources available to the device. Figure 1 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 first gossip will quiet everyone else. However, if not synchronized, it might be that a mote's interval begins just after the broadcast, and it too has chosen a short listening period. This results in redundant transmissions" [8].

The listen-only period made Trickle scale logarithmically with network density, however, it increased delays at the same time. 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, the researchers proposed an optimized version of Trickle (referred to as opt-Trickle in the rest of the paper), 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.

The following is the pseudo-code of optimized Trickle. Plus signs indicate the difference between optimized algorithm and the Original Trickle.

Optimized Trickle

```
  I. Initialization
          $I \leftarrow I_{min}$
  II. Start New Interval
          $I \leftarrow I \times 2$
          $c \leftarrow 0$
          if $I_{max} \le I$ then
                  $I \leftarrow I_{max}$
          end if
+         if from step 6
+                 $t \leftarrow random[0, I_{min})$
+         else
                  $t \leftarrow random[I/2, I)$
+         end if
  III. Received Consistent Transmission
          $c \leftarrow c + 1$
  IV. Received Inconsistent Transmission
          $I \leftarrow I_{min}$
  V. Random Timer Expires
```

```
if $c < kn$ then
    Transmit Scheduled DIO
else
    Suppress Scheduled DIO
end if
```

Figure 3: 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 [7]. Additionally, opt-Trickle still has the listen-only period, and would lead to increased latency, especially in a lossy network. Thus a Levin[7] 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, researchers [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:

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

Figure 4: 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, its 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.

E-Trickle

```
  I. Initialization
          $I \leftarrow I_{min}$
+         $c \leftarrow 0$
  II. Start New Interval
          $I \leftarrow I \times 2$
−         $c \leftarrow 0$
          if $I_{max} \leq I$ then
                  $I \leftarrow I_{max}$
          end if
+         $t \leftarrow random(0 , I)$
−         $t \leftarrow random[I/2, I)$
  III. Received Consistent Transmission
          $c \leftarrow c + 1$
  IV. Received Inconsistent Transmission
          $I \leftarrow I_{min}$
+         $c \leftarrow 0$
  V. Random Timer Expires
+         if $I_{nz} > I$
+                 $kn \leftarrow ( k \times ( 2 \times Inz - I )) / I$
  +       else
  +               kn←k
  +       end if
            if c < kn then
                  Transmit Scheduled DIO
          else
                  Suppress Scheduled DIO
          end if
  +       c ← 0
```

Figure 5: E-Trickle pseudo-code Adapted from [4]

Finally, E-Trickle’s authors pointed out a possible modification for their Trickle version which seemed 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 indicates the difference between ME-Trickle and E-Trickle. The difference is jumping to I_{max} immediately rather than doubling the interval.

ME-Trickle

```
  I. Initialization
        $I \leftarrow I_{min}$
+       c ← 0
  II. Start New Interval
```

```
#−      $I \leftarrow I \times 2$
#+      $I \leftarrow I_{max}$
−       $c \leftarrow 0$
        if $I_{max} \leq I$ then
                $I \leftarrow {}_{Imax}$
        end if
+       $t \leftarrow random(0 , I)$
−       $t \leftarrow random[I/2, I)$
III. Received Consistent Transmission
        $c \leftarrow c + 1$
IV. Received Inconsistent Transmission
        $I \leftarrow I_{min}$
+       $c \leftarrow 0$
V. Random Timer Expires
+       if $I_{nz} > I$
+               $kn \leftarrow ( k \times ( 2 \times I_{nz} - I )) / I$

+       else
+               $kn \leftarrow k$
+       end if
          if $c < kn$ then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO
        end if
+       $c \leftarrow 0$
```

Figure 6: ME-Trickle pseudo-code

The Trickle algorithm variations with pseudo-code and step-by-step breakdown on how the variables work and change are detailed in Appendix A. The source code for the four Trickle algorithms are available online.

3. METHODOLOGY

This project studied the effects of four different Trickle algorithms using the Cooja simulator, in addition to varying several different parameters within RPL and Contiki, specifically DIO minimum interval, DIO doubling, radio duty cycling check rate, and frequency of application messages [1].

3.1 Testing Configuration and Network Topologies

3.1.1 Parameters

This investigation varied the parameters in Table 1 to provide a thorough examination of RPL performance within the Contiki OS. They consist of network stack parameters and Cooja simulator parameters. Each variation of the algorithm used these parameters to perform a series of simulations to isolate a single varied parameter and study the effects of the parameter on the algorithms performance.

Table 1 RPL Parameters and Locations in Contiki and Cooja

Parameters	**Names in Contiki**	**Locations**
RPL Mode of Operation	RPL_MOP_DEFAULT	rpl-private.h
RPL Objective Function	RPL_OF	rpl-conf.h
DIO Min	RPL_DIO_INTERVAL_MIN	rpl-conf.h
DIO Doubling	RPL_DIO_INTERVAL_DOUBLINGS	rpl-conf.h
RDC Channel Check Rate	NETSTACK_RDC_CHANNEL_CHECK_RATE	netstack.h
Send Interval	SEND_INTERVAL	udp-client.c
Reception Ratio	RX Ratio	Cooja
Transmission Ratio	TX Ratio	Cooja
Transmission Range	TX Range	Cooja
Interference Range	INT Range	Cooja
Start Delay	Mote startup delay	Cooja

The testing focused on isolating each one of the parameters to find their effects on the behavior and performance of the four algorithms. One series of tests would vary a single parameter while all of the other parameters remained constant. The test 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. Originally, 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 parameters for the remainder of the tests for consistency and because they represented reasonable and realistic networks.

As the ETX objective function is generally acceptedas the industry standard, all simulations employed ETX.

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

Receive rate	70%
Transmission range	50 meters
Interference range	55 meters
Testing Area	100 meters by 100 meters

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 performance results. Since the tests run for fifteen minutes produced results similar to those run for 20 minutes, we chose fifteen minutes of simulated time as the standard 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. Figure 7 depicts the original layout with the nodes randomly placed.

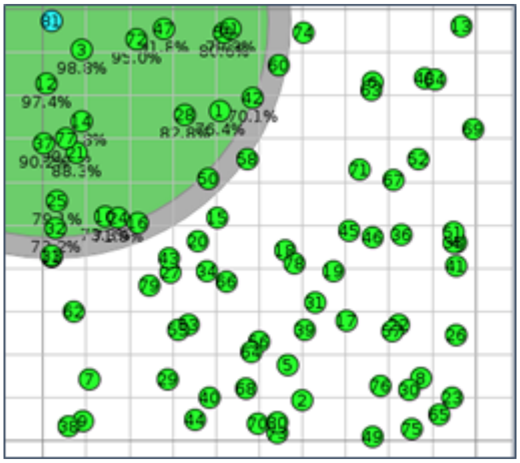


Figure 7: Random topologies with 80 client nodes

The majority of the testing used the above layout to facilitate comparisons toAli’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 topologies for these tests. For each of these sets of simulation tests, the team decided to place 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.

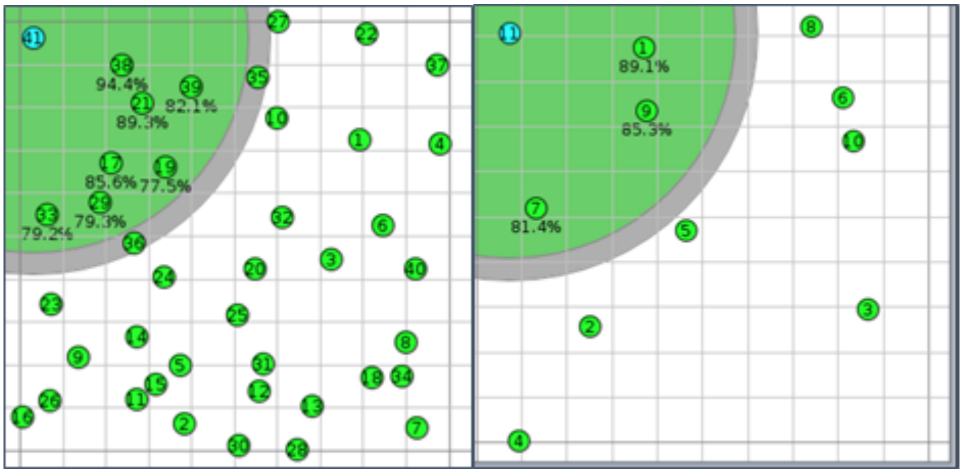


Figure 8: Random topology with 40 client nodes and 10 nodes

After running the random topology, 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 9 illustrates the set up for the grid layouts As previously done, all configurations placed the border router node in the upper left corner of available area to create more interesting results.

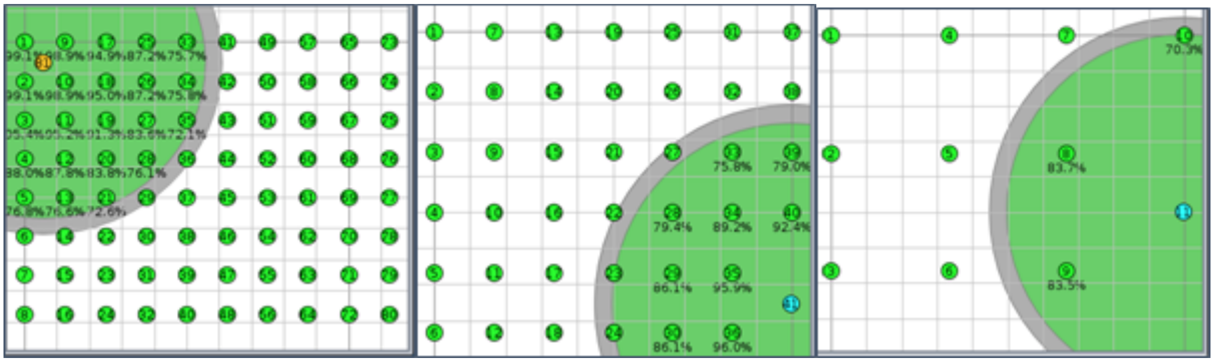


Figure 9: Grid topology with 80, 40, and 10 client nodes

Finally, the team tested a topology which would require a larger number of hops. This meant creating a linear layout of 10 client nodes (see Figure 10). The area is different than in previous simulation such that the nodes can only talk to their adjacent neighbors. Thus the configuration places the nodes venly spaced 40 meters apart with the BR node in the middle.

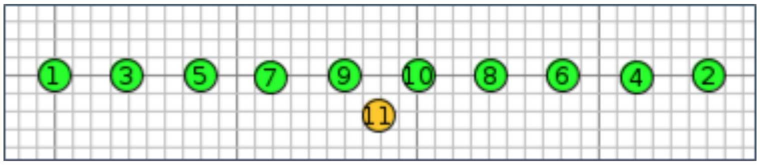


Figure 10: Linear topology with 10 client nodes

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 a user-defined interval. The team used the simulator to load each node with the application and to send a message to the border router at the defined time interval for the duration of the fifteen minute test. 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 the exact same configurations as Ali’s master’s thesis. 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 simulates studied in this research

The team modified the RPL source code in Contiki to write the four Trickle algorithms described in the background chapter. The changes made modifications to the rpl-timers.c file (see Appendix 1). The team replaced 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).

Table 3 Description of Performance metrics

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

The setup time of the network is the amount of time needed for each 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 that they joined the DAG to the simulator output.

The total number of packets is the combined number of application layer messages sent from the client nodes to the server and the RPL routing messages. 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 data points down by a constant value. Cooja tracks these statistics.

The percentage of delivered application packets is the number of successful "hello" 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 averaged all of the delivered percentage.

Radio-on time is the percentage of the simulated fifteen 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 sink 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. 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 highlight each of the performance metrics on which data were collected, as discussed in the previous section. The following results section contains the most interesting results and graphs of our performance metrics for the simulations the team performed. The full set of performance graphs can be found in Appendix B.

4. RESULTS

This chapter discusses and analyze only one set of results for the four Trickle timer algorithms simulating RPL over the Contiki OS with the Cooja simulator. The chapter utilizes the performance metrics detailed in the prior section to provide graphs designed to demonstrate the differences in the Trickle algorithms only for the 80 node simulations. The reader is referred to Daniel Benson's MQP report [2] for the complete presentation and analysis of all the simulations run for this investigation. The remaining set of visualizations are in Appendix B.

4.1 DIO Minimum Interval

The first important RPL parameter iss the DIO Minimum interval. This is the minimum possible interval value in the original Trickle algorithm. 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. DIO Minimum values between two and five provided weak performance results. Thus, this analysis does not include these specific simulated results..

Figures 11-13 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 after testing the values from six to sixteen. All other RPL and Trickle parameters were set to default values.

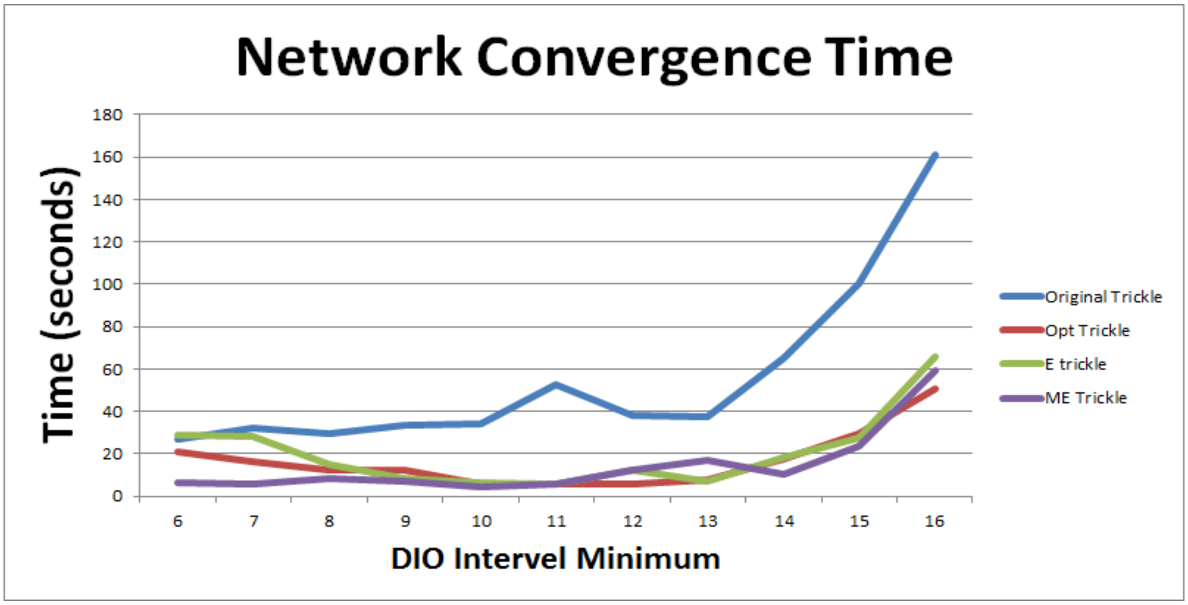


Figure 11: Network Convergence Time versus DIO Minimum

Figure 11 shows that the three new algorithms yield better convergence times than the original Trickle algorithm. ME-Trickle tends to be the best for most DIO minimum choices. The best DIO values are in the range from nine to twelve. The convergence time is much longer as the value of the minimum DIO interval increases because it waits longer to resend if a message fails. Conversely, with a shorter interval the network takes much less time to converge. Network convergence time does not affect the performance significantly after the first few minutes, but it does have a negative effect on the radio-on power usage.

