Modeling Vulnerabilities of Ad Hoc Routing Protocols

Shahan Yang
Electrical and Computer Engineering Department and
Institute for Systems Research
University of Maryland College Park
College Park, MD 20742, USA
syang@isr.umd.edu

John S. Baras
Electrical and Computer Engineering Department and
Institute for Systems Research
University of Maryland College Park
College Park, MD 20742, USA
baras@isr.umd.edu

ABSTRACT
The purpose of this work is to automate the analysis of ad hoc routing protocols in the presence of attackers. To this end, a formal model of protocol behavior is developed in which time is modeled by a set of constraints on the time of occurrence of events, enabling the representation of partially ordered timed events and asynchronous communication. Data variables are represented symbolically, capturing a range of distinct executions in each expression. Given a formal description of Ad Hoc On Demand Distance Vector Routing Algorithm (AODV) and a desired safety property (route stability), an analysis by a naive semi-decision procedure discovers an instance of an attack that leads to a violation of the property.

Categories and Subject Descriptors
C.2.2 [**Computer Systems Organization**]: Computer-Communication NetworksNetwork Protocols[protocol verification, routing protocols]

General Terms
Ad Formal Verification, Ad hoc Wireless Networks, Security

1. INTRODUCTION
Current approaches to securing ad hoc routing protocols focus on using secure signatures to authenticate routing information and some built in mechanisms for resisting or detecting Byzantine attacks. However, authentication is not always sufficient to safeguard the network from malicious nodes, as in the case of publicly accessible wireless networks or in the case of physical compromise, to which mobile devices are thought to be more susceptible. Also, methods for detecting Byzantine attacks are often susceptible to attack themselves.

This work focuses on developing formal models for ad hoc networks routing protocols with the objective of modeling Byzantine or insider attacks. Such a model may aid protocol designers in mitigating their effects or reducing the number of vulnerabilities of this type. In addition, formal models have been used previously in intrusion detection to protect routing in fixed networks. A formal model for ad hoc network routing may also prove useful in constructing intrusion detection systems in a mobile environment.

Proceedings of the 1st ACM Workshop Security of Ad Hoc and Sensor Networks Fairfax, Virginia

2. RELATED WORK
This work is closely related to verification of routing protocols. A notable example is the work on verifying the loop freedom of AODV [2] [8]. The authors used formal models of AODV to discover conditions leading to the formation of routing loops by model checking finite models of AODV in SPIN [6]. They proceeded to design a repair for AODV that eliminates this problem and verified that the repaired version of AODV is loop free under some assumptions (prior to restarting its AODV process, a node must ensure that all of its neighbors detect the restart). The verification combines a mechanically assisted proof using HOL with finite state model checking performed by SPIN.

The approach that this work draws most heavily from is the automated tool Athena, which has been used to verify secrecy and nonrepudiation in security protocols [10]. The tool Athena is based on the Strand model [3], which provides a framework for proving properties of security protocols. Athena has several advantages over other approaches that would be useful in the study of routing protocols. For instance, since the model of variables is symbolic, it is possible to describe an infinite range of executions within a finite expression. Also, partial order reduction is an inherent part of the model. Rather than representing time as a linear or branching structure, time is implicit and only appears as a set of constraints on the causal ordering of events. Athena is so efficient at checking protocols that it is possible to search the space of possible protocols to automatically generate them [9].

As a combination of model checking and theorem proving, Athena uses only symbols and simple operations on these symbols to represent the variables occurring in the system. Athena then uses a model checking procedure to check the satisfiability of these logical formulas that describe scenarios in their system. It manipulates the data variables symbolically, but explores the control paths explicitly. Consequently, only a finite number of control paths can be checked

by Athena. It is possible to show under certain assumptions that this suffices to capture all possible behaviors of a protocol that could satisfy a given formula. However, it is doubtful that the same result applies to the problems addressed in this paper, because in routing algorithms, the number of control paths includes all the infinite possible sequences of topology changes.

An alternative to verifying protocols is subjecting them to automatically generated fault oriented tests [4] [5]. The protocols are modeled as finite state machines and rather than checking the entire state space for violations of the correctness criteria, one only checks states reachable from faults (low level anomalous, but correct behavior). Once incorrect states reachable from faults are identified, a backwards search is performed from the fault to determine if it is reachable. This search is performed for each type of fault and each message of the system and yields a set of tests that lead to error states. The idea of restricting the search to faults helps to mitigate the effect of state space explosion in blind searches of the protocol state space. This work places emphasis on simulation of the system using the test sequences leading to realistic examples of protocol errors.

The example protocol chosen for study is the distance vector routing algorithm AODV. A secure variant of AODV called SAODV has been proposed [11]. This approach assumes that nodes have public and private keys and that each node is able to securely determine if a given public key belongs to another given node. Security is improved over AODV, which has no security provisions by signing routing messages and using hash chains to sign mutable fields of routing messages. However, SAODV does not address the problem of malicious insiders.

3. CRITERIA FOR SECURE ROUTING

Attacks may come in many forms in an ad hoc routing protocol. It is possible that an attacker may compromise nodes and make them behave arbitrarily. It is also possible for an attacker to jam the physical medium. Survivability demands of an ad hoc routing protocol that it is able to maintain service under hostile conditions. Eavesdropping is always possible by passive snooping on the broadcast medium, so it is not an attack that is considered here. The attacks of interest are those that disrupt routing services provided by the protocol even if there exist uncompromised routes through the network.

