ENGINEER- Vol. XXXXI, No. 04 pp. 14-21, 2008
© The Institution of Engineers, Sri Lanka

Novel Decentralized Approach to Discover Automobile Congestion

M. Farazy Fahmy and D. N. Ranasinghe

Abstract: Everyone complains about being stuck in traffic. Many people waste their time being in automobile traffic and millions of gallons of fuel are wasted all around the world. Discovering congestion and disseminating the congestion information can assist drivers to take alternate routes. There are several centralized approaches to discover automobile congestion. Centralized approaches have significant problems. It will not be practical to implement centralized systems in all roads and will not be available everywhere. Decentralized approaches like Vehicular Ad hoc Networks (VANET) play a major role in discovering congestion and disseminating congestion information. Provision of more information to drivers will help them decide on the most effective route. Traffic volume is a valuable parameter for drivers to anticipate the state of the congestion. The moderate amount of research that has been done to date on discovering congestion has been based on GPS. The research proposes an alternative to GPS based system to discover congestion and traffic volume. The proposed idea discovers congestion by observing the neighbour beacons and a tree based counting algorithm is used to count the total number of nodes in congestion. Tree based counting algorithms are used in sensor networks. In sensor networks nodes are fixed, where as in VANET nodes have high mobility. Congestion keeps the node fixed and provides the opportunity to run a tree based algorithm. The simulation results show that the proposed idea can be used to discover congestion and to count the number of vehicles in congestion.

Keywords: VANET, Automobile Congestion, Data aggregation, Tree based algorithm

1 Introduction

Automobile traffic congestion is a major problem in cites and highways. Long queues near traffic lights and junctions, along highways etc are a common sight. Everyone complains about being stuck in traffic. Many people waste their time being in traffic and millions of gallons of fuel are wasted all around the world.

Discovering congestion and disseminating the congestion information can assist drivers to take alternate routes. A significant amount of research has been done on disseminating information but very little on discovering congestion [1] [2].

There are several centralized approaches to discover automobile congestion. However, it is not cost effective and sometimes impossible to implement these systems on all roads and will not be available everywhere. Decentralized approaches to discover automobile congestion has been implemented using GPS equipped vehicles [1] [2].

The number of vehicles caught in a probable congestion would be valuable information for the drivers. This paper proposes a method to discover congestion and to count the number of vehicles stuck in congestion. It is assumed that all vehicles would be equipped with wireless devices and these devices listen for the beaconing packets of their neighbours (nodes that can receive each other's beaconing packets) and decide whether they are in congestion. If the device is in congestion it runs the counting algorithm on receiving the beaconing packet. The counting algorithm then develops a tree similar to aggregation algorithms in Sensor networks. In VANET's the root is not defined whereas in a sensor network it is known.

The proposed idea could be a low cost alternative for the GPS based congestion discovering methods. Unlike GPS based systems this will be available everywhere and at all times where as GPS will not work under tunnels and in cloudy weather conditions. GPS could be used in our approach to provide additional and accurate information.

The research developed a simple mobility model that builds up congestion near a traffic light in a single road with two lanes and tested the algorithm in simulation. Figure 1 shows a block diagram of congestion with a node arrival rate of λ and with a node leaving rate of μ.

Eng. M. Farazy Fahmy, B.Sc Eng Electrical & Electronics, AMIE(Sri Lanka), Presently Manager Software IP Cores, Tier Logic (Pvt.) Ltd
Dr. D. N. Ranasinghe, B.Sc. (Elect. Eng.), M.Sc., DIC (Lond), Ph.D.(Cardiff), Senior Lecturer and Head Department of Computational and Intelligent Systems University of Colombo School of Computing

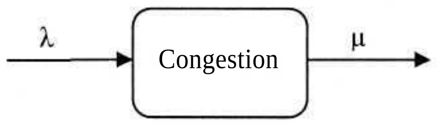


Figure 1 - Typical M/D/1 queueing model

If A = μ	No congestion.
If μ = 0	A traffic block where nodes do not move.
If A > u > 0	Congestion building up

Since the discovery of congestion and counting would take a finite time there should be a threshold value for (A - u). At values below the threshold the counting algorithm will not be successful. Based on additional experiments and simulations threshold values were derived for A and μ.