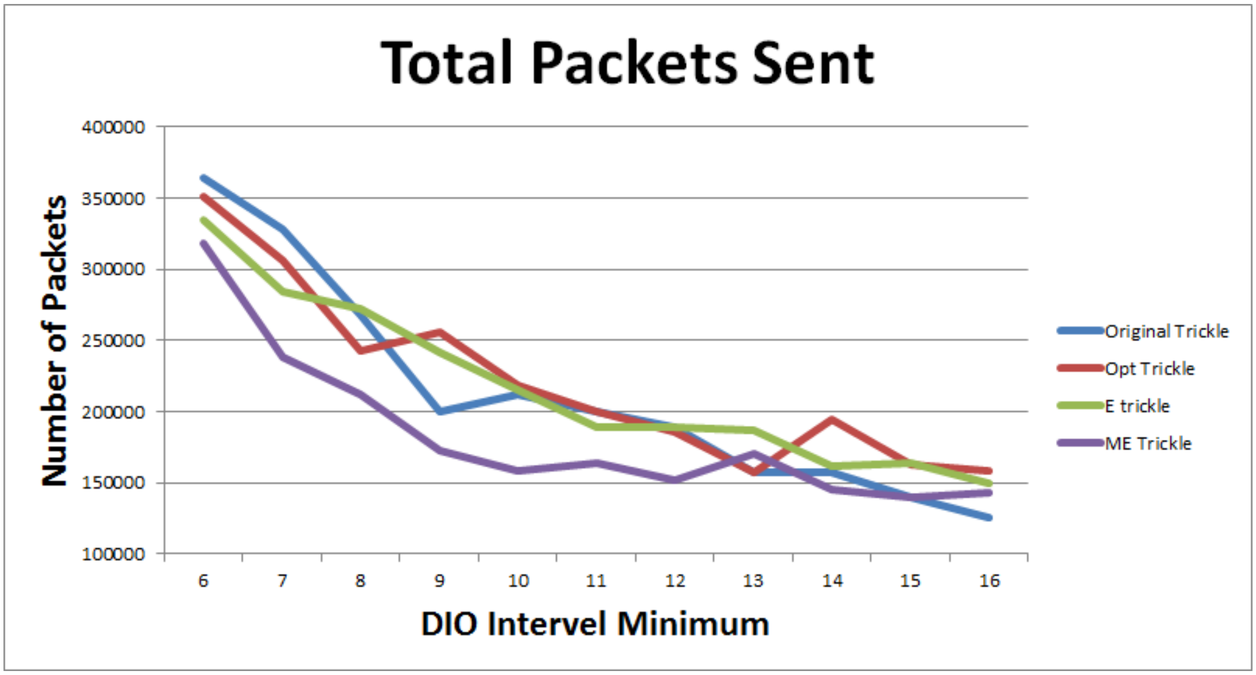


Figure 12: Total Number of Packets versus DIO Minimum Interval

Figure 12 compares the total number of packets sent by the four Trickle variants over the entire fifteen minute simulation as DIO Minimum varies. This includes 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 a larger number of packets are sent, there are much largerer send queues at each of the sensor nodes and many more packet collisions due to transmission interference. This causes many nodes to issue multiple resends before a packet is received successfully. This in turn has a very negative effect on the number of packets sent, as well as all of the other performance metrics measured.

Across all of the values, ME-Trickle continues to maintain the best performance. This is due to the maximization of the interval after a single consistent interval. This trend continues through the other performance metrics.

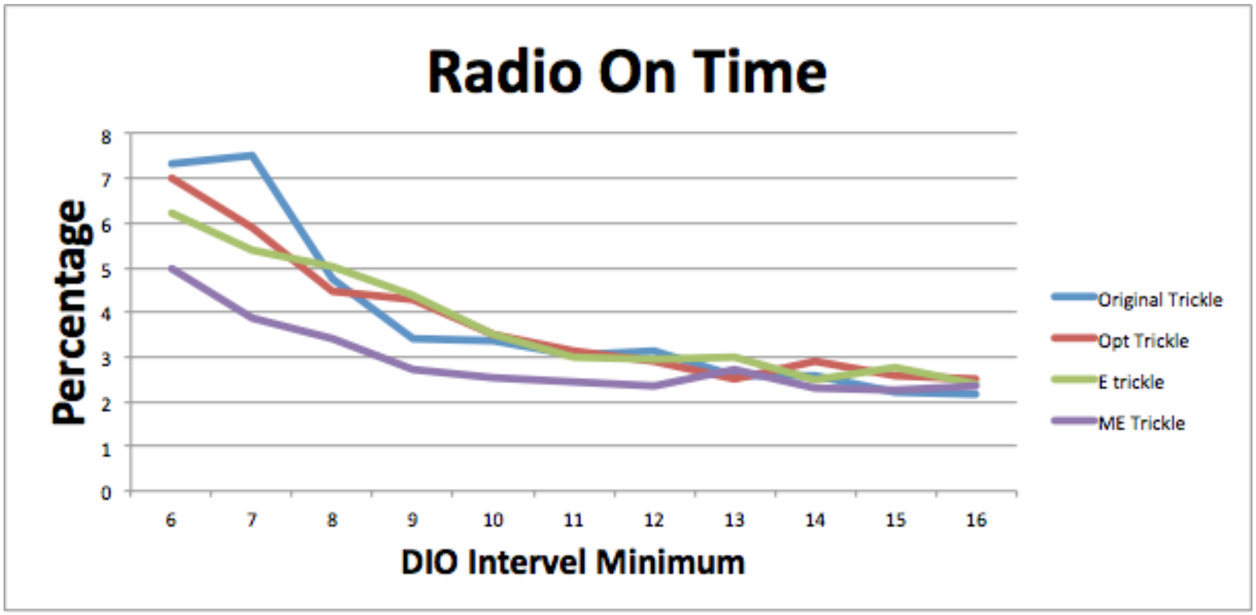


Figure 13: Radio On Time versus DIO Minimum Interval

As shown in Figure 13, the average percentage of radio on time has a direct correlation to the number of packets sent. Due to the large number of packets being sent for the lower DIO Miminum intervals, there is a large increase in the percentage of radio on time. While the radio on percentage is still rather low at around 7% for a DIO Minimum interval of six, it is large in comparison, nearly twice the value, of the larger interval values. As with the previous performance metrics, ME-Trickle has consistently better performance than the other algorithms.

5. CONCLUSIONS

Over the course of the two months of the MQP experiments, the team ran over 300 distinct tests for nearly 400 hours of simulations. Those simulations show ME Trickle generally performs the best under the limited test scenarios. The following section discusses possible next steps for the continuation of this research activity.

5.1 Future Work

Due to the time constraints on the project, there are a number of tests that the team was not able to run. Many things could be done to further evaluate the performances of the four algorithms. There was not time to vary some of the other important parameters. For example, all of tests used a fixed wireless receive probability of 70%, which is very low. Thus, more tests could be performed with higher receive probabilitiesto fully understand how they 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 to receive probability testing, 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 conclusive enough to draw strong, reasonable conclusions about the effectiveness of different values is various scenarios. These simulation results paired with varied packet receive probabilities could yield interesting conclusions.

Another subject of future work could focus on additional data collection. With the project's time limitations, the team was not able to collect data on the number of network hops that a packet takes on average. Although they were able to estimate this by judging the node layout, a more scientific stratefy would be to write a script to look into the packet headers to determine hop counts. When compared with other studied parameters, these simulations could provide more robust conclusions and more tests to back up the previously stated findings. Perhaps the best advancement of this project would be devising an automated mechanism for starting a series of Cooja simulations which facilitated changing the parameters in the source code in each successive test. This automation would allow future researchers to save testingtime 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.

In addition, 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 packet delivery ratio and lower setup time than the other algorithms. However, in general, ME-Trickle did not perform well in a network with a large number of hops. Therefore, the team would like to propose another optimization of the ME-Trickle algorithm.

The idea behind ME-Trickle is that the DIO interval jumps from minimum to maximum if there is no inconsistent transmission. However, this idea would not work very well in a larger network with more hops because there could be more inconsistent transmissions that will not be caught 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 will then jump to the maximum possible value. This proposed version of algorithm should be able to better handle high hop counts scenarios since it has a shorter response time to inconsistent data, while also maintaining low energy consumption by maximizing the DIO interval.

REFERENCES

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

[2] B. Daniel, “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 nad 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] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, “RFC 6206: The trickle algorithm,” *IETF*, 2011.

[8] 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

Original Trickle

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 counter c to 0, and assigns a randomly selected value in the interval to the variable t, chosen from the range [I/2, I).

3) Upon receiving a consistent message, Trickle increments its counter 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 message is transmitted.

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 Im_{ax}. Trickle sets the interval size I to Imax 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.

```
I. Initialization

        $I \leftarrow I_{min}$

II. StartNewInterval

        $I \leftarrow I \times 2$

        $c \leftarrow 0$
        if $I_{max} \leq I$ then
                $I \leftarrow I_{max}$
        end if
        $t \leftarrow$ random[I/2, I)
III. ReceivedConsistentTransmission
        $c \leftarrow c + 1$
IV. ReceivedInconsistentTransmission
        $I \leftarrow I_{min}$
V. RandomTimerExpires
         if $c < k_n$ then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO
        end if
```

Opt-trickle

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 counter c to 0, and if **assigns t randomly from**

[0, I_{min}): begins from step 6

[I/2, I): begins from step 5 or step 1

3) Upon receiving a consistent message, Trickle increments its counter 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 message is transmitted.

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 detected 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.

```
I. Initialization
        $I \leftarrow I_{min}$
II. StartNewInterval
        $I \leftarrow I \times 2$
        $c \leftarrow 0$
        if $I_{max} \leq I$ then
                $I \leftarrow I_{max}$
        end if
```

```
+       if from step 6
+               t ← random[0, $I_{min}$)
+       else
                t ← random[I/2, I)
+       end if
III. ReceivedConsistentTransmission
        c ← c +1
IV. ReceivedInconsistentTransmission
        I ← $I_{min}$
V. RandomTimerExpires
         if c < $k_n$ then
                Transmit Scheduled DIO
        else
                Suppress Scheduled DIO
        end if
```

E-TRICKLE

1) The algorithm begins 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} and **sets the counter c to 0.**

2) When an interval starts, the algorithm assigns a randomly selected value in the interval to the variable t chosen from the range **[0, I]**.

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

4) **At the randomly selected time t. Trickle increases the value of the redundancy factor k if needed.** And then if the counter c is greater than or equal to the redundancy parameter, k, Trickle suppresses its scheduled message. Otherwise the message is transmitted. **In both cases, the algorithm reset c to 0.**

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 (Imax) and re-executes the steps from step2.

6) Upon detecting inconsistent transmission, Trickle resets I to (I_{min}), if it was not already set to (I_{min}), resets c to 0 and starts a new interval as in step 2.

```
I. Initialization
        $I \leftarrow I_{min}$
+       $c \leftarrow 0$
II. StartNewInterval
        $I \leftarrow I \times 2$
−       $c \leftarrow 0$
```

```
        if $I_{max} \le I$ then
                $I \leftarrow I_{max}$
        end if
+       $t \leftarrow random(0 , I)$
−       $t \leftarrow random[I/2, I)$
III. ReceivedConsistentTransmission
        $c \leftarrow c +1$
IV. ReceivedInconsistentTransmission
        $I \leftarrow I_{min}$
+       $c \leftarrow 0$
V. RandomTimerExpires
+       if $I_{nz} > I$
+               $k_n \leftarrow ( k \times ( 2 \times I_{nz} - I )) / I$
+       else
+               $k_n \leftarrow k$
+       end if
         if $c < k_n$ then
                Transmit Scheduled DIO
```

```
          else
                    Suppress Scheduled DIO
          end if
+         $c \leftarrow 0$
```

ME-Trickle

1) The algorithm begins 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} and **sets the counter c to 0.**

2) When an interval starts, the algorithm assigns a randomly selected value in the interval to the variable t chosen from the range **[0, I]**.

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

4) **At the randomly selected time t. Trickle increases the value of the redundancy factor k if needed.** And then if the counter c is greater than or equal to the redundancy parameter, k, Trickle suppresses its scheduled message. Otherwise the message is transmitted. **In both cases, the algorithm reset c to 0.**

5) When the interval I expires, trickle **maximizes** 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 step2.

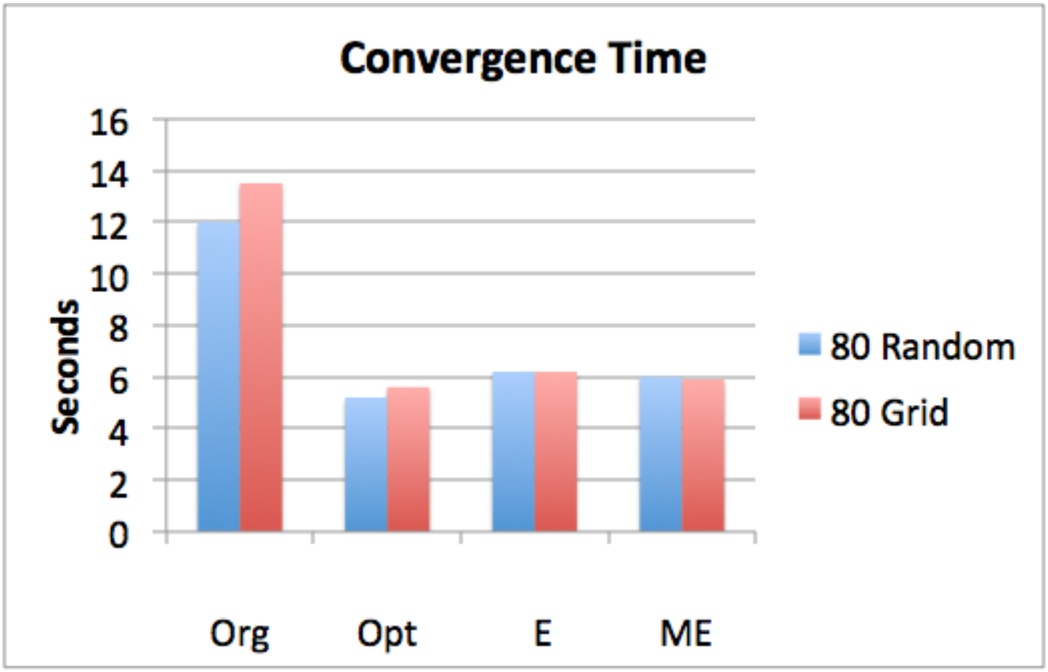
6) Upon detecting inconsistent transmission, Trickle resets I to (I_{min}), if it was not already set to

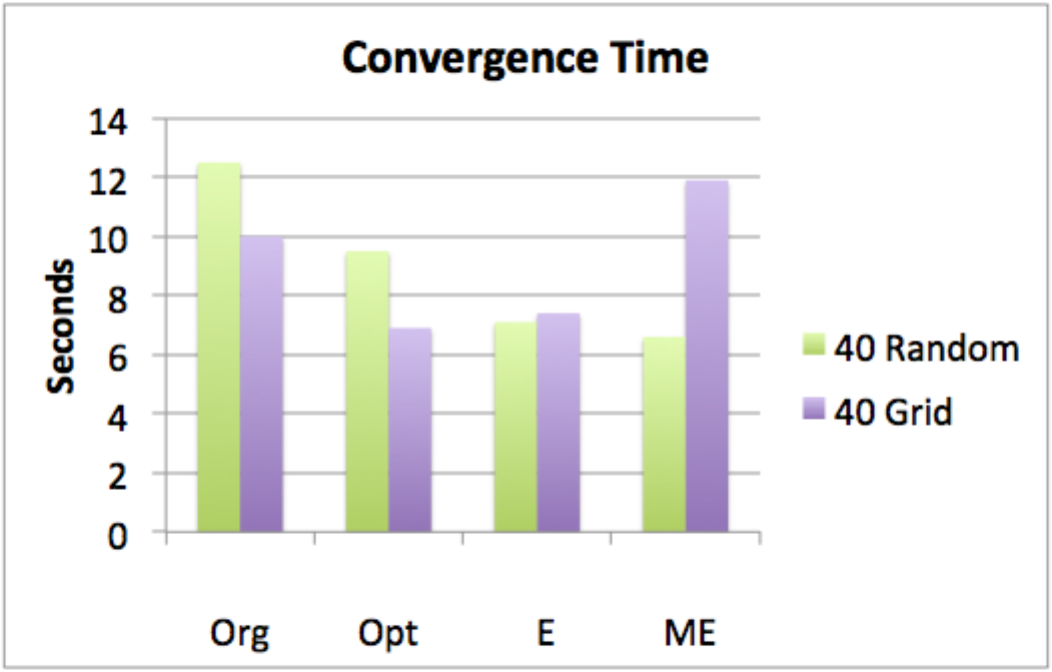
(I_{min}), resets c to 0 and starts a new interval as in step 2.

