Simulation of the APEC Wireless Ad Hoc Network Authentication Protocol

Jason Wright and May Chaffin

Chaf6886@uidaho.edu

***Abstract*— The APEC (Authentication Protocol with Embedded Certificates) wireless ad hoc network authentication protocol was designed to provide secure communication using frequency and time slots for collision avoidance and authentication. In order to test the performance and security of the APEC protocol, the authors simulated it using NS-2 (Network Simulator 2). This paper describes the implementation of the simulation and provides analysis of its performance.**

***Keywords* — Authentication, frequency division multiplexing algorithms (FDMA), simulation, space-time representation, time division multiplexing algorithms (TDMA), time-slotted frequency-hopping spread spectrum.**

I. INTRODUCTION AND BACKGROUND

Wireless mobile ad hoc networks (MANET) increase communication abilities without being restricted by wires. The side effect is that the communication medium cannot be physically restricted either. Security issues arise from the ease of intercepting radio waves. Wireless routing protocols exasperate this problem by requiring that each mobile node act as a router. New routing and authentication protocols continue to be proposed in an effort to solve the reliability and security problems that afflict MANETs.

Packet scheduling protocols address problems caused by packet collisions and interference on the same communication channel. A MAC protocol in a multi-access medium is essentially a distributed scheduling algorithm that allocates the channel to requesting nodes.

Random access MAC protocols require the nodes to dynamically compete for a time slot with other nodes. This is a flexible and efficient method of managing the channel in a fully distributed way, but suffers from collisions and interference.

Multiplexing algorithms are created to allow multiple nodes to communicate over a shared radio wave without interfering with each other. Fixed-assignment channel access methods such as frequency band or spread spectrum code statically allocate a certain time slot to a pair of nodes. This method prevents collisions and interference, but is not as efficient.

Frequency division multiplexing algorithms (FDMA) can avoid collisions by dividing the entire spectrum into frequency bands. Packets can then be sent on different frequency channels at the same time without interfering with each other.

Time division multiplexing algorithms (TDMA) shares one frequency band between a few nodes. Each user is allowed to transmit in predetermined time slots. Hence, channelization of users in the same band is achieved through separation in time.

Mobile wireless TDMA design considerations include:

- Number of logical channels (number of time slots in TDMA frame)
- Frequency
- Maximum delay spread

Recently, an alternative approach to certificate-based authentication using both time and frequency multiplexing has been proposed [1]. The proposed authentication protocol for wireless ad hoc networks with embedded certificates (APEC), reformulates the authentication problem as a geometric splitting of time and space. This protocol removes the need for wireless nodes to explicitly exchange certificates in the authentication process using a collision-avoidance policy for data transmission.

The wireless network simulator, Network Simulator 2 (NS-2), was used to evaluate the APEC protocol. Simulating the APEC protocol requires combining FDMA and TDMA methods for managing the wireless communication channel. NS-2 provides this capability through the MAC layer and channel objects.

For a wireless network, the MAC layer must contain a certain set of functionalities such as carrier sense, collision detection, and collision avoidance. Since these functionalities affect both the sending and receiving sides, they are implemented in a single Mac object. For sending, the Mac object must follow a certain medium access protocol before transmitting the packet on the channel. For receiving, the MAC layer is responsible for delivering the packet to the link layer. Rules for sending and receiving packets can be based on time and other channel attributes such as frequency.

The Channel class simulates the actual transmission of the packet at the physical layer. The basic Channel implements a shared medium with support for contention mechanisms. It allows the MAC to carry out carrier sense, contention, and collision detection. If more than one transmission overlaps in time, a channel raises the collision flag. By checking this flag, the MAC object can implement collision detection and handling.

New MAC layer protocols can be added to NS-2 by defining new rules for transmitting and receiving packets on the channel. A new APEC MAC layer protocol was created for NS that can switch between multiple frequency channels. The APEC object implements time and frequency collision avoidance by controlling the time slot and channel packets can be sent and received on.

