Using the ASTRAL Model Checker to Analyze Mobile IP

Zhe Dang and Richard A. Kemmerer
Reliable Software Group
Computer Science Department
University of California
Santa Barbara, CA 93106 USA
+ 1 805 893-4232
{dang,kemm}@cs.ucsb.edu

ABSTRACT
ASTRAL is a high-level formal specification language for real-time systems. It is provided with structuring mechanisms that allow one to build modularized specifications of complex real-time systems with layering. The ASTRAL model checker checks the satisfiability of critical requirements of a specification by enumerating possible runs of transitions within a given time bound.

This paper discusses the mechanism of the model checker and how it can be used to analyze encryption protocols. Several classic benchmarks have been investigated, including the Needham-Schroeder public-key authentication protocol and the TMN protocol, and a number of attacks were uncovered. This paper focuses on using ASTRAL to specify Mobile IP and testing the specification using the model checker.

Keywords
Encryption protocols, Formal methods, Formal specification and verification, Real-time systems, Timing requirements, State machines, ASTRAL.

1 INTRODUCTION

ASTRAL is a high-level formal specification language for real-time systems. It is provided with structuring mechanisms that allow one to build modularized specifications of complex systems with layering[3]. A real-time system is modeled by a collection of process specifications and a single global specification. Each process specification consists of a sequence of levels; each level is an abstract view of the process being specified.

ASTRAL has been successfully used to specify a number of interesting real-time systems, including a CCITT system[4], a complex wide-area phone system by composing several ASTRAL specifications[3], a hardware description language[2] and a robot control system[1]. In[5], the use of ASTRAL to specify cryptographic protocols is demonstrated, and the ASTRAL model checker is used to uncover a number of attacks in the Needham-Schroeder Public Key Protocol and the TMN protocol.

The model checker reported here is part of the ASTRAL Software Development Environment (SDE)[6], which is an integrated set of design and analysis tools based upon the ASTRAL formal framework. The SDE also includes a syntax-directed editor, a specification processor, a verification condition generator, a browser kit, and a mechanical theorem prover. The model checker is an updated version of the prototype reported in[5]. In the prototype, the model checking process was realized by simulating the ASTRAL abstract machine and enumerating all possible execution branches in order to check the critical requirements in a specification. The new model checker, in contrast, generates customized C++ code for each specification. This code is actually a prototype implementation of the specified system and a control module to enumerate all the branches of execution of this implementation up to a system time bound set by the user (Thus, in this case, enumerating all the branches is equivalent to enumerating all the reachable states.). This code generator approach takes advantage of the fact that ASTRAL is a modularized specification language and that the mapping from an ASTRAL specification to C++ classes is quite natural. For example, an ASTRAL process instance can be translated into a C++ class instance. Data types in ASTRAL, like LIST, SET, etc., can also be implemented efficiently in C++. Thus, the code generator makes it possible to model-check a realistic protocol within a reasonable amount of time. Therefore, more realistic specifications can be validated.

In this paper the main focus is on using the ASTRAL specification language to specify Mobile IP[11] and testing the protocol using the ASTRAL model checker. The paper is organized as follows. The next section summarizes the Reliable Software Group's experience with an earlier prototype of the ASTRAL model checker. This is followed by a brief overview of Mobile IP. After that, in section 4, an introduction to the ASTRAL specification

language is presented along with a partial specification of Mobile IP. The specification includes specifying the mobile nodes, the home agents, the foreign agents, the network and the intruder. Several security properties are also formulated in the specification. Validation of the specification is presented in section 5. Finally, conclusions drawn from this work are discussed.

2 EARLIER MODEL CHECKER EXPERIENCES

The Reliable Software Group's experiences with an earlier prototype of the ASTRAL model checker are reported in [5]. This model checker simulated the ASTRAL specification and enumerated all of the possible execution branches up to a user determined time bound. In this section the results of using this model checker on two benchmark protocols are briefly summarized. The protocols that were analyzed are the Needham-Schroeder public-key authentication protocol and the TMN protocol.

Needham-Schroeder Public Key Protocol

The Needham-Schroeder Public Key Protocol [10] serves to exchange nonces between the initiator and the responder using public keys for mutual authentication. The full protocol has the following seven steps:

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_b, B\}_{K_s^{-1}}$
3. $A \rightarrow B : \{N_a, A\}_{K_b}$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{K_a, A\}_{K_s^{-1}}$
6. $B \rightarrow A : \{N_a, N_b\}_{K_a}$
7. $A \rightarrow B : \{N_b\}_{K_b}$

However, by assuming that each participant already knows all the public keys, one can simplify the protocol into 3 messages:

1. $A \rightarrow B : \{N_a, A\}_{K_b}$
2. $B \rightarrow A : \{N_a, N_b\}_{K_a}$
3. $A \rightarrow B : \{N_b\}_{K_b}$

To initiate the protocol, initiator A generates a nonce N_a, encrypts it with its own ID under responder B's public key K_b, and sends it to B. Then B knows N_a by decrypting the message, and generates a nonce N_b, encrypts N_b and N_a under A's public key K_a. When A receives the second message, A knows that B got the initial message by comparing the nonce N_a in these two messages. After A sends the last message to B, in the same way, B authenticates A.

