ASAND: Asynchronous Slot Assignment and Neighbor Discovery Protocol for Wireless Networks

Fikret Sivrikaya, Costas Busch, Malik Magdon-Ismail, Bülent Yener
Computer Science Department, Rensselaer Polytechnic Institute
Troy, NY 12180
E-mail: {sivrif,buschc,magdon,yener}@cs.rpi.edu

Abstract

Slot assignment is crucial for time-multiplexed channel access schemes. All existing approaches for slot assignment assume that the network nodes are synchronized with high precision, which is costly to maintain. We propose a novel slot assignment protocol which does not rely on time synchronization among nodes in the network. As a byproduct, each node discovers its neighbors along with a local schedule of their channel access times, which is a crucial piece of information for many networking protocols. We describe our approach, present the Asynchronous Slot Assignment and Neighbor Discovery protocol (ASAND), and then provide the OPNET Modeler implementations and simulation results for our protocol. ASAND works in a completely distributed manner and assigns a time interval for each node in the network which is collision-free in its 2-neighborhood. Hence, it eliminates the collisions between neighbor nodes and the hidden-terminals without any explicit time synchronization in the network.

1. Introduction

A Medium Access Control (MAC) protocol specifies how nodes in a network access a shared communication channel. As networking ultra-small-scale devices starts to become a widespread application with an increasing number of participating nodes, the need for low-power operation increases. A major source of power consumption in wireless networks is the radio component, due to the transmission/reception of packets and idle listening. Contention-free MAC protocols eliminate collisions in the wireless medium by assigning each node a *spatially unique* slot for transmission. Eliminating collisions reduces power dissipation due to the retransmission of collided packets. Moreover, scheduled access with contention-free MAC protocols enables nodes to easily schedule powering on/off their radio transceivers to prevent idle listening and save significant amount of battery power. However, assigning and maintaining conflict-free slots among nodes in a large wireless network is a complex task and existing approaches so far assume a central coordinator and/or build upon existing synchronization among nodes in the network.

In this paper, we propose a novel slot assignment protocol that does not rely on time synchronization among nodes in the network. The Asynchronous Slot Assignment and Neighbor Discovery protocol (ASAND) works in a completely distributed manner and assigns a time interval for each node in the network which is collision-free in its 2-neighborhood. Hence it eliminates the collisions between neighbor nodes and conflicts between hidden-terminals without any explicit time synchronization in the network. During the slot assignment process, each node discovers its neighbors and builds a local schedule of their channel access times.

In the next section, we present the network model and assumptions on which we build our protocol. We describe our basic approach for asynchronous slot assignment in Section 2, and then present the ASAND protocol in detail in Section 3. We provide an OPNET Modeler implementation of ASAND in Section 4, which is followed by the simulation results of Section 5. Section 6 discusses some practical issues related to the actual implementation of the protocol, and Section 7 concludes the paper.

1.1. Network Model

We consider a wireless network of n stationary devices equipped with identical radio transceivers; hence, each node in the network has the same *transmission radius*, which determines the set of nodes it can communicate directly. A wireless network is typically represented by a graph $G = (V, E)$, in which two nodes are connected if they can communicate directly. When the transmission range is the same for all nodes, G becomes a unit disk graph. We assume that the given graph G solely represents the communication and interference among nodes, i.e. a node can directly communicate with or have interference on only its immediate neighbors in graph G.

If two nodes u and w are adjacent and send messages simultaneously, their messages collide at u and w, and they receive noise instead of actual messages. If two nodes u and w are not adjacent and have the same common adjacent node v, then when u and w transmit at the same time, their messages collide in v (*the hidden terminal problem*).

The k-neighborhood of a node v, denoted $\Delta_k(v)$, is the set of nodes whose shortest path to v has length at most k. We denote the number of nodes in the k-neighborhood by $\delta_k(v)$ (i.e., $\delta_k(v) = |\Delta_k(v)|$), and the maximum k-neighborhood size by δ_k (i.e., $\delta_k = \max_v \delta_k(v)$). We refer to 1-neighbors simply as neighbors. Nodes do not have any prior information on their local neighborhood sizes, but we assume that all nodes are provided with an upper bound on δ_2, the maximum 2-neighborhood size in the network. This bound will be preserved despite the topological changes in the network. This assumption is necessary for the nodes to initialize their frame sizes in our protocol, and was also used earlier in similar work, such as the distributed coloring algorithm in [1].