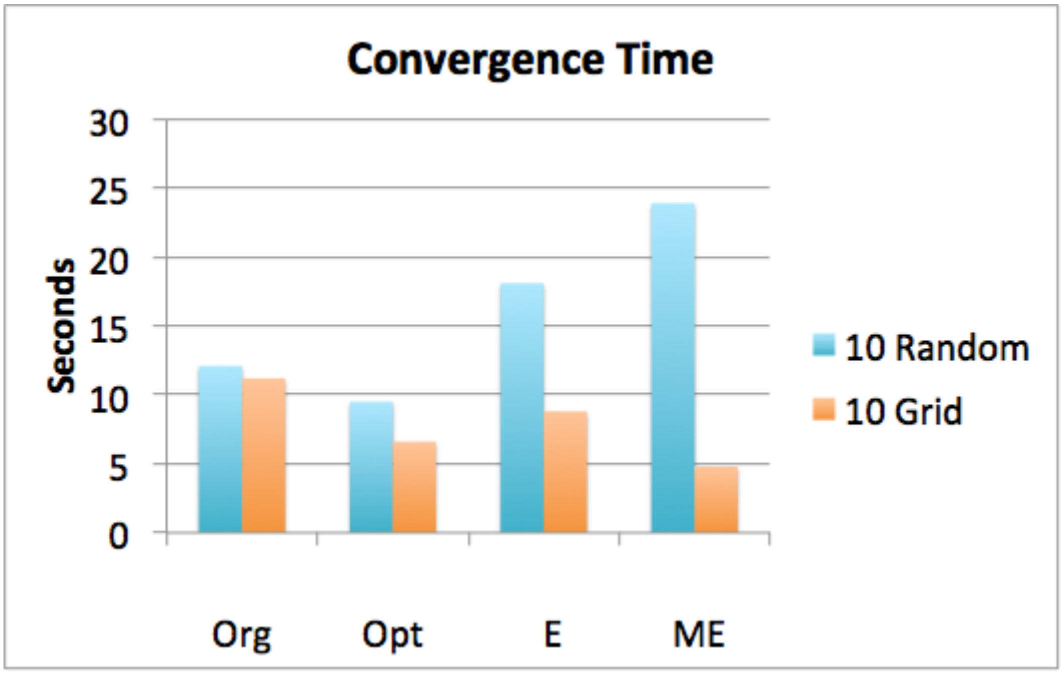
```
I. Initialization
        $I \leftarrow I_{min}$
+       $c \leftarrow 0$
II. StartNewInterval
−       $I \leftarrow I \times 2$
+       $I \leftarrow Imax$
−       $c \leftarrow 0$
        if $I_{max} \leq I$ then
                $I \leftarrow I_{max}$
        end if
+       $t \leftarrow random(0 , I)$
−       $t \leftarrow random[I/2, I)$
III. ReceivedConsistentTransmission
        $c \leftarrow c +1$
IV. ReceivedInconsistentTransmission
```

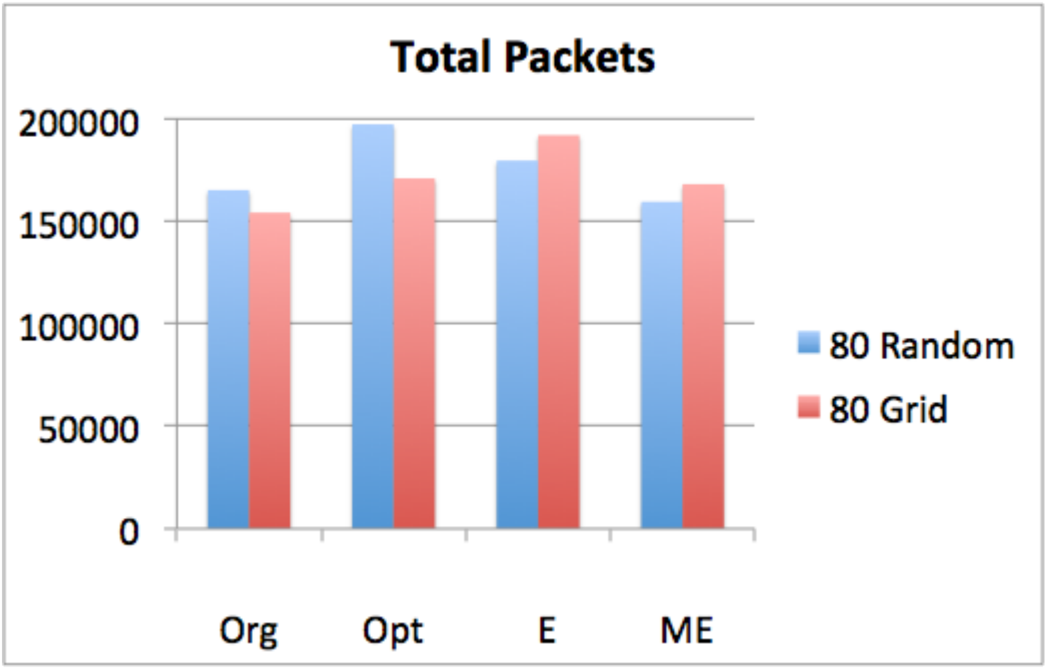
```
            $I \leftarrow I_{min}$
+           $c \leftarrow 0$
V. RandomTimerExpires
+           if $I_{nz} > I$
+                       $k_n \leftarrow ( k \times ( 2 \times I_{nz} - I )) / I$

+           else
+                       $k_n \leftarrow k$
+           end if
              if $c < k_n$ then
                        Transmit Scheduled DIO
            else
                        Suppress Scheduled DIO
            end if
+           $c \leftarrow 0$
```