2 Related Work

Research on VANET has taken many different paths. A reasonable amount of work has been focused on simulation [4]. Information forwarding and exchanging is another area where research is being carried out. Studies have been performed on different forwarding schemes and what information should be forwarded. Whilst more work has been carried out on Multi-hop routing [13] [14], entertainment systems such as multiplayer games or streaming music, very little research on VANET has been directed towards discovering congestion.

TrafficView [1] and StreetSmart [2] address the issue of discovering and disseminating congestion. They use a vehicle based GPS system to discover congestion. In essence a networking car based GPS devices creates a sensor network which measures the congestion of the roads.

Although MANETs typically experience much more topological change than their wired counterparts, nodes are still not completely independent. They may, for example, be grouped due to physical or environmental constraints. These groups of nodes create an opportunity for aggregation similar to that found in wired networks. Automobile congestion will create an opportunity for aggregation.

A significant amount of research has been done on data aggregation algorithms for sensor networks. Tree based aggregation methods are common in sensor networks. Tree based methods construct a routing tree. In most cases sink (node requesting data) becomes the root. Work on Location Based Data Aggregation for Vehicular Networks has been done in [3]. In [3] it is mentioned that *"Due to the above reasons, routing tree-based aggregation is not applicable to the vehicular network"*. Tiny Aggregation Service (TAG) is proposed for aggregation in low-power, distributed, wireless environments [6]. An energy-aware spanning tree construction algorithm (E-Span) is proposed in [7]. E-Span is a distributed protocol. It creates a tree structure which includes all source nodes and contains no cycles. The node with the highest energy is selected as the root. Lifetime-Preserving Tree (LPT) has been proposed in [8].In LPT, nodes which have higher residual energy are chosen as the aggregating parents. LPT also includes a self-healing feature by which the tree will be reconstructed again whenever a node is no longer functional or a broken link is detected. Some recent work proposed the partition of network into small adjacent grids or clusters on which data aggregation is performed. Specifically, a cluster head is associated with each cluster so that all the nodes belonging to the same cluster can aggregate their readings through this node. The work in [5] periodically selects cluster heads according to a hybrid of the residual energy and node degree, resulting in a set of energy-rich cluster heads being uniformly distributed across the network.

The proposed approach develops a routing tree based counting algorithm. Routing tree is developed when the vehicles are in congestion and the node having the largest node Id becomes the root.

3 System Model

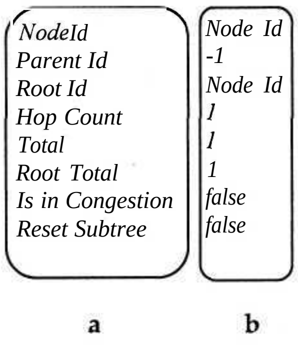
The research has two aspects, discovering congestion and counting the number of vehicles in congestion.

The research proposes a simple mechanism to discover congestion where unlike in TrafficView or StreetSmart it does not use GPS. It is assumed all the vehicles are equipped with short range wireless devices that can sense the vehicle speed and detect whether the vehicle is parked or not. Each node has a unique Id which is an integer and does the beaconing in random intervals. Nodes listen for the neighbours beacons and decide that it is in congestion. Likewise all the nodes decide their congestion state.

Once the node has decided it is in congestion it runs the counting algorithm upon receiving the beacon. The Algorithm develops a tree similar to aggregation algorithm in Sensor Networks. But in VANET's root is unknown.

Algorithm will select the node having the largest Id as the root. Selecting the root and counting happens at the same time. Each node will have the total of its sub tree. So the leaf nodes will have the total as 1 and the root will have the total count. If no beacons are received for the last 10 intervals that neighbour is considered as a lost neighbour and the parent of the lost neighbour will adjust its total and the children will reset the sub tree.

Each node will have its current neighbours list, previous neighbours list and the last set of beacons received from its children. Each beacon will have 8 parameter values as shown in figure 2.



a	b
NodeId	*Node Id*
Parent Id	*-1*
Root Id	*Node Id*
Hop Count	*1*
Total	*1*
Root Total	*1*
Is in Congestion	*false*
Reset Subtree	*false*

Figure 2 - (a) Beacon packet structure. (b) Default values of the beacon

4 Implementation

The counting algorithm can be categorized as an asynchronous distributed algorithm. A node will have different states. Initially it is in an Unknown state and then it comes to Congestion state next it get added to the tree and becomes a Child or a Parent and finally a Root.