Given the above assumptions, it would be desirable for secure routing protocols to provide routing service even in the presence of attackers. Most of the secure routing algorithms require authentication of routing messages. Based on whether or not they may generate these authenticated messages, there are two categories of attackers, outsiders and insiders. A malicious outsider is unable to generate authentic routing messages. However, he is able to replay messages that are generated by legitimate parties and to degrade communication between nodes within his broadcast range by jamming the lower layers of communication. A malicious insider can perform all the attacks that an outsider can. Additionally, a malicious insider has the keys necessary to generate authentic messages for his own identity. If malicious insiders cooperate and share their keys, each insider

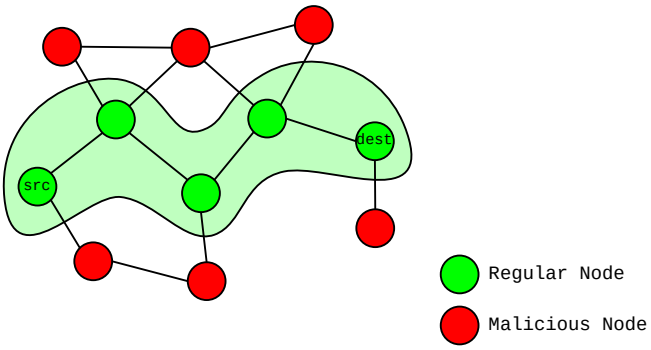

Figure 1: Uncompromised path.

may generate any message appearing to originate from any of the compromised nodes.

In secure environments, it is possible to apply sufficient safeguards to the nodes themselves to prevent compromise, and it is only necessary to consider outsider attacks. However, malicious insiders are a more realistic threat in less controlled environments, such as public access networks.

An ideal routing protocol should be able to provide routing service in the presence of any number of attackers, as long as there is a stable, uncompromised path through the network. There are two criteria that define an uncompromised path through the network expressed as safety and liveness conditions. For safety, every node in the path is normal, that is non-malicious and its key has not been disclosed. If there are malicious nodes in the path, then there is always a way for that node to block the route discovery process along that path. For liveness, communication must be possible along the path. This is necessary because it is possible for malicious nodes not belonging to the path to compromise it by either jamming lower protocol layers or denial of service attacks at the routing message level. For example, continuous routing messages that fill message queues or require processing may prevent legitimate routing messages from ever being processed, as a malicious node is unlikely to respect fairness requirements. More precisely, there must be an upper bound on the amount of delay between nodes in the path.

These requirements are very strong and it may be the case that for certain routing algorithms to operate in the presence of attackers, it is necessary for there to be redundancy in the paths available. It could also be the case that routing through compromised nodes is the only option or the most efficient. This work currently addresses neither of these cases.

In both cases, with or without malicious insiders, a possible sufficient condition for security comprises two assertions: safety and liveness. The safety condition asserts that discovered routes are stable. Otherwise, paths that are discovered can be broken by the attacker as they are formed. This can result in indefinite denial of routing service. The liveness condition asserts that it is possible to discover the route.

These conditions can be expressed formally with sentences

of CTL. The safety condition can be expressed with the sentence

$$\mathbf{A}\Box(R \rightarrow \mathbf{A} \bigcirc R)$$

where R is a state predicate describing the state in which the routing table has an uncompromised route to the destination. The precise definition of R depends on the specific topology and how the routing protocol defines routes. The liveness condition is expressed in terms of a collection of assertions parameterized on a well founded set W. The state predicate R is similar to the one defined above and asserts that in the current state, the route exists. The predicates φ_i where $i \in W$ are a chain of assertions that describe a set of actions, whose inevitable outcome is R. The liveness condition is given below.

$$\mathbf{A}\Box(\mathbf{A}\varphi_i \ \mathcal{U} \ R \vee \bigvee_{j<i} \varphi_j)$$

The definition of state predicates φ_i and W depends on the particular protocol and topology. These two rules are complete for describing safety and liveness properties [7], so these sentences should be sufficient for describing any protocol.

4. EXTENDED STRANDS MODEL

The Strand model exploits the nonbranching behavior of security protocols where each participant executes a fixed sequence of input and output events. However, this will not be entirely appropriate for routing protocols. Since the behavior of routing protocols contains branches, it is necessary to extend the Strand model. This work presents an extension of the strand model that enables it to capture the behavior of routing protocols for the purposes of verification.

4.1 Partial Order Semantics and CTL

The model of concurrent execution for the proposed model is similar to the Strand model, where events are partially ordered by the transitively closed causality relation. Since constructs from temporal logic are used, it is necessary to ensure that the semantics are compatible. The technical issue concerns true concurrency. In CTL, every execution, corresponding to one path through the tree, must admit a *total* ordering of events. This is in conflict with the idea of modeling events by a system of constraints because in the constraint model it is possible for two events to occur exactly simultaneously. For very straightforward reasons, this technical discrepancy is immaterial for the systems under consideration.

Let $<$ be the binary causal relation on the time of occurrence of events. There are three categories of events in the partially ordered model of time: causally independent, causally dependent and mutually causally dependent.

