Towards Verification of the PANA Authentication and Authorisation Protocol using Coloured Petri Nets

Steven Gordon

Sirindhorn International Institute of Technology
Thammasat University, Thailand
steve@siit.tu.ac.th

Abstract

The Extensible Authentication Protocol (EAP) allows a server to request authentication information from a client. In order to transport EAP messages over an IP network, the Protocol for Carrying Authentication for Network Access (PANA) has been developed. This paper applies a protocol engineering methodology using Coloured Petri nets (CPNs) as a step towards formally verifying the design of PANA. State space analysis of a simple PANA configuration shows that the current specification has removed deadlocks discovered in previous PANA versions. Furthermore, state space and language analysis of PANA for different client retransmission limits leads to two important conjectures: the state space size (number of nodes, arcs) can be expressed as a polynomial in terms of the retransmission limit; and the protocol language is independent of the retransmission limit. The results suggest parametric verification is applicable to PANA. Finally, ideas for automatically validating the CPN model against the original specification are discussed.

1 Introduction

The Extensible Authentication Protocol (EAP) [1] is a framework for performing authentication in computer networks (see Figure 1). A typical usage scenario, as illustrated in Figure 2, involves a server (known as *authenticator* in EAP) initiating an authentication request to a *peer*. The peer responds to this, and any subsequent requests, until the authenticator determines the procedures to be a success (the peer is authenticated for network access) or failure (the peer is denied access to the network). In practice a third entity, the *authentication server*, may be utilised for storage of credential information. EAP is designed to support different authentication methods (e.g. MD5, TLS) and to operate over different (non-IP-based) network technologies. For example, a laptop can authenticate with a wireless LAN access point using IEEE 802.11i, or a home PC can authenticate with a dial-in server using EAP over the Point-to-Point Protocol (PPP).

In order to allow EAP to be carried over IP networks, PANA has been developed and released as IETF Request For Comments (RFC) 5191. The Protocol for Carrying Authentication for Network Access [8] is a lower layer for EAP, and PANA itself uses UDP as a lower layer. In addition to the protocol definition in [8], the PANA Working Group has developed a state-table model of PANA published as RFC 5609 [7]. Although the state-table model is for informative purposes, combined with the protocol definition, it provides a detailed explanation of the behaviour of PANA. However, as with many distributed protocols, it is important that the PANA specification is accurate and unambiguous. This is particularly important for an authentication protocol, where small errors or an ambiguous specification may lead to implementations with potentially damaging security flaws.

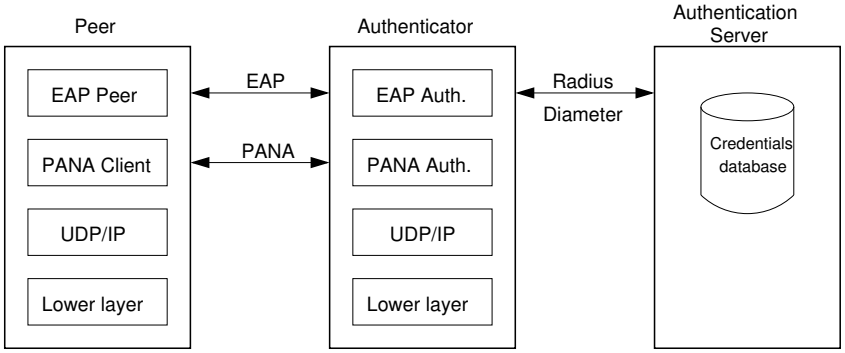


Figure 1: EAP framework

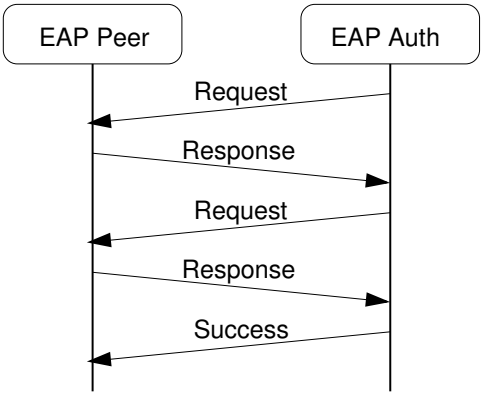


Figure 2: Typical EAP message sequence

Most research on PANA involved its application to wireless networks [18, 17, 5], especially performance analysis of PANA re-authentication during handovers [4, 9]. Little effort has been directed to the formal analysis of PANA, including security analysis. The overall aim of this research is to verify the design of PANA to ensure a complete and correct specification is available. To do so a protocol engineering methodology [2] is applied utilising Coloured Petri nets (CPNs) [14]. There are numerous examples of a protocol engineering methodology applied to other protocols (e.g. [19, 16, 11, 12]). The steps followed in this paper include:

1. Modelling of the PANA protocol specification using CPN Tools [15].

2. Simulation of the PANA CPN model to investigate specific scenarios. CPN Tools is used to step through sequences of events, and combined with BRITNeY [21] to automatically generate message sequence charts.

