Technical Report: Connectivity In Task Graph Based Ad Hoc Networks*

S. Abu Ayyash, P. Basu, W. Ke, and T.D.C. Little
Department of Electrical and Computer Engineering
Boston University
8 Saint Mary's St., Boston, MA 02215, USA
617-353-9877
{*saayyash,pbasu,ke,tdcl*@bu.edu

MCL Technical Report No. 03-30-2003

Abstract– In this document we report on a set of simulation experiments that were conducted to evaluate the Task Graph (TG) protocol as outlined in [1]. The experiments allowed us to identify sources of poor performance due to TCP and other TG protocol related timers. Through these experiments we were able to choose appropriate timer values and achieve better results. We also measured performance for specific graph formulations, mainly different size k-ary graphs, under different conditions (such as speed and node density). Our experiments led us to pose some question regarding the feasibility of instantiating a TG for a specific outlined scenario. Lack of resources or connectivity can be predicted by parsing scenario files prior to the actual simulation experiment. We were able to measure a probability of successful instantiation under different scenarios thus bringing us closer to the task of TG admission control.

Keywords: Connectivity, wireless ad hoc networks, admission control

*This material is based upon work supported by the National Science Foundation under Grant No. ANI-007384. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

The TG protocol layer depends on the various layers and their behavior under various conditions of contention, density and mobility. It is designed in a way to control that dependency and to avoid large delays were other protocols are stabilizing and reacting to their control packet loss or delays. However, the interplay with the various timers and control paths make it hard to predict the behavior analytically and requires some experimentation to isolate scenarios were the performance might suffer. This work was initially an exploratory investigation into sources of TG instantiation delay times which were higher than expected. We followed that thread as we understood more about the behavior of the various underlaying protocols in addition to the TG protocol. Once the delays were reduced and the core behavior of the TG was as expected, we experimented with various TG structures, and went on to establish limits of performance with respect to the given scenarios.

The report summarizes a set of experiments which were conducted over a two month period starting in November 2002 and ending in March 2003. As mentioned above, the initial goal was to verify results of instantiation times observed in ns task graph (TG) simulations by parsing simulation trace files for evidence of the sources of delays. We focused on delays incurred due to TCP timers. Once we adjusted the TCP timeout values appropriately, we further investigated causes of high instantiation times especially in large TGs. Consequently, we found that the TTL timer values needed to be fine tuned to reduce the effects of excessively large ring search delays. The next set of experiments aimed at characterizing instantiation delays which were expected to be logarithmic in nature as a function of the number of nodes in the TG. We studied binary and k-ary trees and came to some conclusions with respect to the causes of some unexpected bi-modal behavior. This led to some questions about the connectivity of nodes and hence the feasibility of trying to instantiate a TG under certain connectivity and node density conditions. Finally we generated a probability of success curve based on 100 scenarios per point plotted. We witnessed a phase transition change in this probability around $1700 \times 1700 km^2$ for various scenarios. The significance of this behavior will be discussed briefly in later sections.

In the following sections we present a set of results along with a discussion verifying and characterizing the behaviors seen. In Section 2.1, we present our experiments on TTL and TCP timers. In Section 2.2 we study performance as a function of TG structure, size and depth. In Section 2.3 we evaluate connectivity and in Section 2.4 we generate a probability of success curve for various scenarios. Finally we conclude with a brief statement as to the possible future direction of this work.

2 Experiments

2.1 TTL and TCP Time-Out Timers

We focused our attention on a large TG case which took as long as 37 seconds to instantiate. We simulated a 127 node TG (binary tree of depth 6) on a 225 node (15x15) *static* grid where every node has 4 or lesser neighbors (boundary nodes). To ensure that we increased the area to 3000x3000 since the radio range is 250m. We predicted that two major sources of the incurred delay could be the TTL and TCP time-out values.

TTL Time-Out Timer As it turned out, the TTL timer used in an expanding ring search primitive used for the discovery of devices was indeed a big culprit. The following rule was used. (1) set the timer T = 0.5s and broadcast with TTL=1 (2) if no response from a candidate within T, set TTL++; T = TTL*T, and then rebroadcast. This process is continued until a response comes. Generally if there is a bottleneck path in the TG (from root to a leaf containing a node that is reasonably large number of hops (h) away from its parent, the parent will take at least 0.5*h*(h-1)/2 seconds. Since there is a large path of length 9 in the simulations, this alone accounts for 18 secs of delay (out of 37). If there are 2 such bottleneck nodes in the path, it results in most of the delay. Hence the expanding ring search parameters were adjusted to reduce this delay. First the initial timer was halved to 0.25 secs, which lowered the instantiation time to 29 secs.

We also investigated another scheme described as follows: set the initial timer to 0.5sec but always keep it = 0.5s (i.e. do not multiply it by 2 if it goes off before a response comes). This scheme further improved the instantiation time to 21 seconds which is almost half of the original time. Even further reducing the timer value to 0.25 sec brought the instantiation time down to 16 seconds.

TCP Time-Out Timer In continuing our investigation into the source of delays, we isolated source-destination pairs in the packet level simulation trace files and plotted delays suffered by TCP packets (static case). We noticed that the initial TCP time-out timers where set at 6 seconds in ns, which did not correspond to recommendations made in RFC2988 of 3 seconds . Once we adjusted the timer values to 3 seconds (see Figure 1 for the isolated TCP delay values vs. an approximate time series represented by all unique sequences of source destination communications), and along with the new TTL timer scheme described above we observed that the instantiation times were drastically reduced. But it also reduced the monotonicity of results. For e.g. for a TG size, N =31, the entire instantiation happens in

