

Specification Synthesis for Monitoring and Analysis of MANET Protocols

Natalia Stakhanova Samik Basu Wensheng Zhang Xia Wang Johnny Wong
Department of Computer Science, Iowa State University, Ames, IA 50011 USA
{ndubrov, sbasu, wzhang, jxiawang, wong}@iastate.edu

Abstract

This paper introduces an approach to automatic synthesis of the specification models of routing protocol behavior from the observed flow of the network traffic. In particular, our technique generalizes the monitored sequences of routing messages constructing a high-level abstract view of the protocol. The basis of our method is similar to Inductive Logic Programming technique that derives a sound hypothesis from the individual examples. The preliminary experiments are presented on the example of AODV ad-hoc routing protocol.

1 Introduction

Mobile Ad-hoc Network (MANET) is a network of loosely connected (without any central infrastructure) aggregation of mobile computing nodes where messages are transferred from one part of the network to another via multi-hop wireless links. Each mobile node acts as a router, cooperatively constructing routes and forwarding messages for other nodes.

Being inherently decentralized and dynamic, ensuring and enforcing security and normalcy in MANET via run-time monitoring is a challenging task and is the focus of intense research in recent years. Run-time monitoring aims at discovering the abnormalities by comparing the execution against some specific set of normal and/or abnormal behavior. It can be broadly classified into three categories (a) misuse-based (b) anomaly-based and (c) specification-based [13]. The misuse-based technique relies on pre-specified attack signatures, and any execution sequence matching with a signature is flagged as abnormal. An anomaly-based approach, on the other hand, typically depends on normal patterns, and any deviation from normal is classified as malicious or faulty. A specification-based technique operates in a similar fashion to an anomaly-based method detecting deviations from the specified legitimate system behavior. However, as opposed to anomaly detection, a specification-based approach requires user guid-

ance in developing a model of valid program behavior in a form of specifications. This process, though tedious and reliant on user-expertise, is more accurate than an anomaly-based technique that suffers from a high rate of false positives [13].

In this context, we propose to develop *models* of MANET protocols via run-time monitoring. On one hand, the technique is close to anomaly detection which relies on automated learning of run-time patterns. On the other hand, our technique aims to generalize the learnt patterns in a way that incorporates in the models certain degree of abstraction. In this sense, it is close to *specification models* which are manually synthesized and represent high-level abstract view of the protocol.

Our approach is based on Inductive Logic Programming (ILP) method that induces a hypothesis from individual observations and background knowledge. In our setting we derive an abstract model of protocol behavior from the examples of its executions, where each example represents a sequence of routing messages initiated during a single route discovery. Taking advantage of the commonality of routing traffic, we develop a generalization algorithm for constructing *models* of routing protocols in automated fashion and present a generated specifications for AODV protocol.

Our model can be effectively employed for:

1. detection of anomalous run-time behavior of the protocol. The execution of the protocol can be monitored against the generated model of valid behavior and any deviation will be flagged as an anomaly.
2. detection of intrusive behavior based on the developed specification model of invalid protocol behavior.
3. validation of the completeness/correctness of existing manually developed specifications. Such specifications may be incomplete due to un-intentional oversight of the specifier and complexity of the protocol. Our generated model can potentially identify these holes in the specifications.
4. analysis of protocol statically via model checking. Often certain minor oversight in the implementation of the protocol may result in incorrect behavior.

Model checking techniques have the potential to identify such problems if a manageable specification of the implementation is provided. In fact, a typical automated verifier (model checkers, e.g. [4]) can take as input protocol specifications in the form of a model as synthesized by our technique.

2 Related Work

As specification-based approach showed to be beneficial in many areas including software testing [2, 3] and intrusion detection [7, 18, 12, 13, 5], a significant amount of research has been focused in this direction. One of the major downsides of the specification-based approach is the necessity to develop the system specifications manually. As this process is time-consuming and error-prone, automatic generation of specifications is highly beneficial.

As such, Wagner and Dean [18] employed a static analysis to automatically derive program specifications based on control-flow analysis of code, pushdown automaton and 2-gram sequences of system calls derived from control-flow graph. Ko [7] presented a machine-learning approach for developing security specifications automatically based on *Inductive Logic Programming* method. His work was focused on generating program behavioral specifications at the system call level using an ILP tool, *Progol* and aimed to reflect security properties of the programs. Although his approach has showed the advantages of automatic development of specifications, it lacked the ability to represent ordering of program operations.

Our work was inspired by this technique. In addition to modeling the ordering of events in program executions, we extend Ko's approach by considering a network setting, specifically, network routing protocols.

