A Correctness Proof of the SRP Protocol *

Huabing Yang, Xingyuan Zhang, and Yuanyuan Wang
PLA University of Science and Technology, P.R. China
yanghuabing@gmail.com, xyzhang@public1.ptt.js.cn, wangyy2005@gmail.com

Abstract

The correctness of a routing protocol can be divided into two parts, a liveness property proof and a safety property proof. The former requires that route(s) should be discovered and data be transmitted successfully, while the latter requires that the discovered routes have some desired characters such as containing only benign nodes. While safety properties are relatively easier to prove, the proof of liveness properties is usually harder. This paper presented a liveness proof of a secure routing protocol, SRP [11] in Isabelle/HOL [10]. The liveness property proved says that if a data package needs to be sent, then it will be sent and then received, and finally, the sender will receive an acknowledgement sent back by the receiver. There are three main contributions in this paper. Firstly, a liveness property is proved for a secure routing protocol, and this has never been done before. Secondly, our validation model can deal with arbitrarily many nodes including malicious ones, and nodes are allowed to move randomly. Thirdly, a fail set is defined to restrict the attackers' actions, so that the safety properties used to prove the liveness property can be established. The paper explains why it is reasonable to prevent malicious nodes from performing the events in fail set.

Index Terms: Correctness, Liveness property, Response property, Secure routing protocol, SRP protocol, Isabelle.

1 Introduction

An *ad hoc network* is a group of wireless mobile computers, in which individual nodes cooperate by forwarding packets for each other to allow them to communicate beyond direct wireless transmission range. Several secure routing protocols [4–6, 11] for ad hoc network have been proposed in recent years. However, each of these protocols treats only a subset of security threats, and most of them have not been formally verified. However, secure routing protocols require rigorous formal verification, so that people can trust and make use of them in real application.

To the best of our knowledge, there have been some formal analysis of ad hoc network routing protocols [1–3, 9, 14, 15, 17], but only paper [2] and paper [17] dealt with *secure* routing protocols. Additionally, all of these verifications only treat safety properties. It seems that we are the first to deal with liveness properties. Furthermore, many of these cited works, such as [14, 15], are done with model checking. Since model checking technology does not scale well, these works can only deal with models with very small number of mobile nodes. For instance, paper [14] only discusses a 5-node model.

Liveness property is very important to a routing protocol, especially to a *secure* routing protocol. In this paper, we focused on the liveness property of a secure routing protocol, SRP [11]. Since SRP is only a route discovering protocol which does not consider data transmission, we combine SRP with a secure message transmission protocol, SSP [12]. SSP is a restricted version of SMT [12]. In this paper, we use SRP to name the combination of SRP and SSP. The meaning of the liveness propety proved in this paper is that if a data package needs to be sent, then the data will be sent, and the receiver will receive this data and send back an acknowledgement, which will eventually be received by the sender. This property can be formulated with a LTL (Linear Temporal Logic) formula of the form $?\sigma \models \Box(?P \longrightarrow \Diamond ?Q)$. According to Manna and Puneli [8], formalae of this form represent response properties.

The liveness proof of a secure routing protocol is harder than the one for non-secure routing protocol, because the behavior of malicious nodes must be taken into account. If there exist some malicious nodes in the system and if they can do everything, even the securest routing protocol can not work normally. So we add a condition that the malicious node can not execute any event in a *fail* set. This paper explains why it is reasonable to prohibit

*This research was funded by National Natural Science Foundation of China, under grant 60373068 'Machine-assisted correctness proof of complex programs'

1-4244-0054-6/06/$20.00

the events in the *fail* set from happening in section 4.

In paper [18], a liveness proof method is proposed, to deal with general liveness properties. Our method extends Paulson's inductive protocol verification approach [13][1] The feasibility of our method has been shown by paper [16], which proved the liveness property of an elevator control system. In this paper, we use the method to deal with a more realistic example, that is, the liveness of SRP. The model proposed in this paper can deal with networks with arbitrarily many mobile nodes, which is the virtue of the theorem proving approach.