2. Basic Approach for Asynchronous Slot Assignment

In our approach, each node independently divides its local time into frames, which are further divided into time slots. We assume a constant packet size; larger messages are fragmented and smaller messages are padded with extra bits. The duration of a time slot corresponds to the time it takes to send one packet.

The number of slots in a time frame, called the *frame size* and denoted by Λ, is set to $2\delta_2$, which is common for and known to all nodes in the network. Note that the nodes may use the provided upper bound on δ_2, instead of its actual value, for setting the frame size. Unlike conventional slot assignment protocols, the time frames and even the time slots at different nodes need not be aligned. However, since frame sizes are the same, a time slot at some node i will always coincide with the same time slots (at most two of them) at some other node j, as illustrated in Figure 1.

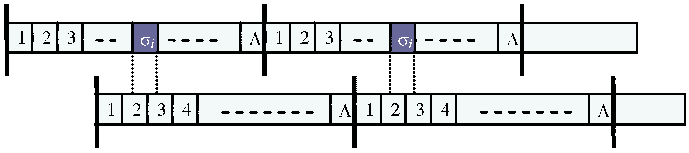

Figure 1 – Independent discretization of the local time at two nodes. Node i's slot (σ_i) always coincides with slots 2 and 3 at another node j.

The basic idea behind ASAND protocol is demonstrated in Figure 2. A node first discretizes its local time into equal sized *frames*, each consisting of Λ slots. The main task for node i is to select a conflict-free time slot in its frame. When this occurs we say that the node is *ready*.

Initially, node i is nonready and it selects randomly and uniformly a slot σ_i in its frame. In slot σ_i, node i broadcasts a "beacon" message m_i to its neighborhood. In all other slots in the current frame, it listens to the channel and marks each slot in which a garbled signal is received. Then in the next frame, i transmits in all slots that were marked in the previous frame. We call this technique *conflict reporting*, and it effectively forces hidden terminals to refrain from obtaining overlapping time slots.

If node i's initial broadcast in slot σ_i was collision-free, then i concludes that none of its neighbors has selected an overlapping time slot. Then, i broadcasts a second time in slot σ_i in the next frame. If this is also collision-free, then i concludes that none of its *2-hop neighbors* selected a time slot conflicting with σ_i. Hence, the random slot selected by node i is conflict-free in its 2-neighborhood and i becomes *ready*. Otherwise, when i detects a collision in either its first or second broadcast, it goes back to the initial state and tries a new random slot.

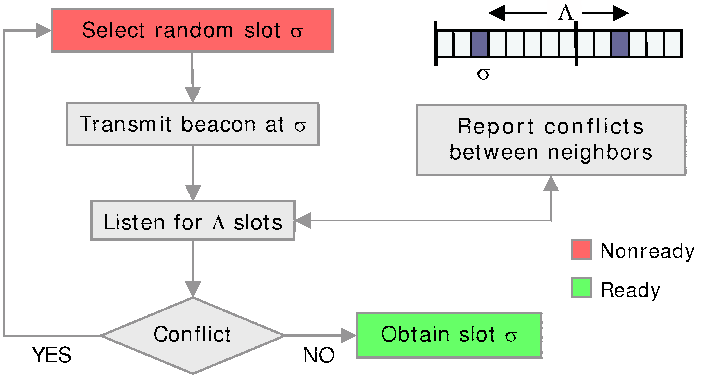

Figure 2 - A schematic outline of the basic idea of ASAND.

When all neighbors of a node are also ready, we say that the node is *ready-1*, at which point it receives no collisions but only conflict-free beacon messages from all of its neighbors. The node thus completes the discovery of its neighbors along with a local schedule of their channel access times when it becomes *ready-1*.

This basic approach is a very high level description of the operation of ASAND, which makes it easy to introduce the protocol, but also brings up some fundamental issues that need to be addressed before an actual implementation. First, it is usually not possible to detect collisions for a wireless device during its own transmission. In Section 6, we propose a novel collision detection scheme based on a special modulation scheme utilizing unique ids of nodes.

