Exploiting Opportunistic Overhearing to Improve Performance of Mutual Exclusion in Wireless Ad Hoc Networks

Ghazale Hosseinabadi and Nitin H. Vaidya

Department of ECE and Coordinated Science Lab.
University of Illinois at Urbana-Champaign
Urbana, IL, 61801, USA.
{ghossei2,nhv}@illinois.edu

Abstract. We design two mutual exclusion algorithms for wireless networks. Our mutual exclusion algorithms are distributed token based algorithms which exploit the opportunistic message overhearing in wireless networks. One of the algorithms is based on overhearing of token transmission. In the other algorithm, overhearing of both token and request messages is exploited. The design goal is to decrease the number of transmitted messages and delay per critical section entry using the information obtained from overheard messages.

Keywords: Wireless networks; opportunistic overhearing; mutual exclusion.

1 Introduction

A wireless ad hoc network is a network in which a pair of nodes communicates by sending messages over wireless links. Wireless ad hoc networks have fundamentally different characteristics from wired distributed networks, mainly because the wireless channel is a shared medium and messages sent on the wireless links might be overheard by the neighboring nodes. The information obtained from the overheard messages can be used in order to design distributed algorithms, for wireless networks, with better performance metrics. Although existing distributed algorithms will run correctly on top of wireless ad hoc networks, our contention is that efficiency can be obtained by developing distributed algorithms, which are aware of the shared nature of the wireless channel. In this paper, we present distributed mutual exclusion algorithms for wireless ad hoc networks.

Distributed processes often need to coordinate their activities. If a collection of processes share a resource or collection of resources, then often *Mutual Exclusion (MUTEX)* is required to prevent interference and ensure consistency when accessing the resources. In a *distributed* system, we require a solution to *distributed* mutual exclusion. Consider users who update a text file. A simple means of ensuring that their updates are consistent is to allow them to access

the file only one at a time, by requiring the editor to lock the file before updates can be made. A particularly interesting example is where there is no server, and a collection of peer processes must coordinate their access to shared resources amongst themselves.

Mutual Exclusion is a well known problem in distributed systems in which a group of processes require entry into the *critical section (CS)* exclusively, in order to perform some critical operations, such as accessing shared variables in a common store or accessing shared hardware. Mutual exclusion in distributed systems is a fundamental property required to synchronize access to shared resources in order to maintain consistency and integrity. To achieve mutual exclusion, concurrent access to the CS must be synchronized such that at any time only one process can access the CS. The proposed solutions for distributed mutual exclusion are categorized into two classes: token based [1], [2], [3] and permission based [4], [5], [6]. In token based MUTEX algorithms, a unique token is shared among the processors. A processor is allowed to enter the CS only if it holds the token. In a permission based MUTEX algorithm, the processor that requires entry into the CS must first obtain the permissions from a set of processors.

In this paper, we design mutual exclusion algorithms for wireless networks. Most of the existing MUTEX algorithms are designed for typical wired networks [1],[2],[4],[5],[6]. Design of mutual exclusion algorithms for mobile ad hoc networks had received some interest in the past few years [3], [7]. Although the underlying network in these algorithms is wireless, the proposed algorithms are only mobility aware solutions, where the goal is to deal with the problems caused by node mobility, such as link failures and link formations. In this work, we show that the broadcast property of the wireless medium can be exploited in order to improve the performance of the MUTEX algorithms in wireless networks. To the best of our knowledge, this work is the first in which opportunistic overhearing is exploited to improve the performance of MUTEX in wireless networks.

In this work, we present two token based mutual exclusion algorithms that are designed for wireless networks. Network nodes communicate by transmitting unicast messages. Since the channel is wireless, a unicast message transmitted from node i to node j might be overheard by neighbors of node i, for example node k. In this case, node k is not the intended receiver of the message, but it has overheard the message due to the shared nature of the wireless medium. We design our algorithms such that the neighboring nodes that overhear messages can learn more recent information about the current status of the algorithm.