The complete ASTRAL specification of the protocol can be found in the appendix of [5]. There are three process types in the specification: ProcessInitiator, ProcessResponder, and ProcessIntruder. The processes ProcessInitiator and ProcessResponder perform the message passing described in the protocol via the intruder process, ProcessIntruder. ProcessIntruder, serves both as the network and as the intruder. It has legitimate status and can either perform legal activities such as initiating, sending and receiving messages allowed by the protocol, relaying messages to other participants, or it can maliciously produce messages by combining nonces, keys, and user_ids that it knows. Only if the message is encrypted under the intruder's own public key, does the intruder know the content of the message. Otherwise, the intruder can only store, relay or delete a message.

The intruder can arbitrarily delay any messages at will by putting nonsense messages on the network that other honest participants will never acknowledge. The timing condition is not critical in this specification though a duration of 1 time unit was used for each transition (in ASTRAL null duration transitions are not allowed).

The following well-known attack:

1. $A \rightarrow I : \{N_a, A\}_{K_i}$
2. $I(A) \rightarrow B : \{N_a, A\}_{K_b}$
3. $B \rightarrow I(A) : \{N_a, N_b\}_{K_a}$
4. $I \rightarrow A : \{N_a, N_b\}_{K_a}$
5. $A \rightarrow I : \{N_b\}_{K_i}$
6. $I(A) \rightarrow B : \{N_b\}_{K_b}$

was uncovered in 135 seconds, exploring 3657 states with time bound 11 (t starts from 0). The system configuration was 1 initiator, 1 responder, and 1 intruder.

TMN Protocol

The second benchmark protocol that was analyzed using the prototype model checker is the TMN protocol [12]. This protocol is intended to be used in mobile phone systems for session key distribution. It is as follows:

$A \rightarrow S : B, \{N_a\}_{K_s}$

$S \rightarrow B : A$

$B \rightarrow S : A, \{N_b\}_{K_S}$

$S \rightarrow A : B, N_b + N_a$

At the end of the protocol, nonce $\{N_b\}$ is taken as a shared secret between initiator A and responder B.

Similar to the case of the Needham-Schroeder protocol, the ASTRAL model of the TMN protocol is composed of processes representing initiators, responders, servers, and the intruder.

An attack where the intruder can impersonate other legitimate users was found when using the simplest setup, which includes 1 initiator, 1 responder, 1 server, and the intruder. The attack:

$A \rightarrow S : B, \{N_a\}_{K_s}$

$S \rightarrow B : A$

$I(B) \rightarrow S : A, \{N_i\}_{K_S}$

$S \rightarrow A : B, N_i + N_a$

was found in 120 seconds, after searching 3300 states using time bound 8.

The details of the specifications and the analysis of both of these benchmark protocols can be found in [5]. As is the case with many benchmarks, neither of these protocols is very complex. In addition, neither can be considered to be a real-time protocol. To demonstrate the advantages of ASTRAL as a high-level real-time specification language it was necessary to apply the model checker to a more complex time-dependent protocol. The Mobile IP protocol, which is presented in the next section, served this purpose.

3 OVERVIEW OF MOBILE IP

Internetworking protocols like TCP/IP behave awkwardly when a host migrates between networks. IP version 4 assumes that a node's IP address uniquely identifies its physical attachment to the Internet. Therefore, when a mobile node is away from its own home network it will not communicate with other nodes. This is because its IP address is not local to the foreign subnet that it enters. Mobile IP[11], however, provides a mechanism that allows a mobile node to send and receive packets without changing its IP address, even when it is located in a foreign subnet.

The operations of Mobile IP are roughly as follows. When a mobile node(MN) is located on its home network, it operates without mobility services. When an MN changes its physical point of network attachment(e.g., when it migrates from its home network to a foreign network or from one foreign network to another), a registration protocol is employed to keep its home agent (HA), the MN's home host or router, informed about its current attachment. An MN detects its movement by the Agent Discovery service of Mobile IP. The registration protocol is composed of an exchange of a Registration Request and a Registration Reply message between the MN and its HA, possibly through a Foreign Agent(FA), which is a dedicated router on the foreign network that provides mobility services. Mobility agents, which are HAs and FAs, periodically send Agent Advertisement messages to their local subnets, and MNs optionally send Agent Solicitation messages to detect the availability of a mobility agent. When an MN receives an Agent Advertisement, it determines its own current location. If it is on a foreign network, it acquires a care-of address from an Agent Advertisement sent from the corresponding FA. The registration procedure requires the following four messages for an MN to register via an FA.