While ensuring completeness of the developed specifications is a common difficulty of the specification-based models, only a few approaches have attempted to address this problem [14]. More recently, several techniques applied specification-based approach to detect attacks against routing protocols, specifically, AODV [16] and OLSR [17, 10] based on the manually developed specifications.

3 Synthesis of specification models

The central tenet of our technique is that specifications of (MANET) protocols can be synthesized from the flow of the network traffic. A specification, in this context, is a form of a graph where the nodes represent the configuration of the protocol and directed-edges between nodes define how the protocol evolves from one configuration to another. Such a specification model will explicitly include all possible *monitored* behavior of the

protocol that is safe. Furthermore, we generalize the specification with an attempt to include additional behavior of the protocol that is *not known* to be anomalous. The basis of our specification synthesis (and generalization) is, in principle, similar to *inductive logic programming* approach presented in the next section.

3.1 Inductive Logic Programming

Given a set of examples and background knowledge or facts in the domain of the examples, Inductive Logic Programming (ILP) aims at *inducing* a hypothesis which when interpreted in the context of background knowledge *deduces* the examples [8].

Formally, if B denotes the background knowledge, E denotes the set of examples and H is the generated hypothesis, then

$$\forall e \in E : B \wedge H \models e \text{ Completeness property} \quad (1)$$

The above states that all examples can be deduced (modeled by, \models) from the hypothesis *and* the background knowledge.

In addition to examples, a set of negative examples may also be provided and the requirement imposed on the hypothesis is that it must not lead to given negative examples. If \bar{E} be the set of negative examples:

$$\forall \bar{e} \in \bar{E} : B \wedge H \not\models \bar{e} \text{ Consistency property} \quad (2)$$

As with completeness, consistency is also defined only with respect to the negative examples in \bar{E} . In the current context, the synthesized specification can be viewed as an hypothesis which will be shown to be both complete and consistent with respect to monitored correct behavior (examples) and known anomalous behavior (negative examples) respectively.

3.2 Routing Flows to Specifications

We model the specification of the protocol, under consideration, using the set of *request-reply* routing flows. A flow is represented by a sequence of routing messages initiated by *route request*. Each message in a sequence represents a sequence state and includes the information corresponding to the routing message (depending on the routing protocol this information can include source and destination IP addresses, hop count, sequence numbers, etc.). An example of such flow in AODV protocol can be $\langle \text{RREQsent} \rightarrow \text{RREPsent} \rightarrow \text{RREPreceived} \rangle$ denoting sending of a request, followed by sending of the corresponding reply, followed by receipt of the reply. Formally, a specification model is defined as follows:

Definition 1 (Specification Model) A *specification model* $M = (S, s^0, T, A)$ where S is a set of states,

$s^0 \in S$ is the start state, T is a set of directed edges ($S \times A \times S$) and A is the set of edge labels.

In the above, S are states of the form RREQsent and T contains transitions of the form $\text{RREQsent} \xrightarrow{a} \text{RREPsent}$ where $a \in A$ is information corresponding to RREQsent . The specification model can be viewed as a *hypothesis* that is generated from a set of example sequences E taking into consideration a set of negative example sequences \bar{E} . The construction procedure ensures the following model property.

$$\forall s_1^e \xrightarrow{a_1} s_2^e \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} s_n^e \in E : \exists s_1 \xrightarrow{b_1} s_2 \xrightarrow{b_2} \dots \xrightarrow{b_{n-1}} s_n \in M : s_1 = s^0 \wedge \forall i : s_i^e = s_i \wedge b_i = a_i$$

The above ensures that all sequences in the example set are also present in the synthesized model (completeness—see Equation 1). Furthermore,

$$\forall s_1^{\bar{e}} \xrightarrow{a_1} s_2^{\bar{e}} \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} s_n^{\bar{e}} \in \bar{E} : \forall s_1 \xrightarrow{b_1} s_2 \xrightarrow{b_2} \dots \xrightarrow{b_{m-1}} s_m \in M : s_1 = s^0 \wedge \exists i : s_i^{\bar{e}} \neq s_i \vee b_i \neq a_i$$

The above ensures that the negative examples are not included as any sequence in the synthesized specification model; either the states are not identical or the edge-labels are inconsistent. This follows from the required consistency property of the model (Equation 2).

3.2.1 Algorithm for Specification Synthesis

Figure 1 presents an algorithm for developing A specification model (Definition 1) from sequences of examples. The algorithm consists of two major procedures. `verticalMerge` merges states in *one* sequence leading to generation of loops in the result. The resultant sequence is a generalized version of the original sequence—the former is a *substring* of the latter. The procedure `horizontalMerge` on the other hand merges sequences resulting from the vertical merging procedure. This leads to (partial) sharing of sequence-states/transitions between multiple sequences.