We call our algorithms *Token Overhearing Algorithm (TOA)* and *Token and Request Overhearing Algorithm (TROA)*. *TOA* is based on the MUTEX algorithm designed by Raymond [1]. In Raymond's algorithm, messages are transmitted over a static spanning tree of the network. *TOA* is based on overhearing of the token transmission and the spanning tree maintained by the algorithm changes when token transmission is overheard by the neighboring nodes. *TROA* is based on Trehel-Naimi's algorithm [2]. In Trehel-Naimi's algorithm, when a node requires entry to the CS, the node sends a request message to the last known owner of the token. In *TROA*, overhearing of both request and token

messages are exploited in order to obtain recent information about the latest token holder in the network. The performance metrics that we aim to improve in this work are the number of transmitted messages and delay per CS entry. Our mutual exclusion algorithms satisfy three correctness properties: 1) *Mutual Exclusion*: at most one processor is in the CS at any time; 2) *Deadlock free*: if any processor is waiting for the CS, then in a finite time some processor enters the CS; 3) *Starvation free*: if a processor is waiting for the CS, then in a finite time the processor enters the CS.

The remainder of this paper is organized as follows: We first describe the network model in Section 2. In Section 3, we present *TOA*. *TROA* is described in Section 4. Simulation results are presented in Section 5.

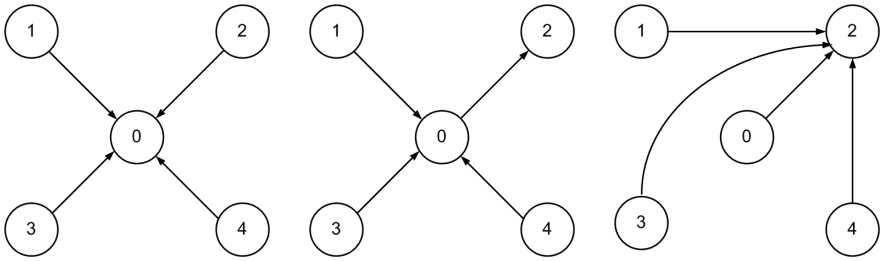
2 Network Model

We consider a network of n nodes, communicating by message passing in a wireless ad hoc network. Each node has a unique identifier, i, $0 \leq i \leq n-1$. Messages transmitted in the network are unicast messages. We assume that lower layers of the network, such as MAC layer and transport layer, ensure reliable delivery of unicast messages. To ensure reliability, retransmission mechanism is used in lower layers in case packets are lost due to noise or interference. Since the network is wireless, a unicast message from node i to node j might be overheard by neighbors of node i, such as node k. We assume that if such an opportunistic overhearing happens, node k does not discard the overheard message; instead it uses the information included in the message. We do not assume that the unicast message of node i to node j is delivered reliably to the neighbors of node i. Instead, the overhearing is opportunistic, meaning that if the neighboring nodes overhear messages not intended for them, they exploit the information included in the messages. We assume that network nodes do not fail and each node is aware of the set of nodes with which it can directly communicate.

3 Token Overhearing Algorithm (TOA)

Token Overhearing Algorithm (TOA) is based on Raymond's algorithm [1]. Raymond designed a distributed token based mutual exclusion algorithm in which requests are sent over a static spanning tree of the network, towards the token holder. The tree is maintained by logical pointers distributed over the nodes and directed to the node holding the token. At each time instance, there is a single directed path from each node to the token holder. When a node has a request for the token, a sequence of request messages are sent on the path between the requesting node and the token holder. The token is sent back over the reverse path to the requesting node. The direction of the links over which the token is transmitted is reversed. In this way, at each time instance, all edges of the tree point towards the token holder.

Similar to Raymond's algorithm, *TOA* uses a spanning tree of the network over which messages are passed. But unlike Raymond's algorithm, the spanning



(a) Initial tree (b) Raymond's algorithm (c) TOA

Fig. 1. Example execution of Raymond's algorithm and *TOA*

tree in *TOA* is dynamic and changes if token transmission is overheard by the neighboring nodes. Sender and receiver of the token are specified in the token message. When token is sent from node i to node j, any other node k that overhears transmission of the token, changes its parent in the tree.