The paper is organized as follows. Section 2 presents the system model, which consists of two concurrent subsystems: *srp* and *attacker fail*. The sub-system *srp* describes the activities of benign nodes, and the sub-system *attacker fail* describes the activities of malicious nodes. Section 3 introduces SRP briefly, and gives the formal description of the sub-system *srp*. Section 4 describes the *fail* set and the sub-system *attacker fail*. Section 5 formulates the liveness part of the correctness of SRP and describes the liveness proof. Section 6 concludes.

2 Concurrent systems

According to the definition of a concurrent system in [18], the type of concurrent systems is $('a\ list \times 'a)\ set$ and a concurrent system is written as *cs*. The expression $(\tau, e) \in cs$ means that the event e is legitimate to happen under system state (event list) τ, according to *cs*. Under such a definition of the concurrent system, the composition operator '||' can be defined naturally as:

$$cs_1 \parallel cs_2 \equiv cs_1 \cup cs_2$$

The intuition behind this definition is that, in a concurrent system consisting of sub-systems cs_1 and cs_2, an event e is legitimate to happen iff it is eligible to happen either according to cs_1 or according to cs_2.

Using '||', a complex concurrent system *cs* can be decomposed into many sub-systems as:

$$cs \equiv cs_1 \parallel cs_2 \parallel cs_3 \parallel \ldots$$

In this paper, the SRP protocol is composed of two subsystems: *srp* and *attacker fail*. The sub-system *srp* describes activities of the benign nodes, and the sub-system *attacker fail* describes activities of the malicious nodes. Since there exist both benign nodes and malicious nodes in the system, the SRP protocol is modeled as:

$$srp \parallel attacker\ fail$$

[1] Paulson's approach can only be used to prove safety properties, i.e. properties about finite execution traces.

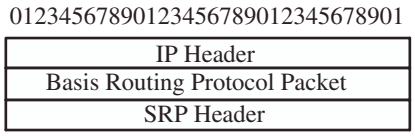

Figure 1. SRP route request packet

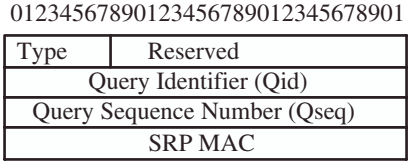

Figure 2. SRP header

3 The SRP protocol and its formalization

3.1 Overview

The SRP protocol is a secure routing protocol for mobile ad hoc networks, based on some reactive routing protocols such as the DSR protocol [7]. SRP assumes the existence of a security association (SA) between the source S and the destination D, which can be achieved through a shared key $\kappa_{S,D}$ between S and D. And the intermediate nodes do not need to validate the control message.

We assume there exist N nodes in the system. And N can be an arbitrarily large natural number. Thus our model can deal with arbitrarily many nodes.

types *Node* = *nat*
— We use natural number to represent the addresses of nodes, and *0* to indicate a broadcasting address.
consts N :: *nat* — N expresses node number.
axioms *N2*: $1 < N$ — We assume N is greater than *1*.

The source initiates a route discovery by broadcasting a route request packet as shown in Figure 1. SRP adds an additional header called SRP header to the underlying routing protocol packet. The SRP header is shown in Figure 2.

The query identifier *Qid* is a random 32 bit identifier generated by S. It is used by the intermediate nodes as a means to identify the request. Since *Qid* is an output of a secure pseudo-random number generator and is unpredictable by the attackers, it can provide protection against attackers who fabricate requests only to cause subsequent requests to be dropped.

The query sequence number *Qseq* is a 32 bit sequence number maintained by the source node (S) for each destination (D), with which it has a security association. It

increases monotonically for every route request generated by S for D, thus allowing D to detect outdated requests. *Qseq* is initialized at the establishment of the SA and is not allowed to wrap around.

The *Qid* and *Qseq* are represented as natural number:

types *Qid* = *nat*
types *Qseq* = *nat*

The SRP *MAC* is a 96 bit value calculated using the shared key and the non-mutable fields of the message. Therefore the SRP *MAC* not only validates the integrality of the message but also authenticates the origin of the packet because the attackers do not know the shared key.

datatype *NonMutableField* =
NQP Node Node Qid Qseq Node list
— The non-mutable fields in the route request message and route reply message.
| *NDA Node Node Data Node list*
— The non-mutable fields in the data message and acknowledgement message.

types *MAC* = *nat*
consts *crypt* :: *Key* ⇒ *NonMutableField* ⇒ *MAC*
— An example of the *MAC* for route request is *crypt* $\kappa_{S,D}$ (*NQP S D qid qseq* []) .