3. Functional property verification from state space analysis. CPN Tools is used to inspect terminal states and identify possible deadlocks, as well as bounds on communication channels.

4. Generation and inspection of the PANA protocol language (i.e. the possible ordering of interactions between PANA and the higher layer, EAP). Obtaining the protocol language is necessary in verifying that PANA is a faithful refinement of the service that EAP assumes is provided by the lower layer. However in this work, there is not yet a formal definition the PANA service offered to EAP. Hence only manual inspection of the protocol language is used at this stage.

Note that this paper does not attempt a formal security analysis (from a cryptographic viewpoint) of PANA. In fact, such analysis depends largely on EAP and other authentication methods, as PANA is only a protocol that carries EAP messages.

In previous work [13] a CPN model and initial analysis of PANA was presented. This was based on RFC5191 and draft version 6 of the PANA state tables. Using state space analysis of the simplest configurations of PANA (no retransmissions, no optimisation of the initiation procedure), a problem with aborting sessions was identified. Since then the PANA state tables have been updated (from version 6 through to 13, which is now published as RFC 5609). This paper uses an updated CPN model of the latest version of PANA. In addition, new configurations are analysed, in particular with retransmissions and initiation optimisation.

The remainder of this paper is organised as follows: Section 2 describes PANA and EAP in further detail. Section 3 provides an overview of the CPN model of PANA. Results from the formal analysis of the PANA Authentication and Authorisation Phase are presented in Section 4. Discussion of the approach, results and ideas for future work are given in Section 5.

2 EAP and PANA

2.1 EAP

EAP is a request/response protocol where only a single packet is in-flight at once, i.e. the authenticator cannot send a new request until the response from the previous request is received. The requests contain authentication challenges to the client. EAP assumes the lower layer (in this paper, PANA) will provide in-order delivery of packets, however it does not require the lower layer to be reliable, provide security or remove duplicates. A typical scenario, as illustrated in Figure 2, involves one or more EAP Request/Response exchanges (always initiated by the authenticator) followed by a final EAP Success or EAP Failure message, depending on the authentication information supplied by the client.

2.2 PANA

The role of PANA is to transport EAP messages between peer (referred to as PANA Client or PaC) and authenticator (PAA). PANA uses UDP at the transport layer, and hence the service provided to PANA may have packet losses, duplication and re-ordering. An exchange of messages in PANA is a session, which is divided into four phases:

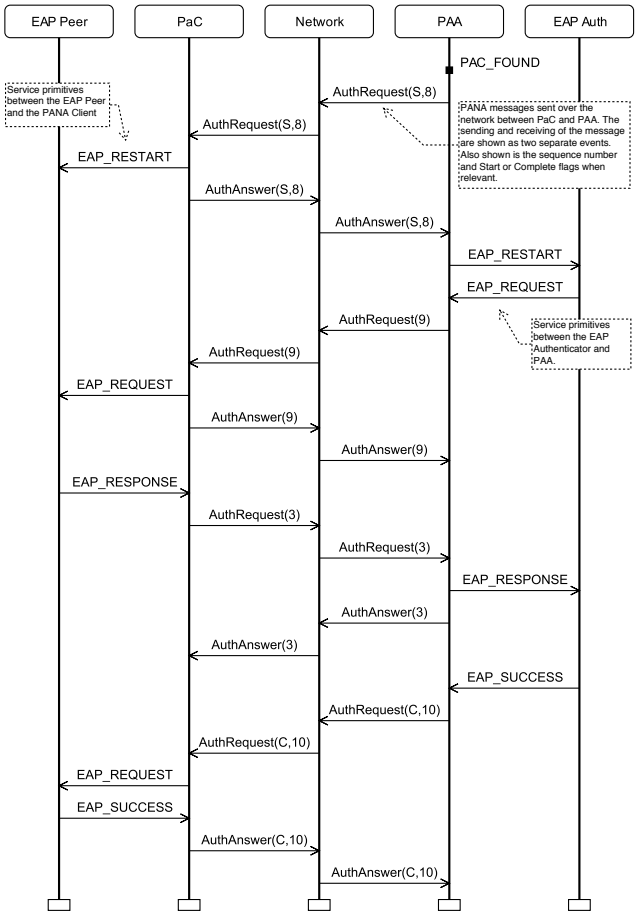
Authentication and Authorisation At the start of a PANA session this phase involves the exchange of EAP messages to perform authentication.