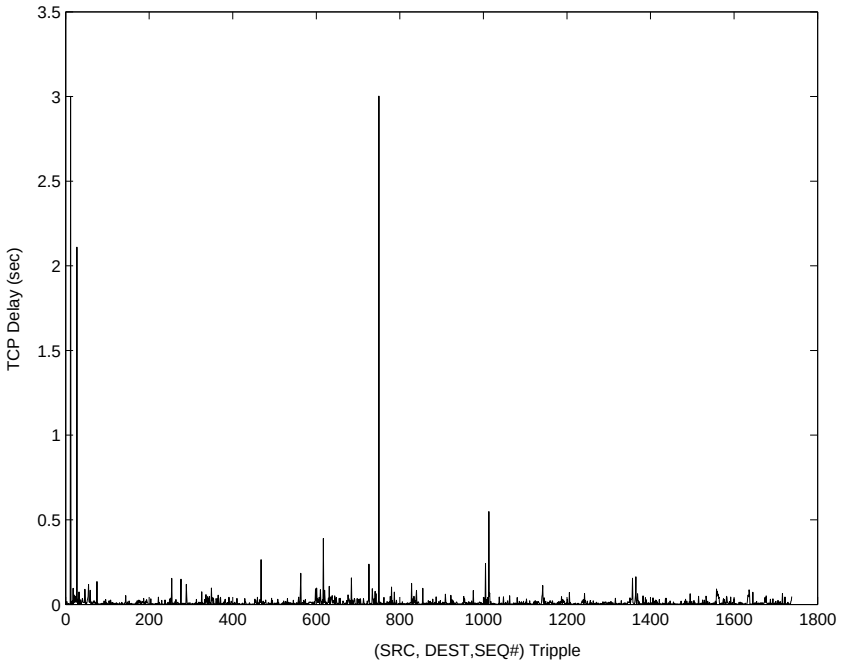

Figure 1: TCP Delay for N=127

1.87 secs whereas for a lesser number of nodes (15), it takes almost 7 secs. see Figure 2.[1]

There was one TCP timeout for N=7 and 63, two timeouts for N=15 and N=127 (see Figure 1). There seems to be no monotonic correlation with the depth of the tree here. The cause of a TCP timeout in a static network with the initial (unadjusted) value = 3secs) is the loss of a SYN packet during the establishment of a connection. A SYN packet can get lost due to collisions. In general, when a packet is lost, TCP waits (up to the current time-out value) before retransmission occurs.

In Figure 3 we show a scatter plot of instantiation times. We used 10 scenario files for each case (number of nodes in the TG). The variance is due to cases were TCP times out. One can see a monotonic increase in the instantiation time as we go down with the number nodes in the TG, for each of the pause time columns.

Through these experiments we were able to isolate TCP delays and to study the effect of the TTL timer. The values chosen were ideal and resulted in very low instantiation times.

[1]All results from ns2 simulations were gathered after 200 seconds of simulation time to allow for some activity time to elapse, especially after large pause times.

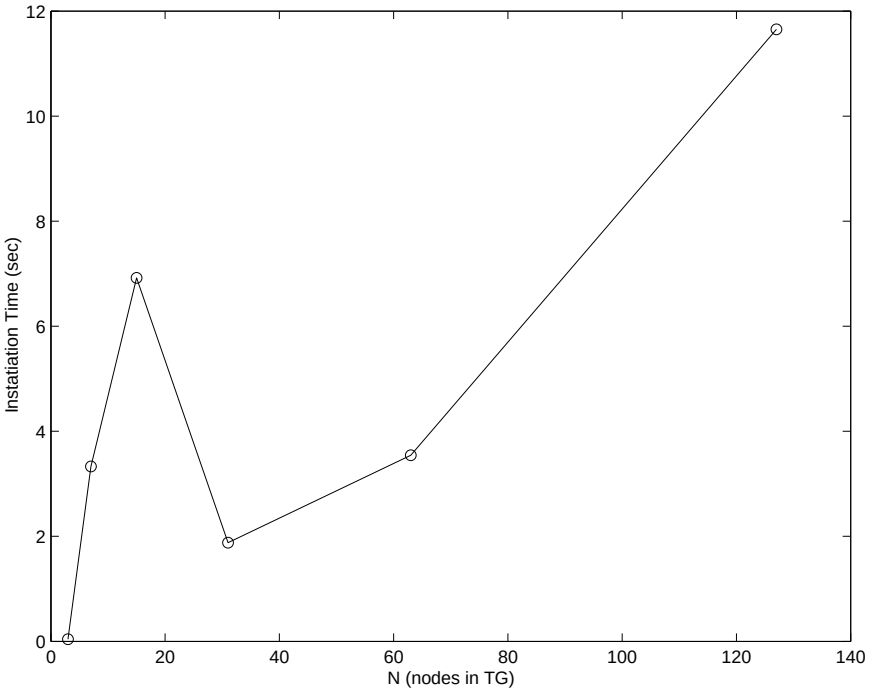

Figure 2: Instantiation Time after adjusting TCP and TTL timers

2.2 Characteristic of Instantiation Times in Static K-ary and Binary graphs

We generated some general k-ary graphs (by varying the depth and the number of leaves) and collected results of instantiation times. We found that for low k in general, there was a bimodal behavior where the instantiation times can be very low or a one TCP timeout value (3sec). Beyond a certain point, congestion and other factors mask this bimodal phenomena. Figure 4 is an instantiation time scatter plot for various k values and number of nodes. In general for a large TG with high k, there are more devices in the neighborhood that are unavailable (already instantiated) and there is also higher contention since more nodes reply at almost the same time, therefore, the average dilation increases which causes more broadcasts due to increased ring search and hence that increases instantiation time.

For more information on this study please refer to [1] section 6.2.1.

2.3 Connectivity: Feasibility of Instantiation

One goal of this effort is to perform a pre-analysis on a scenario file, without having to run an actual simulation, and be able to determine as much as possible the level of connectivity