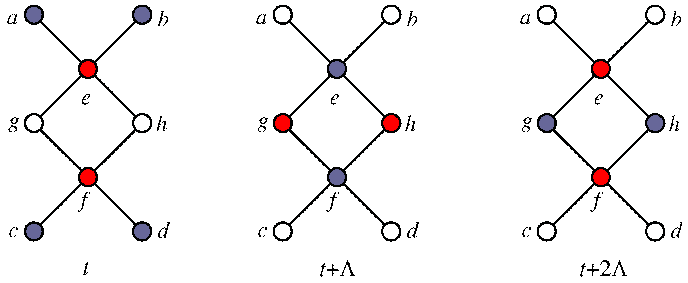

Figure 3 - A simple topology to demonstrate a chain of spurious conflict reports. Blue color (light shade) denotes a node transmitting at that instant, and red color (dark shade) denotes a node detecting a collision. The nodes *e*, *f* and *g*, *h* will indefinitely transmit (report collisions) in this slot every other frame.

Second, the conflict report messages in a time slot may create an indefinite chain of transmissions between alternating nodes, causing the slot to become useless. It is easiest to explain this with an example. Consider the simple topology in Figure 3. Assume that nodes *a, b, c* and *d* all select the same time slot σ. Let *t* be the global time at the instant when *a, b, c, d* all transmit for the first time in this slot. Then *e* detects a collision by the signals from *a* and *b*; *f* detects a collision by the signals from *c* and *d*, and they both report these conflicts at time $t = \sigma + \Delta$, which corresponds to the next occurrence of slot σ. Thus at $t = \sigma + \Delta$, the transmissions of *e* and *f* (which are actually conflict-reports) collide on both *g* and *h*. Since *g* and *h* have no means of differentiating an actual collision (slot conflict) from the collision of conflict-report messages, they transmit conflict-report messages at time $t = \sigma + 2\Delta$. Similarly, now both *e* and *f* spuriously detect a conflict (caused by the conflict-reports of *g* and *h*) and report it at $t = \sigma + 3\Delta$. This chain of messages goes on indefinitely and makes the time slot σ useless in this neighborhood. Moreover, nodes involved in the chain waste significant amount of energy by continuously sending unnecessary conflict-report messages. Another important problem that arises is that the nodes in this neighborhood will not be able to terminate the algorithm as they detect collisions in every time frame due to the slots flooded by spurious conflict reports. We propose a solution to this problem in the next section, where we introduce and present our protocol, ASAND, in more detail.

3. Asynchronous Slot Assignment and Neighbor Discovery Protocol

As explained in the previous section, deterministically reporting every collision may result in chains of redundant messages. Here we present the complete specification for the Asynchronous Slot Assignment and Neighbor Discovery Protocol (ASAND), which introduces the *probabilistic conflict reporting* technique to break such chains. Note that if two hidden terminals u and v select the same time slot, they continuously transmit and collide on the middle node w at this time slot, unless they successfully receive a conflict report and revise their time slots. On the other hand, a conflict report message due to a single collision is sent only once, hence if the conflict reports of u and v collide on w, then w does not detect a collision at the next occurrence of σ (unless some other nodes collide on w at this slot). Probabilistic conflict reporting is based on this observation: a node reports a conflict at time slot σ with a probability based on the number of consecutive collisions at σ. As the number of consecutive collisions detected at slot σ increases, it is more likely that these collisions correspond to a slot conflict, rather than the collision of conflict report messages. Spurious conflict report messages may still be generated occasionally, but the probability of having long chains due to these spurious messages is dramatically reduced.

The pseudocode of ASAND for some node i is shown in Figure 4. Each node executes this algorithm locally, and maintains a local virtual time, t_i, which is basically a counter and is incremented after each time slot using modulo Λ arithmetic.

In ASAND, upon detecting a collision at time t, node w reports this at $t+\Lambda$ with probability p_{report}. Otherwise (with probability $1-p_{\text{report}}$), it remains silent and listens to the channel at $t+\Lambda$. If w again detects a collision at this instant, which is the next occurrence of this slot, then w reports it at $t+2\Lambda$ with probability $2\times p_{\text{report}}$. Depending on the protocol parameter p_{report}, the probability of reporting a conflict will soon reach 1 as long as the hidden terminals continue to collide on w at that slot. Hence a slot conflict between hidden terminals is always reported after at most $1/p_{\text{report}}$ time frames.