(1). The mobile node sends a Registration Request to the prospective FA to begin the registration process. This message contains a care-of address, which indicates the MN's current network attachment, and a lifetime, which is the number of seconds remaining before the registration is considered expired.

(2). The FA processes the Registration Request, and also relays the request message to the MN's HA.

(3). The HA sends a Registration Reply to the FA to grant or deny the request. If the request is granted, the HA will update the mobility binding of the MN by setting the MN's current care-of address and the lifetime of this binding. Therefore, the HA knows the MN's current network attachment if the request is granted.

(4). The FA processes the Registration Reply sent by the HA. If the Registration Reply is granted, the FA will update its visitor list by adding the MN. The FA also relays the Registration Reply to the MN to inform it of the disposition of its request.

After the MN successfully registers with an FA, if a correspondent host (CH) communicates with the MN, the packets sent by the CH to the MN's home IP address (the CH doesn't know the current binding of the MN) are routed to the MN's home network by the standard IP routing mechanism. Packets intercepted by the HA are encapsulated into packets destined to the FA and sent to the FA. On receiving an encapsulated packet, the FA strips off the outer header and delivers the packet to the visiting MN on its local network. Packets sent by the MN to the CH are delivered through standard IP routing mechanism (where the FA is the MN's default router). They are not necessarily sent through the HA. The message exchanges for Mobile IP are illustrated in Figure 1.

4 SPECIFICATION OF MOBILE IP

An ASTRAL system specification includes a global specification and process specifications. The global specification contains declarations of process instances, global constants, nonprimitive types that may be shared by process types, and system level critical requirements. There is a process specification for each process type declared in the global specification. In the Mobile IP specification, four process types are declared in the global specification:

```
PROCESSES
```

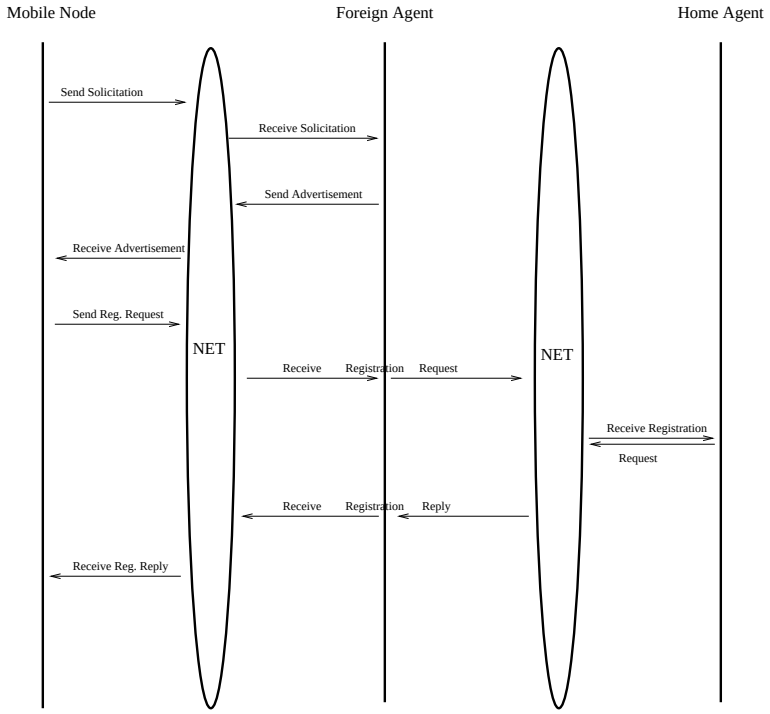

Figure 1: Mobile IP Message Exchange

```
   HomeAgent: array[1..NumberHomeAgent]
                        of ProcessHomeAgent,
   ForeignAgent: array[1..NumberForeignAgent]
                        of ProcessForeignAgent,
   MobileNode: array[1..NumberMobileNode]
                        of ProcessMobileNode,
   Net: ProcessNet.
```

This declaration indicates that there are, for example, `NumberForeignAgent` foreign agent instances of type `ProcessForeignAgent` in the system, where `NumberForeignAgent` is a global constant declared in the constant declaration part:

```
CONSTANT
   NumberForeignAgent: Integer.
```