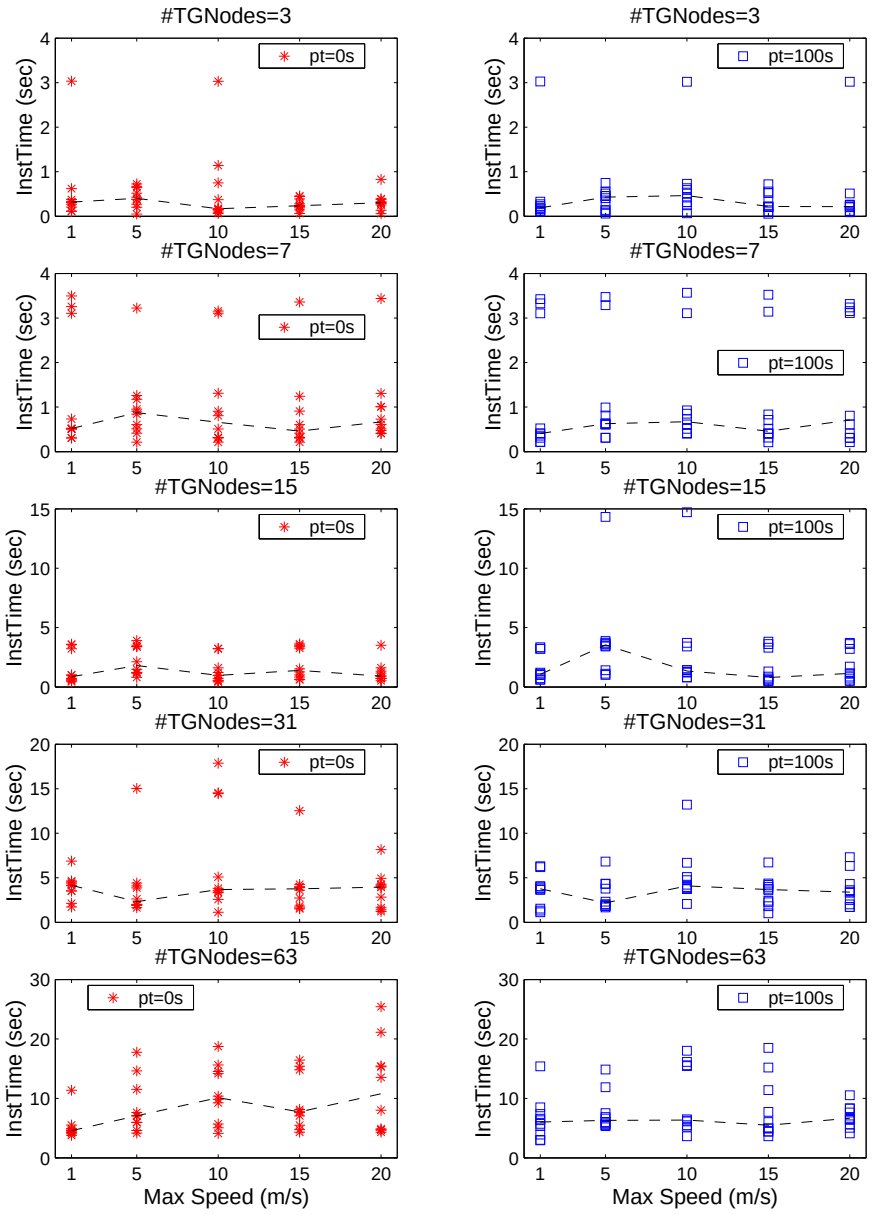

Figure 3: Instantiation Time after adjusting TCP and TTL timers - Scatter Plot with 10 scenarios

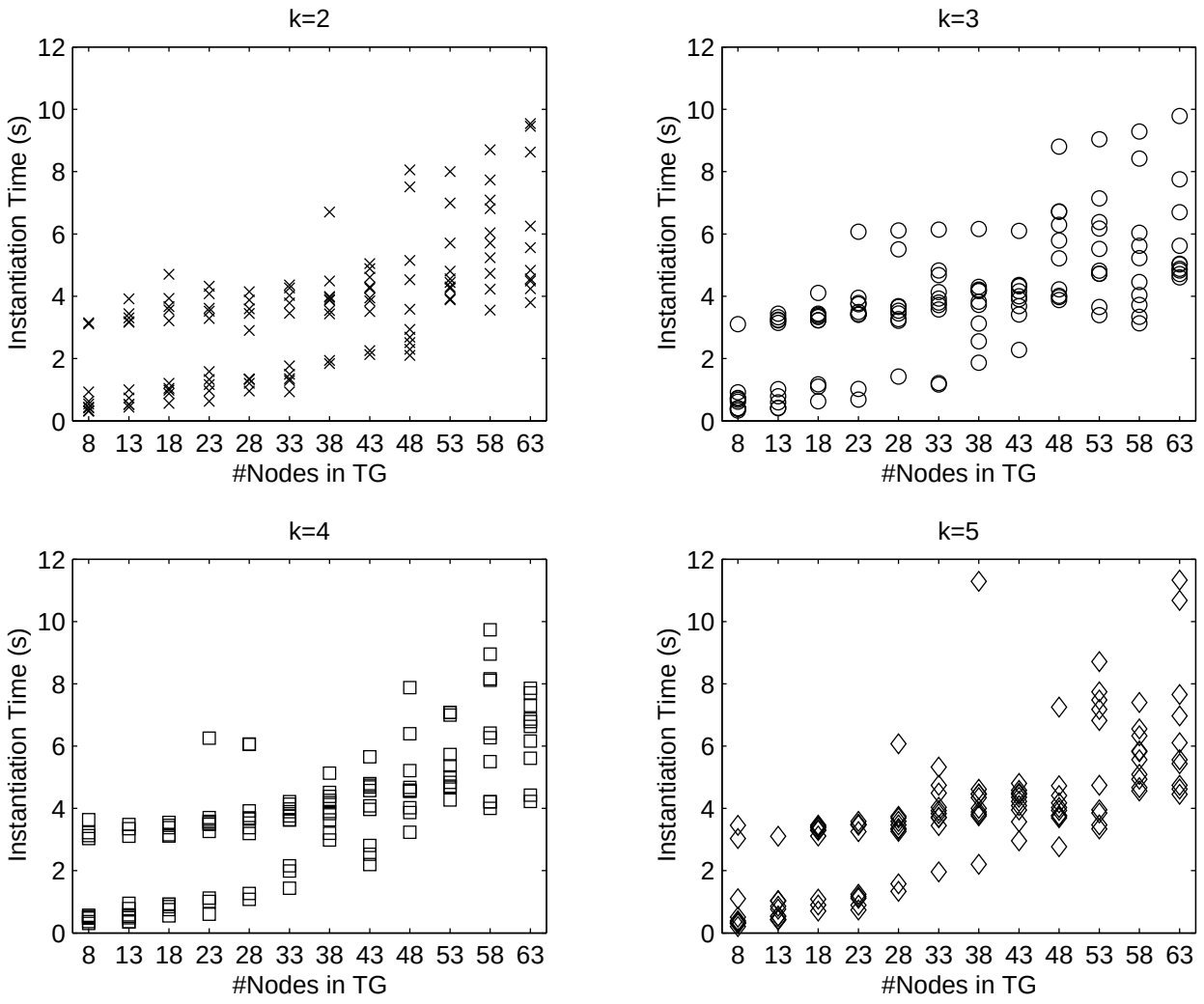

Figure 4: Instantiation Times for k-ary trees.

of the nodes and hence the feasibility of instantiating a task graph. One cause of large delays is if our protocol happens to select a node which soon becomes disconnected from the rest. Time spent in reaction to this disconnection and the repairs that follow add considerably to instantiation delays. A time varying plot of the number of nodes *unreachable* could shed some light onto the scenarios that might cause such delays.