For marking the conflicting time slots, each node i has a local vector C_i of size Λ, which is an integer array indicating the number of recent *consecutive collisions* at each time slot. Initially $C_i[t]=0$ for all $t=0..\Lambda-1$. When node i receives a garbled transmission at its local time t_i, it increments $C_i[t_i]$ by 1. Otherwise, it resets $C_i[t_i]$ to zero. The counter $C_i[t_i]$ is also reset whenever the node sends a conflict-report message in slot t_i.

In its randomly selected time slot σ_i, node i transmits a *beacon message* that includes its unique id, and checks if there exists another transmission on the channel overlapping with its own[1]. If so, it randomly selects another time slot, and resets its *clear* counter to 0. Otherwise, node i uses the current value of its *clear* counter to decide if it can safely obtain σ_i permanently, and then increments the *clear* counter. Note that a neighbor j of node i reports a conflict at time slot σ_i with probability $C_j[\sigma_i]\cdot p_{\text{report}}$.

[1] See Section 6.3 for the implementation details of this.

Therefore j deterministically reports the conflict after receiving $1/p_{\text{report}}$ collisions at σ_i. Thus, if node i's *clear* counter has reached $1/p_{\text{report}}$, σ_i is deterministically collision-free in the 2-neighborhood, hence node i becomes *ready*, by obtaining slot σ_i permanently.

Node i also keeps a vector of neighbors, N_i, of size Λ. Whenever i successfully receives a beacon message from node j at time slot t_i, it adds j to its neighbor list by setting $N_i[t_i]$ to j. Thus the neighbor list and a local schedule of their transmission times are effectively stored using a single vector. Note that the transmission slot of j may overlap with at most two (consecutive) slots in i's frame, so i may mark one or two slots for j in vector N_i. If there were any other slots previously marked for j, they are erased to maintain only the latest slot selections of the neighbors. During the slot assignment, some values may be overwritten by the beacon messages of some other neighbors, but when node i is *ready-1*, i.e. when all of its neighbors are *ready*, their transmission slots must be conflict-free; hence they all occupy distinct entries in vector N_i.

Protocol ASAND (node i)

```
1:  ready ← FALSE
2:  clear ← 0
3:  C_i[t] ← 0,   t=0..Λ-1              // Conflict counters
4:  N_i[t] ← NULL,   t=0..Λ-1           // Neighbor ID list
5:  t_i ← 0              // local virtual time (slot counter)
6:  while not ready-1 do
7:      if clear = 0 then
8:          σ_i ← random(0..Λ-1)
9:      if t_i = σ_i then
10:         send a beacon message
11:         if not ready then
12:             if sensed any other transmission then
13:                 clear ← 0
14:             else if clear++ ≥ 1/p_report then
15:                 ready ← TRUE
16:     else
17:         with probability C_i[t_i]×p_report
18:             send a conflict-report message
19:             C_i[t_i] ← 0
20:         otherwise
21:             listen the channel in slot t_i
22:             if received a garbled message then
23:                 C_i[t_i] ++
24:             else if correctly received <beacon, j> then
25:                 N_i[t_i] ← j
26:     t_i ← (t_i + 1) mod Λ
```

Figure 4 – ASAND Protocol

The termination condition *ready-1* is not coded in Figure 4 to keep the clarity of the main protocol. A node locally checks this condition by keeping a counter that is reset every time a collision is observed or when the time slot marked for a neighbor node is revised. Otherwise, the counter is incremented at each time slot. Once the counter has reached $\Lambda/p_{\text{report}}$, all neighbors must have

become ready as they have not revised their time slot selections and no conflicts were observed during this interval.

4. An OPNET Implementation of ASAND

We have implemented ASAND in OPNET Modeler (v.11.5) as a Medium Access Control (MAC) process model. After the conflict-free slot assignment, the MAC layer informs the upper layer that it is ready to forward any higher layer data for collision-free transmission through the wireless interface. ASAND needs two statistics from the radio receiver; the received power statistics with an adjustable high threshold trigger, and the collision status statistics with rising/falling edge triggers. A sample node model employing the ASAND process is shown in Figure 5.

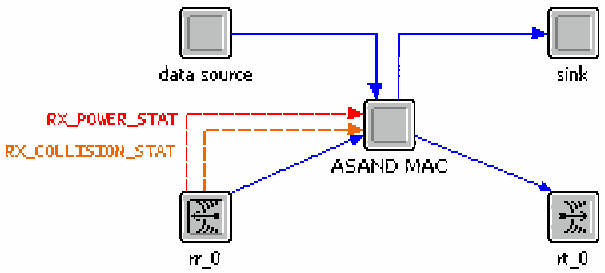

Figure 5 - OPNET node model used to supplement the implementation of ASAND.