Besides the primitive types provided by ASTRAL, user-defined types can be declared in the type declaration part. For instance,

```
TYPE
   MobileID: TYPEDEF i: ID (IDTYPE(i) =
                              ProcessMobileNode),
   MsgKindType: (SolictMsg,AdMsg,RegReqMsg,
                              RegReplyMsg),
   PacketType IS STRUCTURE OF
        (Kind:MsgKindType,
         IP_SAdd:ID,
         IP_DAdd:ID,
         LifeTime:Time,
         CareAdd:ID,
         HAdd:ID,
         HAgent:ID,
         Iden:Time,
         Key:Integer),
   PacketListType IS LIST OF PacketType
```

declares four nonprimitive types: `MobileID`, `MsgKindType`, `PacketType`, and `PacketListType`. The type ID is one of the primitive types of ASTRAL. Each process instance has a unique identifier. The ASTRAL specification function `IDTYPE(i)` returns the type of the process with the identifier `i`. `MobileID` represents all identifiers of process instances of type `ProcessMobileNode`. `MsgKindType` is an enumerated type, which indicates that there are four types of messages in the specification of Mobile IP. A unified abstract packet format `PacketType` is declared for all types of messages in order to make the specification simple while not losing generality. This is a structure type, which has nine fields. The field `Kind` distinguishes the corresponding type of the message it represents. `IP_SAdd` and `IP_DAdd` represent the sender address and the designated receiver address, which are identifiers of the process instances of the sender and receiver. The `LifeTime` field has different interpretations for each specific message type. In a Registration Request message, it represents the proposed life time of the pending registration. Similarly, in an Agent Adver-

tisement message, it is interpreted as the life time of the advertisement itself. In an Agent Advertisement message, `CareAdd` is used to stand for the mobility agent that is available to provide mobile service, while in a Registration Request and Registration Reply message it means the care-of address the mobile node is registered to or is attempting to register with. The `HAdd` and `HAgent` fields represent a mobile node's home IP address and a mobile node's home agent IP address. But in agent discovery messages (i.e., Agent Solicitation messages and Agent Advertisement messages) they are not meaningful. The field `Iden` is a timestamp or a sequence number to protect against replay attacks. The last field, `Key`, represents the unique key shared between an MN and its HA. This key is used to compute the authenticator which is used in registration messages and it keeps the intruder from altering any protected fields. All other IP fields, UDP fields, and Mobile IP fields are omitted since they are only meaningful to specific implementations and are not of interest in formally verifying the security properties of Mobile IP. The type `PacketListType` is used to specify the network, where a communication channel is represented by a list of packets.

ASTRAL specifications also define system critical requirements and assumptions about the behavior of both the other processes in the system and the external environment that interacts with the system. In the Mobile IP specification, a global invariant (a requirement holding in every state that can be reached from the initial state, no matter what the behavior of the external environment is) is defined to specify the correctness of the protocol. The invariant is discussed in detail in the last subsection of this section.

Modeling Mobile Nodes

A mobile node is specified by process type `ProcessMobileNode`. Each `ProcessMobileNode` instance contains the variables

```
VARIABLE
   pX: Real,
   pY: Real,
```

which indicate the current physical location of the node.

An ASTRAL transition is described by pairs of entry and exit assertions, and a non zero duration is assigned to each pair. The entry assertion must be satisfied at the time the transition starts, whereas the exit assertion will hold after the time indicated by the duration from when the transition triggers. Transitions are executed as soon as the entry assertion holds assuming no other transition for that process instance is executing. Exported transitions, however, must be called by the external environment in addition to having their entry assertion satisfied. The migration of an MN is specified by the `Move` transition

```
TRANSITION Move
   ENTRY          [TIME : 1]
                  TRUE
   EXIT
                  pX = pX' + 0.1
   EXCEPT         [TIME : 1]
                  TRUE
   EXIT
                  pX = pX' - 0.1
   EXCEPT         [TIME : 1]
                  TRUE
   EXIT
                  pY = pY' + 0.1
   EXCEPT         [TIME : 1]
                  TRUE
   EXIT
                  pY = pY' - 0.1.
```

In ASTRAL exit assertions, variable names followed by a prime (′) indicate the value that the variable had when the transition fired. Thus, the above transition says that if the first entry/exit pair fires, then after 1 time unit the variable `pX` will be increased by the amount of 0.1 unit.

In contrast to the mobile nodes, the physical locations of HAs and FAs are usually permanent, as specified in the specification. Therefore, changes in an MN's local variables `pX` and `pY` can be used to determine which messages are not receivable because the distance between the MN and the source of the message exceeds the global constant `MAX_SOLICIT_RANGE`.

The local variables of a mobile node process

```
VARIABLE
   pendRegStatus: Boolean,
   pendRegRegIden: Integer,
   pendRegRegLifeTime: Time,
   pendRegSentReq: Time,
   pendRegCareAdd: ID,
```

indicate the current mobility binding of the MN. `pendRegStatus` indicates whether the registration is valid or not. If the registration is valid, the next three fields indicate the identification number of the original registration request corresponding to the current registration, the lifetime of this registration, which was granted by the HA, and the time when the original registration request was sent. `pendRegCareAdd` indicates the foreign agent that the MN is currently registered with.