4.1 Discovering Congestion

Each device does the beaconing in random intervals with the maximum interval of 6 seconds and a minimum of 3 seconds. When a device gets a beacon it adds that node to its neighbours list. If it does not lose any neighbours for a certain period that node decides that it is in congestion. Eventually, all nodes will come to the conclusion that they are in congestion. In discovering congestion the research has considered two scenarios.

1. Node connecting to an already congested set of nodes
2. All the nodes are new (no node is in congestion initially)

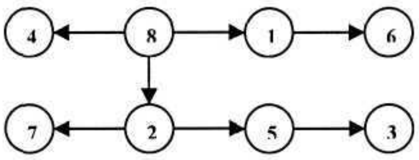
In the second case nodes wait for 40 beaconing intervals to decide their state. In the first case, that is if the new nodes have at least one neighbour in congestion they wait for 2 beaconing intervals before deciding the congestion state.

Algorithm 1: Discovering congestion

```
For each node in previous neighbours list
    Check whether each node exists in the current neighbour list

If all the previous neighbours are in the new neighbours list
    Increase congestion beat count by 1

If all my neighbours are in congestion and congestion beat count > thresh1(=2)
        If not in congestion change the state to congestion

If congestion beat count > thresh2(= 40 )
    If not in congestion change the state to congestion
```

4.2 Constructing the tree

At the start each node will transmit beacons with the default values as shown in Figure 2. When a node receives a beacon with a larger Root Id, it will assign that value to its Root Id and Node Id of the received beacon to its Parent Id. Assigning the Parent Id will attach the node to the tree. Next time when this node transmits the beacon it will transmit the new values. When this runs for a few seconds the tree will be created and every node will have the Root Id and it will be the largest node Id. Circular loops could be built due to high mobility in VANET's. When constructing the tree algorithm eliminates the possibility of building circular loops.



$$\begin{tikzcd}
4 & 8 \arrow[l] \arrow[r] \arrow[d] & 1 \arrow[r] & 6 \\
7 & 2 \arrow[l] \arrow[r] & 5 \arrow[r] & 3
\end{tikzcd}$$

Figure 3 - A sample tree that connect a set of vehicles

4.3 Counting

Counting the nodes also happens at the same time. When a node receives a beacon it will check its Root Id with the received beacons Root Id. If the Root Id of the received beacon is larger, nodes will increase the hop count and assigns the Root total of the received beacon. If a node receives a beacon from the parent with the same Root Id it will update the Root Total. If a node receives a beacon from the child with the same Root Id it will calculate the new total of the node.

Algorithm 2: Constructing tree and counting algorithm that runs when a node receives a beacon

```
if parent says to reset subtree
   assign default values for hopCount, parentId,
     rootId, total and rootTotal
   clear the beacon list
   send beacon
   return

if my childs hop count is greater than my hop count
(checking for loops)
   assign default values for hopCount, parentld,
     rootld, total and rootTotal
   clear the beacon list
   resetSubTree = true
   send beacon
   return

if received beacons rootld > rootld
   rootld = received beacons rootld
   parentld = received beacons nodeld
   total = 1
   hopCount = received beacons hopCount + 1
   rootTotal = received beacons rootTotal
   resetSubTree = false
   if there are any previous beacons remove it from
     the list
   send beacon
   return

if from my child and same rootld
   get the previous beacon from the list
   if there is a previous beacon
       total = total - previous beacon total + current
               beacon total
       replace the new beacon
       if total is different send beacon
   else
       total = total + current beacon total
       add the beacon to the list
       send beacon

if I am the root
   rootTotal = total

if from my parent and same rootld
   rootTotal = received beacons rootTotal
   if rootTotal is different send beacon
```

4.4 Node Leaving the Tree

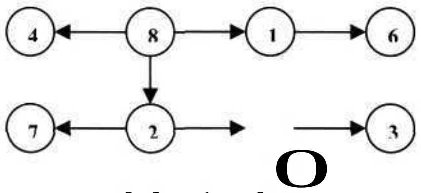


Figure 4 - A node leaving the tree

In figure 4 for example if node 5 leaves, node 3 has lost is parent and node 2 has lost its child. Once the parent discovers that it has lost its child it will subtract that child's total from its total. When a parent is lost child gets disconnected from the tree and it will assign Node Id for Root Id, -1 for Parent Id and resets its sub tree by assigning true to reset SubTree. Child nodes receiving a beacon with resetSubTree = true will assign the default values to the parameters of the beacon and transmits the beacon. So all the nodes below will disconnect from the tree and reconstruction happens when it receives a beacon with a higher root Id.