3.1 Example of Algorithm Operation

An example execution of Raymond's algorithm and *TOA* is illustrated in Figure 1. The network is a wireless ad hoc network composed of five nodes, node 0-4. We assume that the network is single-hop, in which all nodes are in the communication range of each other. Figure 1(a) shows the initial spanning tree of the network, where the token holder is 0. We consider a case where 2 requires entry to the CS and sends a request message to 0. We assume that there is no other pending request in the network. When 0 receives the request of 2, it sends the token to 2. Figure 1(b) shows the spanning tree in Raymond's algorithm after the token is sent to 2. At this point, the direction of the edge between 0 and 2 is reversed. In Figure 1(b), nodes 1, 3 and 4 are two hops away from 2. If any of these nodes, for example node 1, requests to enter the CS, two request messages are sent, one request message from 1 to 0 and one from 0 to 2. It then takes two messages to send the token from 2 to 1. So, total of four messages are sent so that 1 can enter the CS.

We now describe how *TOA* performs when the nodes are initially configured as depicted in Figure 1(a) and node 2 requires CS entry. Like Raymond's algorithm, when 0 receives the request of 2, it sends the token to 2. Since all nodes are in the communication range of each other, token transmission from 0 to 2 might be overheard by 1, 3 and 4. We consider the best scenario for our algorithm, in which all nodes 1, 3 and 4 overhear the token transmission. As a result, 1, 3 and 4 point to 2. Figure 1(c) shows the spanning tree in our algorithm when the token is sent to 2. In this figure, nodes 1, 3 and 4 are only one hop away from the token holder, node 2. If any of these nodes requests entry to the CS, only two messages are transmitted, one request message and one token message. This example shows that in single-hop wireless networks and when requests for the token are initiated separate enough in time, *TOA* might perform better than

Raymond's algorithm. The reason is that as a result of messages overhearing, nodes might be aware of the current token holder in which case they send their requests directly to the token holder. In single hop networks, if every node overhears token transmission, *TOA* is optimal and only two messages, one request message and one token message, are transmitted per CS entry. On the other hand, in Raymond's algorithm four messages might be sent for one CS entry, in single-hop networks. The reason is that, although the token holder and the requesting node are in the communication range of each other, they might not communicate directly, rather they exchange messages through the initial root (e.g. node 0 in this example), simply because Raymond's algorithm uses a static spanning tree.

3.2 Overview of Token Overhearing Algorithm (TOA)

Token Overhearing Algorithm (*TOA*) is based on Raymond's algorithm [1], which is a well-known MUTEX algorithm. Due to the lack of space, we do not present the details of Raymond's algorithm here. We just describe our modification to Raymond's algorithm which is *OverhearToken* (procedure 3.2.1). *OverhearToken* is executed when a node k overhears the transmission of `TOKEN` from *sender* to *receiver*. In this case, k is not the intended receiver of the message, but it has overheard the message. $parent_k$ in the tree becomes *receiver* if k and *receiver* are immediate neighbors, otherwise k chooses *sender* as its parent.

3.2.1 *OverhearToken*

```
1: if type is TOKEN then
2:   if receiver is my neighbor then
3:     parent = receiver of the message
4:   else
5:     parent = sender of the message
```

TOA has three correctness properties; safety, deadlock free and lockout free. The proofs of correctness are omitted here due to the lack of space.

4 Token and Request Overhearing Algorithm (TROA)

We design another mutual exclusion algorithm, called *Token and Request Overhearing Algorithm (TROA)* for wireless networks. *TROA* is based on Trehel-Naimi's algorithm [2]. The objective in designing *TROA* is to find a MUTEX algorithm in which overhearing of both token and request messages is exploited in order to improve the performance. We note that *TOA* is only based on overhearing of token transmission.

Trehel-Naimi's algorithm [2] is a token-based algorithm which maintains two data structures: (1) A dynamic tree structure in which the root of the tree is the last node that will hold the token among the current requesting nodes. This tree is called the *last* tree. Each node i has a local variable *last* which points to the

last probable token holder that node i is aware of. (2) A distributed queue which maintains requests for the token that have not been answered yet. This queue is called the *next* queue. Each node i keeps the variable *next* which points to the next node to whom the token will be sent after i releases the CS.

In Trehel-Naimi's algorithm, when a node i requires entry to the CS, it sends a request to its *last* and then changes its *last* to `null`. As a result, i becomes the new root of the *last* tree. When node j receives the request of node i, one of these cases happens: 1) j is not the root of the tree. It forwards the request to its *last* and changes its *last* to i. 2) j is the root of the tree. If j holds the token, but does not use it, it sends the token to i. If j is in the CS or is waiting for the token, j sets its *next* to i. Whenever j exits the CS, it sends the token to $next = i$.