Three metrics are usually listed at the end of each scenario file: (1) Destinations Unreachable (over the lifetime of the simulation) (2) Route changes (3) Link changes.

We averaged these metrics over 10 runs for a few scenario files with varying mobility patterns. (2) and (3) were smooth but (1) has absolutely no pattern. There were also a few odd points which do not follow any patterns with pause time and max speed.

Prompted by an observation for some particular scenario files (using ad-hockey), it was noticed that the root node was separated from the rest of the network for a very long time, or always. Since the *unreachable* metric used in the scenario files does not reflect the amount of time that a destination was unreachable, we parsed scenario files and recorded the number of destinations unreachable at time t. We then calculated a time average as an indication of the periods of dis-connectivity in the file.

$$mean(U) = \frac{1}{SimTime} \int_0^{SimTime} U(t)\, dt \tag{1}$$

Figure 5 shows the variation of the number of unreachable nodes for all nodes over time for a 25 node scenario with pause: 0.00, maximum speed: 20.00 and $1km^2$ area. The average number of unreachable nodes over the duration of simulation time (100 seconds) was 23.8 nodes.

We decided that a pure average for all the nodes was not satisfactory. Although in a scenario file we do not keep information about the task graph that might get instantiated and hence which nodes to be more focused on, we do know that the root (user) node is always node number (ID) zero. Therefore, we gathered information about another random process namely, *How many nodes exist in the connected component of the user node at a given instant of time?*, see Figure 6 for the same scenario file. This process can identify for us when is instantiation theoretically possible. The average number of nodes in the connected component was 22.5.

Figure 7 shows scatter plots of 10 runs for each mobility pattern (different maximum speeds for pause time zero and pause time 100), 100 nodes and three areas: area I ($1km^2$), area II ($2km^2$) and area III ($4km^2$). Area I and Area II seem to fare well, except for a few