The transition

```
TRANSITION ReceiveAd (j:HomeForeignID)
```

is used to receive Agent Advertisement messages from mobility agents. According to the Mobile IP specification, an MN may also send Agent Solicitation messages to detect the availability of a mobility agent. These solicitations are modeled by the transition

```
TRANSITION SendSolicit.
```

If the MN receives an Agent Advertisement message from a mobility agent other than the currently registered one, the MN will store the advertisement in the variable

```
VARIABLE anotherAgentAd.
```

An MN's loss of contact with its current FA is specified by the following transition:

```
TRANSITION LostContact
  ENTRY          [TIME : 1]
    lostContactWithCurrentFA = FALSE
    & (lifeTimeLastAd + timeReceiveLastAd <= NOW
      | pendRegRegLifeTime + pendRegSentReq <= NOW)
  EXIT
    lostContactWithCurrentFA = TRUE
    & currentAgent = self
    & (pendRegSentReq' + pendRegRegLifeTime' < NOW
                       -> pendRegStatus = FALSE).
```

This states that if the MN fails to receive another Agent Advertisement Message from the current FA before the most recent advertisement expires or fails to receive a Registration Reply message from its HA before the currently pending registration expires, it loses contact with its current FA. If the current valid registration is about to expire, the MN will send a Registration Request message to the mobility agent indicated either in the stored `anotherAgentAd` mentioned earlier or in the pending registration.

The reader should note that according to the Mobile IP specification, the Registration Request message must be authenticated by using the Mobile-Home Authentication Extension. The 128-bit Authenticator value is computed using keyed-MD5 based encryption on the shared secret between the MN and its HA. The authenticator protects all fields of the UDP payload of the Registration Request message except for the UDP header. In the ASTRAL specification, the encryption algorithm is not explicitly specified. In fact, the use of perfect encryption is always assumed. That is, it is assumed that the intruder is not able to break the encryption system. However, the intruder can still replay, redirect, and drop a packet at will. Therefore, based upon the above assumptions, the authenticator is simplified in the specification by including a shared key between the MN and its HA in the Registration Request message and by not allowing the intruder to change any fields of the packet except `IP_SAdd` and `IP_DAdd`, which are the IP source and destination addresses.

The MN is also responsible for receiving a Registration Reply message from an HA, by performing the transition

```
TRANSITION ReceiveRegReply(j:HomeForeignID)
```

Upon receiving a Registration Reply message, the MN will check whether this reply message corresponds to its most recently sent original Registration Request message, whether the original Registration Request's lifetime expired, and whether the key used in this reply message is the key shared with its HA. If this validity check is successful, then the MN will update the `pendRegStatus` variable to be `TRUE`, which indicates that the registration is valid.

Modeling Foreign Agents

A foreign agent is specified by an instance of type `ProcessForeignAgent`, which has local variables

```
VARIABLE
   visitTbStatus(MobileID): Boolean,
   visitTbReqIden(MobileID): Integer,
   visitTbReqLifeTime(MobileID) : Time,
   visitTbTimeRecReq(MobileID): Time,
```

to keep its visitor table information about currently registered MNs inside its local network. `visitTbStatus` indicates whether the registration is valid. If `visitTbStatus(i)` is `TRUE` for some MN `i`, `visitTbReqIden(i)` and `visitTbReqLifeTime(i)` are the sequence number and the lifetime of the corresponding Registration Request message sent by the MN, and `visitTbTimeRecReq(i)` is the time when the Registration Request was received by the FA.

The Foreign Agent process specification in this paper does not include a shared secret association between the FA and HA nor between the FA and a mobile node. This is reasonable because the Mobile-foreign Authentication and Foreign-home Authentication are not mandatory in Mobile IP.

As mentioned above, an FA sends Agent Advertisement messages periodically. It also responds by sending an advertisement when an Agent Solicitation message is received. The major tasks of the FA are to passively relay a Registration Request message to a destination HA, or a Registration Reply message to a destination MN. The transitions for doing this are

```
TRANSITION ReceiveRegReq(j:MobileID)
```

and

```
TRANSITION ReceiveRegReply(j:HomeForeignID).
```

An FA is not allowed to alter the payload of the received packet, except to redirect the packet to the intended receiver. The FA will update its own visitor table upon receiving a Registration Request or a Registration Reply message.