The process model for the ASAND protocol is given in Figure 8, at the end of the paper. The actual protocol –slot assignment and neighbor discovery phase– is separated by annotations for clarity. The contention-free phase is added for completeness, providing the basic functionality of a MAC protocol. This phase simply maintains the virtual time t_i, and transmits any available data from the upper layer in the contention-free slot obtained in the previous phase. It also processes lower layer packets in the remaining slots. Next, we briefly explain the functioning of each state in the ASAND process model.

INIT state: This state initializes state variables and data structures, and reads in the MAC attribute values (frame size, slot size, and conflict-report probability). It also removes the node's receiver from the *receiver group* of its transmitter. This is a useful feature of OPNET Modeler that allows nodes to detect collisions easily. Though this simplification is not realistic for actual implementations, we use it for faster simulations. In Section 6.3, we present a more detailed and realistic model, where we introduce our collision detection scheme.

WAKE UP state: This state simply achieves asynchronous wake-up of nodes by introducing a random delay (within the duration of a single frame) before the node starts its virtual clock and the protocol execution.

NEW SLOT state: This forced state implements the virtual time by scheduling new slot interrupts, and makes a transition into either the TX-REPORT state or the DETECT state until the node becomes *ready-1*. The transition is into the TX-REPORT state if (i) current slot (virtual time) is the same as the node's randomly selected slot, or (ii) if the node probabilistically decides to report a conflict based on the conflict counter associated to the current slot. Otherwise, the transition is into the DETECT state.

TX-REPORT state: In this state, the node broadcasts either a beacon message in its own slot or a conflict-report message in another slot. If the node is broadcasting a beacon message, it also checks for any collisions (overlapping transmissions) with neighbors in this slot. This state is replaced by a set of states in the more detailed and realistic collision-detection model, as explained in Section 6.3.

DETECT state: The node is in the DETECT state whenever it is not broadcasting a beacon or conflict-report message. It listens to the channel and increments a conflict counter associated to any slot in which a collision is detected. These counter values are used for conflict reporting decision in the next frame.

5. Adopting the Right Probability for Conflict Reporting

Setting an appropriate value for p_{report} is an issue that naturally arises with the probabilistic conflict reporting. On one hand, a lower value for p_{report} better eliminates the long chains of spurious conflict reports, but increases the time for a node to become ready. On the other hand, a higher value for p_{report} speeds up the process of becoming ready once a conflict-free slot is chosen, while it may cause more slots to be flooded with spurious conflict reports. In this section we use OPNET simulations of ASAND with different values of conflict-report probability and compare the time to stabilization for each case. In particular, we vary p_{report} from 0.1 to 1 and obtain the average running times for different values of p_{report}.

We randomly generate networks of 500 to 1000 nodes, which are scattered in a unit square area. In order to study various node densities, we use a fixed, normalized transmission radius of r = 0.1 units for each node in the network, utilizing the standard OPNET model RX GROUP CONFIG with an appropriate distance threshold value. Each data point used in the plots of this section is an average over 20 random networks. The frame size is set to $\Lambda=2\delta_2$ for each random network generated. This is accomplished by pre-computing the maximum 2-neighborhood size for each network.

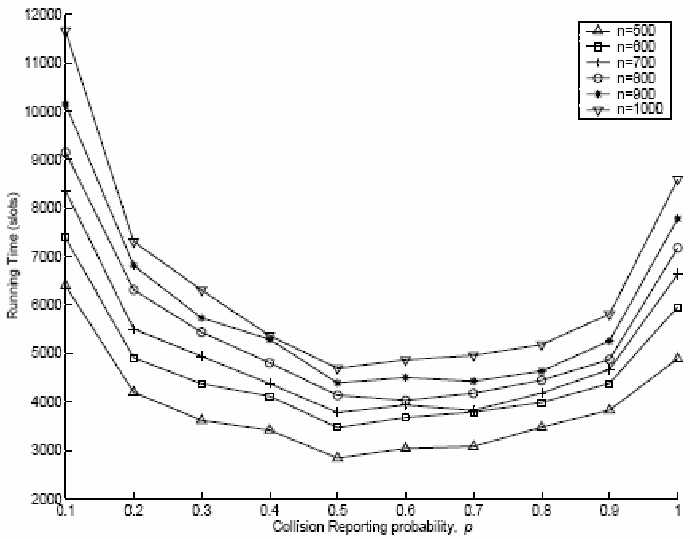

Figure 6 – Running time of ASAND with varying conflict reporting probabilities, shown for different network sizes. For each network size, p=0.5 is approximately the optimal value that minimizes the running time.