Algorithm 3: Algorithm that runs when a node lose its neighbor

```
if my child is lost
   total = total - previous beacon total
   if I am the root
       rootTotal = total
   remove the the beacon from the list
   return

if my parent is lost
   if the parent is the root
       lostRoot = rootld
       total = 1
   assing default values for hopCount, total,
   rootTotal, parentld and rootld
   resetSubTree = true
   clear the beacon list
   return
```

5 Simulation

All the work was tested in simulation environment. The Jist/SWANS a general purpose simulation engine was used to run our simulations [9], [10], [11], [12]. A simple mobility model was developed to test our algorithms. In the mobility model the vehicles travel in two straight lanes in the same direction. The vehicles were introduced in exponentially reducing order and a traffic light was operated in order to build up congestion near the traffic light.

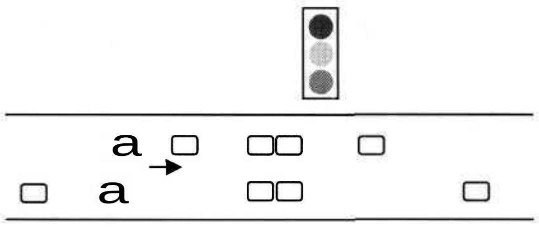


Figure 5 - Simulation Environment

5.1 Simulation Parameters

- Simulation area is defined as 50m width and 2000 m length.
- Distance between lanes is 3 m
- The communication range is 10 m
- Minimum distance between two radio devices 5 m
- Beaconing interval 3 to 6 seconds
- Congestion discovering time is 40 beaconing intervals, that is approximately 3 minutes (120 sec - 240 sec)

6 Simulation Scenarios

The research carried out the simulations considering three different scenarios.

6.1 Scenario 1(\ > 0 and u = 0)

Static Congestion - in this scenario node arrives at the congestion at a particular arrival rate but nodes do not leave the congestion. Time taken to count the total number of nodes after the last node has detected it's in congestion is calculated.

6.2 Scenario 2

In this scenario we introduced different number of nodes and built up the congestion with node leaving rate zero (μ = 0). After all the nodes decided that they are in congestion and counted the total two nodes were allowed to leave the congestion and time taken to count the new total was measured.

6.3 Scenario 3

In this scenario node arrival rate and node leaving rate were varied to find out a possible optimal arrival rate and leaving rate for the algorithm to count the number of nodes successfully.

In the simulation nodes were introduced in,

1. Ascending order of node Id *(Ascending)*
2. Descending order of node Id and *(Descending)*
3. Middle node having the largest node Id. *(Mid)*

7 Simulation Results

7.1 Scenario 1(\ > 0 and M = 0)

Table 1 presents the time taken to count the total number of nodes after the last node has decided it's in congestion.

Table 1: Time taken to count the nodes in scenario 1

Number of Nodes	*Time to Count /(s)*		
	Ascending	*Descending*	*Mid*
10	8.29014	6.92554	5.8841
100	68.1377	32.4640	32.4491
200	86.8807	59.4489	57.4471
300	125.6271	79.5016	78.0087
400	222.1632	114.1569	96.1674
500	246.7267	142.8748	113.549

Figure 6 show that the ascending order of node Id's take more time to count. Largest node Id in the middle takes less time. This is because in ascending order last node is the largest node Id, the counting algorithm has to create the whole tree and then do the counting where as in other two cases it has to only add the last node to the tree.

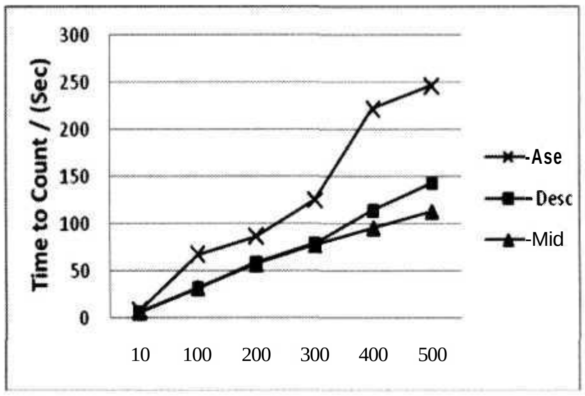


Figure 6 - Time taken to count the nodes in Scenario 1

According to our results in the worst case the algorithm takes less than 250 seconds- to count 500 nodes. In *Mid* order of nodes Id's takes less than 120 seconds to count 500 nodes which we can think would be a likely node arrangement in real life.