If two events, α and β are causally independent, $\neg\alpha < \beta$ and $\neg\beta < \alpha$, then there are three possible executions: $< \alpha, \beta >$, $< \beta, \alpha >$ and $\alpha|\beta$. $\alpha|\beta$ denotes true concurrency of the events α and β. Suppose that the ordering of the events α and β may affect the state reached. This implies that α and β must both affect some common process p. This can only occur if α and β synchronize in the timeline of p so they are causally dependent contradicting the hypothesis. The case where $\alpha|\beta$ is ruled out by the same argument because synchronization in a single process implies a total ordering of events. This behavior is easily captured by the branching time semantics. The argument may be extended to encompass sets of causally independent events.

If two events, α and β are dependent, then they are totally ordered, which easily captured by the branching time semantics.

Finally, if events α and β are mutually causally dependent, that is, they must occur simultaneously, then in the branching time semantics, the events must be treated as a single event with a shared label. This might create difficulties if the causal constraints indicate that events must be concurrent, as in a causal loop. This is impossible though because a loop including α requires an event in process a that precedes α to be caused by α and all events preceding α in a strictly precede it.

4.2 Messages and events

The proposed model, in the context of a LTS has an alphabet Σ consisting generally of tuples of integers. These tuples have special structure however, associated with their semantics. The terms "event", "action" and "message" refer to syntactic constructs rather than the particular events, actions or messages of an execution. The term "event instance" will be used to refer to an actual occurring event.

Events are composed of an action and a message. The action describes what the event does. It may be a communication or a synchronization point in the process. The message embodies the information that is associated with the event.

The following are the defined action symbols representing different kinds of events.

$+$ directed message send.

$-$ message receipt.

$\star$ message broadcast.

$\triangleleft$ state precedence.

$\triangleright$ state succession.

A message is a set of symbols. Each symbol in the message represents some data field that is included in the message. For example, $\sigma = \{s, d, f_1, \ldots, f_m\}$ is a directed message.

An event is formally a tuple (a, V), written aV, where a is one of the action symbols and V is a message. For events with $+$ actions, the message must be directed and include at least an element for both source and destination. For $\star$ actions, the message is broadcast so it must include the source but not necessarily the destination. Finally, for $\triangleleft$ or $\triangleright$ actions, the message must be a set containing a single symbol representing the identity of the process such as $\{p\}$. For example, $\triangleleft\{p\}$ is an event and if σ is a directed message then $+\sigma$ and $-\sigma$ are both events.

4.3 Role

Roles describe implicitly the state space and the behavior of processes in the protocol. In a general sense, this extended role definition combines actions of TLA with the behavioral model of Strands. The TLA action is used to describe the change in state variables, which the Strand model does not have, while Strands are used as a partially ordered, goal driven model of execution.

Formally, a role R is a tuple $R = (X^i, X^f, M, \Phi(L))$.

X^i set of state symbols representing values of the state variables prior to the execution of the role.

X^f set of state symbols representing values of the state variables after the execution of the role.

M a sequence of events. If an event with the action $\triangleright$ occurs in this sequence, then it can only be the first event and it must be of the form $\triangleright\{p\}$ where p is a participant symbol. Similarly, if $\triangleleft$ occurs, then it must be the last event and of the form $\triangleleft\{p\}$.

For some finite sets Σ^d of directed messages and Σ^b of broadcast messages, the rest of the events are of the form $+\sigma_1$, $\star\sigma_2$ or $-\sigma_3$ for some $\sigma_1 \in \Sigma^d$, $\sigma_2 \in \Sigma^b$ and $\sigma_3 \in \Sigma^d \cup \Sigma^b$. For example

$$\langle \triangleright\{p\}, +\sigma_1, \ldots, +\sigma_m, \triangleleft\{p\} \rangle$$

where $\sigma_1, \ldots, \sigma_m \in \Sigma^d$, is such a sequence.

Define the length of a role R, written $l(R)$ as the length of the sequence of events M. Also define the i^{th} event of role R, written $R[i]$, where $1 \leq i \leq l(R)$ as the i^{th} event in the sequence of events M.

Φ a formula of first order logic with free variables from the set of symbols L.

L the complete set of uniquely labeled symbols included in X^i, X^f and M. Let $M = \langle a_1 V_1, \ldots, a_m V_m \rangle$. Let the distinguishable message symbols $\Omega = \bigcup_{i=1}^{m} \{i\} \times V_i$. The following mappings define L.

g_i $X^i \to L_i$ where g_i is 1-1 onto.

g_f $X^f \to L_f$ where g_f is 1-1 onto.

g_M $\Omega \to L_M$ where g_M is onto but generally not 1-1. There is a particular element of L_M that would not have a well defined inverse mapping, though the rest of the elements of L_M would. Let α denote this symbol in L_M. If $a_i = \triangleright$ or $a_i = \triangleleft$, then $V_i = \{p\}$ for some symbol p and $g_M(i, p) = \alpha$. If $a_i = +$ or $a_i = \star$ then V_i contains s where s is a symbol for the sender and $g_M(i, s) = \alpha$. Similarly, if $a_i = -$ and V_i is a directed messsage, then the V_i contains a symbol r corresponding to the recipient and $g_M(i, r) = \alpha$. Every other element of Ω maps to a distinct element of L_M that is not α.