3.2 Messages and events in SRP

The message in SRP is expressed as: *MSG Source Destination Msg-option*. It is defined as:

datatype *Msg* = *MSG Node Node Msg-option*

There are four kinds of *Msg-option*, defined as:

datatype *Msg-option* =
RREQ Node Qid Qseq Node list MAC — Request
| *RREP Qid Qseq SegsLeft Node list MAC* — Reply
| *DATA Data SegsLeft Node list MAC* — Data
| *ACK Data SegsLeft Node list MAC* — Acknowledgement

We do not consider the route error messages because it does not affect the liveness proof.

The type of events that may happen in SRP is defined as:

datatype *event* =
Send Node Msg
— *Send A msg*: Node *A* sends a message *msg*.
| *Recv Node Msg*
— *Recv A msg*: Node *A* receives a message *msg*.
| *Disturb real* × *real real*
— *Disturb* (*x*, *y*) *p*: A disturbance happens at position (*x*, *y*) with the power *p*. The disturbance may result in a failure of data reception.
| *Move Node real real*
— *Move A x y*: Node *A* moves a distance of *x* in horizontal orientation, and *y* in vertical orientation.
| *DataNeedSend Node Node Data*
— *DataNeedSend S D d*: A data *d* comes to node *S*'s network layer from application layer and needs to be transmitted to node *D*.
| *DataRecvd Node Node Data Node list*
— *DataRecvd D S d p*: A data *d* is received by node *D* from node *S*, through the path *p*.
| *Tick*
— *Tick*: It is used to reckon the steps of the system time.

3.3 Describing the sub-system *srp*

The sub-system *srp* consists of fourteen rules, which are established according to SRP. For instance, we use the following rule:

(*τ*, *Move A* r_1 r_2) ∈ *srp*
to express that the nodes can move randomly in the network.

If there exists a data in the sending buffer of the source *S*, and there is no route in the route cache of *S*, then *S* broadcasts a new route request:

⟦ (*data*, *0*) *mem sendbf* (*τ*, *S*, *D*) ∨
(*data*, *2*) *mem sendbf* (*τ*, *S*, *D*) ∨
(*data*, *1*) *mem sendbf* (*τ*, *S*, *D*) ∧
(∃ *x*. *x mem curreq* (*τ*, *S*, *D*) ∧ *RetranstimerOut* ≤ *snd* (*snd x*));
cache (*τ*, *S*, *D*) = []⟧
⟹
(*τ*, *Send S* (*MSG S 0* (*RREQ D* (*ranqid* (*τ*, *S*, *D*))
(*seqmono* (*τ*, *S*, *D*)) [*S*] (*crypt* $\kappa_{S,D}$ (*NQP S D*
(*ranqid* (*τ*, *S*, *D*)) (*seqmono* (*τ*, *S*, *D*)) [])))) ∈ *srp*

When an intermediate node receives such a route request, it extracts the *Qid* value to determine if it has already relayed a packet corresponding to the same request. If not, the intermediate node extracts the *node list* from the request. If this intermediate node already exists in the *node list*, the request is discarded directly. Otherwise, the intermediate node appends its own IP address to the *node list* and rebroadcasts the request message:

⟦ *MSG S 0* (*RREQ D qid qseq ndl mac*) *mem pdREQ* (*τ*, *B*);
B ≠ *D*; (*S*, *qid*, *D*) ∉ *set* (*sentREQ* (*τ*, *B*)); ¬ *B mem ndl*⟧
⟹
(*τ*, *Send B* (*MSG S 0* (*RREQ D qid qseq* (*ndl*@[*B*]) *mac*))) ∈ *srp*

Thus IP addresses of the intermediate nodes keep on accumulating on the route request.

when the destination D receives this request packet, it verifies that the packet has originated from the node with which it has SA. And *Qseq* is compared with *MAXseq*, the maximum query sequence number received from S. If *Qseq* < *MAXseq*, the request is considered to be outdated and is discarded. Else the encrypted hash of the request field is calculated and matched against the *SRP*