7.2 Scenario 2

Table 2 shows that the time taken to count the total number of nodes, after two nodes have left the congestion.

Table 2 - Time taken to count the nodes in scenario 2

Number of Nodes	*Time to Count/(s)*		
	Ascending	*Descending*	*Mid*
10	4.7271	8.7080	1.7173
100	39.0761	50.266	16.9694
200	67.8049	102.4848	32.9563
300	121.6634	167.6030	62.5276
400	152.8112	308.9187	92.3081
500	227.4433	320.1912	100.657

According to figure 7 the descending order of node Id's takes more time to count. This is because the leaving nodes will be the nodes having the largest node Id. So, it has to reset the tree, construct and count the nodes. In the other two cases leaving nodes are the child nodes and it's a matter of deducting the two nodes from the total. Largest node Id in the middle takes less time than the other because the number of hops to the root is almost half compared to the ascending order of nodes.

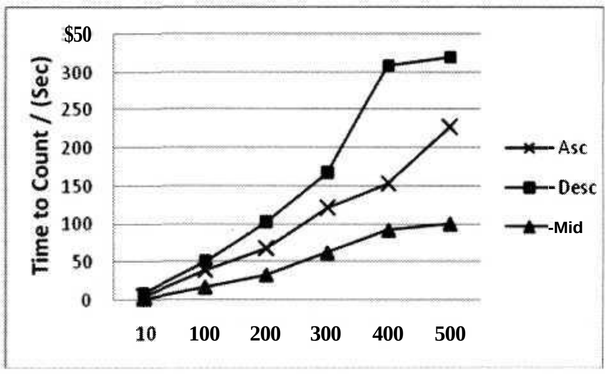


Figure 7 - Time taken to count the nodes in Scenario *2*

According to Table *2,* the algorithm has taken about 320 seconds to count 498 nodes in the worst case after two nodes have left the congestion. In the best case it has taken less than 120 seconds to count 498 nodes.

7.3 **Scenario 3**

To find approximate threshold values for X and μ we performed experiments with 200 nodes. We operated the traffic light, allowed congestion to build up near the traffic light and varied the red and green light intervals. Simulations were done with two different congestion discovery intervals.

The following threshold values for successful counting were derived considering 40 beaconing intervals to discover congestion.

X - 36 nodes per min
μ - 5 nodes per min

In the simulation congestion started to build up in the 9th minute (540 second), counting started in the 11th minute (700 second) and the algorithm counted around 70 nodes in the 12th minute (760 second).

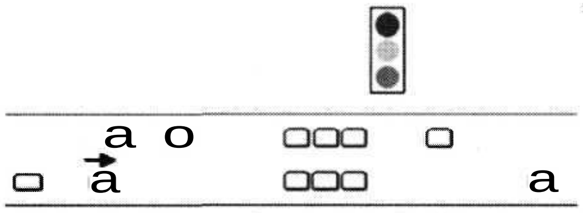


Figure 8 - Congestion starts to build up

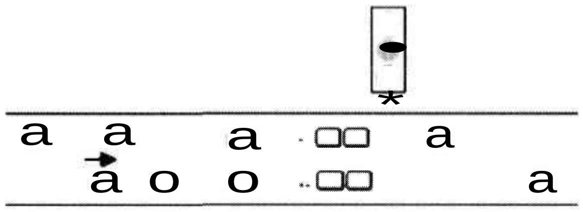


Figure 9 - Counting starts in about 150 seconds

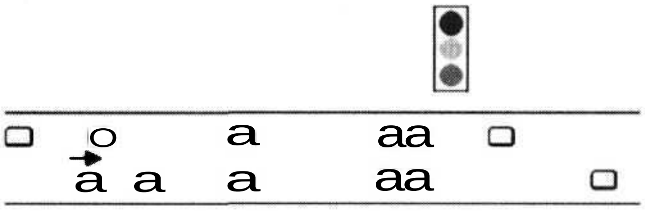


Figure 9 - Algorithm counts 70 nodes in about 60 seconds

Also we tried to determine the threshold values for lower congestion discovery time that is considering 6 beaconing intervals to discover congestion. Following parameters were obtained for successful counting.

X - 36 nodes per min
μ - 24 nodes pre min

If we analyze the two cases we can see that with lesser congestion discovery time it can detect even a very small (slowly developing) congestion.