Modeling Home Agents

A home agent is specified by an instance of type `ProcessHomeAgent`. Each instance includes a mobility binding table, which is indicated by variables

```
VARIABLE
   bindTbStatus(MobileID): Boolean,
   bindTbCareAdd(MobileID): ID,
   bindTbRegIden(MobileID): Time,
   bindTbRegLifeTime(MobileID): Integer,
   bindTbTimeRecReg(MobileID): Time.
```

The `bindTbStatus` indicates whether this binding is valid. If it is valid for some MN `i`, `bindTbCareAdd(i)` indicates the care-of address of the MN, and `bindTbRegIden(i)` and `bindTbRegLifeTime(i)` are the identification and life time values of the received Registration Request message sent by the MN. `TimeRecReg(i)` is the time when the request was received by the HA. Besides sending Agent Advertisement messages similar to what was described for an FA, the major tasks of the HA are sending Registration Reply messages and updating its own binding table, upon receiving a Registration Request from an MN. These tasks are performed by the transition,

```
TRANSITION ReceiveRegReq(j:ForeignMobileID).
```

where the validity check is specified by the following statement in the exit assertion of this transition:

```
HomeAgent[SubNet(Net.chan(j,self)[Net.head(j,self)]
                                        [HAdd])] = self
& Net.chan(j,self)[Net.head(j,self)][Iden]
     + Net.chan(j,self)[Net.head(j,self)][Lifetime]
                                        > NOW
& Net.chan(j,self)[Net.head(j,self)][Key]
       = SHARED_SECRET(self,Net.chan(j,self)
                            [Net.head(j,self)][HAdd])
& bindTbRegIden'(Net.chan(j,self)[Net.head(j,self)]
                            [HAdd])
         < Net.chan(j,self)[Net.head(j,self)][Iden]
```

The term, `Net.chan(j,self)[Net.head(j,self)]`, represents the received Registration Request packet. In the above exit assertion, the first conjunct states that the requesting MN must have the same subnet mask as the HA. The second conjunct states that when the HA receives the Registration Request, this request has not expired due to network delay. The third conjunct states that the authenticator which is the shared key between the HA and the MN is valid, and the final conjunct states that the received request is a fresh request.

Modeling the Network and the Intruder

The underlying network is specified by the unique instance `Net` of type `ProcessNet`. The network contains all packets sent along each channel between two agents, where each channel is declared as a list of packets:

```
VARIABLE chan(AgentID,AgentID):PacketListType.
```

Channels are unidirectional and FIFO, i.e., `chan(i,j)` represents the list of packets sent from `i` to `j`. The transitions `PacketIn` and `PacketOut` are used to deliver packets that are ready for the network and to remove packets that have already been received. `Net` plays the role of an intruder in the sense that the intruder has full control over the network. The intruder can replay, redirect, drop, and inject a packet at will. Thus, the intruder can alter the source and the destination address, but cannot alter the other fields in the packet. This limitation was chosen for the following reasons:

a) Protection from redirection and replay attacks are extensively discussed in the Mobile IP specification. Verification of the protocol when the intruder is allowed to replay and redirect authenticated packets is therefore the primary concern of the specification.

b) As mentioned earlier, the use of perfect encryption is always assumed. Hence, the intruder is not able to break the encryption system. Each Registration Request and Registration Reply message is authenticated with the corresponding shared key. Therefore, if the shared key used between an MN and an HA is not known to the intruder, then any unintended alteration of the authenticated message will be detected by the corresponding receiver when validity checks are performed. This is the reason why the intruder is not allowed to alter any protected fields in Registration Request and Registration Reply messages.

c) The Agent Advertisement messages from a mobility agent and the Agent Solicitation messages from a mobile node are not authenticated in Mobile IP. In the model presented in this paper, the intruder is not allowed to change these unauthenticated messages. Future analysis of the Mobile IP will likely include this possibility.

d) The state space can easily explode if the intruder is able to change other fields in a packet at will.

Based upon the above reasoning, the security property for the Mobile IP specified in this paper is that the protocol is free from replay and redirect attacks when a perfect encryption scheme is employed. Denial of service attacks are not protected by the Mobile IP; therefore, it is less interesting to specify them in the specification.

In the specification, the intruder's behavior is characterized by three transitions. The first

```
TRANSITION IntruderMaliciousPacket(i:AgentID,j:AgentID)
```

produces a new packet with the destination field of a ready-to-deliver packet of agent `i` being changed to agent `j`. This new packet is stored in the local variable `packet`. The second transition

```
TRANSITION IntruderDrop(i:AgentID,j:AgentID)
```

removes a packet from the channel `chan(i,j)` when the channel is not empty. The intruder can drop at most one packet at any time. The third transition

```
TRANSITION IntruderInsert
```

inserts the packet generated by `IntruderMaliciousPacket`, into the corresponding channel when the channel is not full.

The packet generated by `IntruderMaliciousPacket` can be a redirected packet or an unchanged packet. If it is a redirected packet, then this packet is inserted in the message channel when `IntruderInsert` fires, and the network eventually delivers this packet to the redirected destination. Thus, message redirection is modeled. If the generated packet is an unchanged packet and `IntruderInsert` fires after the packet has been sent out to the network by the transition `PacketIn`, then the generated packet is inserted in the message channel and the inserted packet is actually a replay packet. Thus, message replay is modeled.