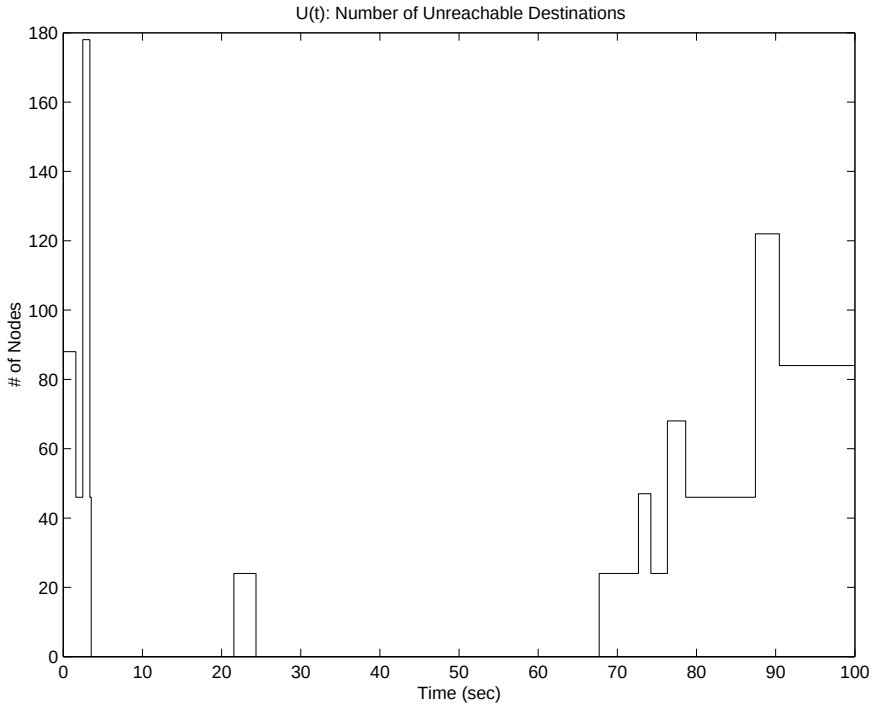

Figure 5: Number of unreachable nodes

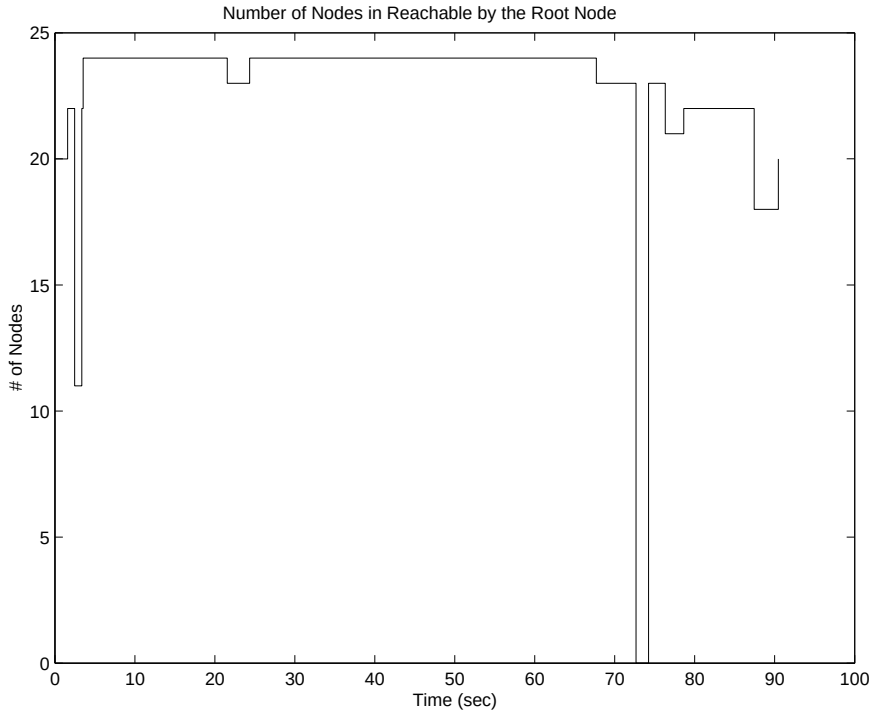

Figure 6: Number of Nodes Reachable by the Root

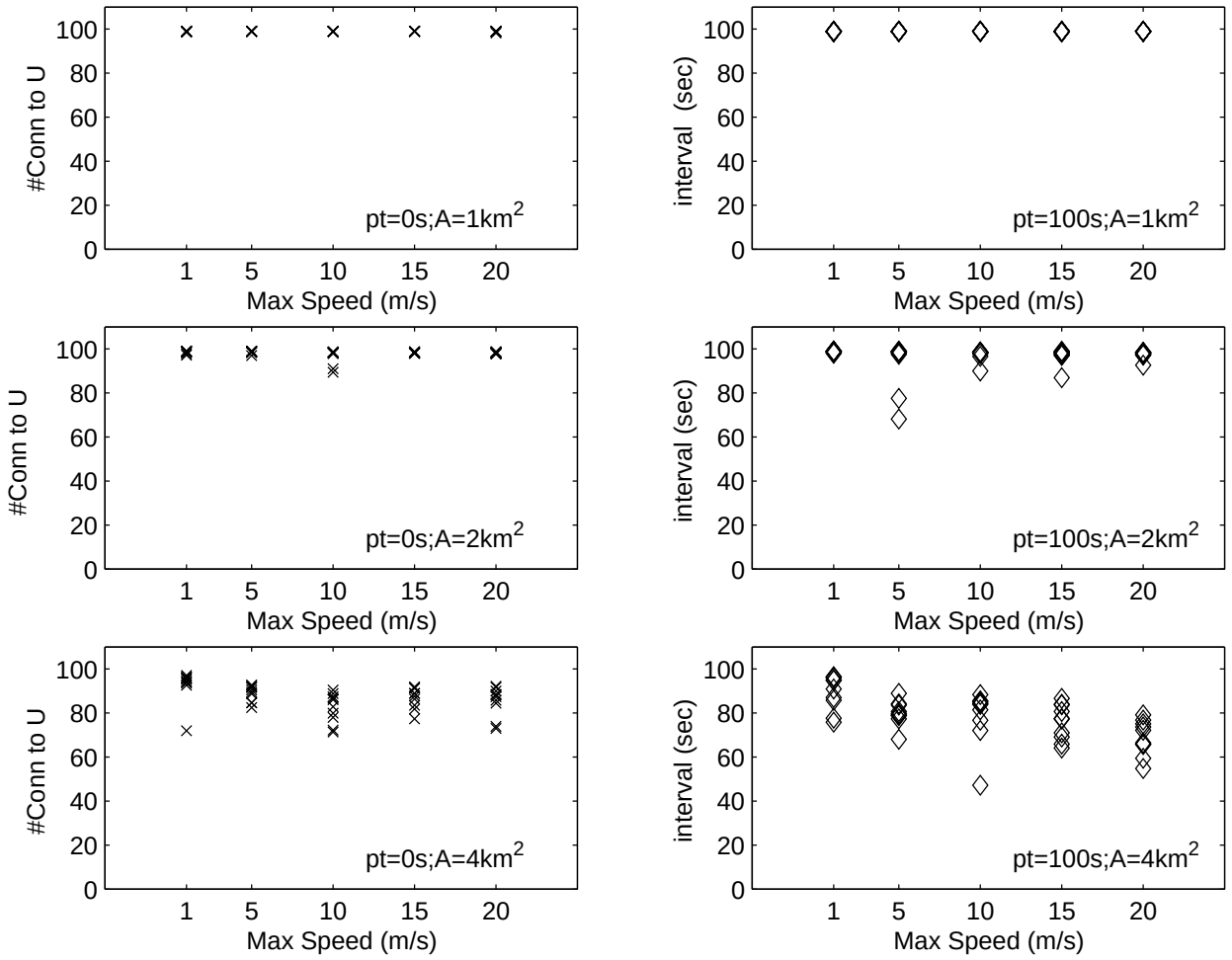

Figure 7: Number of Nodes Reachable by the User node

points in Area II with pause time 100 seconds where a disconnected graph might remain in that state due to the lack of constant mobility which might remedy the situation (compare pause time 0 seconds Area II). Results shown in this figure are a little optimistic, since they are not qualified by how long the user node was able to keep that connected component. Disconnection, especially relatively long ones, can trigger costly re-instantiation measures. Looking ahead in contrast, in Figure 12 we measure the probability of success where success is defined if user connected component can be maintained for a minimum time period T and disconnects are transient ($< 7sec$ where re-instantiation timers are not triggered).

Since the area III case showed the most variability and the worst results, we increased the number of runs to 50 runs to get more confidence in our results. As shown in Figure 8 there was some improvement in the 50 run case over the 10 run with respect to the mean