Figure 6 plots the running time of ASAND, which is the time it takes for all nodes in the network to become ready. The figure clearly demonstrates that the optimal value for p_{report} is around 0.5, regardless of the network size or density. We can also observe the tradeoff mentioned in the beginning of this chapter. As p_{report} gets closer to 1, some time slots get flooded and become useless by the spurious conflict reports, decreasing the availability of conflict-free slots. On the other hand, very low values of p_{report} causes an increase on the running time; since there is not much further gain due to the elimination of spurious collisions, but the number of time frames required to obtain a slot increases inversely with p_{report}. For instance, when p_{report} = 0.1, a node needs to wait for at least 10 time frames before it can obtain a conflict-free slot, whereas the waiting time can be as low as the duration of a single frame when p_{report} is set to its maximum value of 1.

6. Practical Considerations

In this section, we discuss some of the issues to be faced in an actual implementation of ASAND. Essentially, we show that the simplifying assumptions we have made for easier analysis and presentation can be removed, and we discuss the effect of relaxing these assumptions.

6.1. Clock Skew

We have assumed so far that a time slot of one node always coincides with the same slot(s) at another node. In practice, there may be a very small clock skew that can cause the relative times to change. One approach to addressing this problem in our algorithms is to run a skew-correction algorithm (for example [2]) on top of our protocol. Another solution is to maintain safety gaps at the beginning and end of each slot, based on a given maximum skew for the clock oscillators, which is generally available by the manufacturer. Then the actual data packets will be smaller than the control packets used during slot allocation, providing room for error due to drifting clocks.

6.2. Collision Detection between Hidden Terminals

Our model assumes that, in the hidden terminal case, a node detects a collision if two or more nodes within its transmission radius (excluding itself) attempt to transmit. One way to distinguish a collision from random background noise is to place a threshold on the power of the incoming signal. Distorted signals with sufficiently high power are collisions. Since wireless signals attenuate with distance rather than drop to zero at the transmission radius [3], it is possible that many nodes outside the transmission radius transmit at the same time, resulting in a collision detected at the central node, even though none of the nodes are actually colliding. The correctness of the algorithm is not affected; however the convergence time may get affected, because such spurious collisions may occasionally prevent a node from obtaining a time slot, causing it to select a new one.

6.3. Collision Detection in Adjacent Nodes

A fundamental assumption we made throughout this paper is that the nodes are capable of detecting a collision on the wireless channel during their own transmission. However, if the receiver and transmitter of a node operate simultaneously, the node's receiver hears the signals from its own transmitter so strongly that it may not be able to detect any other incoming signal. In this section, we propose a collision detection scheme that remedies this problem, hence justifying our assumption.

Our scheme is based on dividing each (original) time slot into $m = \theta(\log n)$ *mini slots*. Each node u locally generates a unique *binary sequence* b_u of m bits, where the k^{th} most significant bit in b_u corresponds to the k^{th} mini slot in u's original slot. Then, during the slot assignment phase, the transmitter of a node emits signals only in the mini slots with the corresponding bit set in the binary sequence, and remains silent for the rest. This allows the receiver of the same node to sense the channel during those brief periods of "silence".

Consider node u that transmits in its current slot σ_u. Let S_u denote the set of nodes whose transmissions collide with u's transmission. Then, a collision detection scheme must ensure that for some integer k, $1 \le k \le m$, there exists a node $v \in S_u$ such that $b_u(k)=0$ and $b_v(k)=1$. Note that this criterion is simplified for brevity, by assuming that the positions of the mini slots $b_u(k)$ and $b_v(k)$ occur at the same real time. In general, a collision detection scheme should work regardless of the alignment between the time slots and even the mini slots.