MAC. The equality validates the integrality of the request as well as the authenticity of the sender. For each valid request, the destination puts the accumulated route of intermediate nodes into the route reply packet. The Qseq and Qid fields from the route request are copied into the corresponding fields of the reply packet. *MAC* is calculated to preserve the integrality of the reply packet in transit. The *Qseq* and *Qid* fields verify the freshness of the reply packet to the source. We express the above case as the rule *reply-route-request*:

$[\![\ MSG\ S\ 0\ (RREQ\ D\ qid\ qseq\ ndl\ mac)\ mem\ pdREQ\ (\tau,\ D);$
$mac = crypt\ \kappa_{D,S}\ (NQP\ S\ D\ qid\ qseq\ []);$
$maxseq\ (\tau,\ D,\ S) \leq qseq\ ]\!]$
$\Longrightarrow$
$(\tau,\ Send\ D\ (MSG\ D\ S\ (RREP\ qid\ qseq\ (|ndl\ @\ [D]| - 2)$
$(rev\ (ndl\ @\ [D]))\ (crypt\ \kappa_{D,S}\ (NQP\ D\ S\ qid\ qseq$
$(rev\ (ndl\ @\ [D])))))))\ \in\ srp$

When the source S receives the route reply packet, it checks source, destination addresses, the *Qid*, and the *Qseq*. S discards the reply if it does not correspond to the currently pending query. Otherwise, S compares the reply IP *source-route* with the reverse of the route carried in the reply package. If the two routes match, *MAC* is calculated using the non-mutable fields of SRP header and $\kappa_{S,D}$. The successful verification confirms that the request indeed reaches the intended destination and the reply has not been corrupted on the way back from D to S. Furthermore, since the reply packet has been successfully routed and received over the reverse of the route it carries, the routing information has not been compromised during the request propagation.

4 The *fail* set and the definition of *attacker fail*

4.1 The *fail* set

If there exist some malicious nodes who can do everything, such as knowing all shared keys of others, even the securest routing protocol will fail. So we define a *fail* set, and assume the events in which will not happen. Then we can prove some safety properties of SRP, so long as the events fabricated by attackers are not in the *fail* set. And we will explain the *fail* set is defined reasonably. That is to say, SRP is safe enough to withstand any reasonable attackers.

The *fail* set changes with the system state, so it is defined as a function of event list. We divide it into five parts: *fail1*, *fail2*, *fail3*, *fail4*, and *fail5*.

constdefs *fail* :: *event list* $\Rightarrow$ *event set*
$fail\ \tau \equiv (fail1\ \tau \cup fail2\ \tau \cup fail3\ \tau \cup fail4\ \tau \cup fail5\ \tau)$

In the following subsections, we will interpret the five parts of the *fail* set separately.

4.1.1 The *fail1*

We know that no node can receive any message if no node has sent any message. Namely, only if a node has sent a message, can the node's neighbors receive this message. Therefore, the event *Recv B msg* should not happen except that the event *Send A msg* has happened and that the distance between A and B is not more than the sending power of A. This property is described in *fail1*:

consts *fail1* :: *event list* $\Rightarrow$ *event set*
$fail1\ [] = \{e.\ (\exists\ B\ msg.\ e = Recv\ B\ msg)\}$
$fail1\ (Send\ A\ msg\ \#\ \tau) = fail1\ \tau -$
$\{e.\ (\exists\ B.\ dis\ \tau\ A\ B \leq powr\ A \wedge A \neq B \wedge e = Recv\ B\ msg)\}$
$fail1\ (e\ \#\ \tau) = fail1\ \tau$

4.1.2 The *fail2*

We assume that the attackers do not know the shared key between the source S and the destination D. Since the non-mutable parts in the messages are protected with the *MAC* and the *MAC* is encrypted with $\kappa_{S,D}$ or $\kappa_{D,S}$, any message fabricated by attackers will be detected by S or D, except that the attackers only change the mutable parts in the messages, and the change must be very *skillful* for the purpose of not being detected by S or D.