Specifying Security Properties

In the Mobile IP model presented in this paper, the intruder is not allowed to change the protected fields of an authenticated packet; therefore, data integrity of the payload is naturely preserved and there is no interest in specifying it as a critical requirement of the specification. In addition, privacy, which is largely dependent upon a particular implementation, is not in the scope of Mobile IP. Thus, the security properties modeled in this paper concentrate on agreement between HA and MN's views of the current registration status, which is mandatory for Mobile IP to guarantee correct registration.

The correctness of the protocol is expressed by the following global invariant:

```
FORALL i: MobileID, j: HomeID
 ((i.pendRegStatus = TRUE & j = i.HAgentAdd
   & i.pendRegRegLifeTime + i.pendRegSentReq > NOW)
     -> (j.bindTbStatus(i) = TRUE
         & j.bindTbCareAdd(i) = i.pendRegCareAdd
         & j.bindTbRegLifeTime(i)
             + j.bindTbTimeRecReg(i) > NOW)),
```

which says that for every mobile node `i` and home agent `j`, if `j` is `i`'s home agent and `i`'s current registration is valid and isn't expired, then the mobility binding of `i` in its home agent's binding table is valid and hasn't expired either. Furthermore, the mobility agent that `j` believes `i` is currently attached to is actually the one for which `i` is currently registered.

The fact that a home agent registers an MN only if the MN previously sent a Registration Request message requesting the registration is expressed by the local invariant of `ProcessHomeAgent`:

```
FORALL i: MobileID
 ((HomeAgent[SubNet(i)] = self
   & bindTbStatus(i) = TRUE)
      ->   EXISTS t: Time
       (t < bindTbTimeRecReg(i) & t > 0
        & past (i.packet[HAgent],t) = self
        & past (i.packet[Kind],t) = RegReqMsg
        & past (i.packet[Iden],t)
                      = bindTbRegIden(i))),
```

which states that if the HA believes MN `i`'s registration is valid, then MN `i` really sent this Registration Request message to this HA. The request message is uniquely identified by the `Iden` field value, which is a time stamp of the corresponding message.

In a similar manner, the fact that every Registration Reply message is originated from the mobile node's home agent is expressed by the local invariant of `ProcessMobileNode`:

```
pendRegStatus = TRUE
  -> EXISTS t: Time
   (t < NOW & t > 0
    & past(HAgentAdd.packet[Kind],t) = RegReplyMsg
    & past(HAgentAdd.packet[HAdd],t) = self
    & past(HAgentAdd.packet[Iden],t)
                                 = pendRegRegIden).
```

If the MN receives a Registration Reply message, then its HA previously sent this reply message, and it is a

response to the Registration Request of the current registration sent by the MN.

5 VERIFICATION USING THE ASTRAL MODEL CHECKER

The specification was constructed and type-checked using the syntax-directed editor of the ASTRAL SDE. A validated ASTRAL specification can be model checked by invoking the model checker component of the ASTRAL SDE.

The ASTRAL abstract machine is, in general, an infinite state machine with unlimited time bounds for system evolution. However, in order to make the model checker practical, a finite time bound needs to be set by the user. The time bound indicates the maximal depth of the state space that the model checker will search the current specification. Therefore, the model checking process begins by setting a time bound. In addition, if there are system constants or process constants in the specification, the values of these constants also need to be set. The model checker then generates a customized C++ program for the specification. The resulting program performs the model checking process by enumerating all possible states within the given time bound. If a violation of the specification's critical requirements is detected, a trace of the states of the specified system is reported to the SDE. Therefore, a user is able to debug the specification or discover flaws in the protocol.

For the Mobile IP specification, the model checker was able to find a number of violations in preliminary versions of the specification, which were all the result of specification errors. For the current version of the specification, no errors were found under the various constant settings that were tried. Figure 4.1 shows the results of running the model checker on the specification. No violations were found after searching 10^8 states on the three set-ups shown in the figure. The generated C++ code was compiled by the GNU C++ compiler with no optimization flags set. All results run on a Sun Ultra 1 with CPU time measured.

set-up	states searched	lines of C++	time bound	search time (secs.)
1HA, 1FA, 1MN	10^8	12,584	20	12,097
2HA, 2FA, and 2MN from different HAs	10^8	31,358	20	34,467
2HA, 2FA, and 2MN from the same HA	10^8	31,358	20	33,229

Figure 4.1. Test results

6 DISCUSSION

The model-checking technique used in this paper is explicit state exploration. As compared to the old model-checker[5], the new model-checker is about 100 times faster on average. This is because the new one generates C++ code instead of simulating a specification within a virtual machine, as was the case for the old model-checker. Explicit state exploration is also used in the Murphi model-checker[9]. The Murphi compiler generates a C++ program for a Murphi program, which exhaustively generates the reachable states. However, the Murphi description language can only be used to describe finite state asynchronous concurrent systems. ASTRAL aims at specifying real-time (infinite state) systems, by providing rich data types, timing constructs, environmental assumptions and critical requirements within an ASTRAL specification. The model-checker presented in this paper handles a subset of ASTRAL[5], due to the fact that validity checking of a general ASTRAL formula is undecidable. A symbolic model-checker that could handle a larger subset of ASTRAL s highly desirable. This is one of the ongoing efforts in the Reliable Software Group at UCSB.