Trehel-Naimi's algorithm is designed for wired networks in which transmitted messages are not overheard by the neighboring nodes. We modify the algorithm to perform better in wireless networks by exploiting the broadcast property of wireless networks. In *TROA*, nodes can learn more recent information about the last token holder in the network by overhearing of messages not intended for them.

4.1 Data Structures, Messages and algorithm procedures

Since *TROA* and Trehel-Naimi's algorithm are different from each other in so many ways, we describe the details of *TROA* in this section. In *TROA*, each node maintains the following data structures:

- *privilege*: *privilege* is `true` if the node holds the token, and `false` otherwise.
- *requestingCS*: when a node initiates request for the token, its *requestingCS* is set to `true`. *requestingCS* becomes `false` when the node releases the CS.
- *last*: when a node wants to enter the CS, it sends a request to its *last*. *last* of a node might change when the node receives or overhears messages.
- *next*: When a node that is waiting for the token receives a request message from another node, it saves the *initiator* of the request message in its *next*. Later, when the node releases the CS, it sends the token to *next*.
- *numCSEntry*: *numCSEntry* of node i denotes how many times CS entry has happened in the network such that node i is aware of.
- *numReceivedRequests*: it denotes how many REQUEST messages are received by a node while the node was waiting for the TOKEN.

numCSEntry and *numReceivedRequests* are used as counters to determine if a node should change its *last* when it overhears messages. We will present more details later, when we describe algorithm procedures.

There are two types of messages in the algorithm, `REQUEST` and `TOKEN`. A `REQUEST` message includes the following information.

- *initiator*: it is the id of the node that has initiated the request for the token.

– *destination*: it denotes the final destination of the message. Since we consider the general case of multi-hop networks, *destination* is not necessarily a neighbor of *initiator*. In this case, the message is routed on the shortest path between *initiator* and *destination*.
– *numberCSEntry*: when a node transmits a message, it writes its *numCSEntry* in *numberCSEntry* part of the message. *numberCSEntry* is used by nodes that overhear the message to determine if their *last* should be changed or not.

A message of type TOKEN includes the following information.

– *destination*: it denotes the final destination of the token.
– *numberCSEntry*: As we explained before, when a node transmits a message, it includes its *numCSEntry* in *numberCSEntry* part of the message.

We now present the procedures of *TROA*.
Initialization: Procedure 4.2.1 is executed at the beginning of the algorithm by every node i to set the initial value of i's data structures.
RequestCS: Procedure 4.2.2 is called when a node wants to enter the CS. If the node holds the token, it enters the CS. otherwise, it sends a REQUEST to its *last*.

4.2.1 *Initialization*

```
1: last = INITIAL-TOKEN-HOLDER
2: next = null
3: requestingCS = false
4: numReceivedRequests = 0
5: numCSEntry = −1
6: if last == myId then
7:    privilege = true
8:    last = null
9:    numCSEntry = 0
10: else
11:   privilege = false
```

4.2.2 *RequestCS*

```
1: requestingCS = true
2: if (privilege == false) then
3:    send REQUEST to last
4:    last = null
5: else
6:    enter CS
```

OverhearRequest : When a REQUEST message from node i to node j is overheard by node k, Procedure 4.2.3 is executed, in which if some conditions hold, node k changes its *last* to *initiator* of the message.

4.2.3 *OverhearRequest*

```
1: if numberCSEntry > numCSEntry+numReceivedRequests+1 and last ! =
   null and requestingCS == false then
2:    last = initiator
3:    numCSEntry = numberCSEntry−1
4:    numReceivedRequests = 0
```

OverhearToken : When node k overhears the transmission of `TOKEN` from node i to node j, Procedure 4.2.4 is executed.

4.2.4 *OverhearToken*

```
1: if numberCSEntry > numCSEntry + numReceivedRequests and last != null and requestingCS == false then
2:    last = destination
3:    numCSEntry = numberCSEntry
4:    numReceivedRequests = 0
```

ReceiveToken : Procedure 4.2.5 is executed when `TOKEN` is received at its final destination, *destination*. Intermediate nodes on the path that forward the message do not run this procedure.
ReleaseCS: Procedure 4.2.6 is executed when a node exits the CS.