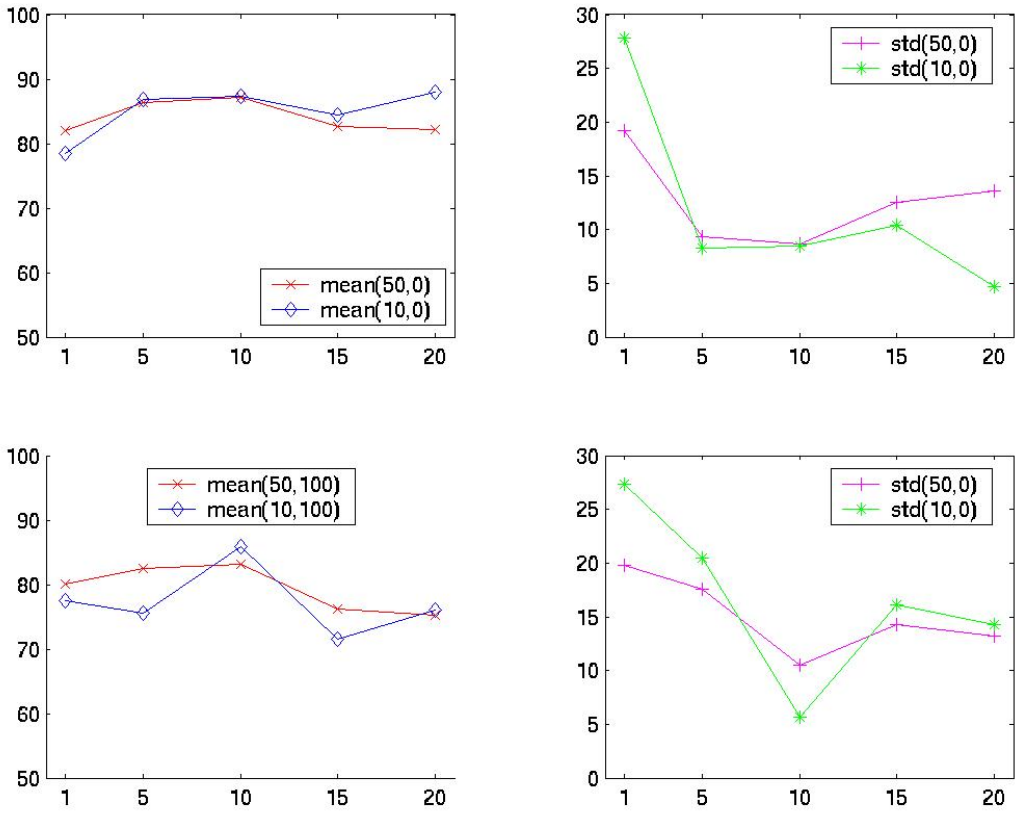

Figure 8: Comparing Mean and STD for 50 vs 10 runs in area III in Figure 7

and variation (of the mean). Hence although there is some improvement in the variance for the 50 scenario case, due to space limitations we decided it was not really necessary at this point.

To further investigate, we report the first interval of time where the user node had 63 or more nodes in its connected component. This gave us an indication of whether it is possible to instantiate and for how long this supposed task graph would remain connected. Figure 9 shows this statistic. It is obvious that area I and area II seem to be favorable. Area III has many cases where 63 is not achieved or achieved for such a small period of time that it would be questionable if a task graph can be instantiated. [2].

Similar to Figure 7, these results are also optimistic since they do not quantify the length of time of disconnects that follow this connected period which could trigger a costly

[2]Figures 9 through 13 were generated based on a mobility model different than the rest, where the minimum speed in setdest was changed from zero to 0.9 of maximum speed

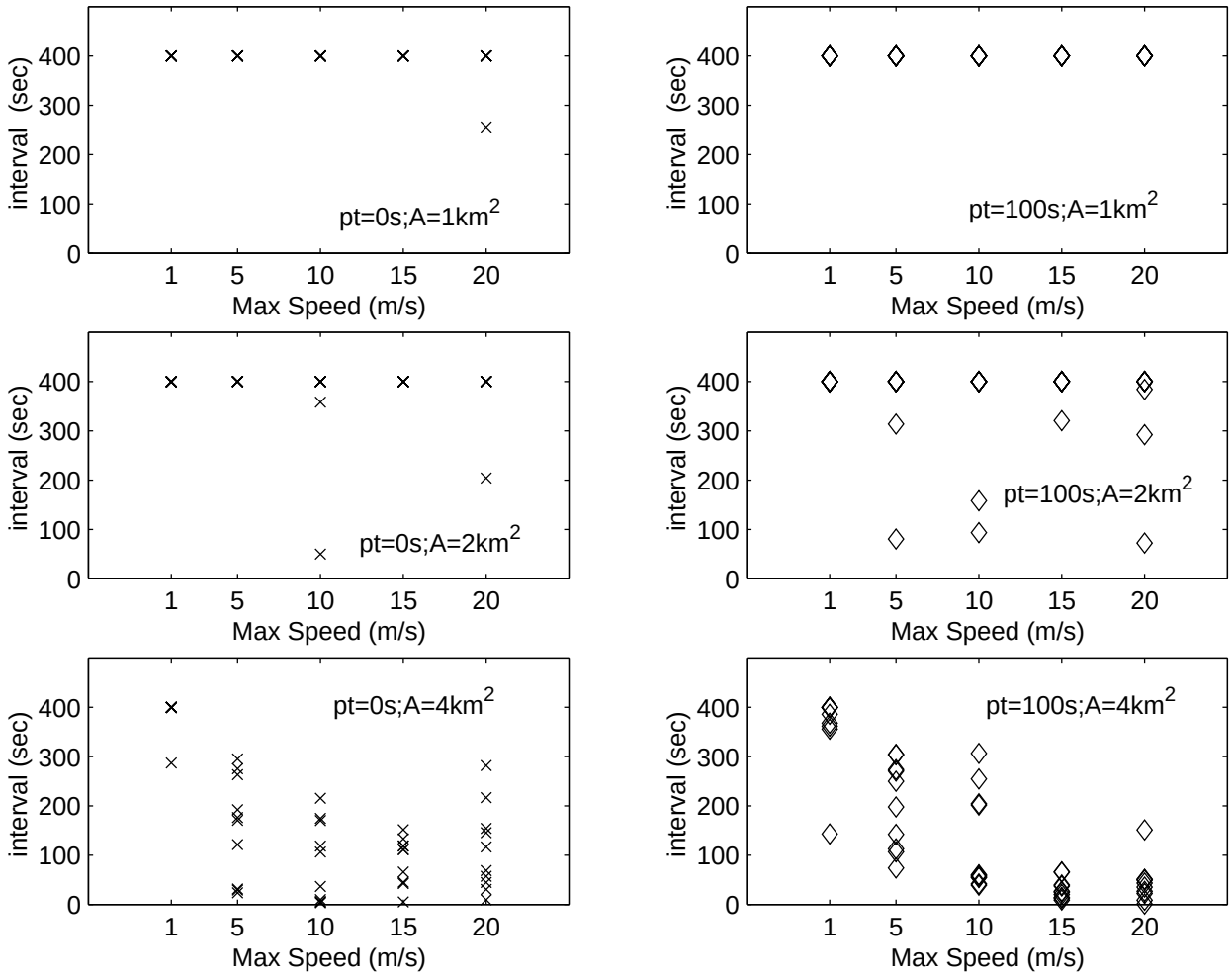

Figure 9: Longest Instantiated Interval with No Disconnects)

re-instantiation.

To make a point clear, we show the results of instantiation times for the area III case, See Figure 10 (the median line is plotted too). Note that although in some cases it shows that instantiation happens, albeit *after* a long time, these results do not say anything about for how long the User node was able to keep that TG instantiated. On the other hand, some cases exhibited a large number of brief disconnects, which could potentially be harmless if the application layer TG protocol does not discover them and react unnecessarily.