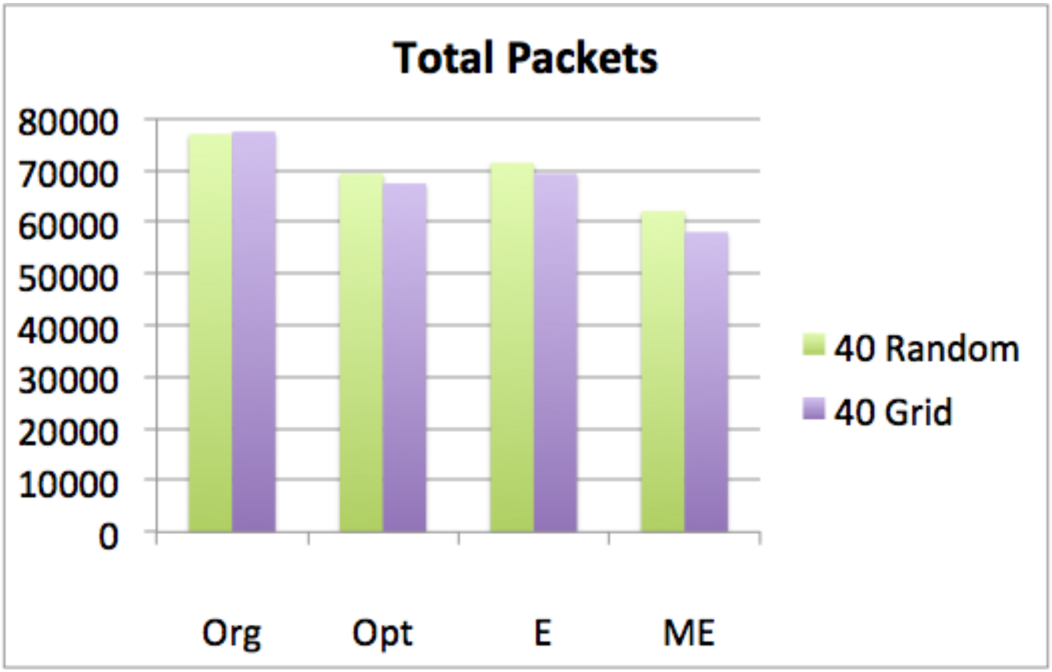
Appendix B

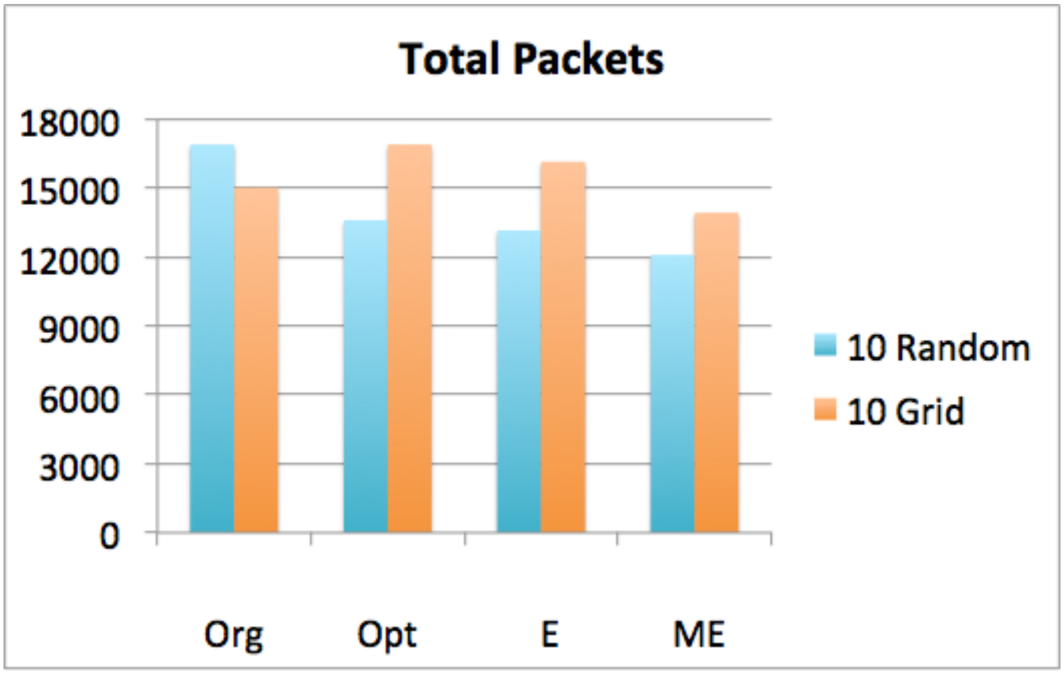

Convergence Time
Seconds
16
14
12
10
8
6
4
2
0
Org
Opt
E
ME
80 Random
80 Grid

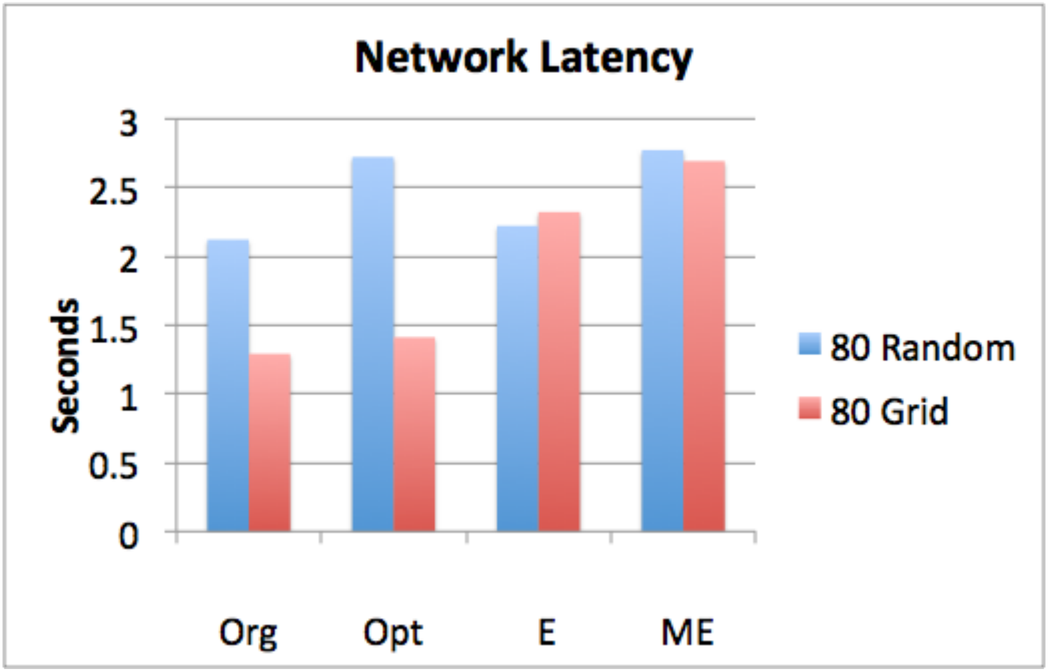

Convergence Time
Seconds
14
12
10
8
6
4
2
0
Org
Opt
E
ME
40 Random
40 Grid

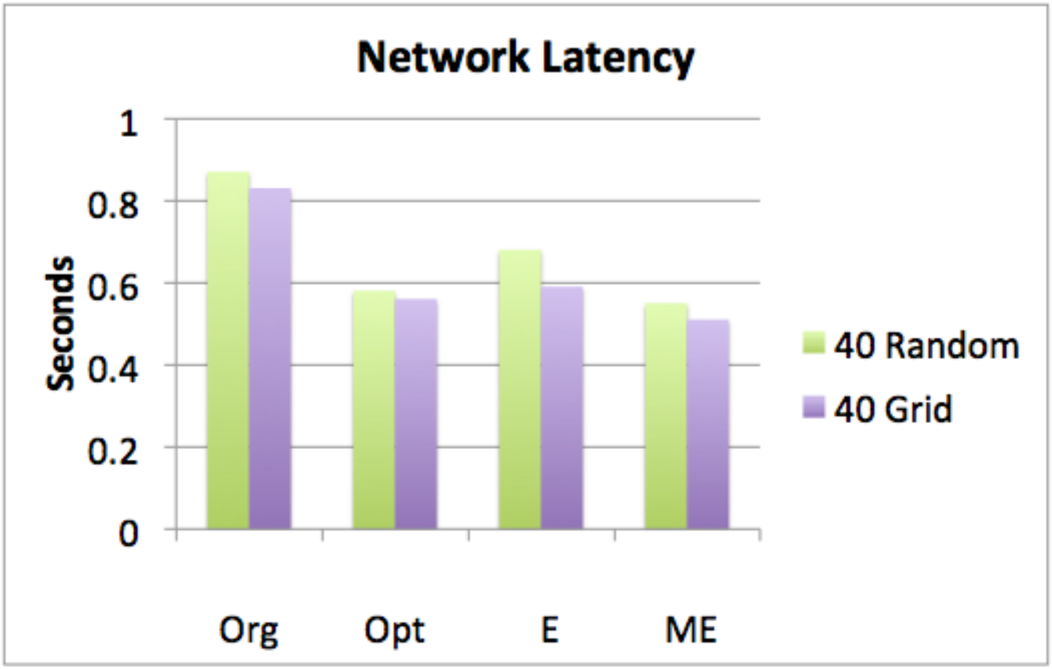

Convergence Time
Seconds
30
25
20
15
10
5
0
Org
Opt
E
ME
10 Random
10 Grid

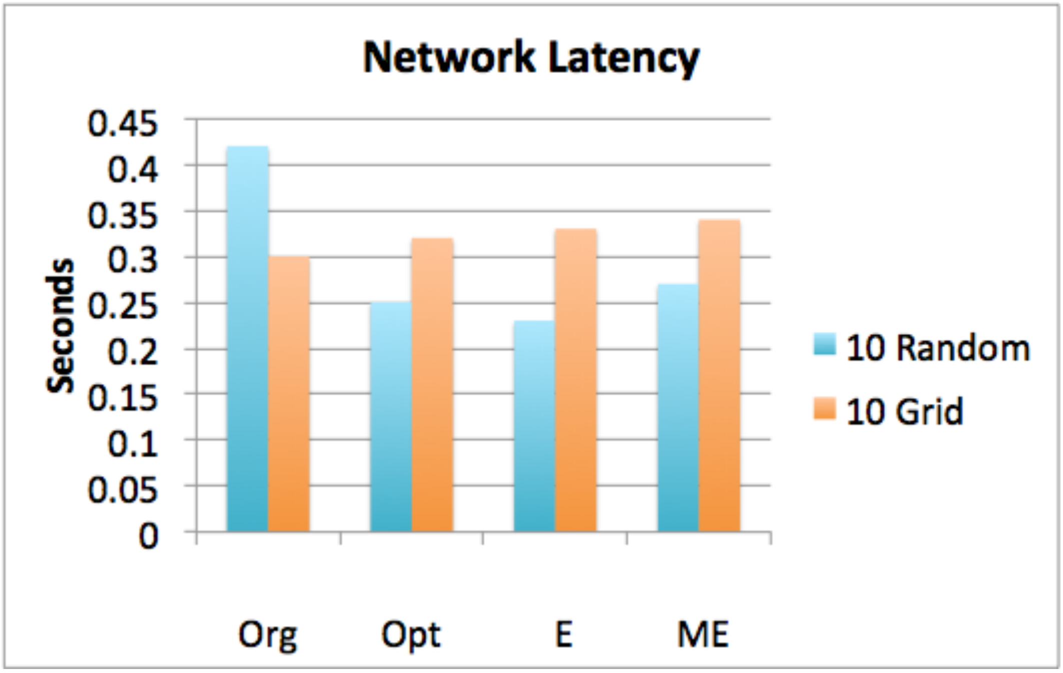

Total Packets
200000
150000
100000
50000
0
Org
Opt
E
ME
80 Random
80 Grid


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

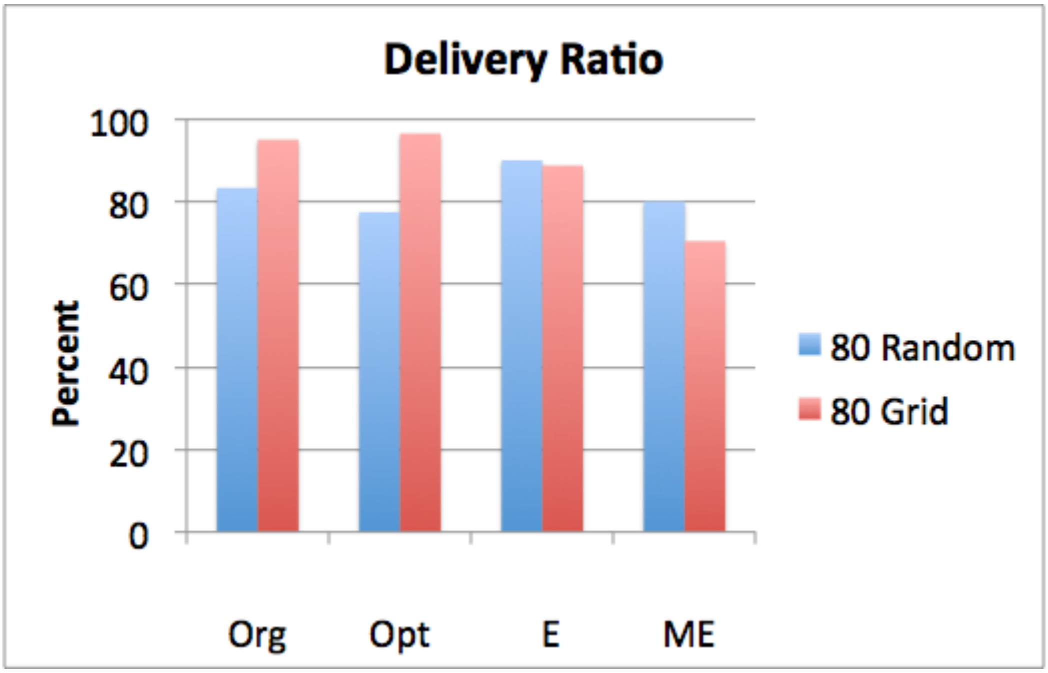

Total Packets
18000
15000
12000
9000
6000
3000
0
Org
Opt
E
ME
10 Random
10 Grid


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

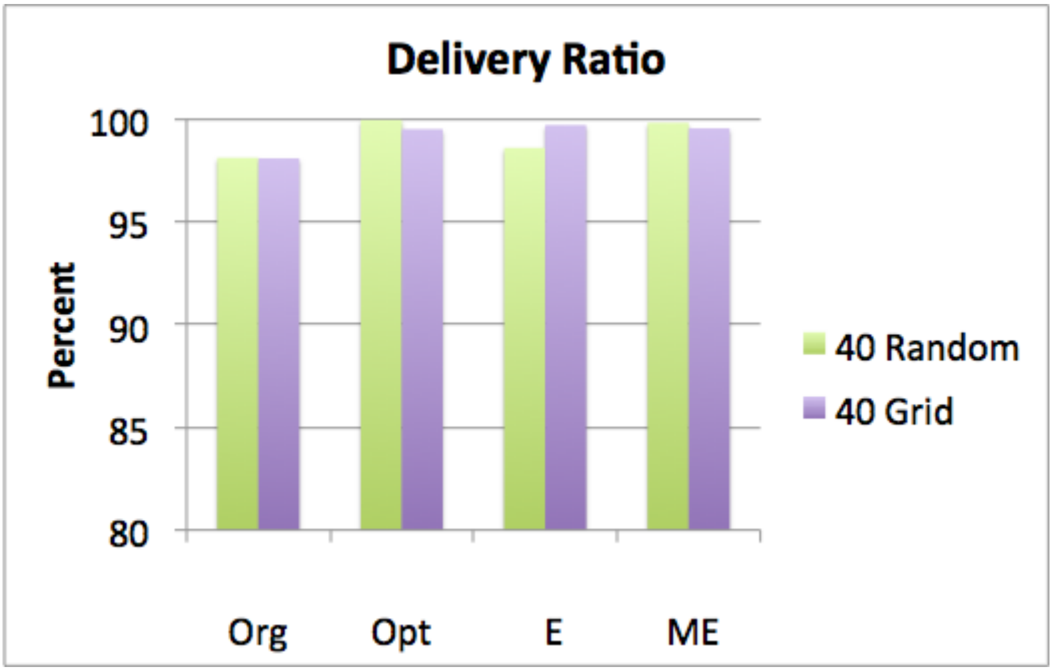

Network Latency
Seconds
1
0.8
0.6
0.4
0.2
0
Org
Opt