All pairs of ranges from L_i, L_f, L_M should have empty intersections. Let g be the union of g_i, g_f, g_M. $L = L_i \cup L_f \cup L_M$, the range of g.

The shared symbol α represents the *id* of the participant, which should remain constant throughout. In constrast, fields associated with state variables and message field values are potentially different and therefore labeled distinctly.

4.3.1 Logical Theories for Φ

The description of roles has so far been syntactic. The actual logical theory of Φ will depend on the protocol being studied. For most protocols, this theory will be some reduct of number theory but it is possible to have other theories, such as the theory of real numbers. The formula $\Phi(L)$ is always decidable because it models the next state function of a protocol for which an effective decision procedure must exist.

It is possible that Φ is merely a set of formulas as with an integer program. For example

$$l_1 + l_1 + l_2 + l_2 + l_2 = 0$$

where $l_1, l_2 \in L$ is such a formula which is equivalent to

$$2l_1 + 3l_2 = 0$$

in terms of what relation it defines on l_1 and l_2. This is a subset of Presburger arithmetic, $\mathfrak{N}_A = (\mathbb{N}; 0, S, <, +)$ for which satisfiability is exactly the same thing as solving an integer program. While decidable, like Presburger arithmetic, no decision procedure is fast enough for very long formulas.

In the case of routing protocols such as AODV and TORA, it suffices to consider the theory of the structure

$$\mathfrak{N}_L = (\mathbb{N}; 0, S, <).$$

This theory admits the elimination of quantifiers, which is stronger than decidability. Some examples of atomic formulas of this structure include

$$\begin{aligned} l_1 &= l_2, \\ l_1 &\leq l_2, \\ l_1 &< l_2, \\ l_1 &+k = l_2 \end{aligned}$$

where $l_1, l_2 \in L \cup \mathbb{Z}$, $k \in \mathbb{Z}$. The formula $l_1 \leq l_2$ is equivalent to the formula $l_1 < l_2 \vee l_1 = l_2$ and the formula $l_1 + k = l_2$ is merely a suggestive way of writing the formula $s^k l_1 = l_2$. It is possible to extend the language to include relations $+k =$ as defined. Observe that on the structure $\mathfrak{N}_L$, $(+k =) \subset (<) \subset (\leq)$ for any $k \in \mathbb{Z}$. The inclusion of these extra relational symbols does not change the expressiveness of the language but they do facilitate the description of an efficient satisfiability procedure for terms of this language to be described in Section 4.10.

BDDs might also be a good choice for the representation Φ. In this representation, each variable represented in L is a set of Boolean variables, and Φ can then be expressed in terms of the characteristic function of a Boolean function on L, which may be easily represented by a BDD.

4.4 Protocol

Define a protocol as the tuple $P = (\Sigma, \Delta)$.

Σ set of messages. $\Sigma = \Sigma^d \cup \Sigma^b \cup \{p\}$ where Σ^d is a set of directed messages, Σ^b is a set of broadcast messages and p is a symbol representing a participant.

Δ set of roles of the protocol where for all $R \in \Delta$ where $R = (X^i, X^f, M, \Phi)$ the messages of events in M are all in Σ.

4.5 Strand

A *strand* is a prefix of an execution of a role. Formally, a strand is a tuple $\theta = (R, k, I)$, where

R a role, $R = (X^i, X^f, M, \Phi(L))$.

k a number $1 \leq k \leq l(R)$ giving the number of events of the role that this strand instantiates.

I some set of instance symbols. I has an element for each symbol occurring in L such that there is a 1-1 onto mapping $h : L \rightarrow I$.

Each instance symbol is unique. For any two strands $\theta_1 = (R_1, k_1, I_1)$ and $\theta_2 = (R_2, k_2, I_2)$ $I_1 \cap I_2 = \phi$. Even though the strand represents only the execution of the first k events of the role, set I still contains all the symbols associated with L, because it might not be possible to express all the applicable constraints from Φ otherwise.

$t = (\theta, i)$, where $\theta = (R, k, I)$ is a *node* of θ if $1 \leq i \leq k$, written $t \in \theta$. t refers to the execution of event $R[i]$.

4.6 Goal binding

A goal is an event required to occur by a particular node in order for that node's event to occur. For a node $t = (\theta, i)$, where $\theta = (R, k, I)$, $1 \leq i \leq k$ and $R[i] = aV$, the the goal is bV, where b is some event depending on the action a as described below.

$a = +$ No goals are associated with actions of this type.

$a = -$ Message receipt has as its goal the sending of the same message, either through a broadcast or a directed send, depending on the message type. If V is a directed message, then the goal is $+V$, if it is a broadcast message, then the goal is $\star V$. There is exactly one goal instance for nodes having this action.

$a = \triangleright$ The single goal of nodes having this kind of event is $\triangleleft\{p\}$.

$a = \triangleleft$ This type of node has no goals.

Binders are also events, and these will satisfy the goals described above. Nodes having certain actions will have binders, and for these nodes the binder is $R[i]$. The number of instances of the binder depends on the binder's action symbol.

$a = +$ Exactly one instance of the binder.

$a = \star$ The number of instances of this binder depends on the topology of the network.

$a = -$ Zero binder instances.

$a = \triangleright$ Zero binder instances.

$a = \triangleleft$ Exactly one instance of the binder.