Given the above observations, we also monitored the longest interval of time after the above interval (the instantiation interval) where the User node lost one or more of it's required 63 nodes. See Figure 11. Again area III shows some problematic cases especially

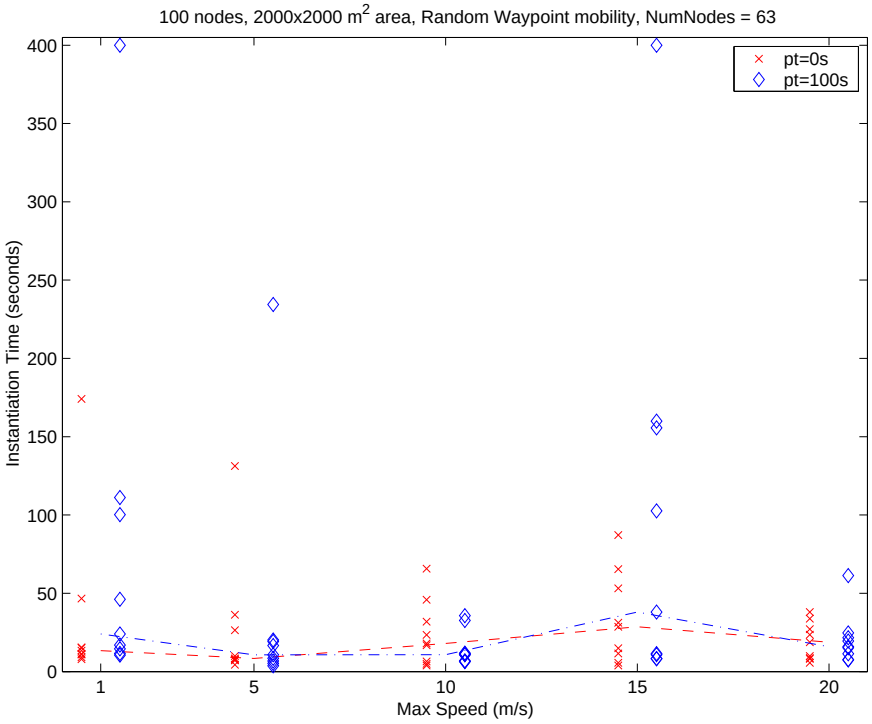

Figure 10: Instantiation Times for a 63 TG with 100 Nodes in Area II

when that interval is longer than 14 seconds (7 seconds in some cases) which is the period of time it takes the task graph to start a re-instantiation phase. Any other period smaller than that might go unnoticed by the task graph and therefore that brief dis-connectivity might be irrelevant.

We make the point here that this analysis has started as a verification effort. We migrated into an attempt at feasibility analysis where we could predict prior to simulation or actual test bed experiment, which cases render a successful instantiation/ application. However, it is clear that although we can identify the unsuccessful cases, there is no telling what might happen in a case that is seemingly successful. The reason for this is that we only monitored the connectivity of nodes to the User node. However, in a distributed component, the children of the User are responsible for maintaining their children (and eventually the leaves of the task graph). Although the User might remain connected to a leaf node, the leaf node's parent might get disconnected and it might take a while for the route change to occur (through the User node), long enough to disrupt the task graph operation.