The remainder of the paper describes the APEC protocol, presents the new APEC addition to NS-2, documents the simulation parameters and test scenarios, and presents the simulation results.

II. SOLUTION APPROACH

The proposed APEC authentication protocol has been documented by Hiromoto et.al. [1,2]. This section summarizes the protocol characteristics that were implemented in the NS-2 simulator.

A. *Initialization*

The following steps describe the APEC initialization sequence. An administrator node initializes authentication by sending one or more public keys to all verified network nodes.

1. Assume that an initial authentication phase has verified the trust of nodes that have joined the cluster.
2. The cluster head sends a public key to all nodes in the cluster.
3. The public key is used by every node to select the appropriate addem, multipliers, etc. to construct a common pseudo-random number generator (PRNG) using a hash table. Two such PRNGs are constructed RandT() and RandF, that produces the sequences for T_i and F_i; respectively.
4. From the public key, a random seed is produce and used to seed RandT(). Each T_i that is generated is in turn used as a seed to generate a corresponding F_i from the pseudo-random number generator RandF().
5. After each complete communication time period, the order of the sequence of timeslots is permuted.

At the end of these steps, a sequence of T_m time-slots with their randomly correlated frequency channel, F_m, are created, and forms m-coordinate, 2-tuples (T_i, F_i). In addition, the details of the PRNG construction are known to each entrusted node in the network; and therefore, allows a deterministic recreation of all present and future m-coordinate pairs. This becomes important when a node cannot send data packets to the cluster head in one hop but instead requires a multi-hop link to reach its destination [2].

B. *Communication*

Communication can be parameterized as a space-time, 2-tuple coordinate basis, (Ti, Fi). Time (time-slot), T_i, is taken as the moment (over the time-slot duration) that a message is sent or received. Space is the radio frequency, F_i, over which the message is sent or received.

Collision avoidance is guaranteed by restricting a clustered network of wireless nodes to communicate only over predetermined, non-overlapping send or receive time-slots. As a consequence, certain types of external attacks can be detected if two or more distinct data packets arrive during the same time-slot.

The collision-avoidance protocol and the two-dimensional representation of communication events within a wireless mobile ad hoc network provide a cryptographic confusion algorithm. This algorithm requires the following set of properties:

Property 1: A unique time-slot and frequency channel pair is assigned to only one node within the network.

Property 2: It is desirable to conceal the selection of the radio frequency channel F_i in each round of a node's communication time-slot sequence.

Property 3: For each communication period a new sequence order for a given node's time-slots T_i are assigned.

Property 4: The length of a time-slot duration depends upon the time-skewing experience by each node's clock. It is assumed that each node is equipped with a GPS device for time synchronization.

The process for concealing the selection of T_i and F_i is performed using a pair of orthogonal PRNGs. Each node must be given a different random number sequence in a deterministic fashion for authentication.

III. IMPLEMENTATION

A. *NS-2 Modification*

The NS-2 network simulator was modified to simulate the proposed APEC protocol. A new MAC class, MacApec, was created to correlate new frequency and time slot sequences for each communication period.

This implementation did not model the head node. It also assumes that the first three APEC initialization steps have been performed successfully. The authentication, key distribution, and PRNG construction phases are therefore assumed to be complete, and each node should possess a common random seed. To simulate this, the MAC object is sent two random seeds from the TCL simulation script during node configuration.

The seeds are bound to the MAC class and used to initialize two new random number generator instances for assigning time slots and frequencies:

```
# random number generator for selecting
# frequencies
bind("rngf_seed_", &rng_f_seed);

LIST_INIT(&chanhead);
rng_f = new RNG;
rng_f->set_seed(RNG::RAW_SEED_SOURCE,
rng_f_seed);

# random number generator for selecting time
# slots

bind("rngt_seed_", &rng_t_seed);

rng_t = new RNG;
rng_t->set_seed(RNG::RAW_SEED_SOURCE,
rng_t_seed);

int MacTdma::rng_f_seed = 1;
int MacTdma::rng_t_seed = 1;
```

Figure 1. Code added to `MacTdma` class header.