Procedure `verticalMerge` (Figure 1) takes as input the given network flow sequence Seq_k and a specification model M and returns a new sequence Seq_m where repetitions in Seq_k are generalized as loops (*unbounded repetitions*).

For all states in the sequence Seq_k , `verticalMerge` iteratively checks whether two consecutive states should be merged (Lines 14-15). Once the states to be merged are found a new transition that represents a self-loop is created and relationships between corresponding parameters in this transition are set (Lines 16-18). The subprocedure `setParameterRelationships()` identifies the relationships ($=$, $<$, $>$, \neq) between all parameters associated with two given states. One of the merged states is removed and

all its out-going transitions are created as out-going transitions of the other. (Lines 18-22).

As an example of procedure `verticalMerge`¹ consider a sequence

$$\left\langle \begin{array}{l} \text{RREQ}(\text{sent}_5) \rightarrow \text{RREQ}(\text{fwd}_6) \rightarrow \text{RREQ}(\text{fwd}_{19}) \rightarrow \text{RREP}(\text{sent}_3) \\ \rightarrow \text{RREP}(\text{fwd}_{19}) \rightarrow \text{RREP}(\text{fwd}_6) \rightarrow \text{RREP}(\text{rec}_5) \end{array} \right\rangle$$

States representing forwarding state (RREQfwd and RREPfwd nodes) can be merged together into (generalized) states RREQfwd and RREPfwd with self-loops. The resulting generalized sequence is

$$\left\langle \begin{array}{l} \text{RREQ}(\text{sent}_5) \rightarrow \text{RREQfwd}(\text{self-loop}) \rightarrow \text{RREP}(\text{sent}_3) \\ \rightarrow \text{RREPfwd}(\text{self-loop}) \rightarrow \text{RREP}(\text{rec}_5) \end{array} \right\rangle$$

To ensure the *Consistency property* defined in Equation 2 we check whether the resultant generalized sequence from `verticalMerge` becomes a superstring of any of the negative examples $\in \bar{E}$ (Line 4). If this is the case, then the generalization is not performed and the original input sequence Seq_k is inserted into a specification model M through `horizontalMerge` procedure which takes as input a sequence Seq_k (either in generalized or in original form) and a specification model M .

For each transition in sequence Seq_k , `horizontalMerge` decides whether a new insertion of this transition and corresponding states in M are necessary (Lines 31-33). Each new insertion of the transition is followed by the adjustment of the parameters associated with this transition (Lines 34-35, 43-45). Lines 36-38, 46-50 and 56-58 ensure that none of the negative examples is included into a specification model M . For an example of `horizontalMerge`, consider two generalized sequences

$$\left\langle \begin{array}{l} \text{RREQ}(\text{sent}_5) \rightarrow \text{RREQfwd}(\text{self-loop}) \rightarrow \text{RREP}(\text{sent}_3) \\ \rightarrow \text{RREPfwd}(\text{self-loop}) \rightarrow \text{RREP}(\text{rec}_5) \text{ and} \\ \text{RREQ}(\text{sent}_{21}) \rightarrow \text{RREP}(\text{sent}_{17}) \rightarrow \text{RREP}(\text{rec}_{21}) \end{array} \right\rangle$$

After performing *vertical merge* the resulting specification model is

$$\left\langle \begin{array}{l} \text{RREQsent} \rightarrow \text{RREQfwd}(\text{self-loop}) \rightarrow \text{RREPsent} \rightarrow \\ \text{RREPfwd}(\text{self-loop}) \rightarrow \text{RREPrec} \end{array} \right\rangle$$

4 Case Study

We demonstrate the validity of our approach on the example of The Ad hoc On-demand Distance Vector (AODV) and the Dynamic Source Routing (DSR) routing protocols. The details on these protocols can be found in [11, 6].