The general idea is that in an execution, all the goals must be satisified by binders. Also, the binders with actions $+$ or $\star$ require goals to be bound to them, while binders with the action $\triangleleft$ do not.

For any goal and binder pair, the binder may bind the goal when the events are equal, meaning that the actions are identical and the sets of variables are identical.

When a binder of node t_1 binds a goal of node t_2, it can be written as $t_1 \rightarrow t_2$. In the special case where $t_1 = \triangleleft\{p\}$ and $t_2 = \triangleright\{p\}$ for some symbol p, this may also be written as $t_1 \prec t_2$. Since each strand may have at most one occurrence of an event with action $\triangleright$ and one occurrence of an event with action $\triangleleft$, there is no confusion in also writing $\theta_1 \prec \theta_2$ if for some $t_1 \in \theta_1$, $t_2 \in \theta_2$ $t_1 \prec t_2$.

Define the binary relation $\rightarrow$ on nodes to hold for pairs of nodes where $t_1 \rightarrow t_2$.

4.7 Causal relation

The causal relation C is a transitive binary relation on nodes. It is possible to interpret this relation as $\leq$ where the node symbols are interpreted as timestamps. C arises from the strands and goal binding as follows.

For a strand $\theta = (R, k, I)$, for each $i \in \{1, \ldots, k-1\}$ and for each $j = i + 1$ the nodes $t_i = (\theta, i)$ and $t_j = (\theta, j)$ satisfy

$$Ct_it_j \wedge \neg Ct_jt_i.$$

Each node of a strand strictly precedes the higher numbered nodes in the same strand.

Given nodes t_1 and t_2 where $t_1 \rightarrow t_2$,

$$Ct_1t_2 \wedge Ct_2t_1.$$

The event associated with node t_1 occurs concurrently with the event associated with node t_2.

A given set of strands and goal bindings between their nodes describes a causal relation. If this causal relation is infeasible by interpreting over $\leq$ and timestamps, then there is no way that this describes an actual execution of the protocol. For example, if the transitive closure of constraints given by the binding relation and the ordering of nodes in a strand includes both Ct_1t_2 and $\neg Ct_1t_2$ then no ordering of t_1 and t_2 can satisfy both constraints.

4.8 Constraint Program

Given a set of strands $\Theta = \{\theta_1, \ldots, \theta_n\}$ and a goal binding relation $\rightarrow$ on nodes occurring in Θ define an equivalence relation E on the instance symbols of each strand in Θ.

For each strand $\theta_i \in \Theta$, let

$$(R_i, k_i, I_i) = \theta_i,$$

$$R_i = (X_i^i, X_i^f, M_i, \Phi_i(L_i)),$$

g_i be the labeling function defined in R_i and h_i be the instantiation function defined in θ_i. By definition for each $i \neq j$ where $1 \leq i \leq n$ and $1 \leq j \leq n$ $I_i \cap I_j = \phi$. Let $U = \bigcup_{i=1}^{n} I_i$. E is an equivalence relation on U satisfying the following.

- For $t_i \rightarrow t_j$ the instance symbols associated with the messages of the two nodes are equivalent under E. Let $(\theta_i, i_i) = t_i$ and $(\theta_j, i_j) = t_j$, where $\theta_i, \theta_j \in \Theta$. Since $t_i \rightarrow t_j$, $R_i[i_i] = aV$ and $R_j[i_j] = bV$ for exactly the same set of symbols V, though a and b will differ. For all $v \in V$

  $$E(h_i(g_i(i_i, v)), h_j(g_j(i_j, v))).$$

- If $t_i \prec t_j$, then the above constraints still apply, along with some additional recursively defined constraints.

  Let $X_0^u = X_j^i$, $\theta_{p0} = \theta_i$. The following hold for $k = 0$ and for all $k > 0$ where $X_k^u \neq \phi$ and $\theta_{pk} \prec \theta_{p(k-1)}$ for some $\theta_{pk} \in \Theta$.

  $$E(h_{pk}(g_{pk}(x)), h_i(g_i(x))) \text{ for all } x \in X_k^u \cap X_{pk}^f$$
  $$X_{k+1}^u = X_k^u \setminus X_{pk}^f$$

  This recursion terminates for any finite set of strands and binding relation with a feasible causal relation.

The equivalence relation E defines a partition on U. For each partition, choose a distinct unused symbol and let e be a mapping from U onto this set of new symbols. $g(x) = g(y)$ iff $E(x, y)$. Consider the set of constraints $\bigcup_{i=1}^{|\Theta|} \Phi_i(I_i)$. Clearly, this is a set of constraints only on U. Apply to each symbol from U occurring in this set of constraints the mapping e. The resulting set of constraints Γ must be feasible in order for the set of strands and binding relation to be a valid execution.

4.9 Semibundle

A semibundle is a tuple $(\Theta, \rightarrow)$.

Θ set of strands.

$\rightarrow$ goal binding relation on nodes occurring in Θ.

For all semibundles, the semibundle's causal relation (Section 4.7) and constraint program (Section 4.8) must be feasible.

A semibundle is a bundle if every goal is bound to a binder and every binder requiring a goal to be bound to is bound.

4.10 Constraint program feasibility

In the case where the model for Φ, the transitions, is an integer program, it is possible to solve it as an integer program. However, integer programs are difficult to solve, and all algorithms are in their worst case exponential in difficulty with the number of variables. Fortunately, AODV and TORA do not require the full expressiveness of integer prorgrams. They can actually be described completely on the structure $\mathfrak{N}_L$.