4.2.5 *ReceiveToken*

```
1: privilege = true
2: numCsEntry = numberCsEntry +1
3: numReceivedRequests = 0
4: enter CS
```

4.2.6 *ReleaseCS*

```
1: requestingCS = false
2: if next != null then
3:    privilege = false
4:    send TOKEN to next
5:    next = null
```

ReceiveRequest : Procedure 4.2.7 is executed when a `REQUEST` message is received at its final destination, *destination*.

4.2.7 *ReceiveRequest*

```
1: if last==null then
2:    if requestingCS==true then
3:       next=initiator
4:    else
5:       privilege=false
6:       send TOKEN to initiator
7: else
8:    send request to last
9:    numReceivedRequests++
10: last=initiator
```

TROA has three correctness properties; safety, deadlock free and lockout free. The proofs of correctness are omitted here due to the lack of space.

5 Simulations

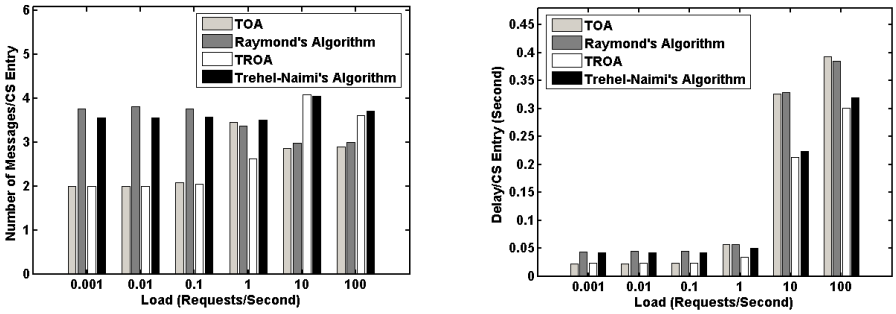
We run simulations to measure the performance of *TOA* and *TROA*. We also simulate Raymond's algorithm and Trehel-Naimi's algorithm to find the improvements obtained by message overhearing. In our simulations, network nodes

are placed uniformly at random in a square area. The node closest to the center of the area is chosen as the initial root of the tree. Messages sent in the network are unicast messages. In order to implement message overhearing, we change the 802.11 MAC layer of ns-2. In the current implementation of ns-2, packets that are received in the MAC layer of node i with MAC destination address different from i's MAC address are dropped. We change ns-2 so that such packets are not dropped, and they are delivered to the application layer of node i. In this work we measure two performance metrics, which we call the cost of the algorithms: 1) Number of messages per CS entry: it is equal to the number of messages transmitted in the network per entry to the CS. 2) Delay per CS entry: the delay is measured as the interval between the time at which a node initiates a request to enter the CS and the time at which the node enters the CS.

Requests for CS entry are assumed to arrive at a node according to a Poisson distribution with rate λ requests/second. When λ is small, no other processor is in the CS when a processor makes a request to enter the CS. In this case, the network is said to be lightly loaded. When λ is large, there is a high demand for entering the CS which results in queueing up of the requests and the network is said to be heavily loaded. The time to execute the CS is 10^{-5} second. Figures 2-4 plot number of messages and delay per entry to the CS against λ in three example networks. λ increases from 10^{-3} to 10^{2} requests/second. Each point in Figures 2-4 is obtained by taking the average of 10 runs of the algorithms. In each run, total number of entry to the CS is $5 * n$, where n is number of nodes in the network. In other words, each point in Figures 2-4 corresponds to the average cost of $50 * n$ entry to the CS.

Figure 2 plots the cost of the algorithms against λ, in a single-hop network. The network is composed of $n = 20$ nodes placed uniformly at random in an area of $100m \times 100m$. Carrier sense range is $250m$. In such a scenario, each node is an immediate neighbor of every other node. We observe that in Figure 2(a), *TOA* outperforms Raymond's algorithm, when λ is small (i.e., under light demand for the token). In single-hop networks and for small λ, approximately 4 messages are transmitted per CS entry in Raymond's algorithm while 2 messages per CS entry are transmitted in *TOA* (as explained in Section 3.1). Figure 2(b) shows that the delay per CS entry is smaller in *TOA* than in Raymond's algorithm, under light demand for the token. Under light demand for the token, when node i makes a request to enter the CS, no other message is transmitted in the network except the messages correspond to the request of node i. In this case, the delay per CS entry is equal to the time required to transmit request and token messages between the requesting node and the token holder, and the wireless channel is available whenever a node wants to transmit a message; i.e. there is no contention in the network.