To construct specifications for these protocols we used traces of valid protocol behavior obtained through

¹Parameters associated with transitions are not shown for brevity. Each state is annotated with action names and the network-node that originated that action; e.g. sent_5 is sent action performed by node 5

<pre> 1: void main(){ 2: for all flow sequences $\in E$ { 3: Seq_m = verticalMerge(Seq_k, M); 4: if ($\exists e \in \bar{E} \mid \text{Seq}_m \models e$) 5: horizontalMerge(Seq_k, M); 6: else horizontalMerge(Seq_m, M); 7: } 8: return; 9: } 10: 11: 12: Seq_m verticalMerge(Seq_k, M) { 13: int i=1; 14: while (i≠k) { 15: if(s_i=s_{i+1}) { 16: if (s_i → s_i ∉ M) { 17: createTransition(s_i, s_i); 18: } // end of if-then 19: setParameters(s_i, s_{i+1}, M); 20: if (i+1 ≠ k) { 21: createTransition(s_i, s_{i+2}); 22: setParameters(s_i, s_{i+2}, M); 23: } // end of if-then 24: remove(s_{i+1}); 25: } // end of if-then 26: else i=i+1; 27: } // end of while 28: return Seq_k; 29: } </pre>	<pre> 28: void horizontalMerge(Seq_k, M){ 29: int i=1; 30: while (i≠k) { 31: if(s_i ∈ M){ 32: if(s_{i+1} ∈ M){ 33: if (s_i → s_{i+1} ∉ M) { 34: createTransition(s_i ∈ M, s_{i+1} ∈ M); 35: setParameters(s_i, s_{i+1}, M); 36: if ($\exists e \in \bar{E} \mid M \models e$) { 37: removeTransition(s_i → s_{i+1} ∈ M); 38: } 39: } 40: } // end of if-then 41: } // end of if-then 42: else{ 43: createState(s_{i+1} ∈ M); 44: createTransition(s_i ∈ M, s_{i+1} ∈ M); 45: setParameters(s_i, s_{i+1}, M); 46: if ($\exists e \in \bar{E} \mid M \models e$) { 47: removeTransition(s_i → s_{i+1} ∈ M); 48: removeState(s_{i+1} ∈ M); 49: } 50: return; 51: } 52: } // end of if-else 53: } // end of if-then 54: else { 55: createState(s_i ∈ M); 56: if ($\exists e \in \bar{E} \mid M \models e$) { 57: removeState(s_i ∈ M); 58: } 59: return; 60: } // end of if-else 61: i=i+1; 62: } // end of while 63: return; 64: } </pre>
--	---

Figure 1. Pseudo-code for *vertical merge* and *horizontal merge*.

ns2 tool [1]. The validity of specifications of routing protocols also depends on the attributes associated with each event in the route request-reply flow (each state in a sequence). To identify the parameters needed to distinguish valid behavior, different feature selection techniques can be applied. In our experiments we used all parameters provided by *ns2* simulator. However, final specification model contains only a subset of those features, the rest was removed due to the irrelevancy and space constraints. Figure 2 presents the generated specifications of valid behavior for AODV protocol.

Generated AODV specification model contains seven states: sending RREQ(RREQs), forwarding RREQ(RREQfwd), sending RREP(RREPs), forwarding RREP(RREPfwd), received RREP(RREPrec), sending ERROR(RERRs), Drop. The model starts with a state RREQs indicating the source node sending out a RREQ message. The transition from one state to another occurs only when the corresponding conditions specified in the text box are satisfied. As such when a RREQ reaches an intermediate node and the conditions specified in the text box titled RREQs → RREQfwd are satisfied, the model enters state RREQfwd; on the other hand, if the conditions

specified in the text box titled RREQs → RREPs are satisfied, it enters a state RREPs. From the state RREPs the model has four valid transitions to the states: RREQfwd, RREPrec, RERRs and Drop.

In addition to valid behavior, we have also considered possible attacks against AODV protocol based on the analysis by [9]. The details of the experiments and the misuse specification models for AODV protocol and the results for DSR protocols can be found in [15].

5 Conclusion

The constructed models of protocols' behavior describe the protocols' message exchange and the relationships among protocol-relevant attributes as reflected in the network traffic. While properties hidden from the network layer are not represented in these models, they can be incorporated into specifications using experts knowledge on system environment. We feel that approach brings significant benefits to intrusion detection field. Constructed specifications can be used in detecting protocol misuses. However, the approach is not designed as a stand-alone intrusion detection tool, although can be used to complement the existing intrusion detection systems.