The constraint program arising from $\mathfrak{N}_L$ is an integer program and can be solved as such, there is a polynomial time algorithm to determine whether or not a diagonal constraint program is feasible.

Consider the atomic formulas of the first order language L_1 having nonlogical symbols $\{s, 0\}$, where s is the unary successor function and 0 is a constant symbol. All atomic formulas of this language have one of the following forms

$$s^n v_1 = s^m v_2$$

$$s^n v_1 = s^m 0$$

where $n, m \geq 0$ and s^n represents a string of n s symbols. Any formula is equivalent to one of the following normal forms

$$v_1 = s^n v_2$$

$$v_1 = s^n 0$$

$$s^n v_1 = 0$$

where $n \geq 0$.

Let $\Gamma \subset Atfm_{L_1}$ be finite and let each formula in Γ be in normal form. Let ϕ be the formula formed by the conjunction over Γ. Let ψ be ϕ existentially quantified over all variables in $Fv(\phi)$. The satisfiability of ψ on the structure $\mathfrak{A} = (\mathbb{Z}, s)$ can be determined by a graphical method.

Let $G = (V, E)$ be the directed graph described as follows. $V = Fv(\phi) \cup \{0\}$. Define $\Gamma \subset Atfm_{L_1}$ where $|\Gamma| < \omega$ and $Fv(\Gamma) \subseteq \{x_1, \ldots, x_n\}$ for some fixed, finite n. Define the directed graph $G = (V, E)$ where $V = \{x_1, \ldots, x_n\}$, and the edges

$$E = \{(x, y) \in V \times V : +1 = xy \in \Gamma\}.$$

Γ is realized in $\mathfrak{A} = (\mathbb{Z}, +1 =)$ iff every simple, undirected loop in G contains an equal number of upstream and downstream edges.

Proof. The goal is to show that Γ is feasible implies that every loop contains an equal number of upstream and downstream edges. It suffices to show the contrapositive. If there are an unequal number of $+1 =$ and $= 1+$ edges in a particular loop, then that loop is infeasible, since for any node x in the loop, it says $x + k = x$ for some nonzero k, which is impossible.

The proof of the converse proceeds by constructing a solution. Assume without loss of generality that the graph is connected because disjoint partitions may be realized independently. Arbitrarily choose some $v \in V$ and some $c \in \mathbb{Z}$. Assign values to neighbors of v exactly as indicated by the edges. Continue in this manner by either breadth first or depth first search throughout the graph. The assignments will all be consistent by hypothesis. $\dashv$

This result generalizes to encompass terms $+k =$ by introducing $k - 1$ artificial variables and applying $+1 = k$ times.

Consider now the language L with nonlogical symbols $\{\leq, +1 =\}$, where both are binary relations. Define Γ as

before, but with L instead of L_1. Define the graph $G = (V, E)$ with $V = \{x_1, \ldots, x_n\}$ as before, but the edges must include labels $\{\leq, +1 =\}$ in order to distinguish between $+1 =$ edges and $\leq$.

$$E = \{(x, y, R) \in V \times V \times \{\leq, +1 =\} : Rxy \in \Gamma\}$$

Definition. A *simple loop* of a an edge labeled graph $G = (V, E)$ is a sequence of edges

$$< (x_1, x_2, \lambda_1), (x_2, x_3, \lambda_2), \ldots, (x_{n-1}, x_n, \lambda_{n-1}) >$$

where

$$\text{for all } 1 \leq i \leq n-1 \ (x_i, x_i + 1, \lambda_i) \in E,$$

$$\text{for all } 1 \leq i \leq n-1, i < j \leq n-1 \ x_i \neq x_j.$$

$$x_1 = x_n$$

Definition. Define the *direction complemented graph* G' of a graph G as $G' = (V, E')$ where

$$E' = E \cup \{(y, x, = 1+) : (x, y, +1 =) \in E\}.$$

Γ is realized in $\mathfrak{A} = (\mathbb{Z}, \leq, +1 =)$ iff every simple loop of the direction complemented graph of G satisfies the following criteria.

- If the loop excludes edges labeled $\leq$, then the number of edges labeled $+1 =$ must equal the number of edges labeled $= 1+$.
- If the loop includes edges labeled $\leq$, then the number of edges labeled $= 1+$ must be greater than or equal to the number of edges labeled $+1 =$.

Proof. It is easy to show necessity by contraposition. To show sufficiency, construct a feasible solution to the problem incrementally, showing that each increment preserves both criteria.

This problem is equivalent to the problem of assigning to each $\leq$ a value $s \geq 0$ such that the $\leq$ is interpreted as $+s =$. Let $e = (x, y, \leq) \in E'$. Formally, the graph after such a substitution has the set of edges

$$(E' \setminus \{e\}) \cup \{(x, y, +s =), (y, x, = s+)\}.$$

In this case, the problem is feasible if after assigning values to all of the $\leq$ edges, the first criterion still holds, as this is equivalent to the feasibility of the program in L_1.

It suffices to show that given that the criteria hold, it is possible to assign some s value to any $\leq$ edge in the graph such that after making the assignment, the resulting graph still satisfies the criteria. Then a solution can be constructed by recursively assigning s values to the result of the previous substitution.