Raymond's algorithm is designed such that, under heavy demand for the token, constant number of messages (approximately 3) are transmitted per CS entry. Detailed explanation can be found in [1]. In Figure 2(a) we observe what we expected, meaning that under heavy demand approximately 3 messages are transmitted in both Raymond's algorithm and *TOA*. As Figure 2(b) shows,



(a) Number of Messages (b) Delay

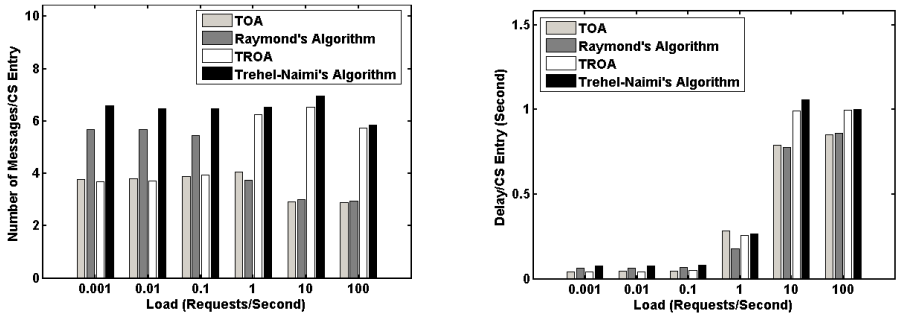
Fig. 2. 20 nodes placed in 100mx100m

both *TOA* and Raymond's algorithm has the same delay when λ is large, simply because both algorithms transmit the same number of messages per CS entry. Delay increases as λ increases, because at each time instant, there is more than one node requiring access to the channel and so packet of a node might be delayed by other nodes that are using the channel.

As Figure 2 shows the cost of *TOA* is half of the cost of Raymond's algorithm under light demand. Under heavy demand both algorithms perform approximately the same. We conclude that the cost is decreased by opportunistic overhearing when demand for the token is light.

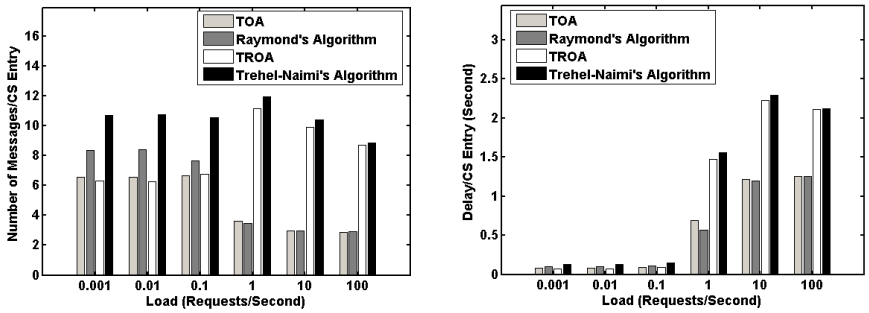
Figure 2 also plots the cost of *TROA* and Trehel-Naimi's Algorithm. As Figure 2(a) shows, in *TROA* two messages are transmitted per CS entry under light demand. The reason is that in *TROA*, nodes send their request to the last token holder, which is known to them because of message overhearing. So, only two messages, one request message and one token message, are transmitted per every CS entry. On the other hand, in Trehel-Naimi's algorithm, a requesting node does not necessarily know which node is the last token holder, since a node does not receive messages exchanged between other nodes. In such a case, a sequence of request messages are transmitted until the request of the requesting node is received by the token holder. We conclude that under light demand, cost of *TROA* is less than the cost of Trehel-Naimi's Algorithm.