The default value was added to "tcl/lib/ns-default.tcl".

```
Mac/Tdma set rngf_seed_ 1
Mac/Tdma set rngt_seed_ 1
```

Figure 2. Code added to `tcl/lib/ns-default.tcl` file.

The two seeds and PRNG's correspond to RandF() and RandT() included in the description of APEC. The default for both of the seeds is the value 1, but this can be changed for each instantiation of the Mac/Tdma class. The pseudo random number generators used are the default ones available from the RNG class in NS-2. There are not necessarily cryptographic quality PRNG's, but they produce a uniform distribution from *a* to *b* (a < b). To alter the PRNG, one needs to create a subclass of the RNG class and implement the appropriate functions.

The MAC class is documented in the NS manual. The Mac object simulates the medium access protocols that are necessary in shared medium environments such as the wireless and local area networks.

On the sending side, the Mac object is responsible for adding the MAC header and transmitting the packet onto the channel.

When a Mac object receives a packet via its `recv()` method, it checks whether the packet is outgoing or incoming. On the receiving side, the Mac object asynchronously receives packets from the classifier of the physical layer. For an incoming packet, the MAC object does its protocol processing and passes the packet to the link-layer.

For an outgoing packet, the Mac object fills in the rest of the MAC header with the source MAC address and the frame type. It then passes the packet to its `send()` method, which carries out the medium access protocol. For the basic Mac object, the send method calls `txtime()` to compute the transmission time, then invokes `Channel::send` to transmit the packet. Finally, it schedules itself to resume after the transmission time has elapsed. After MAC protocol processing, it passes the data packet to the link layer.

The Channel class simulates the actual transmission of the packet at the physical layer. The basic Channel implements a shared medium with support for contention mechanisms. It allows the MAC to carry out carrier sense, contention, and collision detection. If more than one transmission overlaps in time, a channel raises the collision flag. By checking this flag, the MAC object can implement collision detection and handling.

Since the transmission time is a function of the number of bits in the packet and the modulation speed of each individual interface (MAC), the Channel object only sets its busy signal for the duration requested by the MAC object. It also schedules the packets to be delivered to the destination MAC objects after the transmission time plus the propagation delay. The `send()` method allows the MAC object to transmit a packet on the channel for a specified duration of time. The `hold()` method allows the MAC object to hold the channel for a specified duration of time without actually transmitting any packets [3].

NS-2 already contains a TDMA class which is supposed to statically allocate time slots for all nodes attached to a channel. We modified TDMA to create a new time slot and frequency sequence every communication period.

Normally, only one channel can be attached to a node. The frequency can be changed, but this doesn't affect whether the packets are still received or not. A node is attached to a channel, regardless of the frequency. We added a channel list to the TDMA object to represent multiple frequency channels for the MAC to switch between.

```
class Channel;

struct MacTdmaChannelList {
  LIST_ENTRY(MacTdmaChannelList) next;
  Channel *channel;
};
```

Figure 3. Code added to `MacTdma` class definition.

This requires a change to the channel class so that the MAC object is able to add and remove nodes from the channel. Normally nodes are kept in sorted order (by their X coordinate) to make searches faster when deciding which nodes can hear one another. So many additions and deletions happen with channel hopping that we must be careful to ensure that these caching assumptions are kept. Marking the list as un-sorted forces a sort before evaluation. This addition is shown in Figure 4.

```
void
WirelessChannel::addNodeToList(MobileNode
*mn)
{
  MobileNode *tmp;
    // create list of mobilenodes for this
channel
  if (xListHead_ == NULL) {
    //fprintf(stderr, "INITIALIZE THE LIST
xListHead\n");
    xListHead_ = mn;
    xListHead_->nextX_ = NULL;
    xListHead_->prevX_ = NULL;
    mn->prevX_ = tmp;
    mn->nextX_ = NULL;
  }
  sorted_ = false;
  numNodes_++;
```

Figure 4. Code added to `WirelessChannel::addNodeToList` method.