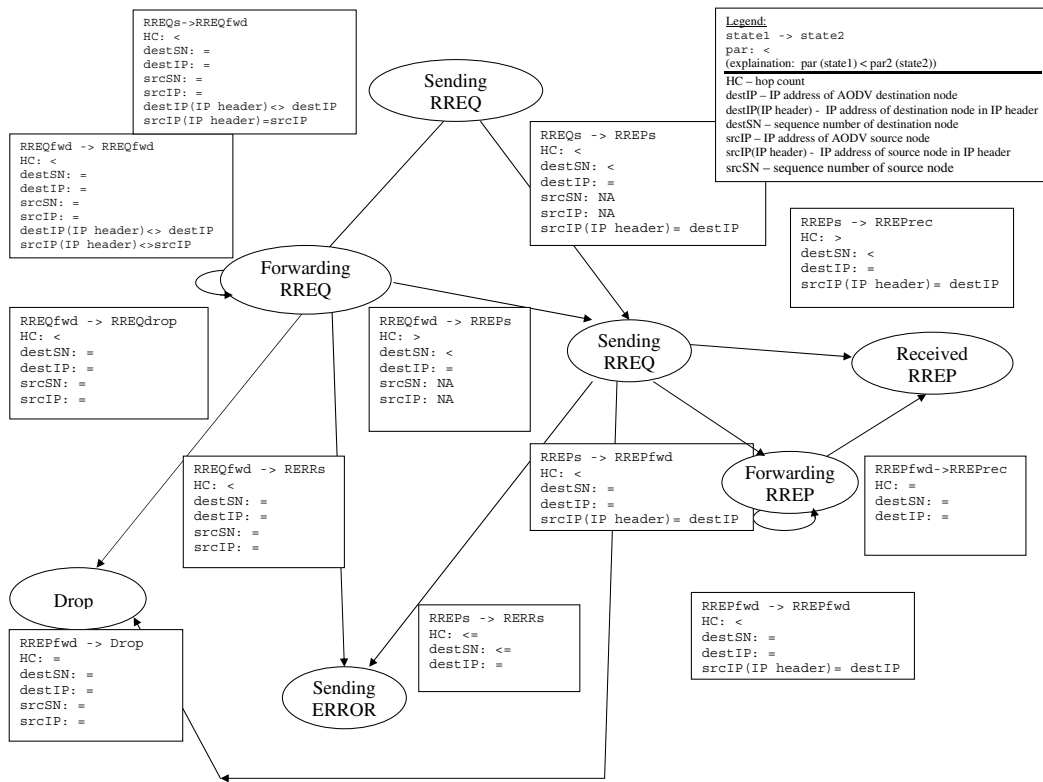


Figure 2. Valid behavior of AODV protocol

As a future avenue of our research work we envision a formal evaluation of generated specifications with regards to their completeness and consistency.

References

- [1] The network simulator ns2. Online. <http://www.isi.edu/nsnam/ns>.
- [2] J. Chang and D. J. Richardson. Structural specification-based testing: automated support and experimental evaluation. In *ESEC/FSE*, 1999.
- [3] Y. Chen, R. L. Probert, and D. P. Sims. Specification-based regression test selection with risk analysis. In *CASCON*, 2002.
- [4] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 1997.
- [5] S. P. Joglekar and S. R. Tate. Protomon: Embedded monitors for cryptographic protocol intrusion detection and prevention. *Journal of Universal Computer Science*, 2005.
- [6] D. B. Johnson, D. A. Maltz, and Y.-C. Hu. The dynamic source routing protocol for mobile ad hoc networks (dsr), 2004.
- [7] C. Ko. Logic induction of valid behavior specifications for intrusion detection. In *SP*, 2000.
- [8] S. H. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 1994.
- [9] P. Ning and K. Sun. How to misuse AODV: A case study of insider attacks against mobile ad-hoc routing protocols. In *Ad Hoc Networks*, volume 3, pages 795–819, 2005.
- [10] J.-M. Orset, B. Alcalde, and A. R. Cavalli. An EFSM-based intrusion detection system for ad hoc networks. In *ATVA*, 2005.
- [11] C. E. Perkins, E. M. Royer, and S. R. Das. Ad hoc on demand distance vector (AODV) routing. Routing IETF, Internet Draft, 2003.
- [12] J.-P. Pouzol and M. Ducasse. Formal specification of intrusion signatures and detection rules. In *CSFW*, 2002.
- [13] R. Sekar et al. Specification-based anomaly detection: a new approach for detecting network intrusions. In *CCS*, 2002.
- [14] T. Song et al. Using ACL2 to Verify Security Properties of Specification-based Intrusion Detection Systems. In *ACL2 workshop*, 2003.
- [15] N. Stakhanova et al. Specification synthesis for monitoring and analysis of MANET protocols. Technical Report 07-02, Iowa State University, 2007.
- [16] C.-Y. Tseng et al. A specification-based intrusion detection system for AODV. In *SASN*, 2003.
- [17] C.-Y. Tseng et al. A specification-based intrusion detection model for OLSR. In *RAID*, 2005.
- [18] D. Wagner and D. Dean. Intrusion detection via static analysis. In *SP*, 2001.