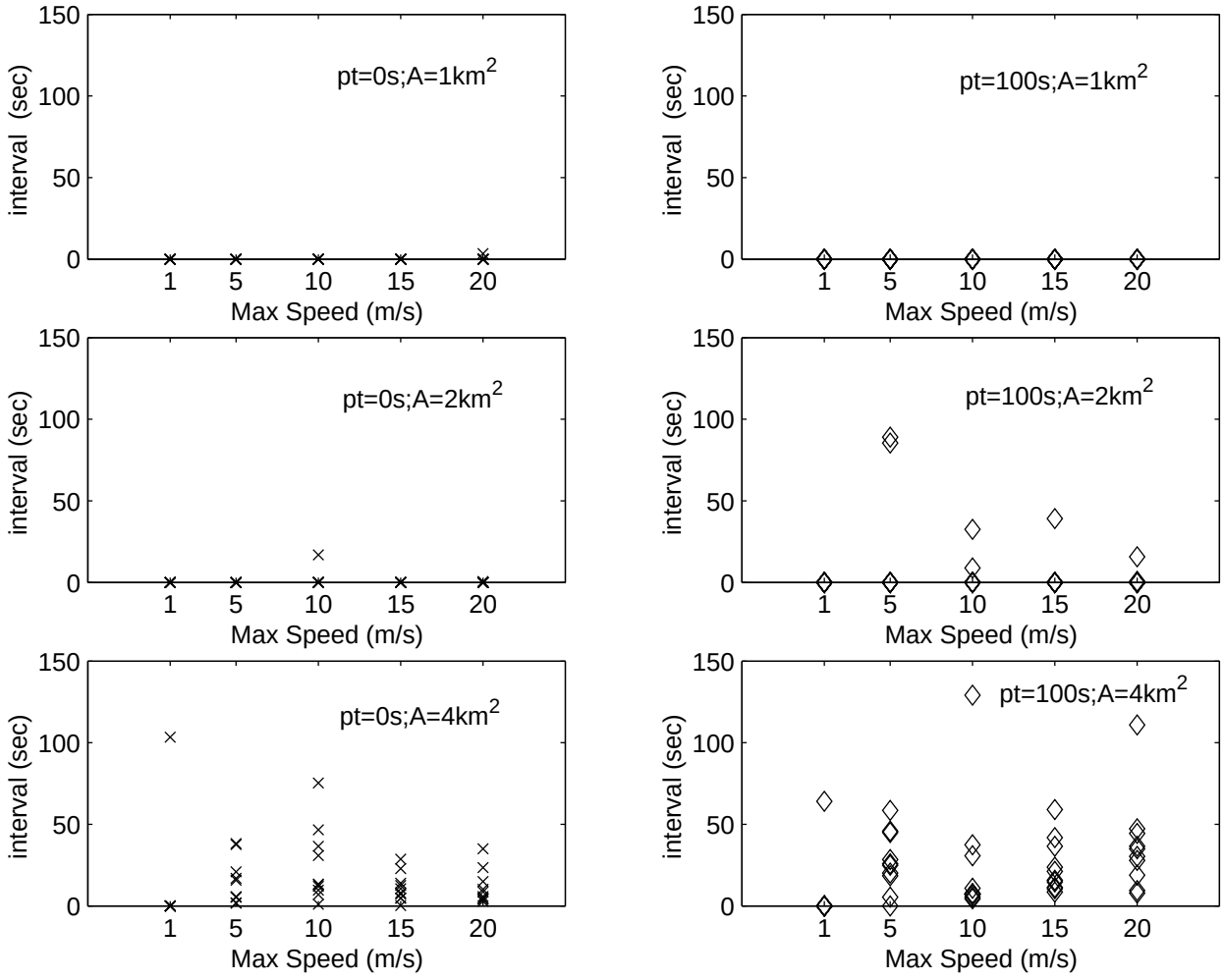

Figure 11: Longest Disconnected Interval After instantiation

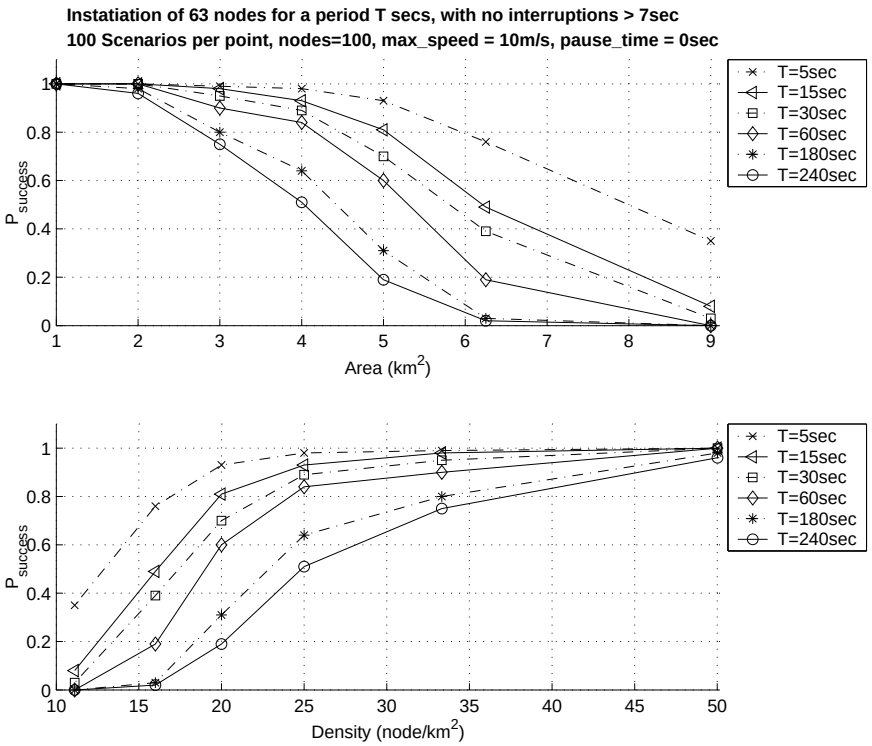

Figure 12: Probability of Success of instantiation

2.4 Probability of Success

To get a sense of what is the probability of successful instantiation of a task graph, we tried to answer the following question:

For 100 nodes going at an average speed of 10m/s, with no pause time, can we instantiate a task graph of size 63 for a period of T seconds with interruptions or gaps no larger than 7 seconds in various areas?

We characterized the behavior within the above described parameters empirically by calculating a probability of success which is an average of success (a 1) or failure (a 0) for a 100 scenarios. We observed a phase transition point around $3km^2$ for most plots (Figure 12). This transition indicates a region where performance starts degrading rapidly making it unfeasible to attempt instantiating a TG-based application.

In Figure 13 we note that the variance is somewhat small which roughly indicates that 100 scenarios are sufficient to produce good confidence in the results. However to really establish a confidence measure in the probability of success measure, one needs to repeat this $100-scenario$ experiment many times and calculate the probability of success for each. The variance among these probabilities is what we really need, however this effort is excessively

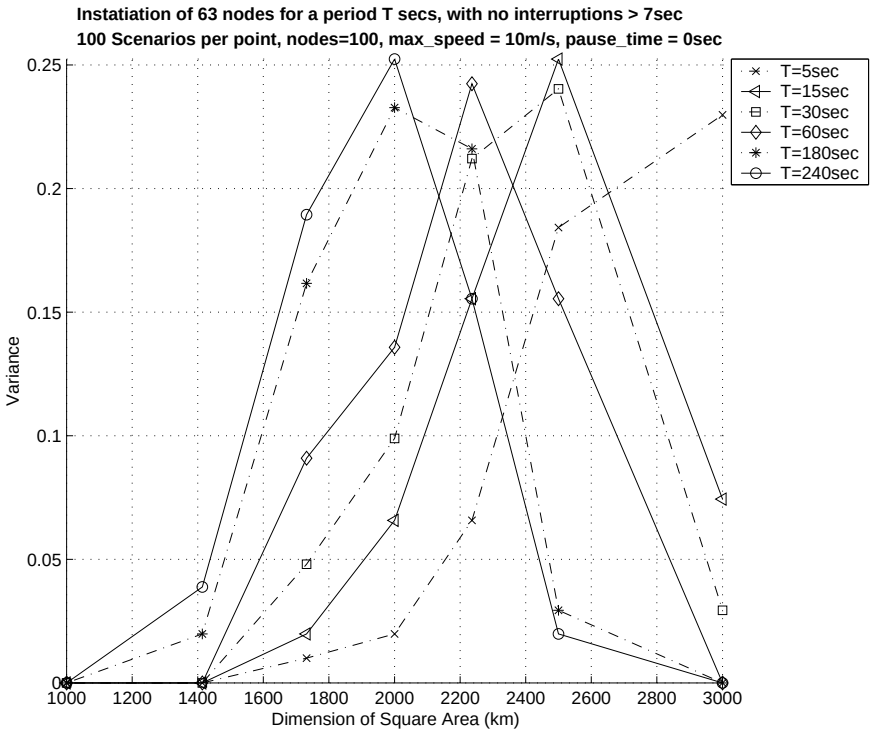

Figure 13: Variance for probability of success for each point over 100 scenarios

time consuming and instead we hope to determine an analytical model based on perculation theory in our future work efforts.

3 conclusions and Future Work

Through these simulations/ experiments, we have gained knowledge about the interplay between TG protocol timers and other timers such as TCP and TTL. We have identified sources of TG instantiation delay and characterized the performance for different size, density and speeds of nodes. We see the connectivity experiments as a stepping stone into understanding the workings of an admission control policy. An interesting behavior that was witnessed in this set of experiments plus else where [1] in our work (static case), was the transition phase shift at roughly the same critical points for a similar set of scenarios. Our future work will focus on using such results to formulate admission control policies for any scenario and TG requirements.

References

[1] P. Basu. *A Task Based Approach for Modeling Distributed Applications on Mobile Ad Hoc Networks*. PhD thesis, Boston University, May 2003.