The `for` loop in the `WirelessChannel` object removeNodeFromList`()` method was also modified. This corrects a minor error in the linked list handling. Without this change, the last item in the node list for a channel can never be deleted. This patch will be mailed to the NS-2 maintainers.

```
Void WirelessChannel::
removeNodeFromList(MobileNode *mn) {
MobileNode *tmp;
// Find node in list
~~for (tmp = xListHead_; tmp->nextX_ != NULL;
tmp=tmp->nextX_) {~~
for (tmp = xListHead_; tmp != NULL;
tmp=tmp->nextX_) {
  if (tmp == mn) {
    if (tmp == xListHead_) {
      xListHead_ = tmp->nextX_;
        xListHead_ = tmp->nextX_;
        if (tmp->nextX_ != NULL)
          tmp->nextX_->prevX_ = NULL;
      } else if (tmp->nextX_ == NULL)
        tmp->prevX_->nextX_ = NULL;
      else {
        tmp->prevX_->nextX_ = tmp->nextX_;
        tmp->nextX_->prevX_ = tmp->prevX_;
      }
      numNodes_--;
      return;
    }
  }
  fprintf(stderr, "Channel: node not found
in list\n");
}
```

Figure 5. Modifications to `WirelessChannel::removeNodeFromList` method.

The channel list is created when the new MAC object is constructed. This new code is shown in Figure 6.

```
if (strcmp(argv[1], "add-channel") == 0) {
  Channel *c = (Channel
*)TclObject::lookup(argv[2]);
  if (c == 0)
    return (TCL_ERROR);
  MacTdmaChannelList *l = new
MacTdmaChannelList;
  l->channel = c;
  LIST_INSERT_HEAD(&chanhead, l, next);
  return (TCL_OK);
}
```

Figure 6. Code added to `MacTdma::command` method.

This allows the MAC object to switch frequency channels each time slot. The function, `chooseNewChannel()`, is called each timeslot. This new method is shown below:

```
MacTdma::chooseNewChannel() {
  Tcl& tcl = Tcl::instance();
  Node *node;
  Phy *phy;
  Channel *chan;
  int i;
  MacTdmaChannelList *l;

  if (chanhead.lh_first == NULL) {
    // if there's no channel list, there's
nothing to do!
    return;
  }

  phy = netif_;
  chan = (WirelessChannel *)netif_-
>channel();
  node = netif_->node();

  /* step 1. remove the node from the
current channel */
  tcl.evalf("%s remove-node %s",
      chan->name(), node->name());

  /* step 2. find new channel */
  for (i = 0, l = chanhead.lh_first; l !=
NULL; l = l->next.le_next)
    i++;

  long r = rng_f->rand_int(0, i-1);

  for (i = 0, l = chanhead.lh_first; l !=
NULL;
     l = l->next.le_next, i++)
    if (i == r)
      break;

  chan = l->channel;

  // step 3. use new channel
  phy->setchnl(NULL);
  tcl.evalf("%s channel %s", phy->name(),
  chan->name());
  tcl.evalf("%s add-node %s", chan->name(),
  node->name());
}
```

Figure 7. New `MacTdma::chooseNewChannel` method.

The original TDMA object assigns time slots one time by calling the `re_schedule` method when it is constructed. We edited the `re_schedule` method to randomly assign time slots and call it at the beginning of each new communication period in the `slotHandler()` method.