Replacing a $\leq$ labeled edge e may affect the criteria as follows.

1. Loops having e as the only $\leq$ labeled edge become loops containing only $+1 =$ and $= 1+$ labeled edges. These resulting loops must satisfy the first criterion.
2. Loops containing e along with other $\leq$ labeled edges must still satisfy the second criterion after the substutition occurs.
3. It is possible that the substitution creates loops that did not previously exist because it adds the reverse edge $(y, x, = s+)$. The resulting loop must contain at least one $\leq$ labeled edge where the direction differed from the removed $\leq$ labeled edge so the second criterion applies.

These effects constrain the value of s to be substituted. If the criteria hold prior to the substitution, these constraints are feasible.

Lemma. The set of edges in any finite loop is the union of the sets of edges of some number of simple loops.

Proof. Recursively decompose the arbitrary finite loop into smaller and smaller loops until they are simple. The only difference between a simple loop and an arbitrary loop is that simple loops contain only one repeated vertex, the beginning and end. The arbitrary loop may revisit multiple vertices.

If the loop is not simple, then there is some vertex, call it v, that is not the beginning or end but is traversed twice. The path between the two occurrences of the same vertex is a loop. The path starting from the beginning and up to the first instance v skipping the v- to-v loop then continuing to the end is another loop. If either of these resulting loops is not simple, then recursively reapply the process. The recursion terminates because each stage of the recursion monotonically decreases the size of the loops and the initial loop is finite. $\dashv$

Corollary. If every simple loop of a graph satisfies the criteria, then every arbitrary finite loop satisfies the criteria.

Proof. Both criteria are closed under summation, and every simple loop satisfies the criteria so every arbitrary loop that can be decomposed into simple loops also satisfies the criteria. $\dashv$

If any loop l containing e has effect (1), then s will be equal to the number of edges labeled $= 1+$ minus the number of edges labeled $+1 =$. $s \geq 0$ holds because of the second criterion. It is easy to verify that this substitution does not invalidate any of the criteria for other loops containing e by considering the fact that for any loop containing e, the same loop not including e but traversing the remainder of l must still satisfy both criteria. It is necessary to apply the corollary here in case the loop described here is not a simple loop.

Otherwise, only effects (2) and (3) apply. Effect (2) places an upper bound on s, while (3) places a lower bound. For every loop where effect (2) applies, the value s must be less than or equal to the number of edges labeled $= 1+$ minus

the number of edges labeled $+1 =$. There is some loop for which this difference is minimal, though always greater than or equal to zero by the second criterion. This is the upper bound on s.

Let $(x, y, \leq) = e$. When effect (3) applies, there is a simple path from x to y that contains at least one $\leq$ labeled edge. s plus the number of $= 1+$ labeled edges minus the number of $+1 =$ labeled edges along this simple path must be greater than or equal to zero. In other words s must be greater than or equal to the number of $+1 =$ labeled edges minus the number of $= 1+$ labeled edges.

The simple path of effect (3) runs parallel to and in the same orientation as e. Then there is a loop consisting of the simple path and the loops having effect (2). These loops must satisfy the criteria by the corollary, which guarantees the lower bound is less than or equal to the upper bound. $\dashv$

This result can be used to reason about $<$ and $=$ constraints. For $<$ constraints such as $x < y$, introduce an artificial variable a_0 and represent the constraint as the pair of constraints $a_0 + 1 = y$ and $x \leq a_0$. For $=$ constraints, such as $x = y$, remove the constraint and substitute throughout the rest of the constraints x every time y appears.

4.11 Algorithm for checking feasibility

An adaptation of the Floyd-Warshall all pairs shortest paths algorithm [1] determines the feasibility of the constraint program of Section 4.10 in polynomial time.

5. SEARCH PROCEDURE

The search procedure follows directly from the search procedure in Athena. For example, a disruption attack can be modeled by instantiating a pair of participants with a path between them. Then instantiate a node with the goal term where one of the participants no longer has the other one in its routing table. Additionally, instantiate a set of nodes corresponding to the initial conditions of the system. Continuously bind unbound goals in all possible ways by instantiating roles and binding to existing nodes in the system until all goals are bound. If a bundle is eventually discovered, a corresponding execution exists and the property has been disproved. On the other hand, verifying the property requires that all branches of the backwards reachability search converge. In Athena, this was claimed to be undecidable in general because each instantiation of a role potentially creates new unbound goals (though in Athena the possible instantiations are in fact bounded making the procedure decidable still, which is not the case here). It is possible to force convergence by assuming a bound on the length of executions.

5.1 Protocol specification language

The messages are specified in a message file with a very simple syntax, the message name, followed by the names of the message fields where each message is separated by a new line.

While roles are described as sequences of events and a set of atomic formulas taken in conjunction to constraining the values of state variables and event message fields, the actual specification language allows for the description of roles in terms of atomic formulas joined by the logical connectives and some programmatic connectives. The logical connectives supported are the usual Boolean operations $\wedge, \vee, \neg$ denoting and, or and not respectively. For convenience, the operator $\Rightarrow$ represents the programmatic construct if-then-else.