Several previous works[5][7][9] show that a model-checker is a useful tool in verifying security protocols such as the Needham-Schroeder public-key authentication protocol and the TMN protocol. However, the protocols considered are simple in the sense that the security properties for those protocols can be specified in a high level, in which many details of the low level network communications can be abstracted away. For a much more complex protocol like Mobile IP, it is worthwhile to see how much the model-checking technique is applicable.

In a recent publication, McCann and Roman also present a Mobile UNITY specification of Mobile IP[8]. Their specification consists of a set of parameterized programs: *mobile-node*, *home-agent*, *foreign-agent* and *network*. Such a partition is similar to ours. A detailed level of communication model is specified, along with protocol correctness properties. But without any discussions of security properties, the intruder's behaviors are not specified in the specification. Therefore, the correctness properties are those without presences of intruders. Besides, Mobile UNITY does not provide an automatic verification tool at current time.

7 CONCLUSIONS

This paper presents an ASTRAL formal specification of the Mobile IP protocol. This protocol is particularly interesting because its definition includes timing requirements. For example, a mobile node needs a time reference in order to decide whether its current regis-

tration is going to expire, and a timestamp mechanism is needed in order to protect against potential replay attacks. Because ASTRAL is a specification language for real-time system, it has the necessary timing primitives to make the specifications of these requirements straightforward.

The rich data types of the ASTRAL specification language also provided the data structures needed to specify a scalable Mobile IP specification. Thus, unnecessary unfaithful simplifications could be avoided. Simplifications were made at several places in the Mobile IP model presented, but they were made to reduce the state space exploration of the model checker. They were not due to any lack of expressive power in the ASTRAL language.

The model checker runs that were made on the current version of the Mobile IP specification have not revealed any errors in the protocol. The next step will be to apply the ASTRAL theorem prover to the specification to see if the model can be fully verified or if any errors can be discovered. The full version of the ASTRAL specification is available at `http://www.cs.ucsb.edu/~dang`.

ACKNOWLEDGEMENTS

The authors would like to thank Professor Sarah Mocas from Portland State University for helpful discussions on Mobile IP. Thanks are also given to Paul Kolano for many discussions on the ASTRAL SDE.

REFERENCES

[1] K. Brink, L. Bun, J. van Katwijk, and W. J. Toetenel, *"Hybrid specification of control systems," First IEEE International Conference on Engineering of Complex Computer Systems*, Ft. Lauderdale, Florida, November 1995.

[2] G. Buonanno, A. Coen-Porisini, and W. Fornaciari, *"Hardware specification using the assertion language ASTRAL," Proceedings of the Advanced Research Workshop on Correct Hardware Design Methodologies*, Torino, Italy, June 1991.

[3] A. Coen-Porisini, C. Ghezzi, and R. A. Kemmerer, *"Specification of real-time systems using ASTRAL," IEEE Transactions on Software Engineering*, Vol. 23, No. 9, 1997, pp. 572-598.

[4] A. Coen-Porisini, R. A. Kemmerer, and D. Mandrioli, *"A formal framework for ASTRAL intralevel proof obligations," IEEE Transactions on Software Engineering*, Vol. 20, No. 8, 1994, pp. 548-561.

[5] Z. Dang and R. A. Kemmerer, *"Using the ASTRAL model checker for cryptographic protocol analysis," Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers University, 1997.

[6] P. Z. Kolano, Z. Dang, and R. A. Kemmerer, *"The design and analysis of real-time systems using the ASTRAL software development environment,"* to appear.

[7] G. Lowe, B. Roscoe, *"Using CSP to detect errors in the TMN protocol," IEEE Transactions on Software Engineering*, Vol. 23, No. 10, 1997, pp659-669.

[8] P. J. McCann and G. -C. Roman, *"Modeling Mobile IP in Mobile UNITY,"* to appear in *ACM Transactions on Software Engineering and Methodology.*

[9] J. C. Mitchell, M. Mitchell, and U. Stern, *"Automated analysis of cryptographic protocols using Murφ," Proceedings of the IEEE Symposium on Security and Privacy*, 1997, pp141-151.

[10] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, Vol. 21, No. 12, December 1978, pp. 993-999.

[11] C. Perkins, *"IP mobility support", Network Working Group Request for Comments: 2002*, October 1996.

[12] M. Tatebayashi, N. Matsuzaki, D. B. Newman, "Key Distribution Protocol for Digital Mobile Communication Systems," *Advances in Cryptology - CRYPTO '89*, LNCS 435, G. Brassard, ed. Springer-Verlag, 1991, pp. 324-333.