E
ME
40 Random
40 Grid

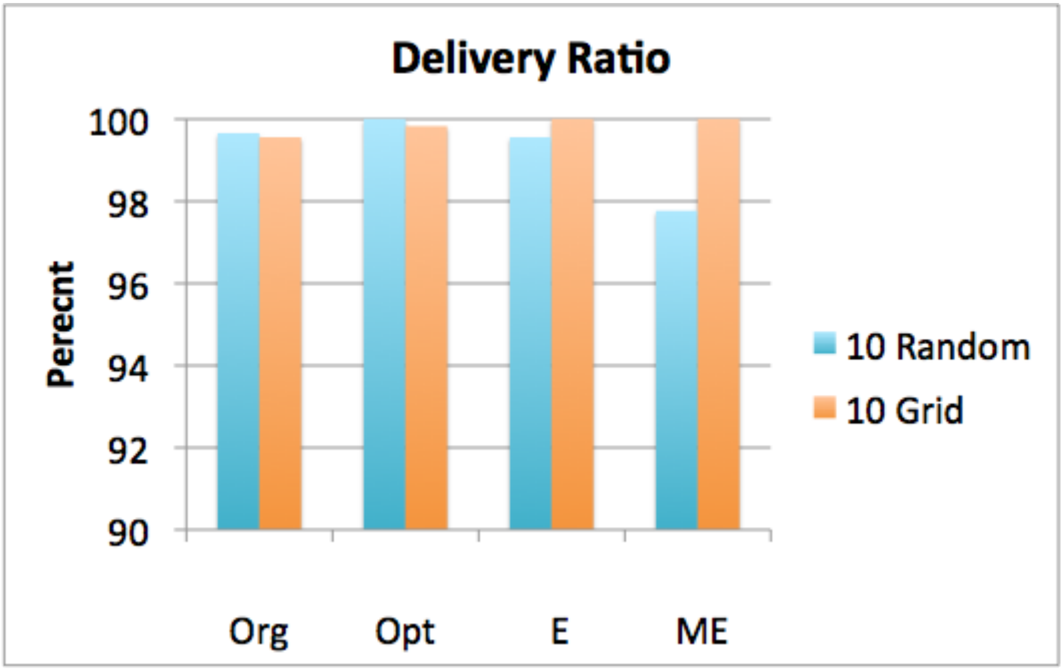


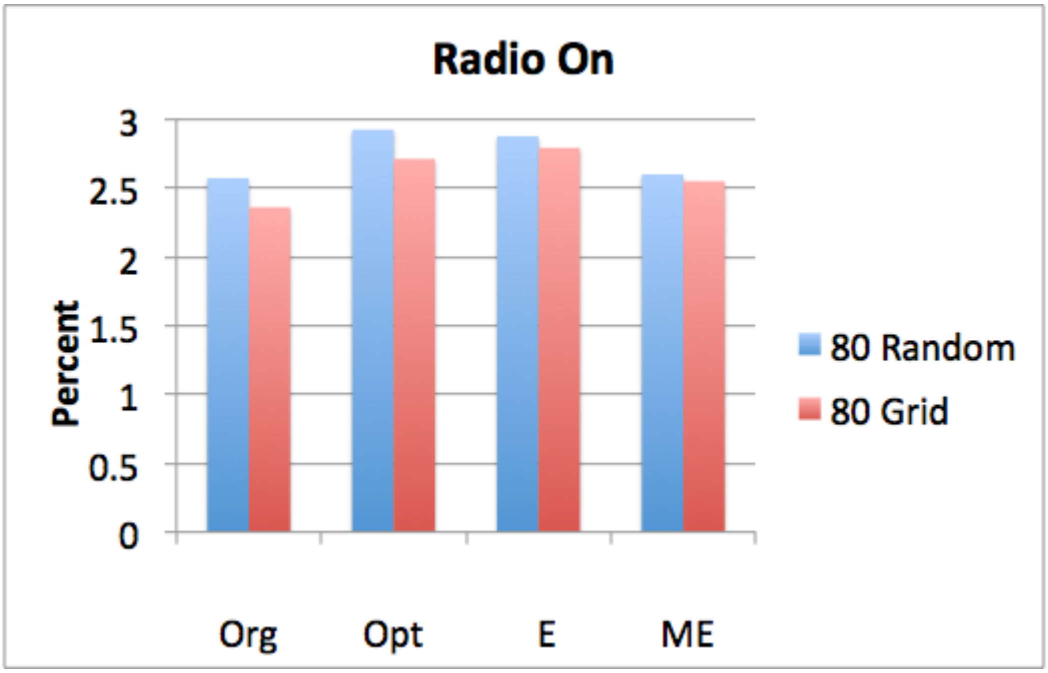
Network Latency

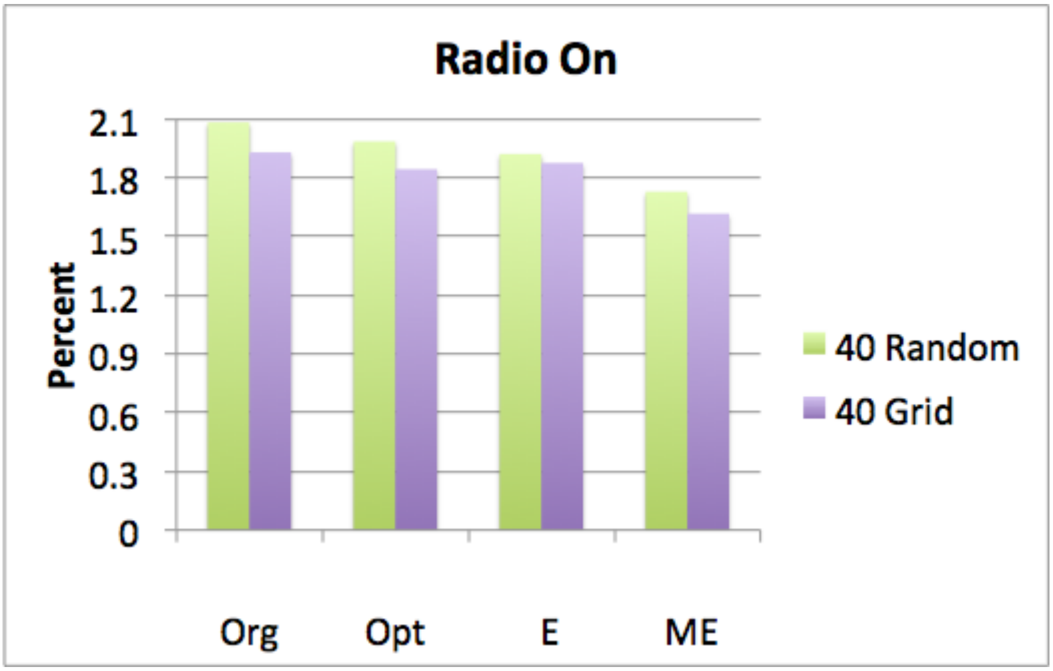


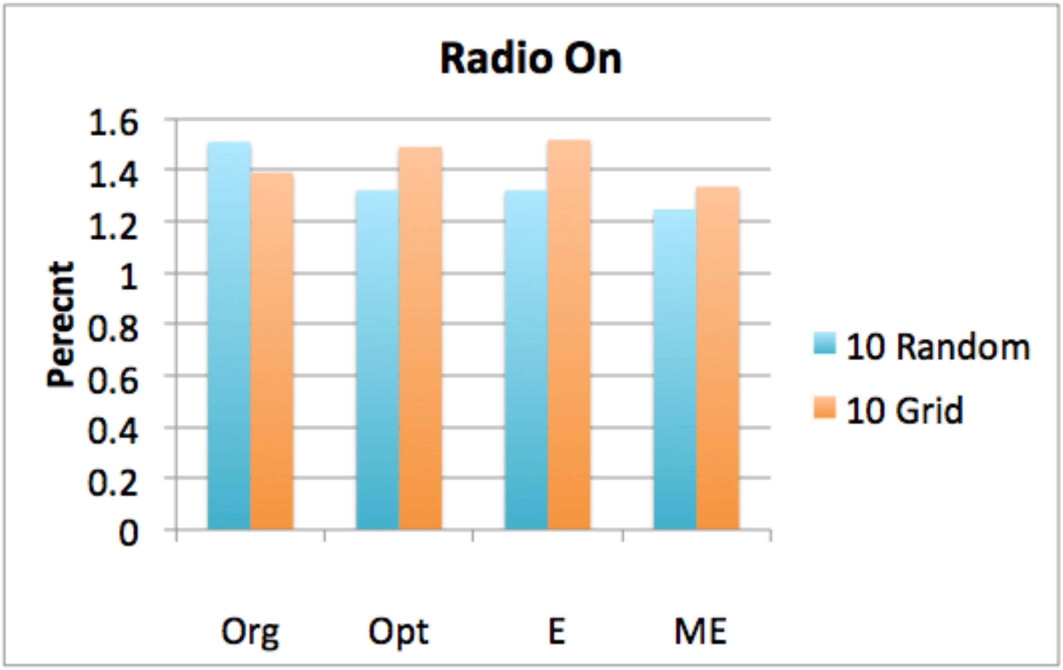
Delivery Ratio

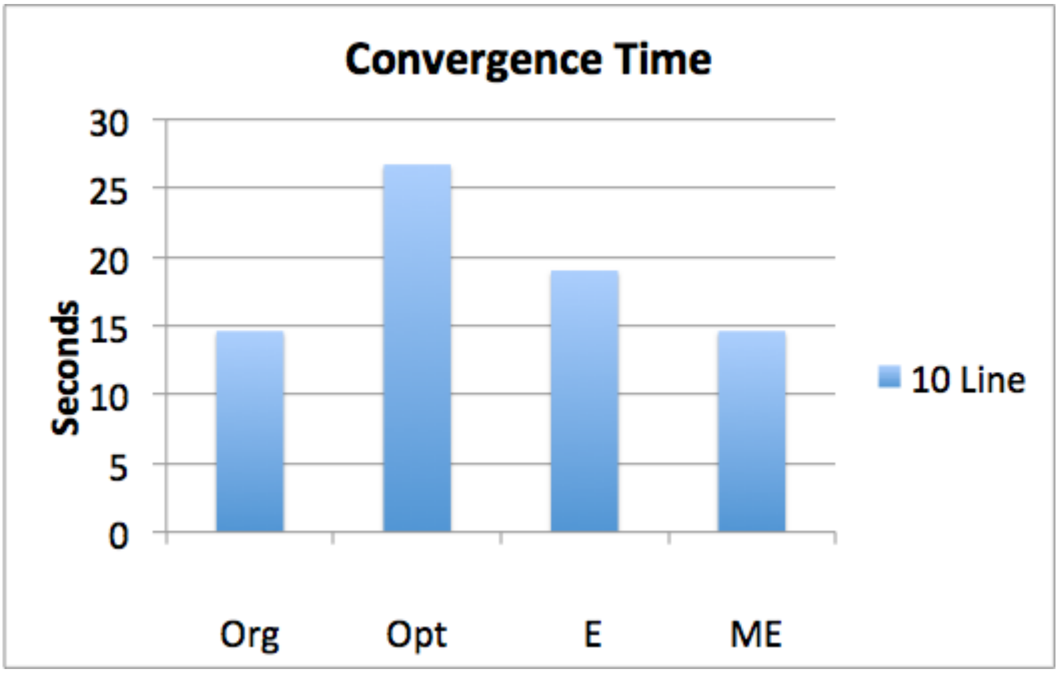

Delivery Ratio
Percent
100
95
90
85
80
Org
Opt
E
ME
40 Random
40 Grid

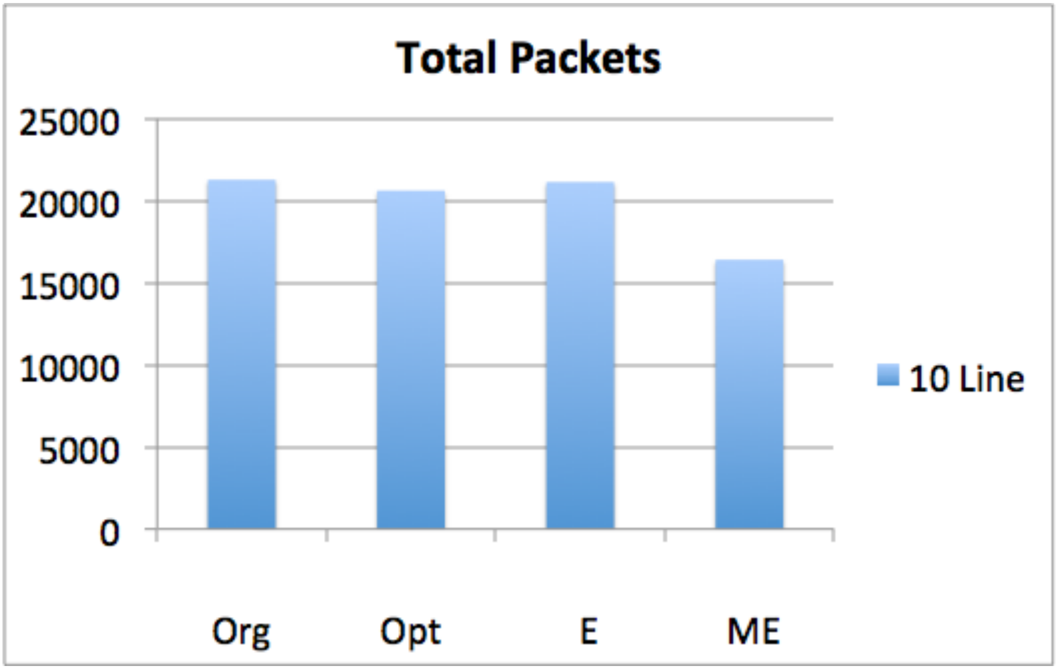

Delivery Ratio
Perecnt
100
98
96
94
92
90
Org
Opt
E
ME
10 Random
10 Grid

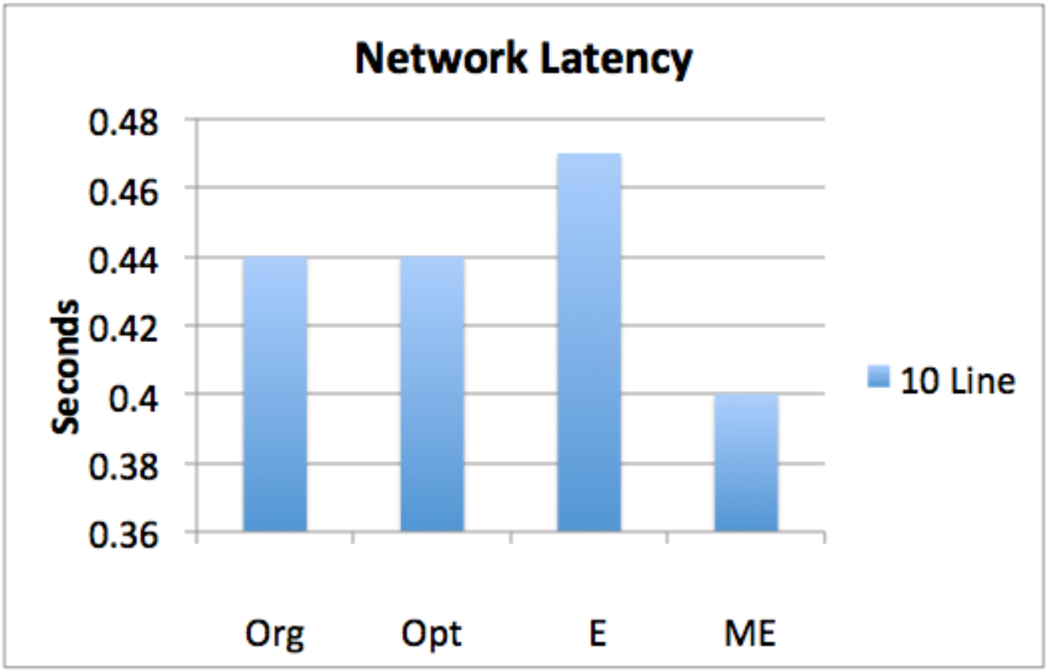

Radio On
Percent
3
2.5
2
1.5
1
0.5
0
Org
Opt
E
ME
80 Random
80 Grid

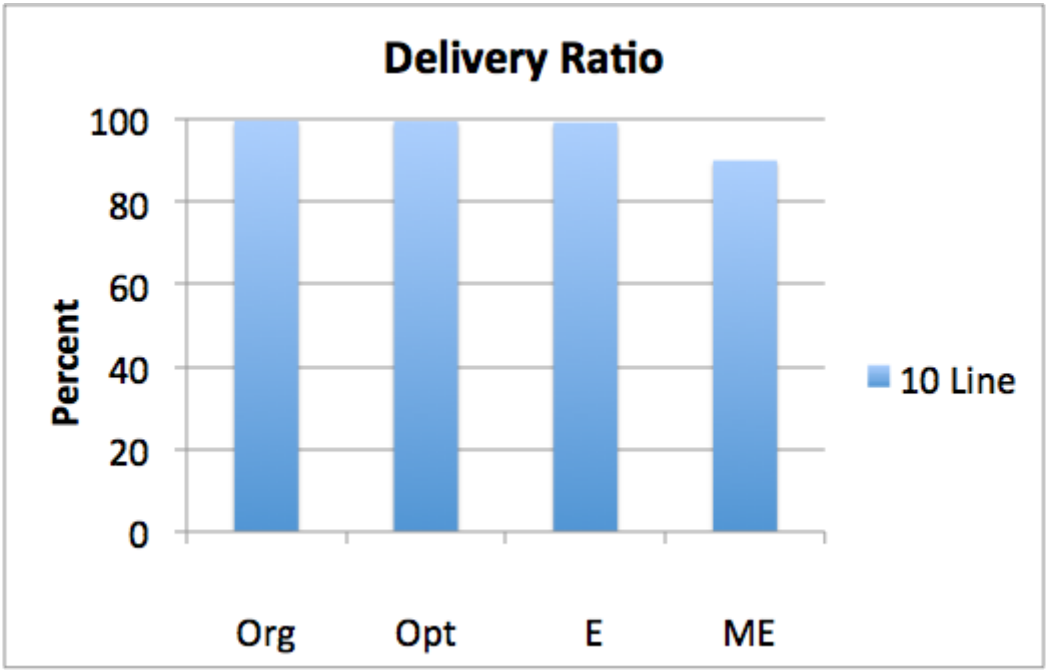

Radio On
Percent
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0
Org
Opt
E
ME
40 Random
40 Grid

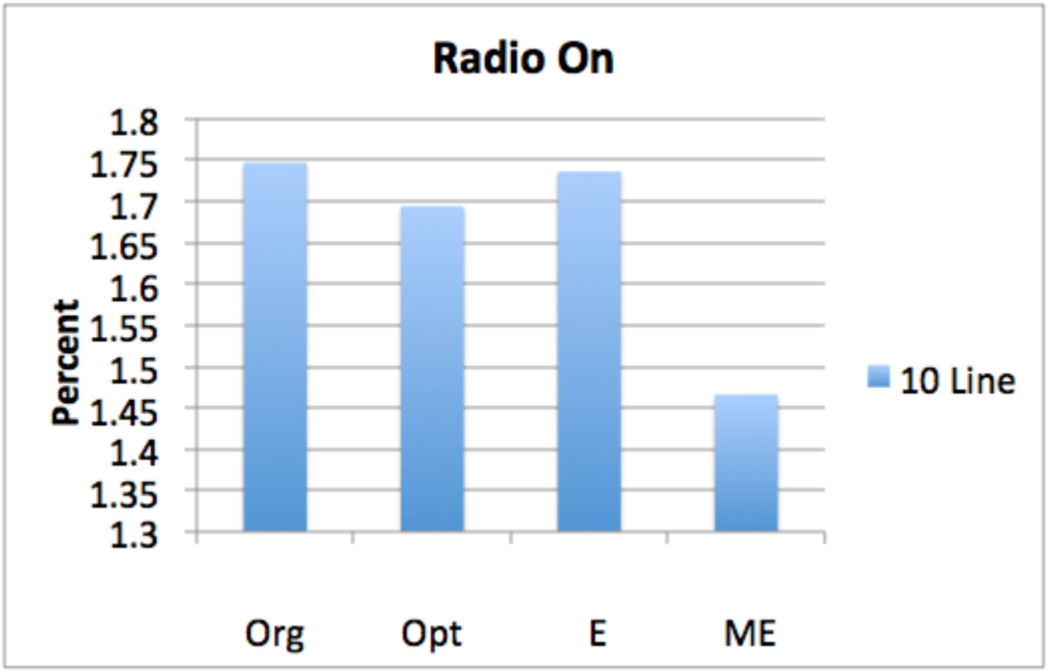

Radio On
Perecnt
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Org
Opt
E
ME
10 Random
10 Grid