The interpretation of this programmatic connective $\Rightarrow$ differs fundamentally from logical implication. It denotes that if the first argument is true, then the latter argument must also be true. However, when the first argument is false, the second argument is not executed. In the case where $\Rightarrow$ has three arguments, the third argument represents the else clause and is invoked when the first condition evaluates to false. In practice, the $\Rightarrow$ separates into two separate roles, one in the case where the condition evaluates true, and the other when the condition evaluates false. This can lead to a multiplication of roles because a formula may contain multiple instances of $\Rightarrow$ terms where each possible combination forms a distinct role.

After the $\Rightarrow$ construct is removed from the formulas by separating it into its components, the remaining formula consists of atomic formulas joined by $\wedge, \vee, \neg$. Converting this to disjunctive normal form (DNF), then taking each individual conjunct yields the roles. Negations appearing on the atomic formulas can be incorporated into the formulas themselves by reversing the inequality or equalities. If the reversal is of an equality, the role splits into two, one where the equality is substituted with $>$ and one with $<$.

6. IMPLEMENTATION

The implementation is mixed Ruby (an object-oriented scripting language) and C. Ruby has many desirable features in a language: iterators, blocks and closures, built-in regular expressions and garbage collection. It also has a clean interface to native C. The core of the search engine is in native C for performance and the rest of the implementation, of which a large portion is a parser for the language, is in Ruby.

7. RESULTS

The test scenario comprises four participants, one of which is an intruder. The scenario intializes the network to a state where the hop counts to the destination are the actual distances and the route is valid. This state represents R in the formalization given previously. A simple way to invalidate this condition is by specifying that one of the hop counts is less than its actual distance to the destination. This goal state is an element of $\neg R$.

The goal binding search procedure examines possible executions that can lead to the goal state being reached and discovers that it is possible to reach such a state by a forged RREP emitted by the intruder. While this might be obvious to an analyst examining the protocol, this example demonstrates that it is possible to automate such reasoning.

8. FUTURE WORK

There are two significant problems that we intend to address in the immediate future. The first is that while topology changes in ad hoc networks may impact on security, the

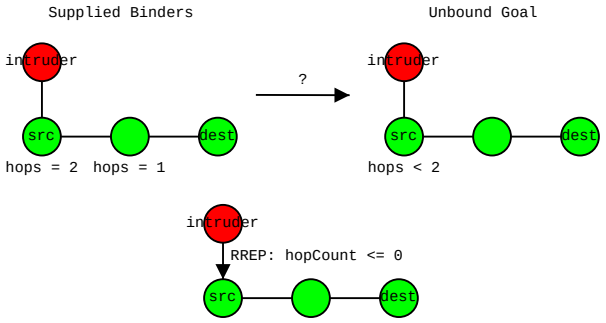

Figure 2: Example scenario.

analysis currently applies to only fixed networks. This might not be a severe problem since most ad hoc routing protocols assume stable topologies and cannot guarantee service in changing topology. Therefore, if an ad hoc network does not achieve routing under changing topologies in the presence of intrusions, it is not possible to conclude that it is a security violation of the behavior. This leads immediately to the other significant problem, which is to develop a specification of what security means in routing protocols. Clearly, the sufficient conditions analyzed in this work are not necessary, and in fact too strong for realistic routing protocols.

In addition to the specific questions above pertaining to ad hoc networks, there are also limitations imposed by state space explosion on the number of nodes that can be analyzed tractably. Model checking, even using state space reduction techniques such as symbolic representation and partial order reduction seems better suited for analyzing local properties of protocols. The success of previous routing protocol verification studies depends on theorem proving. It might be necessary to resort to some theorem proving to analyze the security of routing protocols, but the disadvantage would be that it would be impossible to automate. Further model checking techniques such as state abstraction might also help to mitigate these problems.

These results so far only demonstrate a violation of the safety property of SAODV in the presence of malicious insiders. The more difficult question of liveness has not been explored and neither has the question of how SAODV performs in the absence of malicious insiders. The same questions can be asked of the other secure routing protocols.

9. ACKNOWLEDGEMENTS

Research partially supported by the U.S. Army Research Office under Award No. DAAD19-01-1-0494.

10. REFERENCES

[1] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice-Hall, Inc., 1992.

[2] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM (JACM)*, 49(4):538–576, 2002.

[3] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2,3):191–230, 1999.

[4] Ahmed Helmy, Deborah Estrin, and Sandeep K. S. Gupta. Fault-oriented test generation for multicast routing protocol design. In *FORTE*, pages 93–109, 1998.

[5] Ahmed Helmy, Deborah Estrin, and Sandeep K. S. Gupta. Systematic testing of multicast routing protocols: Analysis of forward and backward search techniques. In *Proceedings of IEEE ICCCN*, October 2000.

[6] Gerard J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[7] Zohar Manna and Amir Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming*, 4(3):257–290, December 1984.

[8] D. Obradovic. *Formal Analysis of Routing Protocols*. PhD thesis, University of Pennsylvania, 2001.

[9] Adrian Perrig and Dawn Song. A first step towards the automatic generation of security protocols. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '00)*, pages 73–83, San Diego, CA, February 2000. Internet Society.

[10] Dawn Xiaodong Song, Sergey Berezin, and Adrian Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[11] Manel Guerrero Zapata and N. Asokan. Securing ad hoc routing protocols. In *Proceedings of the ACM workshop on Wireless security*, pages 1–10. ACM Press, 2002.