A check was added to the constructor to make sure that there were more time slots than nodes.

```
max_slot_num_ = max_node_num_;
//make sure have empty spot for the special
slot
if(max_slot_num_ <= max_node_num_)
  max_slot_num_ = max_node_num_+1;
```

Figure 8. Code added to `MacTdma` class constructor.

```
void MacTdma::re_schedule() {
  int slot, node;
  // Record the start time of the new
schedule.
  start_time_ = NOW;
  /* Seperate slot_num_ and the node id:
     we may have flexibility as node number
changes.
  */
  ~~//slot_num_ = slot_pointer++;~~
~~  //tdma_schedule_[slot_num_] = (char)
index_;~~
  //need to know who's talking when, so
assign all the slots
  //tdma_schedule_ is initialize to -1 each
round
  for (slot=0; slot<max_slot_num_; slot++)
    tdma _schedule_[slot] = -1;

  //assign each node to a slot, if it is
taken, step till find empty slot
  for (node=0; node < active_node_;
node++){
  //We'll keep the special 1st slot to make
it easy
    slot = rng_t->rand_int(1, max_slot_num_-
1);
    while (tdma_schedule_[slot] >= 0 ) {
      slot++;
      if( slot >= max_slot_num_) slot=1;
    }
    tdma_schedule_[slot] = node;

    //if my slot:
    if(node == index_) slot_num_ = slot;  }
}
```

Figure 9. Modifications to `MacTdma::re_schedule` method.

We then replaced the TDMA send method with a call to our `RealTxPktTdmaTimer start` method which is derived from the `MacTdmaTimer`. (The basic MAC send method calls txtime() to compute the transmission time, then invokes Channel::send to transmit the packet.)

This change prevents a node from transmitting before the receiver is ready. Without it, during the beginning of a timeslot, a node will choose a new channel and if it happens to be its transmit slot, transmit. If the other nodes have not yet had a chance to change channels (even if they are supposed to do so at the same instant), the packet will not be received. This is a function of the discrete-event nature of NS-2. To work around this problem, instead of sending immediately after changing channels, the `send()` function is rescheduled 0 seconds into the future. This has the effect of scheduling it after all of the other nodes have changed channels.

```
chooseNewChannel();

// Make a new presamble for next frame.
if ((slot_count_ == active_node_) ||
(slot_count_ == FIRST_ROUND)) {

  radioSwitch(ON);

  //actually re-schedule the slots
  re_schedule();

  makePreamble();
  slot_count_ = 0;
  return;
}
// If it is the sending slot for me
if (slot_count_ == slot_num_) {

// We have to check the preamble first to
avoid the packets coming in the middle.
if (tdma_preamble_[slot_num_] !=
NOTHING_TO_SEND)
  //send();
  mhRTX_.start((Packet *)&realTxIntr, 0.0);
else
  radioSwitch(OFF);

slot_count_++;
return;
}
```

Figure 10. Code added to `MacTdma::slotHandler` method.

The `RealTxPktTdmaTimer` class definition and handle method are shown below.

```
class RealTxPktTdmaTimer : public
MacTdmaTimer {
public:
  RealTxPktTdmaTimer(MacTdma *m) :
MacTdmaTimer(m) {}
  void handle(Event *e);
};

class MacTdma : public Mac {
  friend class RealTxPktTdmaTimer;
private:
  RealTxPktApecTimer mhRTX_;
}
```

Figure 11. `RealTxPktTdmaTimer` class definition and constructor.

An instance of the `RealTxPktTdmaTimer` class was added to the `MacTdma` class definition.

```
MacTdma::MacTdma(PHY_MIB* p) :
  Mac(), mhSlot_(this), mhTxPkt_(this),
mhRxPkt_(this), mhRTX_(this) {
```

Figure 12. `RealTxPktTdmaTimer` instance added to `MacTdma` class constructor.

```
void RealTxPktTdmaTimer::handle(Event *e)
{
  busy_ = 0;
  paused_ = 0;
  stime = 0.0;
  rtime = 0.0;
  mac->realSend(e);
}
```

Figure 13. `RealTxPktTdmaTimer::handle` method.

The new `MacTdma realSend` method in Figure x calls send after all the nodes have switched channels.

```
void
MacTdma::realSend(Event *e) {
  send();
}
```

Figure 14. `MacTdma::realSend` method.