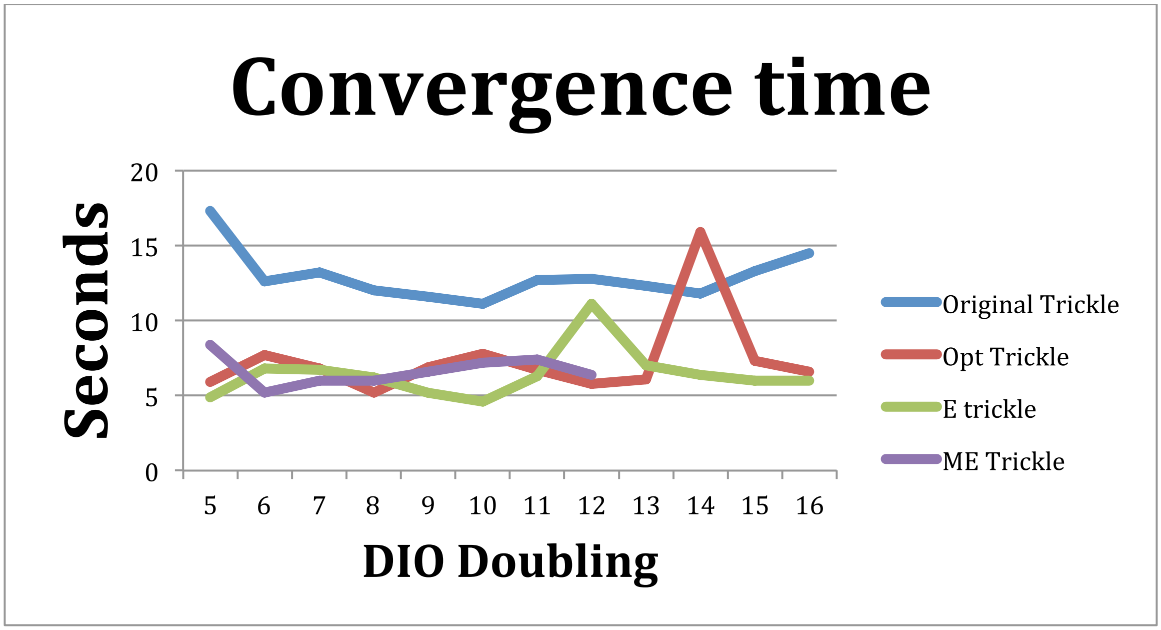

Convergence Time
Seconds
30
25
20
15
10
5
0
Org
Opt
E
ME
10 Line

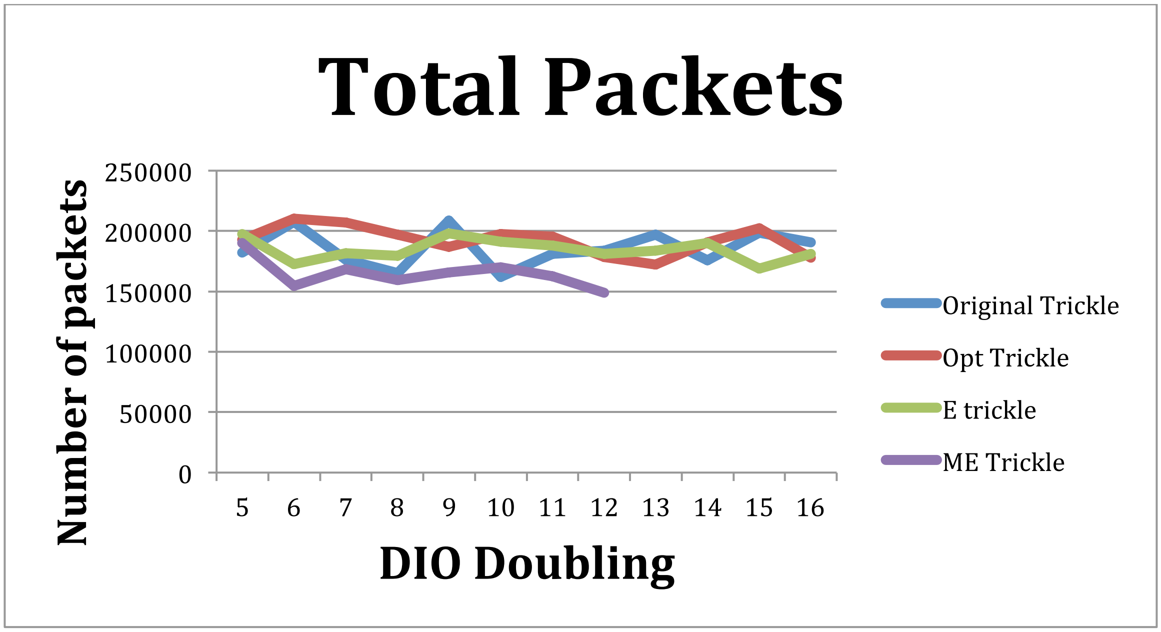

Total Packets
25000
20000
15000
10000
5000
0
Org
Opt
E
ME
10 Line

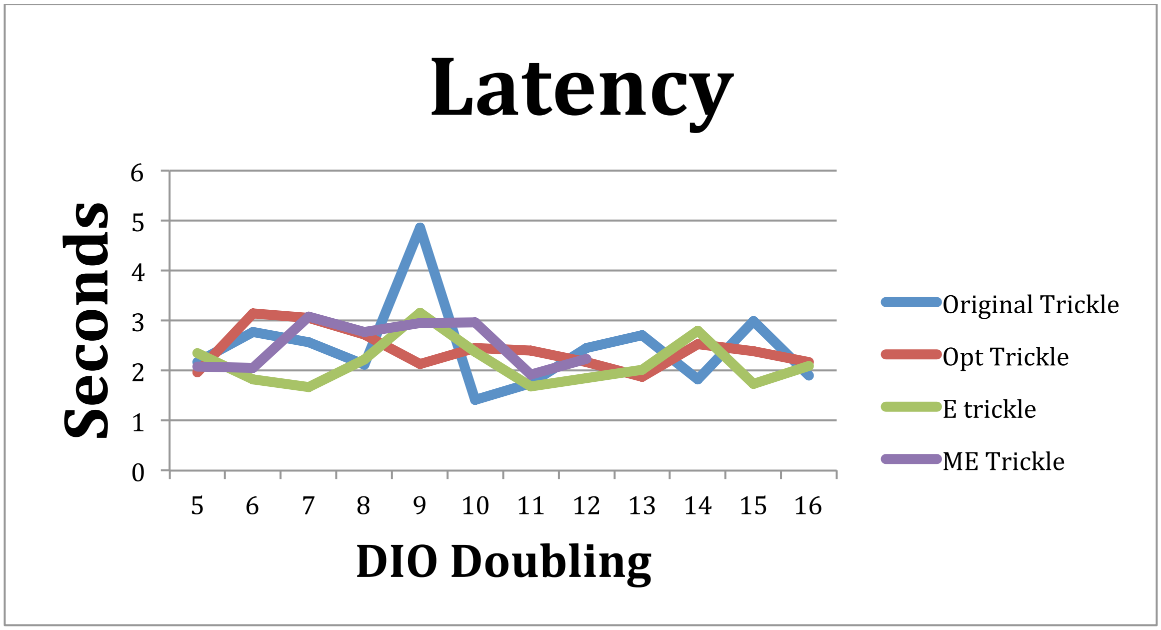

Network Latency
Seconds
0.48
0.46
0.44
0.42
0.4
0.38
0.36
Org
Opt
E
ME
10 Line

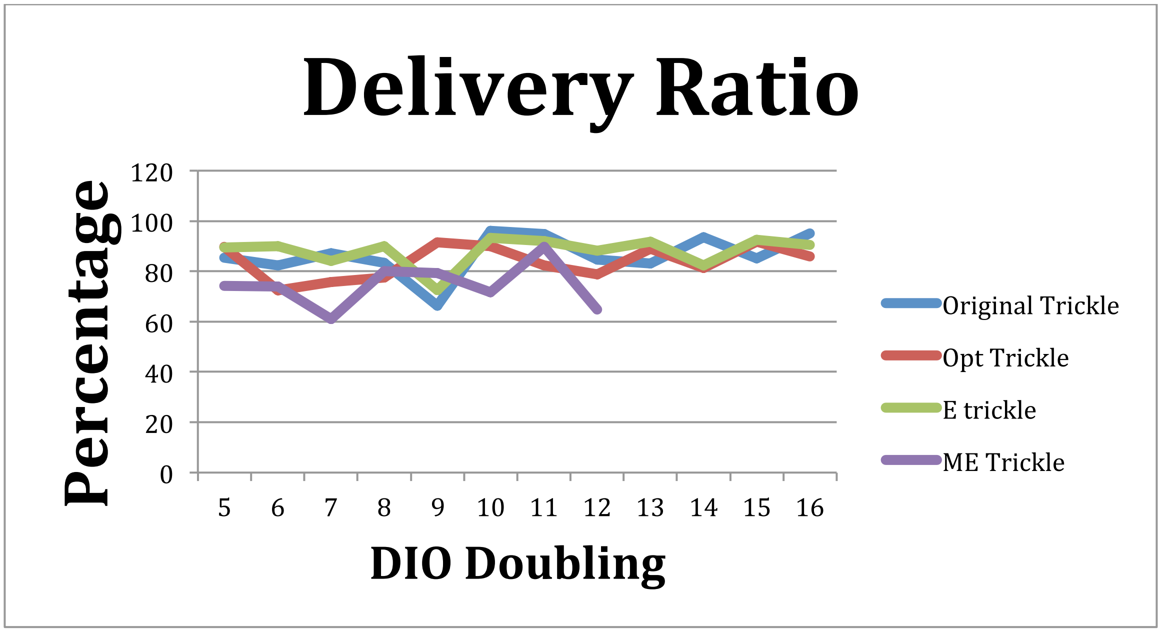

Delivery Ratio
Percent
100
80
60
40
20
0
Org
Opt
E
ME
10 Line

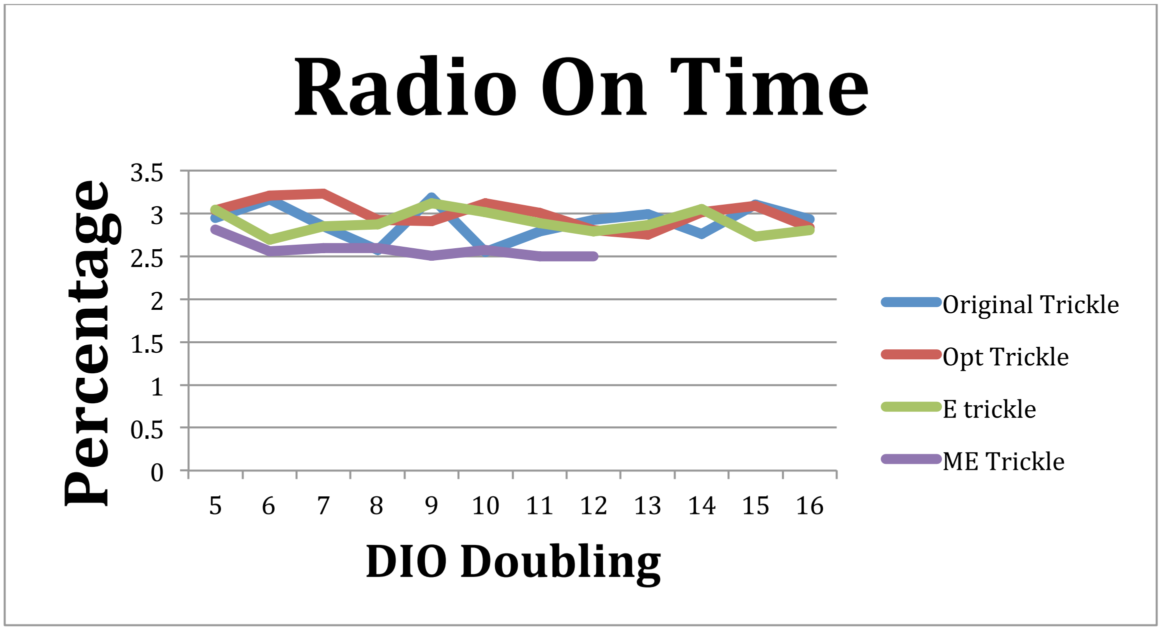

Radio On
Percent
1.8
1.75
1.7
1.65
1.6
1.55
1.5
1.45
1.4
1.35
1.3
Org
Opt
E
ME
10 Line

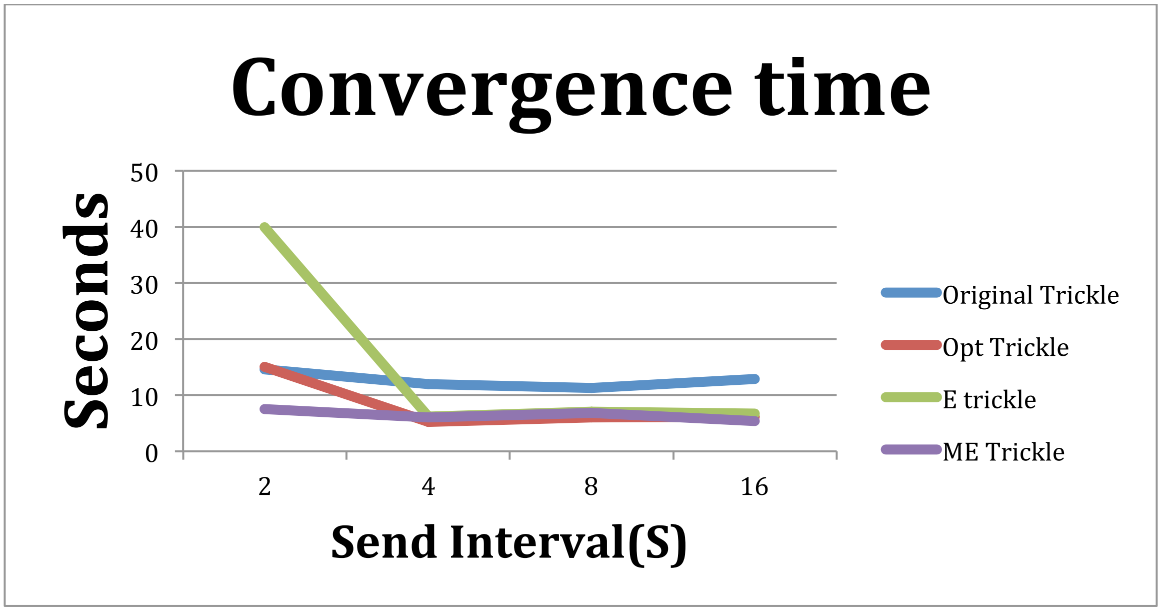

Convergence time
Seconds
20
15
10
5
0
5
6
7
8
9
10
11
12
13
14
15
16
DIO Doubling
Original Trickle
Opt Trickle
E trickle
ME Trickle

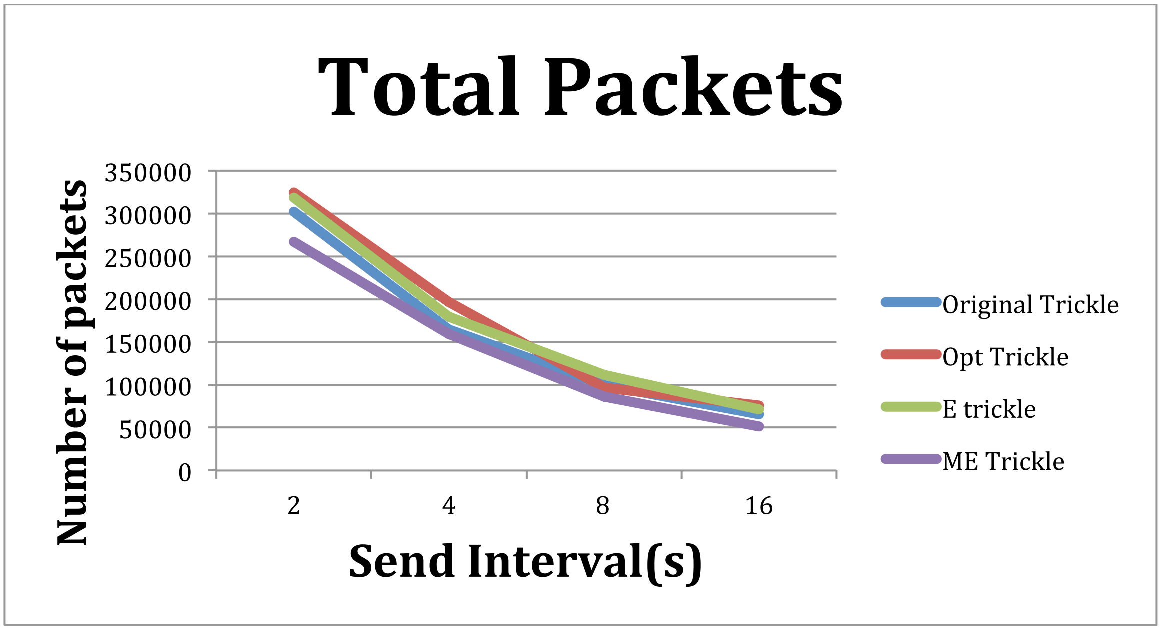

Total Packets
Number of packets
250000
200000
150000
100000
50000
0
5
6
7
8
9
10
11
12
13
14
15
16
DIO Doubling
Original Trickle
Opt Trickle
E trickle
ME Trickle

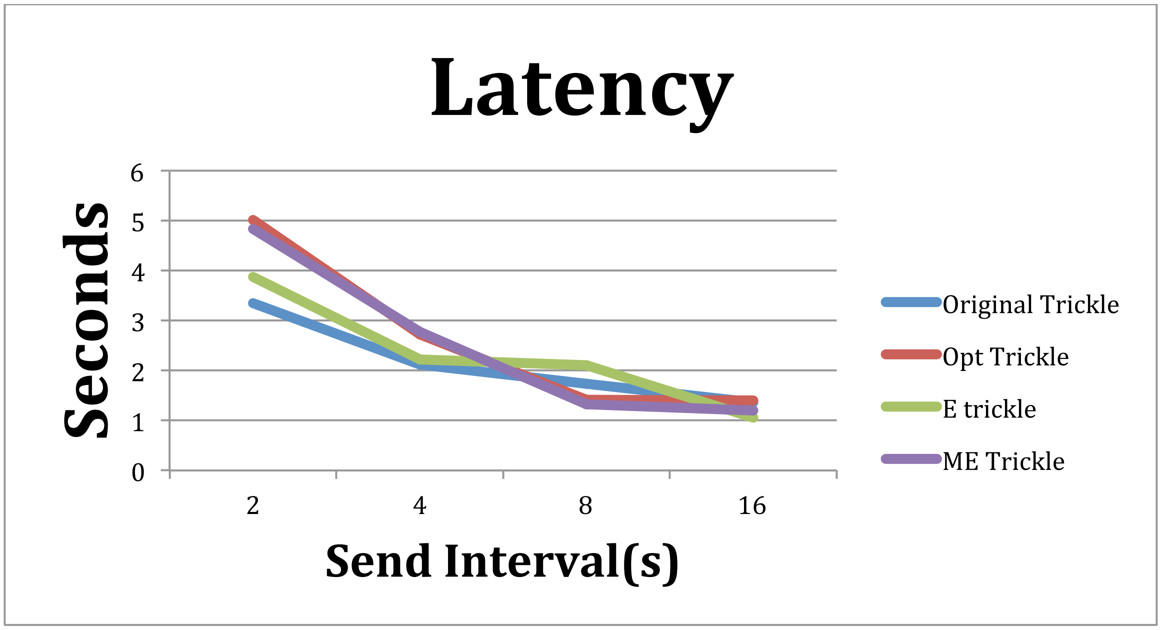

Latency
Seconds
6
5
4
3
2
1
0
5
6
7
8
9
10
11
12
13
14
15
16
DIO Doubling
Original Trickle
Opt Trickle
E trickle
ME Trickle