In our collision detection scheme, each node u chooses its binary sequence as $b_u = \left(\mathrm{id}_u 00 \overline{\mathrm{id}_u} 01\right)^{(2)}$, where id_u corresponds to a unique identifier assigned to node u, and $\overline{\mathrm{id}_u}$ to its bitwise complement. We assume that there are as many identifiers as the number of nodes n. Hence, the length of each id is $\log n$ bits (leading bits of smaller ids are padded with 0s). The square power means that each bit is (consecutively) repeated twice, resulting in a total sequence length of $m = 4\log n+8$ bits. We have proved in [4] that with this scheme, a node has always a mini slot with binary value 0 in which at least one of its neighbors are transmitting whenever they have conflicting slots, which is sufficient to detect the conflict in that mini slot –either by a clear message received from one neighbor or as a collision received from multiple neighbors.

Every binary sequence in the proposed scheme has exactly $2\log n+2$ bits of 1, with the remaining $2\log n+6$ bits being 0. Hence, a node may utilize $2\log n+2$ mini slots for transmission in each slot. If the length of the longest control message is $\kappa \log n$ bits, then the duration of a mini slot should be set such that approximately $\kappa/2$ bits can be transmitted during each mini slot. After the stabilization phase, nodes can fully utilize their time slots to transmit data packets.

We have implemented this collision detection scheme in OPNET at the process model level. Though in reality, a transmitting node would modulate its signal using this scheme, similar to the on/off keying (OOK) modulation, we used mini packets (fragmented beacon messages) and separate transmissions in each mini slot for the purpose of simulating the collision detection scheme. The required changes in the state transition diagram of the ASAND process model is outlined in Figure 7. Besides some additions to the INIT state, the only modification in the model is to replace the unforced TX_REPORT state with

three new states; forced TX_REPORT state, mini_tx state and mini_rx state, which are described next.

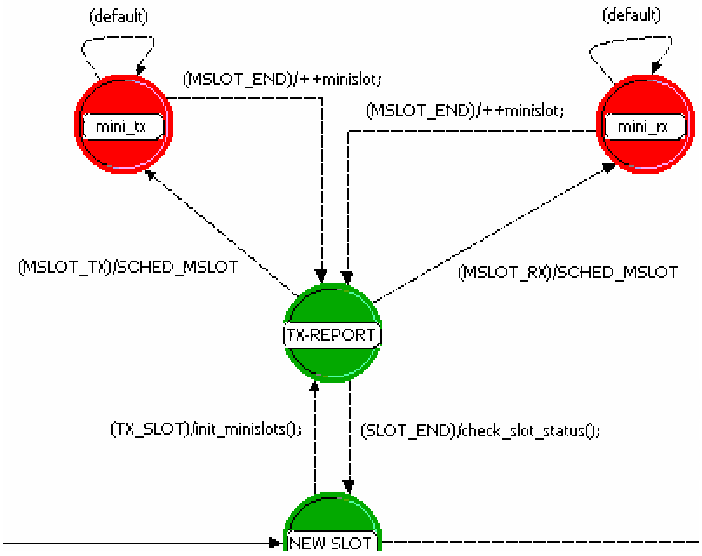

Figure 7 – The states used for the collision detection scheme in detailed ASAND process model. Top three states replace the TX-REPORT state in the previous model.

TX-REPORT state: This state now implements the mini slot counter, decides on which mini slots to transmit/listen based on the binary sequence, and makes a transition accordingly to either the mini_tx state or the mini_rx state.

mini_tx state: This state corresponds to the mini slots with an associated bit value of 1 in the binary sequence. The node transmits one fragment of either a beacon or a conflict-report message.

mini_rx state: This state corresponds to the mini slots with an associated bit value of 0 in the binary sequence. If the current slot is the node's own slot and a transmission is detected, this state sets necessary flags to indicate a collision with a neighbor.

We have used this detailed model to justify the correctness of the collision detection scheme by verifying that nodes obtain conflict-free slots regardless of the relative alignment of their time frames and slots. After this verification, the basic model presented earlier was used in our simulations since it provides a significant speed-up.

7. Conclusion

We have proposed a distributed and asynchronous slot assignment and neighbor discovery protocol, ASAND, for wireless networks. We have presented and described an OPNET process model for ASAND, which we used in our simulations to verify the correctness of the protocol and to obtain the optimal probability for reporting collisions. The results indicate that distributed and asynchronous slot assignment with reasonable frame sizes is possible in wireless networks.

In this paper, we have focused on the practical implementation of ASAND and its simulation model with OPNET Modeler. Although we considered a stationary network in this paper, it is possible to extend our protocol to capture networks with changing topologies. The reader is referred to reference [4] for such variations of the proposed protocol and their formal analyses.