For route request messages, only when the malicious intermediate nodes change the *message sender* and the *accumulated node list* synchronously, can the change not be detected by the destination. It is described in *fail2*:

consts *fail2* :: *event list* $\Rightarrow$ *event set*
$fail2\ [] = \{e.\ (\exists\ A\ S\ D\ qid\ qseq\ ndl.$
$e = Send\ A\ (MSG\ S\ 0\ (RREQ\ D\ qid\ qseq\ ndl$
$(crypt\ \kappa_{S,D}\ (NQP\ S\ D\ qid\ qseq\ []))))))\}$
$fail2\ (Send\ A\ (MSG\ S\ D0\ (RREQ\ D\ qid\ qseq\ ndl\ mac))\ \#\ \tau) =$
$(if\ D0 = 0 \wedge mac = crypt\ \kappa_{S,D}\ (NQP\ S\ D\ qid\ qseq\ [])$
$then\ fail2\ \tau -$
$\{e.\ (\exists\ n < |ndl|.\ e = Send\ (ndl\ !\ n)\ (MSG\ S\ 0$
$(RREQ\ D\ qid\ qseq\ (take\ (Suc\ n)\ ndl)\ mac)))\}$
$else\ fail2\ \tau\ )$
$fail2\ (e\ \#\ \tau) = fail2\ \tau$

4.1.3 The *fail3*, *fail4*, and *fail5*

For route reply messages, data messages, and acknowledgement messages, only that the malicious intermediate nodes change the *message sender* and the *segments-left* synchronously, can the change be not found by the source. the *fail3* describes the change on reply messages:

consts *fail3* :: *event list* $\Rightarrow$ *event set*
$fail3\ [] = \{e.\ (\exists\ A\ S\ D\ qid\ qseq\ segl\ ndl.$
$e = Send\ A\ (MSG\ D\ S\ (RREP\ qid\ qseq\ segl\ ndl$
$(crypt\ \kappa_{D,S}\ (NQP\ D\ S\ qid\ qseq\ ndl)))))\}$
$fail3\ (Send\ A\ (MSG\ D\ S\ (RREP\ qid\ qseq\ segl\ ndl\ mac))\ \#\ \tau) =$
$(if\ mac = crypt\ \kappa_{D,S}\ (NQP\ D\ S\ qid\ qseq\ ndl)$
$then\ fail3\ \tau -$
$\{e.\ (\exists\ n.\ n < |ndl| \wedge 0 < n \wedge e = Send\ (ndl\ !\ n)$

$(MSG\ D\ S\ (RREP\ qid\ qseq\ (n-1)\ ndl\ mac)))\}$
else fail3 τ)
fail3 $(e\ \#\ \tau) =$ *fail3* τ

The *fail4* and the *fail5* are defined almost the same as the *fail3*, so we do not give their definitions here.

4.2 The definition of *attacker fail*

We define a concurrent system *attacker* G as follows:

consts *attacker* :: $('a\ list \Rightarrow 'a\ set) \Rightarrow ('a\ list \times 'a)\ set$
ak: $e \notin G\ \tau \Longrightarrow (\tau, e) \in attacker\ G$

In this system, any event may happen if only the event is not in the event set $G\ \tau$. We assume that the malicious nodes should not produce any event in the *fail* set. So the concurrent system *attacker fail* can exactly describe the behaviour of attackers, who can produce any event in anytime, except those in the event set *fail* τ.

5 Liveness proof of SRP

5.1 Liveness Description

The informal description of the liveness propety proved in this paper is that if a data package needs to be sent, then the data will be sent, and the receiver will receive this data and send back an acknowledgement, which will eventually be received by the sender. When a data package needs to be sent from S to D, there exist three cases.

Firstly, if the route cache of S is not empty, and the first route in the cache is good[2], the data will be transmitted successfully using this route. Then D will send out an acknowledgement when it receives this data, and S will receive this acknowledgement eventually.

Secondly, if the route cache is empty, a route discovery will be performed, and at least one path (good route) will be discovered because we assume there will exist paths between the source and the destination[3]. And then the data will be transmitted using this newly discovered route.

Thirdly, if the route cache is not empty, and the first route in the cache is broken, S will transmit the data using this broken route. The transmition will fail and S will retransmit this data after a period of fixed time. While the retransmision times is larger than a threshold value, S will delete this broken route from its cache. As the size of the cache is limited, S will delete all of the broken routes, and use a good route to send the data. If all routes in the cache are broken, S will clear its cache, and find a new path through a route discovery.