B. *NS-2 Simulation*

The following code shows how to select the TDMA class and set the desired number of time slots and seeds for the random number generators.

```
# Set MAC type to the new TDMA MAC
set val(mac)            Mac/Tdma;

# Set the number of time slots to 12
$val(mac) set max_node_num_ 12

# Seed the random number generators
Mac/Tdma set rngf_seed_ $seed1
Mac/Tdma set rngt_seed_ $seed2
```

Figure 15. Example of setting values in simulation scenario.

In the following code, each node is configured with 20 channels. This is done by getting the physical interface, PHY, from the node, and using it to fetch the MAC object corresponding to the PHY. Then the new *add-channel* MAC method is used to add the channel to the MAC. The order that channels are added is significant. Permuting the order will cause the communication scheme to fail (most of the time).

```
# Create 20 frequency channels to switch
between

for {set i 0} {$i < 20} {incr i} {
  set c [new $val(chan)]
  for {set j 0} {$j < $val(nn)} {incr j} {
    set phy [$node_($j) set netif_(0)]
    set mac [$phy up-target]
    $mac add-channel $c
  }
}
```

Figure 16. Example code to add multiple channels to nodes.

An experiment was performed to test the new APEC protocol and compare it to the NS-2 802.11 agent. The nodes were confined to a 9 meter by 9 meter region with radio ranges of 3 meters.

The simulation parameters used are listed in Table 1. All other values were default.

Parameter	**Value**
Number of conversations allowed between source and destination nodes	1
Radio range	3 m
Traffic source	CBR
Interval	20 packets / sec
Packet size (data)	64
Maximum number of packets in interface queue	50
Nodes confined to square region	9m X 9m
Simulation time	900 sec

Table 1: Simulation parameters

The node configuration parameters are listed in Table 4. All other values were default.

Parameter	**Value**
Network interface type	Phy/WirelessPhy
Antenna model	Antenna/OmniAntenna
Radio propagation model	TwoRayGround
Channel type	Channel /WirelessChannel
MAC type	Mac/802_11 Edited Mac/Tdma
Antenna model	OmniAntenna
Radio propagation model	TwoRayGround
Interface queue type	Queue/DropTail /PriQueue
Link layer type	LL

Table 2: Node configuration parameters

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The first simulation scenario was created using 10 wireless communication nodes. The wireless nodes were positioned so that a communication route was formed between the source and destination nodes. Figure 17 shows the 10 node scenario.

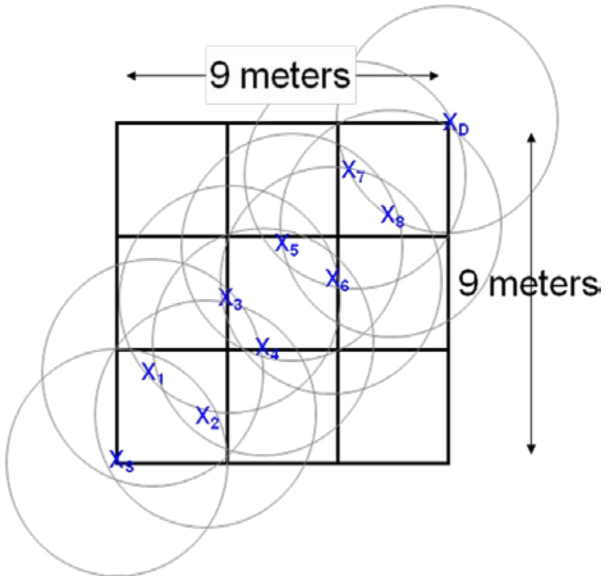

Fig.17. Eight intermediate nodes between source and destination.

UDP over CBR traffic was sent from node X_S to node X_D at a rate of 20 packets per second. Only one conversation was allowed between the source and destination node.

The wireless communication nodes were kept stationary so each simulation run had the same results since everything was deterministic.