7.4 **Message Complexity**

Message complexity depends on the number of hops, the order in which the nodes got connected to the tree, the order of the node ids and the communication range. For example if we take two nodes they will require five messages to count the total as two. The five messages required for the counting are as follows.

- 2 messages to discover the neighbour
- 1 message to get connect to the tree
- 1 message for the parent to count the total
- 1 message to notify the child the total

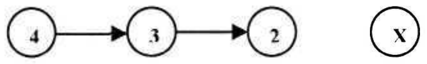


Figure 10 - New node X get connected to 3 nodes who are already connected and know their total.

For example let's consider counting node X in Figure 8 where *X* is less than 4. It will require two messages to discover neighbour, one message to get connect to tree, three messages to count the total to the root and three messages to notify the exact total. In aggregate, it will require 9 messages to count the new total with node *X, if X < 4.* If X> 4, two messages to discover neighbour and about 15 messages to arrange the tree, perform the counting and the notification.

7.5 Time Complexity

The algorithm that runs on receiving a beacon will have to get the previous message from its message list. So the time depends on the size of the message list or in other words number of neighbours. Therefore this is a linear time algorithm with O(n) complexity. If the neighbour size is very small we can think it as a constant time algorithm. In our case the neighbour size is around 3 – 5, we can approximate the time complexity as O(l). The algorithm that runs on leaving a node will also have the same time complexity as above.

8 Conclusion & Future work

VANET based discovering is more flexible compared to centralized congestion discovering mechanisms which is available only in specific places like traffic lights and junctions.
This paper proposes a new VANET based method of discovering congestion and counting the number of vehicles in congestion. Our simulation results have shown that the proposed idea can be used in real life. A simple mechanism was used to discover congestion which can be developed further and through simulation we have shown that a tree based algorithm can be used to count the number of vehicles in congestion. For example if a vehicle leaves a congestion of 500 vehicles (congestion of about 2500 meters) it takes only about 100 seconds to recount the total which is practically feasible in real life.
Proposed idea can be further improved by improving the equipment fixed in the vehicle. Equipment with the ability to sense the speed and the parked status of the vehicle would be ideal. Without this kind of device a block of vehicles moving more or less together would erroneously detect as congestion. Also it will help to detect the vehicles parked in a parking area. The proposed idea with this kind of equipment would give better results.

Acknowledgement

We are grateful to Thabotharan Kathiravelu, a research student at Uppsala University for helping us on directing towards the JiST/SWANS simulator, and helping in installing the simulator.

References

1. Tamer Nadeem, Sasan Dashtinezhad, Chunyuan Liao, "TrafficView: Traffic Data Dissemination using Car-to-Car Communication"

2. Sandor Dornbush, Anupam Joshi, "StreetSmart Traffic: Discovering and Disseminating Automobile Congestion Using VANET's"

3. Chai Chen, "Location Based Data Aggregation in Mobile Ad hoc Networkds"

4. David Choffnes and Fabin E. Bustamante. "Straw - an integrated mobility and traffic model for vanets. (June 2005)."

5. C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of Network Density on Data Aggregation in Wireless Sensor Networks," in Proceedings of 22nd International Conference on Distributed Computing Systems, Jul. 2002, pp. 457-458.

6. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks," in Proceedings of the 5th symposium on Operating systems design and implementation, Dec. 2002, pp. 131-146.

7. M. Lee and V. Wong, "An energy-efficient spanning tree algorithm for data aggregation in wireless sensor networks," in Proc. of IEEEPacRim'05, Victoria, BC, Aug. 2005.

8. Mac Lee and Vincent W. S. Wong, "LPT for Data Aggregation in Wireless Sensor Networks"

9. Rimor Barr, "JiST - Java in simulation Time User Guide"

10. Rimon Barr, "SWANS - Scalable Wireless Ad hoc Network Simulator User Guide"

11. Rimon Barr,Zygmunt J. Haas, Robbert, Renessel "JiST: An efficient approach to simulation using virtual machines"

12. Rimon Barr,Zygmunt J. Haas, Robbert, Renessel "Scalable Wireless Ad hoc Network Simulation"

13. M. Dikaiakos, S. Iqbal, T. Nadeem, and L. Iftode. Vitp: "An information transfer protocol for vehicular computing"

14. Z. Chen, H. Kung, and D. Vlah. Ad hoc relay wireless networks over moving vehicles on highways.