Access Once authentication is successful network access is provided. In this phase either PaC or PAA may test for the liveness of the session (which has a limited lifetime).

Re-authentication May be performed to maintain the session liveness.

Termination Either PaC or PAA may terminate a session. If a session isn't terminated gracefully, then a timeout on the PANA session will result in the termination.

PANA communications are implemented as a series of request and answer messages. To explain the Authentication and Authorisation phase consider the example scenario in Figure 3(a) (which is generated from our CPN model in Section 3). It shows the



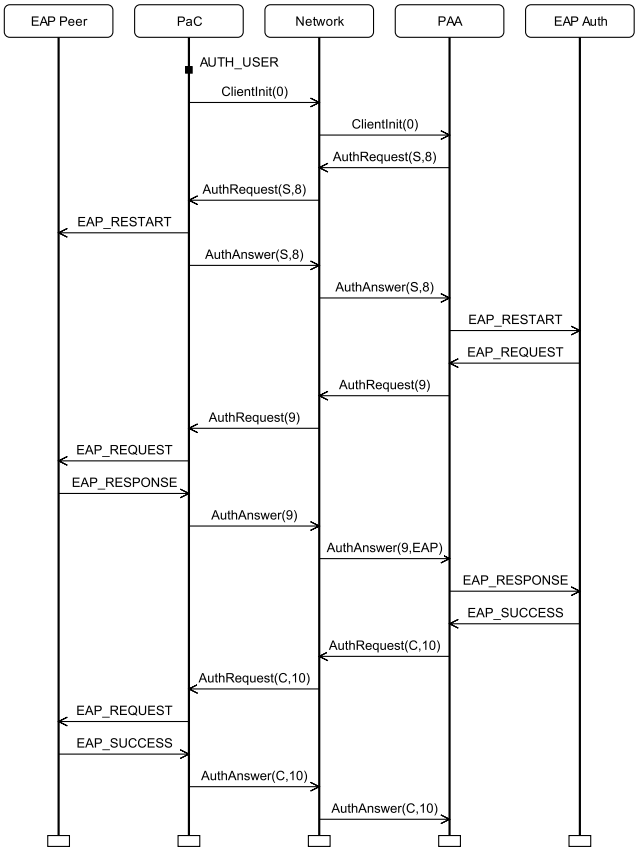


Figure 3: Message sequence chart of PANA Authentication phase: (a) no piggybacking; (b) with piggybacking

communication between EAP entity and corresponding PANA entity, as well as between PANA entities across the network.

The PANA session can be initialised by either the PAA or PaC (Figure 3(a) shows an example initiated by PAA, whereas Figure 3(b) illustrates PaC initiation, as well as piggybacking). The methods for each entity learning about the presence of the other is out side of the scope of PANA. For a PAA-initiated-session, after it discovers the presence of a PaC, it sends an AuthRequest message to start the session (the 'S' flag indicates this message is to start the session). This initial AuthRequest is used to force a restart of the EAP session at the Peer. The PaC responds with an AuthAnswer, which results in the EAP session at the Authenticator restarting.

The EAP Authenticator then initiates the authentication with an EAP Request. This triggers the PAA to send an AuthRequest carrying the EAP Request method (e.g. the authentication challenge). Upon receipt of the AuthRequest, the PaC passes the EAP Request method to the EAP Peer and replies with an AuthAnswer.

The PaC sends the response to the challenge in an AuthRequest (which is also acknowledged by the PAA with a AuthAnswer). This sequence of EAP requests and responses (and AuthRequest and AuthAnswer messages) may repeat until the authentication is complete. Finally the EAP Authenticator will send a Success or Failure method indicating the result of authentication. The EAP Success is shown in Figure 3(a), which is carried in an AuthRequest with the Complete flag set. Once the AuthAnswer is received by the PAA, both PAA and PaC have the PANA session established and the Access phase is entered. Note that the AuthAnswer messages can be considered as acknowledgements of the AuthRequest messages. They do not necessarily carry the answer to the authentication challenge (i.e. the EAP response). Other relevant details of PANA include:

- 32-bit sequence numbers are used to maintain ordering and perform error detection. The sequence numbers at PAA and PaC are independent. An outgoing request message contains a sequence number, and the corresponding answer message must have the same sequence number.