The MAC and routing protocols were varied to test how they affected performance. The simulation scenario was repeated using the NS-2 802.11 and APEC MAC agents, Mac/802_11 and Mac/Tdma. The APEC MAC agent was simulated with both DSR and AODV routing protocols. The AODV timeslot count was set to 12.

The throughput for each MAC and routing protocol combination is listed in Table 3.

Routing Protocol	APEC	802.11
AODV	5.714 Kb/s	10.240 Kb/s
DSR	5.718 Kb/s	10.240 Kb/s

Table 3. Average throughput for 10 wireless nodes.

Throughput was measured every 5 seconds. 802.11 traffic flows at a consistent rate, but APEC packets are sent at different timeslots each frame. Figure 18 compares the 5 second interval throughput using 802.11 and APEC with both DSR and AODV.

APEC decreases throughput by approximately 50% in this scenario. This is due to the fact that there is only one sending node waiting an average of 12 timeslot periods between packets. With the default timeslot length calculation, this has the effect of decreasing the send rate from 20 to 10 packets a second for this scenario.

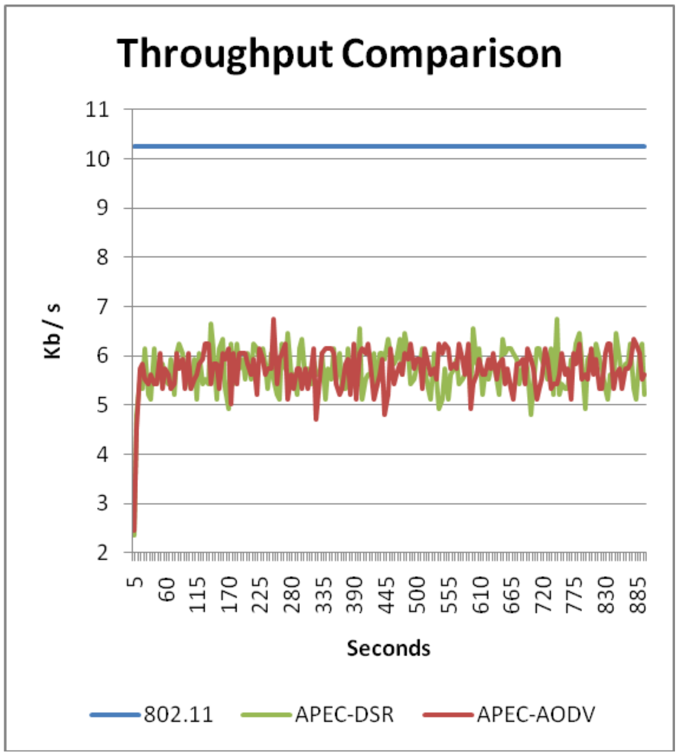

Fig.18. Interval throughput.

Another simulation was performed varying the number of time slots used. Increasing the number of available time slots independently of the number of nodes increases the cryptographic complexity since an attacker must now choose the correct timeslot from a wider field of choices. The odds of guessing the correct timeslot is related to the number of possible slots. Increasing the number of time slots also decreases overall throughput. This follows from the fact that each node will choose exactly one time slot to transmit in during each communication period. The remaining slots are simply not used. In this simulation, the number of nodes is constant (8), and the throughput of a TCP connection between two nodes that must be routed by all of the intervening nodes is used. The data uses a CBR source (20 packets/second, 64 bytes/packet). The results of this simulation are shown in Table 4. There is no variance in the simulated values. This is because the wireless nodes are stationary, and APEC ensures that every node gets exactly one transmit slot during each communication period and the channel is always clear (no collisions).

# slots	Throughput	Varianc e
10	5.971	0.0
20	2.697	0.0
30	1.747	0.0
40	1.276	0.0
50	0.946	0.0
60	0.612	0.0

Table 4. Average throughput for 10 wireless nodes.

The data from Table 4 is graphed below in Figure 19. Throughput drops drastically from 10 to 20 slots, but appears to decrease almost linearly past this point. The authors expect that as the number of time slots is increased further, throughput will asymptotically approach zero. A balance is necessary between cryptographic complexity and performance. Increasing the number slots much greater than the number of nodes has a dramatic effect on performance.