Figure 2(a) shows that when demand for the token increases, the number of messages transmitted in *TROA* increases. The reason is that requests for the token from different nodes are initiated close to each other, and so a node might not know the latest status of the algorithm when it initiates a request. For example, we consider a case where node i sends a request to the token holder, node j. If another node k initiates a request before it overhears the request of node i, node k sends its request to node j which is not the last requesting node any more. Node j will forward the request of node k to node i and so one extra request message is transmitted. Figure 2(b) shows that delay per CS entry in *TROA* is always less than Trehel-Naimi's algorithm, when λ is small, simply because fewer messages are transmitted in *TROA*. When



(a) Number of Messages (b) Delay

Fig. 3. 40 nodes placed in 500mx500m



(a) Number of Messages (b) Delay

Fig. 4. 60 nodes placed in 800mx800m

λ increases, delay of both *TROA* and Trehel-Naimi's algorithm increases, as a result of contention between nodes on accessing the wireless channel. Considering Figure 2, we conclude that in single hop networks, opportunistic overhearing improves the performance the most when demand for the token is light, and the improvement is about 100%.

Figures 3 and 4 plot the cost of the algorithms in multi-hop networks. Figure 3 plots the cost in a network of 40 nodes placed randomly in an area of $500m \times 500m$. The underlying network in Figure 4 is 60 nodes placed randomly in an area of $800m \times 800m$. Figure 3 shows that in this network topology, under light demand for the token (small λ), the cost of *TOA* is less than the cost of Raymond's algorithm. Comparing *TOA* and Raymond's algorithm in Figure 2 and Figure 3, we observe that the improvement obtained by message overhearing has decreased in Figure 3. The reason is that in a multi-hop network, nodes do not overhear all transmitted messages in the network and so they are not able to learn the latest status of the algorithm, i.e. they might not know which node currently holds the token. As Figure 3 shows and as we explained before, under heavy demand (large λ), Raymond's algorithm and *TOA* has almost the

same cost. Figure 3(b) shows that the delay of Raymond's algorithm and *TOA* increases when λ increases. This is because of the contention between nodes in accessing the wireless channel.

As we observe in Figure 4, the cost of *TOA* is still less than the cost of Raymond's algorithms, although improvement percentage has decreased. This shows that in Raymond's algorithm, as the size of the network increases, the improvement percentage obtained by exploiting message overhearing decreases. As Figures 3 and 4 show, in these networks when λ is small, *TROA* outperforms Trehel-Naimi's algorithm and the improvement is still significant. We conclude that the effect of exploiting message overhearing in different MUTEX algorithms is not always the same; instead it highly depends on the design of the algorithm.

6 Conclusion

We design two distributed token based mutual exclusion algorithms for wireless networks, called *TOA* and *TROA*. Our algorithms exploit the shared nature of the wireless channel in which nodes can overhear the messages not intended for them. We measured the performance of our algorithms as well as Raymond's algorithm and Trehel-Naimi's algorithm through simulations in ns-2, in networks of different sizes and under various rates of the demand for the token. We discussed under what conditions the performance of the considered MUTEX algorithms is improved by exploiting message overhearing.

Acknowledgments. This work was supported in part by Boeing and the authors would like to thank Boeing.

References

1. Raymond, K.: A tree-based algorithm for distributed mutual exclusion. ACM Trans. Comput. Syst. vol. 7, pp. 61–77 (1989)
2. Naimi, M., Trehel M.: A distributed algorithm for mutual exclusion based on data structures and fault tolerance. In: Conference Proceedings of Sixth Annual International Phoenix Conference on Computers and Communications, pp. 33–39 (1987)
3. Walter, J., Welch, J., Vaidya, H.: A mutual exclusion algorithm for ad hoc mobile networks. In: Wireless Networks, vol. 7, pp. 585–600 (2001)
4. Manivannan, D., Singhal, M.: An efficient fault-tolerant mutual exclusion algorithm for distributed systems. In: Proceedings of the International Conference on Parallel and Distributed Computing Systems, pp. 525–530 (1994)
5. Agrawal, D., Abbadi, A.: An efficient and fault-tolerant solution for distributed mutual exclusion. In: ACM Trans. Comput. Syst., vol. 9, pp. 1–20 (1991)
6. Singhal, M.: A heuristically-aided algorithm for mutual exclusion in distributed systems. In: IEEE Transactions on Computers, vol. 38: pp. 651–662 (1989)
7. Wu, W., Cao, J., Raynal, M.: A dual-token-based fault tolerant mutual exclusion algorithm for manets. In: Proceedings of the 3rd international conference on Mobile ad-hoc and sensor networks, pp. 572–583 (2007)
8. Bulgannawar, S., Vaidya, N.: A distributed k-mutual exclusion algorithm. In: ICDCS, pp. 153–160 (1995)