The formal expression of the liveness property is theorem *send-will-recv*:
$[\![(srp \parallel attacker\ fail) \vdash \sigma;$
$PF\ (srp \parallel attacker\ fail)\ \{\!|F\ S\ D\ data,\ E\ S\ D\ data,\ M|\!\}\ \sigma]\!]$
$\Longrightarrow$
$\sigma \models \Box(\langle(\!|DataNeedSend\ S\ D\ data|\!)\rangle \hookrightarrow$
$\Diamond\langle(\lambda\tau.\ \exists p.\ (\!|Recv\ S\ (MSG\ D\ S\ (ACK\ data\ 0\ p$
$(crypt\ \kappa_{S,D}\ (NDA\ D\ S\ data\ p)))))|\!)\ \tau)\rangle)$

The conclusion of *send-will-recv* is a response property. It says that if the event *DataNeedSend S D data* happens, then there exists a path p and the event *Recv S* (*MSG D S* (*ACK data 0 p* (*crypt* $\kappa_{S,D}$ (*NDA D S data p*)))) will eventually happen[4].

The premise of *send-will-recv* is a *Parametric Fairness (PF)* assumption. The explanation and the definition of *PF* are given in paper [18]. The *PF* assumption can ensure that the concurrent system *srp* $\parallel$ *attacker fail* runs fairly . In unfair executions, even though the event *DataRecvd D S data p* is enabled infinitely many times, if it never happens, then D will never receive any data. The fairness assumption is necessary to prevent such occasions from happening for infinitely many times.

5.2 Liveness proof

5.2.1 Overview

According to the *resp-rule* [18]:

$[\![\ RESP\ ?cs\ ?F\ ?E\ ?N\ ?P\ ?Q;$
$?cs \vdash ?\sigma;\ PF\ ?cs\ \{\!|?F,\ ?E,\ ?N|\!\}\ ?\sigma]\!]$
$\Longrightarrow ?\sigma \models \Box\langle ?P\rangle \hookrightarrow \Diamond\langle ?Q\rangle$,

if we let
$?cs = srp \parallel attacker\ fail,$
$?P = (\lambda\tau.\ (\!|DataNeedSend\ S\ D\ data|\!)\ \tau),$
$?Q = (\lambda\tau.\ \exists p.\ (\!|Recv\ S\ (MSG\ D\ S\ (ACK\ data\ 0\ p\ (crypt\ \kappa_{S,D}$
$(NDA\ D\ S\ data\ p)))))|\!)\ \tau),$
we can gain:
$[\![RESP\ (srp \parallel attacker\ fail)\ ?F\ ?E\ ?N$
$(\!|DataNeedSend\ S\ D\ data|\!)\ (\lambda\tau.\ \exists p.\ (\!|Recv\ S\ (MSG\ D\ S$
$(ACK\ data\ 0\ p\ (crypt\ \kappa_{S,D}\ (NDA\ D\ S\ data\ p)))))|\!)\ \tau);$
$srp \parallel attacker\ fail \vdash \sigma;\ PF\ (srp \parallel attacker\ fail)\ \{\!|?F,\ ?E,\ ?N|\!\}\ \sigma$
$\Longrightarrow$
$\sigma \models \Box(\langle(\!|DataNeedSend\ S\ D\ data|\!)\rangle \hookrightarrow$
$\Diamond\langle(\lambda\tau.\ \exists p.\ (\!|Recv\ S\ (MSG\ D\ S\ (ACK\ data\ 0\ p$
$(crypt\ \kappa_{S,D}\ (NDA\ D\ S\ data\ p)))))|\!)\ \tau)\rangle)$

And then, we only need to prove the premise *RESP* in

[2] We let nodes choose the first route in its cache to send data.

[3] It is possible that there is no path sometimes in the network by reason of nodes' movement, so we assume that the maximum time interval of non-existent path has an upper limit. The following axiom expresses this assumption clearly:
axioms mobility: *expath S D* $\tau \vee (\exists\tau'.\ expath\ S\ D\ \tau' \wedge time\ \tau' - time\ \tau \leq RetranstimerOut)$