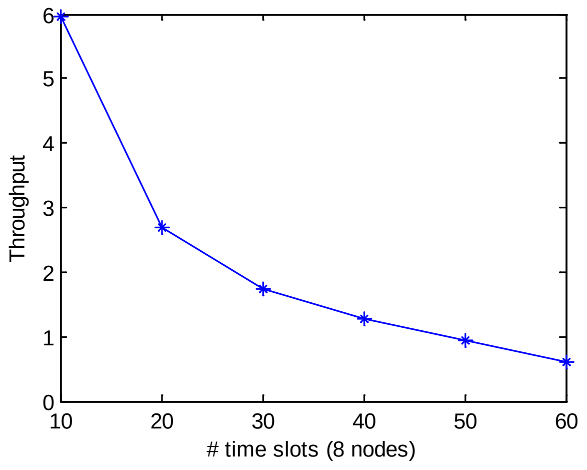

Figure 19. Average throughput for 8 wireless nodes.

V. Conclusion

An authentication protocol for wireless ad hoc networks with embedded certificates (APEC) has been proposed as alternative approach to certificate-based authentication using both time and frequency multiplexing. This protocol removes the need for wireless nodes to explicitly exchange certificates in the authentication process using a collision-avoidance policy for data transmission.

The wireless network simulator, Network Simulator 2 (NS-2), was used to evaluate the APEC protocol. Simulating the APEC protocol required combining FDMA and TDMA algorithms for managing the wireless communication channel.

A new MAC layer protocol was added to the NS-2 network simulator by editing the existing TDMA protocol. New rules for transmitting and receiving packets on the channel were added to the NS-2 MAC layer and channel objects. This new APEC MAC layer protocol implements time and frequency collision avoidance by controlling the time slot and channel that packets are sent and received on.

Simulations comparing the throughput of APEC to a normal 802.11 MAC show a roughly 50% decrease in performance with APEC. This decrease in performance is related to the cryptographic complexity offered by the APEC MAC (and not available in the 802.11 MAC).

Further simulations showed the effect of increasing the number of available timeslots per communication period while maintaining a constant number of nodes. The data shows that performance drops sharply, so a balance must be struck between cryptographic complexity and the performance decrease.

VI. References

[1] Hiromoto, Robert E, ``APEC - An Authentication Protocol with Embedded Certificates," IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Rende (Cosenza), Italy, September 21-23, 2009.

[2] Robert E. Hiromoto and J. Hope Forsmann, ``An Authentication Protocol for Wireless Ad Hoc Networks with Embedded Certificates," Fourth International Workshop on Artificial Neural Networks and Intelligent Information Processing, Funchal, Madeira - Portugal, May 14-15, 2008.

[3] Fall, Kevin and Varadhan, Kannan, "The *ns* Manual", http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf, January 6, 2009.

VII. Appendix

New Classes and Methods

`mac/mac-tdma.h`

```
class Channel;

struct MacTdmaChannelList {
  LIST_ENTRY(MacTdmaChannelList) next;
  Channel *channel;
};

class RealTxPktTdmaTimer : public MacTdmaTimer
{
public:
  RealTxPktTdmaTimer(MacTdma *m) :
MacTdmaTimer(m) {}
  void handle(Event *e);
};

/* Tdma Mac layer. */
class MacTdma : public Mac {

  friend class RealTxPktTdmaTimer;

  public:

  void realSend(Event *e);


  void chooseNewChannel();

  RealTxPktTdmaTimer mhRTX_;

  RNG *rng_f;
  static int rng_f_seed;
  RNG *rng_t;
  static int rng_t_seed;

  LIST_HEAD(,MacTdmaChannelList) chanhead;
  Event realTxIntr;
 };

int MacTdma::rng_f_seed = 1;
int MacTdma::rng_t_seed = 1;
```