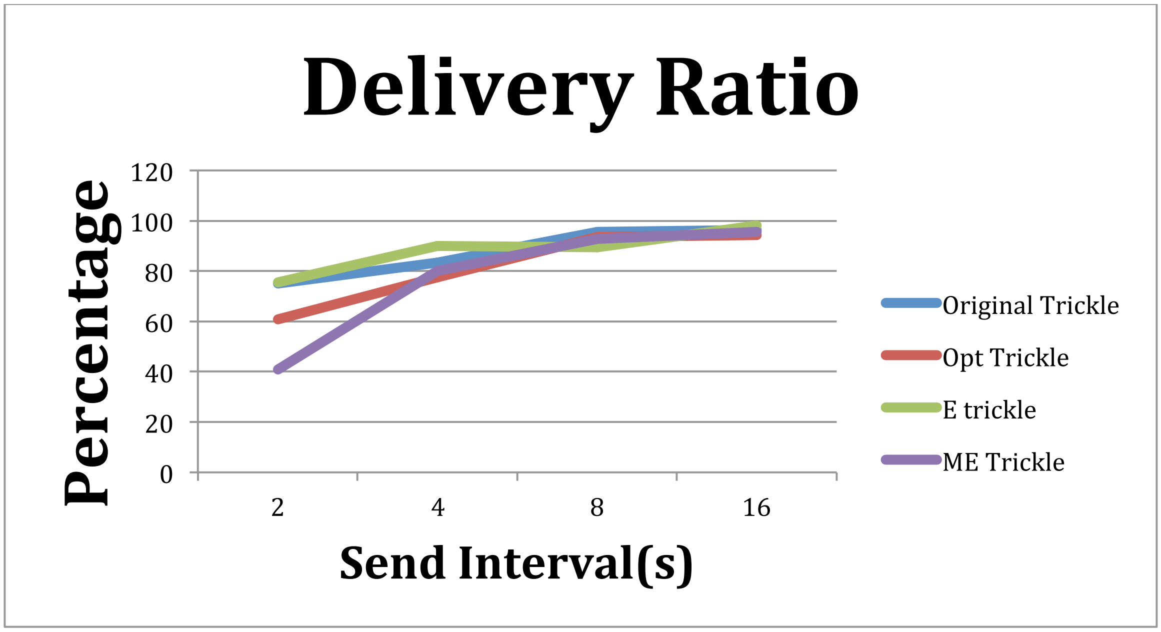

Delivery Ratio
Percentage
120
100
80
60
40
20
0
5
6
7
8
9
10
11
12
13
14
15
16
DIO Doubling
Original Trickle
Opt Trickle
E trickle
ME Trickle

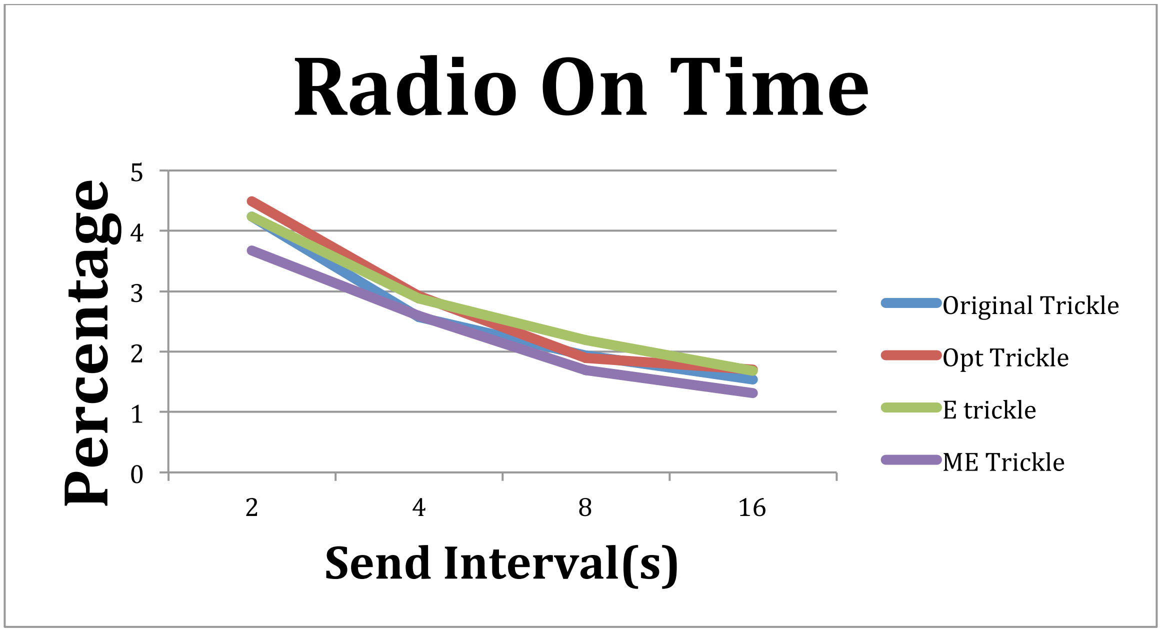

Radio On Time
Percentage
3.5
3
2.5
2
1.5
1
0.5
0
5
6
7
8
9
10
11
12
13
14
15
16
DIO Doubling
Original Trickle
Opt Trickle
E trickle
ME Trickle

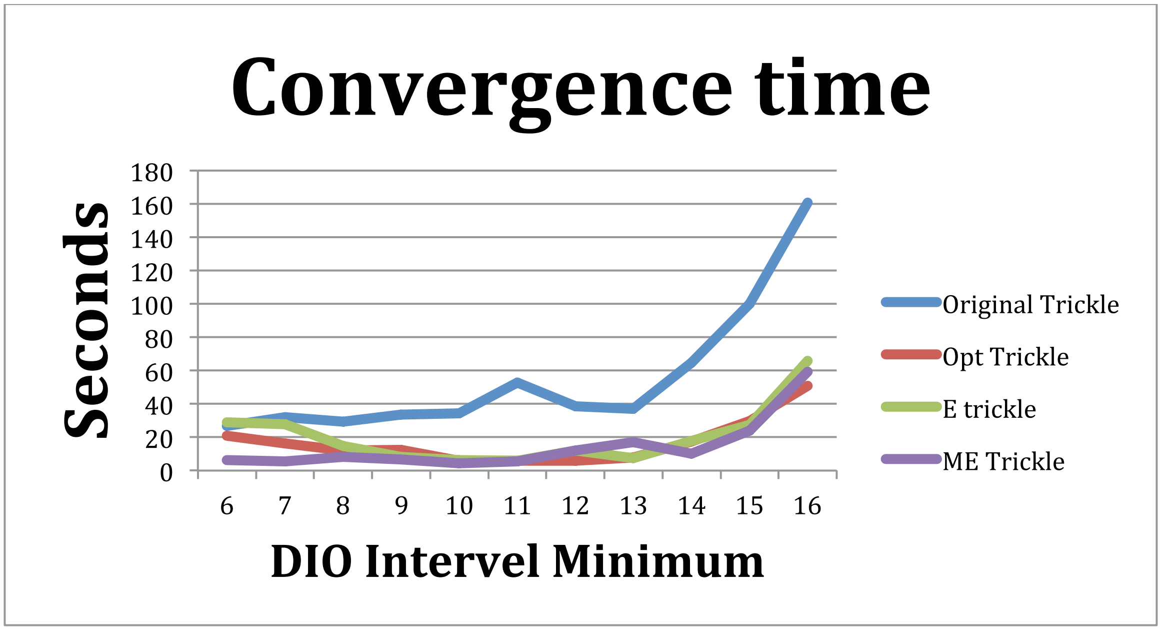

Convergence time
Seconds
50
40
30
20
10
0
2
4
8
16
Send Interval(S)
Original Trickle
Opt Trickle
E trickle
ME Trickle

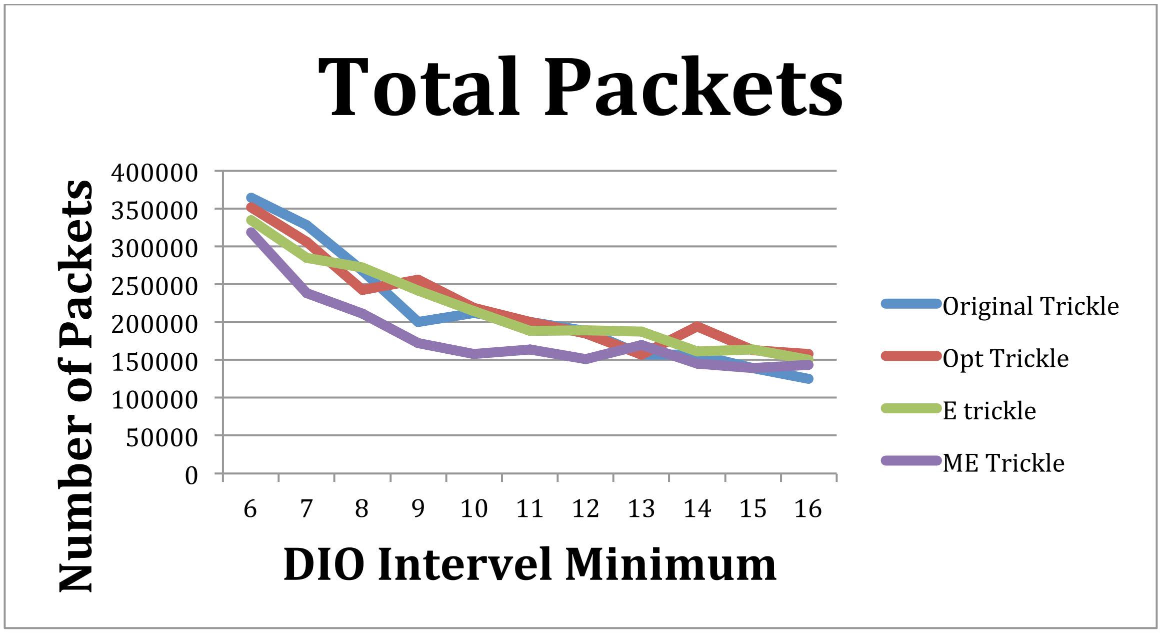

Total Packets
Number of packets
350000
300000
250000
200000
150000
100000
50000
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

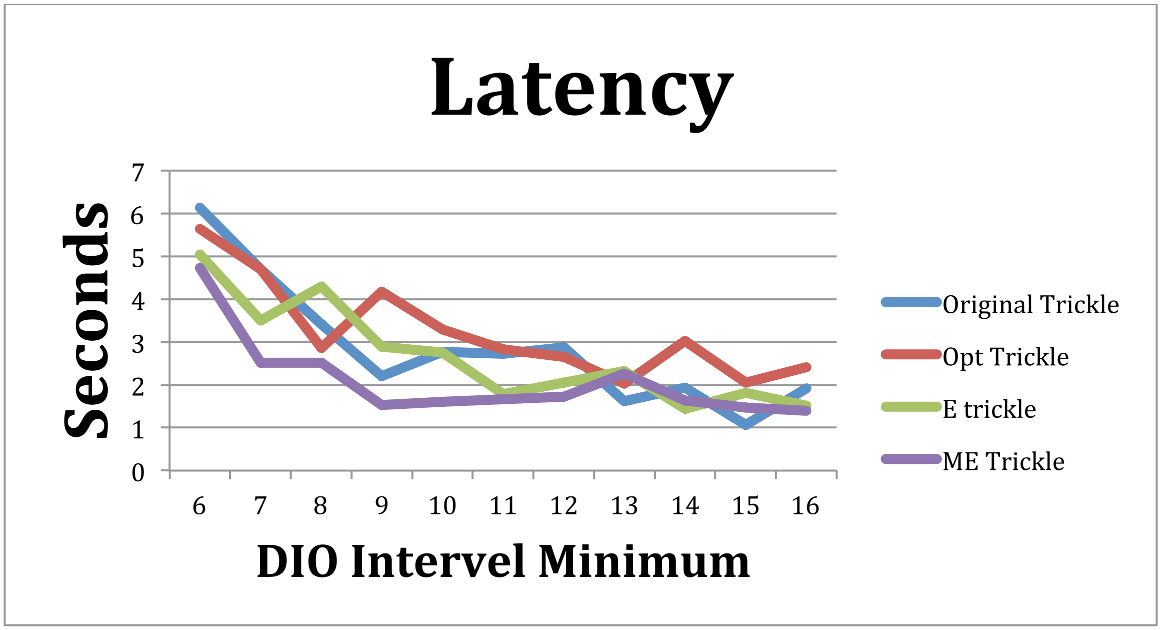

Latency
Seconds
6
5
4
3
2
1
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

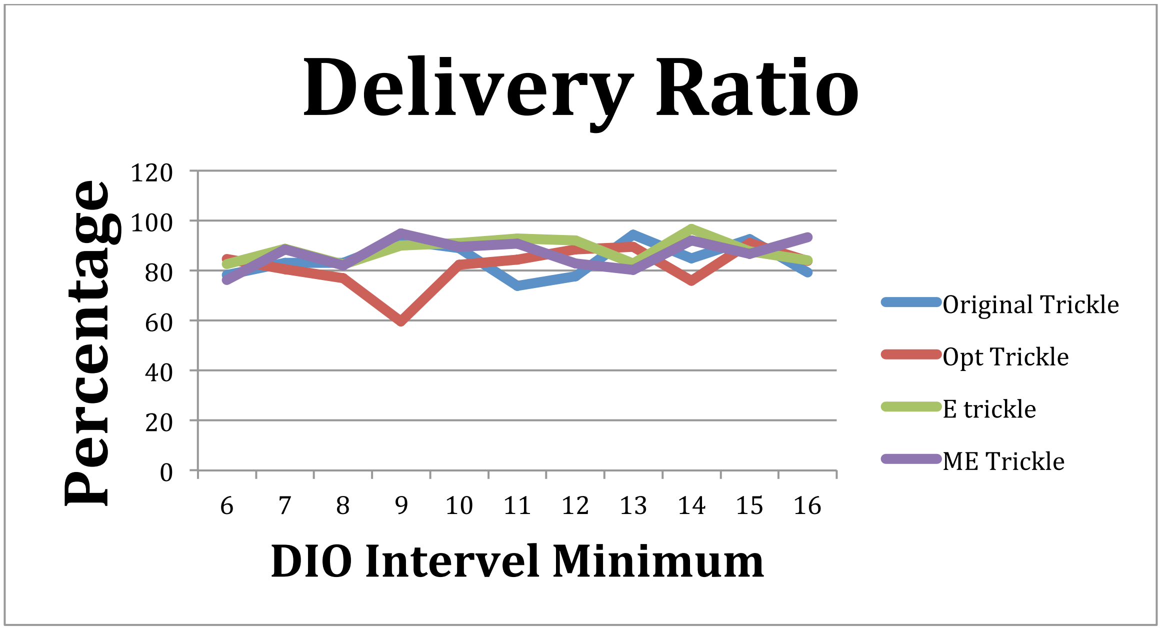

Delivery Ratio
Percentage
120
100
80
60
40
20
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