References

[1] T. Moscibroda, R. Wattenhofer, "Coloring Unstructured Radio Networks," 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Las Vegas, Nevada, USA, July 2005.

[2] K. Arvind, "Probabilistic Clock Synchronization in Distributed Systems," IEEE Trans. Parallel Distrib. Syst., vol. 5, no. 5, pp. 474--487, 1994.

[3] R. Crane, "Propagation Handbook for Wireless Communication System Design," CRC Press LLC, 2003.

[4] C. Busch, M. Magdon-Ismail, F. Sivrikaya and B. Yener, "Contention-Free MAC Protocols for Asynchronous Wireless Sensor Networks," Submitted to the Journal of Distributed Computing (under revision). Available at http://www.cs.rpi.edu/~sivrif/academic/papers/JDC_CFMAC.pdf

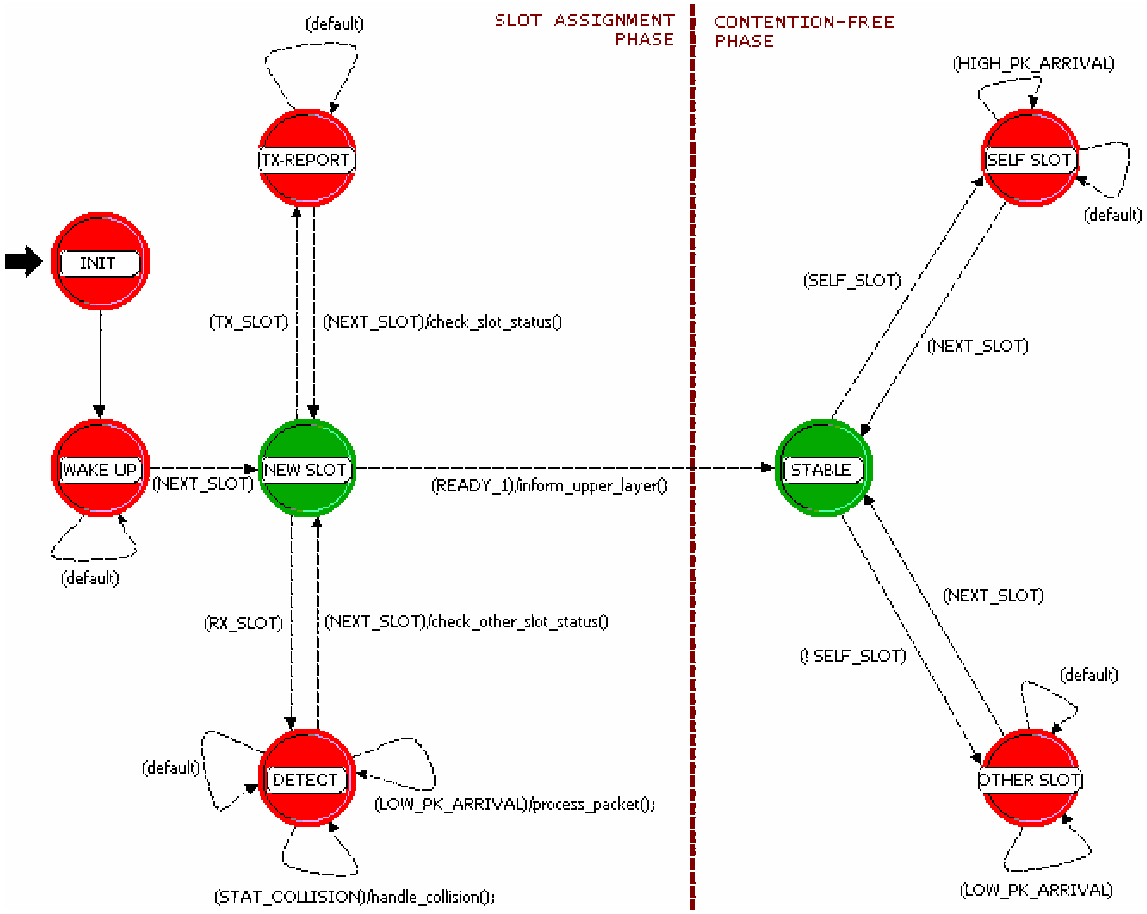

Figure 8 – OPNET Process model for ASAND.