[4] $(\!|e|\!)\ \tau$ means that the last event of the event list τ is e.

```
locale RESP =
  fixes cs :: ('a list × 'a) set
  and F :: 'a list ⇒ nat
  and E :: 'a list ⇒ 'a
  and N :: nat
  and P :: 'a list ⇒ bool
  and Q :: 'a list ⇒ bool
  assumes mid: [[cs ⊢ τ; ⌈P ⟼¬Q ∗⌉ τ; ¬ Q τ]]
          ⟹ 0 < F τ ∧ F τ < N
        and fd: [[cs ⊢ τ; 0 < F τ]]
          ⟹ τ [cs> E τ ∧ F (E τ # τ) < F τ
```

Figure 3. The definition of *RESP*

```
locale RESP1 =
  fixes cs :: ('a list × 'a) set
  and TR :: 'a list ⇒ 'a list
  and N :: nat
  and P :: 'a list ⇒ bool
  and Q :: 'a list ⇒ bool
  assumes path: [[cs ⊢ τ; ⌈P ⟼¬ Q ∗⌉ τ; ¬ Q τ]]
          ⟹ |TR τ| < N ∧ Q ((TR τ) @ τ) ∧
               cs ⊢ (TR τ) @ τ
```

Figure 4. The definiton of *RESP1*

order to get the theorem *send-will-recv*, which is the formal expression of the liveness property.

The definition of *RESP*, given in Figure 3, expresses some requirements on the underlying state-transition system. *RESP* requires F to be a measuring function which returns the distance from the current state to the desired Q-state. *RESP* also requires function E to be a strategy for choosing the next eligible event to happen, so that the happening of the selected E-event will decrease the F-measurement. The N is an upper bound of F. The existence of F, E and N will ensure the desired liveness property. The detail explanation of the *RESP* is in paper [18].

We only need to find two functions F, E and a natural number N to prove the *RESP* premise. We have successfully found such functions F, E in the liveness proof of an elevator control system [16]. However, it is more difficult to find such functions with regard to SRP. So we use another locale *RESP1*, whose definition is shown in Figure 4.

Comparing with *RESP*, *RESP1* only needs one function *TR*, and there exists only one assumption *path*. The *path* assumption ensures that there exists a finite, valid list $TR\ \tau$ leading to the desired Q-state .

we can substitute *RESP1* for *RESP*, since we can prove the lemma *resp-from-1*:

RESP1 cs TR N P Q $\Longrightarrow \exists F\ E.$ *RESP cs F E N P Q*

Now what we need to do is to prove the *RESP1* in order to get the conclusion.

5.2.2 Proving the *RESP1*

In order to prove the *RESP1*, we should first find a finite valid event list $TR\ \tau$ and a natural number N, and prove the *path* assumption of *RESP1*.

From above discussion, we know that there are three cases when a data comes to S and needs to be sent. In the first case, the first route in S's route cache is good. We let *?TR* equals to the event list of the process of transmitting this data, and the process of receiving data and transmitting acknowledgement.

In the second case, S's route cache is empty. We let *?TR* equals to the event list of the process of discovering the route, and transmitting the data using the route just discovered, and the process of receiving data and transmitting acknowledgement.

In the third case, the first route in S's cache is broken. We let *?TR* equals to the event list of the process of deleting broken route(s), and the process of discovering new route (if needed), and the process of transmitting this data, and the process of receiving data and transmitting acknowledgement. Figure 5 illustrates a state transition diagram for a network with four nodes S, A, B, and D .

We can find such a valid event list $TR\ \tau$ according to above description, and the length of it is finite because the size of route cache and the number of nodes is finite. Once found $TR\ \tau$, the *path* assumption of *RESP1* is provable. The proof is straight forward with the definition of *TR* in mind, although it is a little boring.

6 Conclusion

At present, there are few formal verifications for *secure* routing protocols, and there is no liveness verification for ad hoc routing protocols of any kind. In this paper, we presented a liveness proof of the secure routing protocol SRP. In our model, any node can move randomly, and there exists disturbance which may result in a failure of data reception. For the purpose of the liveness proof, we proposed a *fail* set to reasonably restrict the attackers' behaviours, and we explained why the definition of *fail* set is reasonable.

References

[1] K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *Journal of the ACM*, 49(4):538–576, 2002.

[2] L. Buttyan and I. Vajda. Towards provable security for ad hoc routing protocols. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 94–105, New York, NY, USA, 2004. ACM Press.