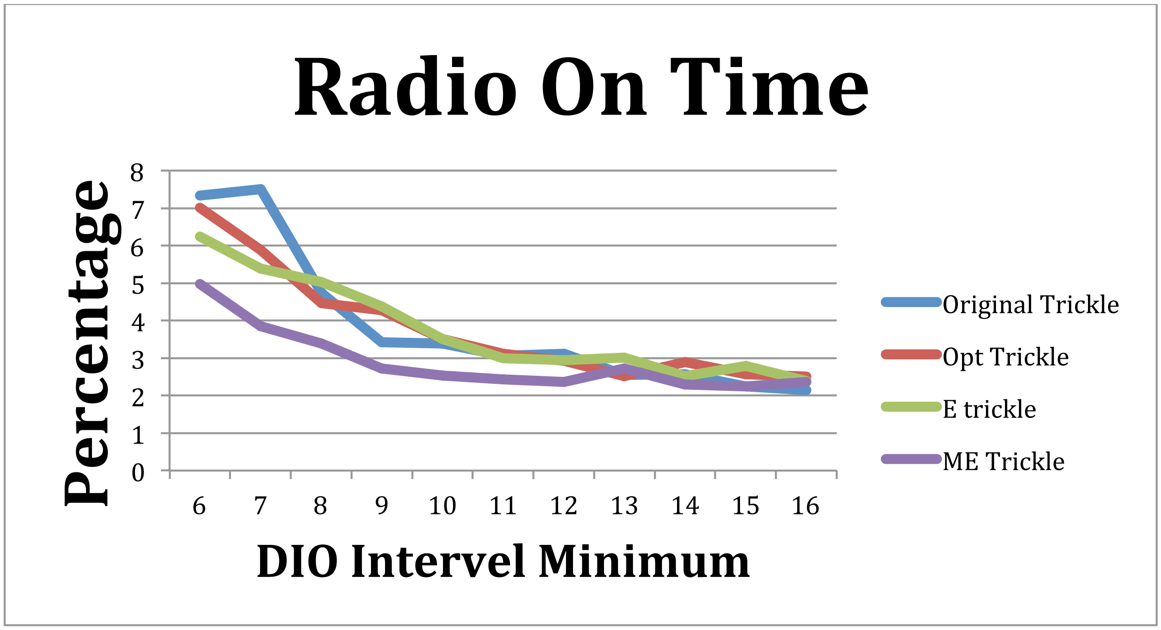

Radio On Time
Percentage
5
4
3
2
1
0
2
4
8
16
Send Interval(s)
Original Trickle
Opt Trickle
E trickle
ME Trickle

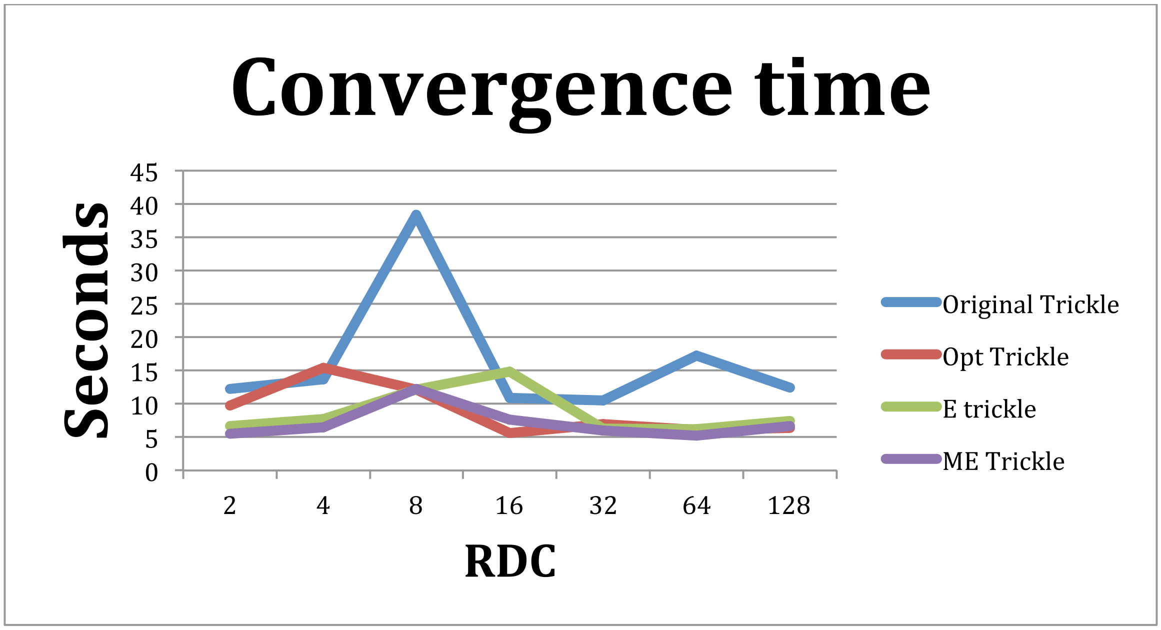

Convergence time
Seconds
180
160
140
120
100
80
60
40
20
0
6
7
8
9
10
11
12
13
14
15
16
DIO Intervel Minimum
Original Trickle
Opt Trickle
E trickle
ME Trickle

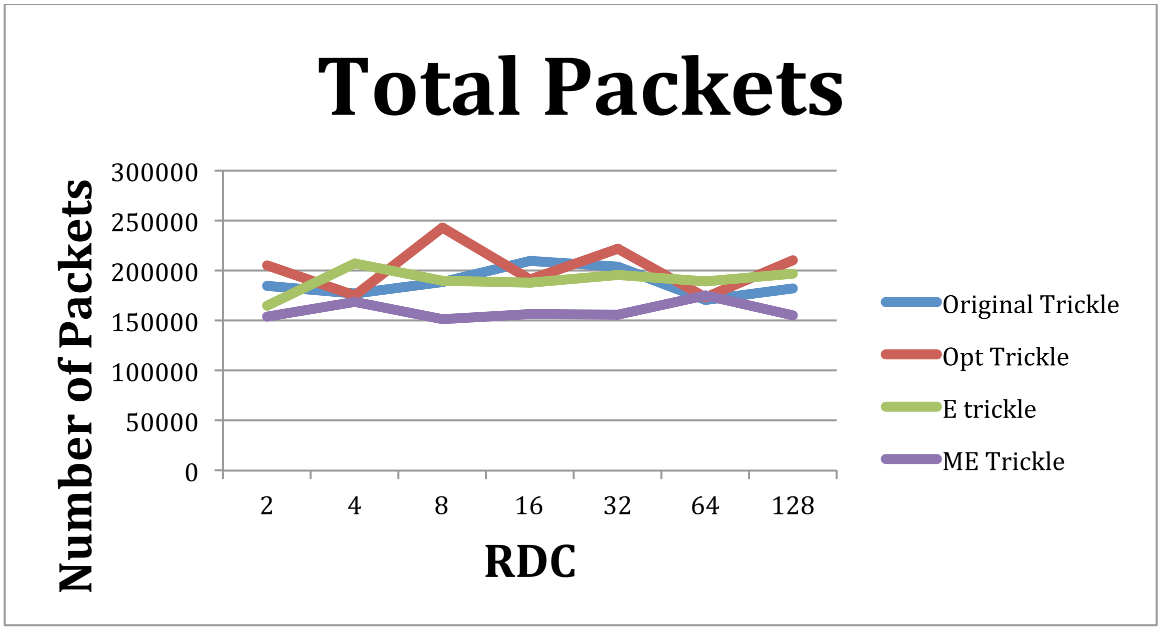

Total Packets
Number of Packets
400000
350000
300000
250000
200000
150000
100000
50000
0
6
7
8
9
10
11
12
13
14
15
16
DIO Intervel Minimum
Original Trickle
Opt Trickle
E trickle
ME Trickle

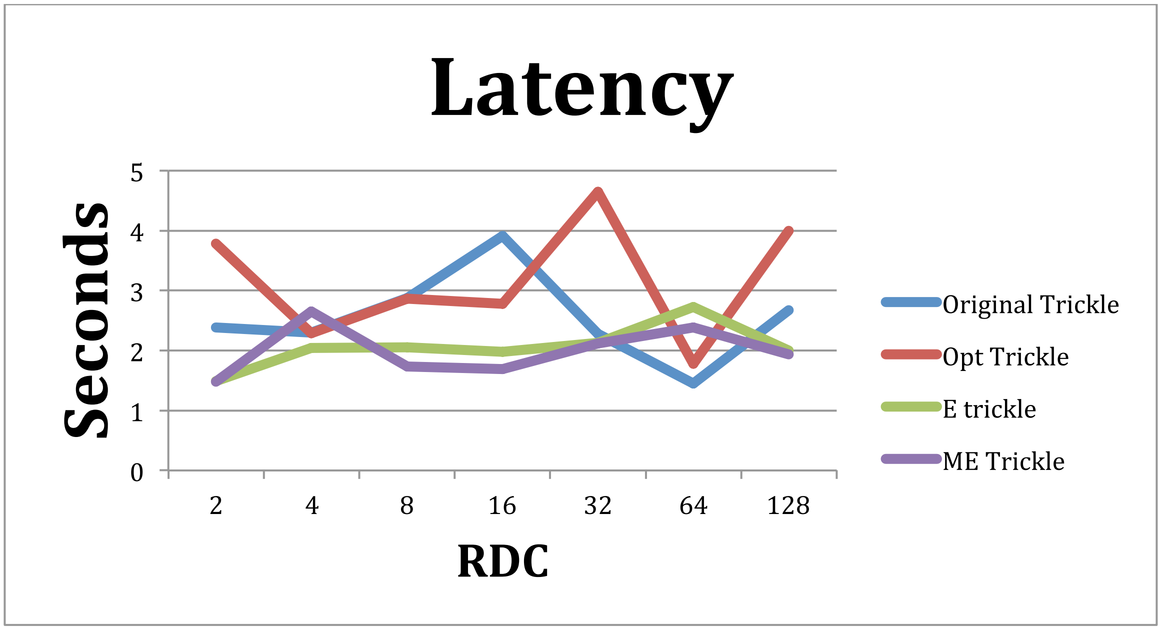

Latency
Seconds
0
1
2
3
4
5
6
7
6
7
8
9
10
11
12
13
14
15
16
DIO Intervel Minimum
Original Trickle
Opt Trickle
E trickle
ME Trickle

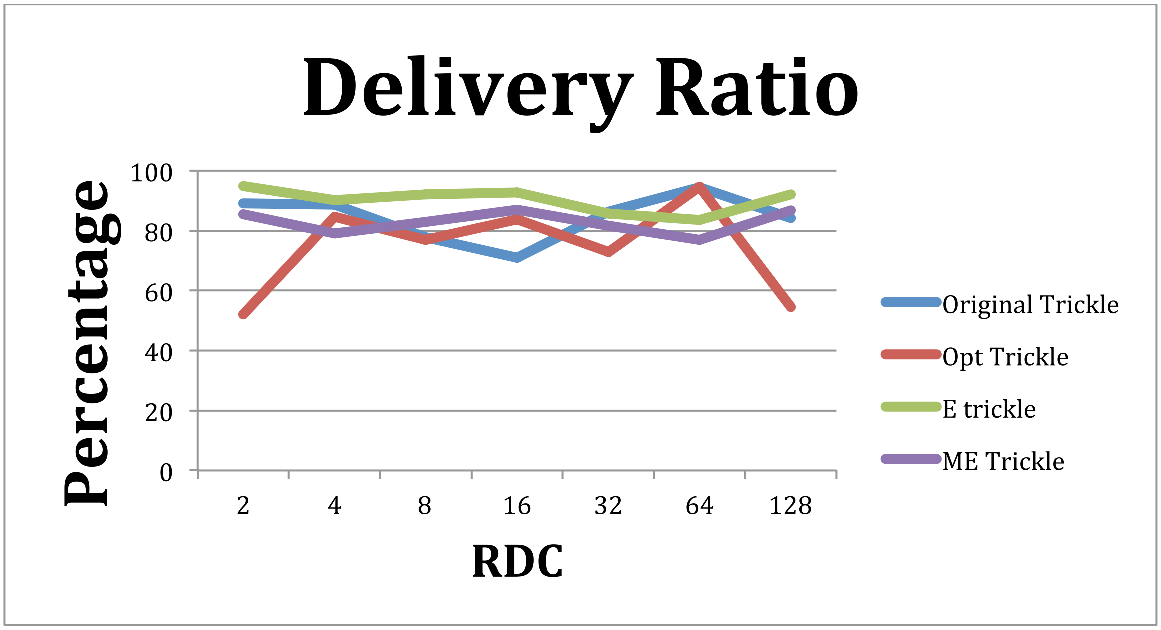

Delivery Ratio
Percentage
0
20
40
60
80
100
120
6
7
8
9
10
11
12
13
14
15
16
DIO Intervel Minimum
Original Trickle
Opt Trickle
E trickle
ME Trickle

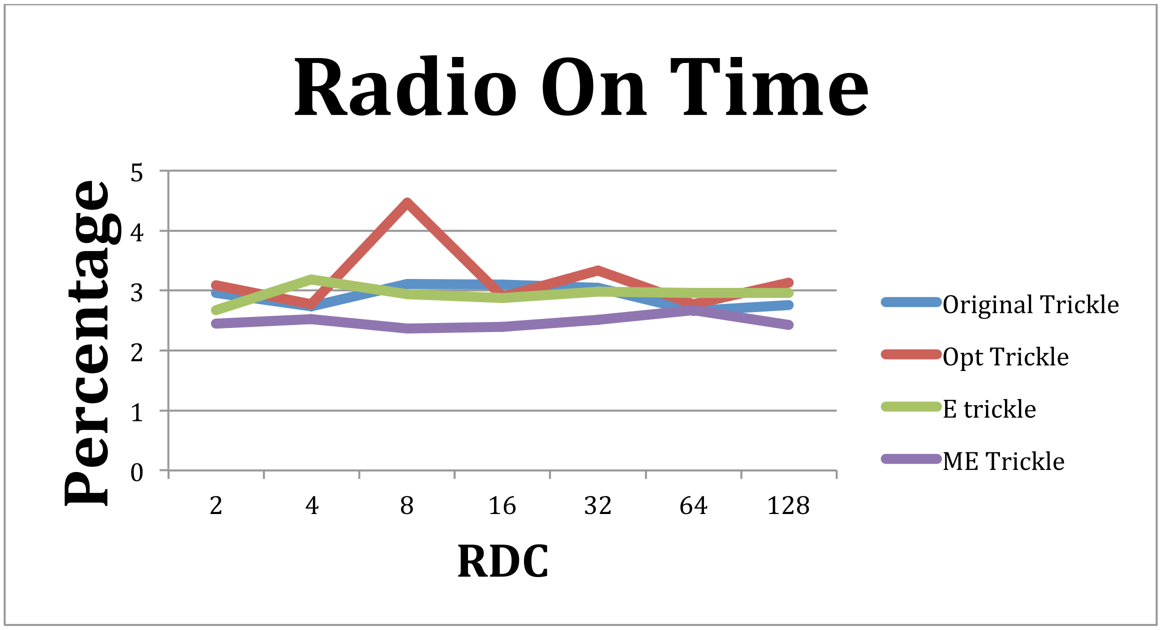

Radio On Time
Percentage
0
1
2
3
4
5
6
7
8
6
7
8
9
10
11
12
13
14
15
16
DIO Intervel Minimum
Original Trickle
Opt Trickle
E trickle
ME Trickle


Convergence time
Seconds
0
5
10
15
20
25
30
35
40
45
2
4
8
16
32
64
128
RDC
Original Trickle
Opt Trickle
E trickle
ME Trickle


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