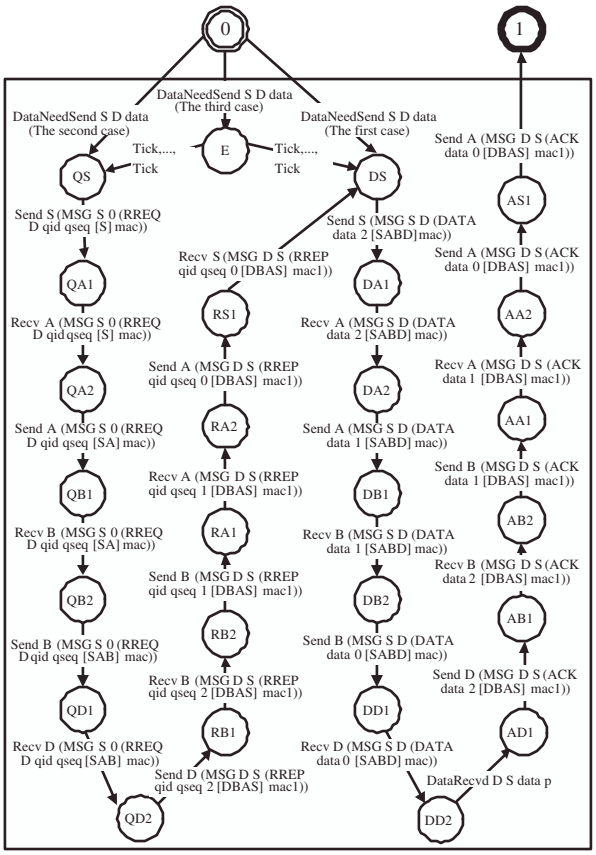

Figure 5. The state transition diagram for a four-node network

[3] A. R. Cavalli, C. Grepet, S. Maag, and V. Tortajada. A validation model for the dsr protocol. In *ICDCS Workshops*, pages 768–773, 2004.

[4] B. Dahill, K. Sazgiri, B. N. Levine, E. Belding-Royer, and C. Shields. A secure routing protocol for ad hoc networks. In *10th Conference on Network Protocols*, 2002.

[5] M. Z. Guerrero and N. Asokan. Securing ad hoc routing protocols. In *ACM Workshop on Wireless Security (WiSe) in conjunction with ACM MobiCom*, Atlanta, Georgia, sep 2002.

[6] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. In *MOBICOM*, pages 12–23, 2002.

[7] D. B. Johnson, D. A. Maltz, and Y.-C. Hu. The dynamic source routing protocol for mobile ad hoc networks (dsr). *Internet Draft: draft-ietf-manet-dsr-10.txt*, July 2004.

[8] Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comput. Sci.*, pages 91–130, 1991.

[9] S. Nanz and C. Hankin. Static analysis of routing protocols for ad-hoc networks. In *Proceedings of the 2004 ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'04)*, pages 141–152, 2004.

[10] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[11] P. Papadimitratos and Z. J. Haas. Secure routing for mobile ad hoc networks. In *Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS)*, pages 193–204, 2002.

[12] P. Papadimitratos and Z. J. Haas. Secure data transmission in mobile ad hoc networks. In *WiSe '03: Proceedings of the 2003 ACM workshop on Wireless security*, pages 41–50, New York, NY, USA, 2003. ACM Press.

[13] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[14] R. Renesse and A. H. Aghvami. Formal verification of ad-hoc routing protocols using spin model checker. In *IEEE MELECON*, Dubrovnik, Croatia, May 2004.

[15] S. C. Tanara Lauschner, Autran Macedo. Formal verification and analysis of a routing protocol for ad-hoc networks. 2000.

[16] H. Yang, X. Zhang, and Y. Wang. Liveness proof of an elevator control system. In *The 'Emerging Trend' of TPHOLs 2005*, Oxford University Computing Lab. PRG-RR-05-02, pages 190–204, 2005.

[17] S. Yang and J. S. Baras. Modeling vulnerabilities of ad hoc routing protocols. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 12–20, New York, NY, USA, 2003. ACM Press.

[18] X. Zhang, H. Yang, and Y. Wang. Liveness reasoning for inductive protocol verification. In *The 'Emerging Trend' of TPHOLs 2005*, Oxford University Computing Lab. PRG-RR-05-02, pages 221–235, 2005.