- Request messages are retransmitted if an answer is not received within a specified time. The session is terminated if too many retransmissions occur.

- PANA messages contain 16 bytes of fixed size header (e.g. flags, message type, sequence number, session identifier) as well as a variable number of Attribute-Value Pairs (AVPs). AVPs include: the actual EAP message; authentication data; session lifetime; and other security related information.

- Optional piggybacking of messages allows either PaC or PAA to send a single PANA message that represents both an answer and a request. For example in Figure 3(a) without piggybacking, PaC sends an AuthAnswer(8) followed by AuthRequest(3). With piggybacking turned on (Figure 3(b)), the PaC sends a single message, AuthAnswer(9) which also includes the EAP response.

2.3 EAP/PANA Interface

In order to verify if the PANA protocol correctly interacts with EAP, it is necessary to understand the interface between the two layers. The EAP state-machines [20] specify the variables used for communication between EAP and a lower layer. This information is summarised in Figure 4. As an example, when the EAP Authenticator sends an EAP Request, the eapReq flag will be set to true and the Request method will be included in eapReqData.

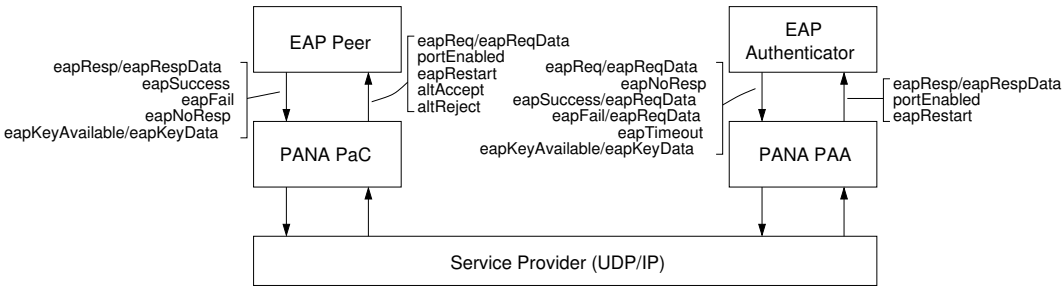


Figure 4: EAP/PANA interface based on [20]

In addition to the EAP-defined interface, the PANA state-tables [7] describes its own set of variables and procedures used for communication between PANA and EAP. As an alternative to the two separate set of variables defined by EAP and PANA, *service primitives* can be used to describe the interface between the two layers. Table 1 is an attempt to define the service primitives that correspond to information found in the EAP standard [20] and PANA standard [8, 7]. The table lists the EAP-defined variables, the PANA-defined variables, as well as the newly defined service primitives. However from the available specifications it is difficult to define all valid sequences of primitives, and hence orderings are not yet defined. In Section 4 the service primitives are used in determining the PANA protocol language, i.e. the possible sequences of service primitives.

Table 1: EAP/PANA interface variables and service primitives

No.	Entity	EAP	PANA	Primitive
1	Peer/PaC	-	AUTH_USER	CAuthUser
2	Peer/PaC	eapRestart	EAP_RESTART	CRestart
3	Peer/PaC	eapReq	EAP_REQUEST	CRequest
4	Peer/PaC	eapResp	EAP_RESPONSE	CResponse
5	Peer/PaC	eapSuccess	EAP_SUCCESS	CSuccess
6	Peer/PaC	eapFail	EAP_FAILURE	CFailure
7	Peer/PaC	-	-	CTimeout
8	Peer/PaC	-	ABORT	CAbort
9	Auth/PAA	-	PAC_FOUND	APacFound
10	Auth/PAA	eapRestart	EAP_RESTART	ARestart
11	Auth/PAA	eapReq	EAP_REQUEST	ARequest
12	Auth/PAA	-	-	AResponse
13	Auth/PAA	eapSuccess	EAP_SUCCESS	ASuccess
14	Auth/PAA	eapFail	EAP_FAILURE	AFailure
15	Auth/PAA	-	EAP_TIMEOUT	ATimeout
16	Auth/PAA	-	ABORT	AAbort
17	Peer/PaC	-	DISCARD	CDiscard
18	Auth/PAA	-	DISCARD	ADiscard

3 Coloured Petri Net Model of PANA

3.1 Model Hierarchical Structure

A CPN model of PANA has been created based on the state tables in [7]. The model consists of 23 pages, 63 transitions and 7 places (however not all transitions are relevant for the Authentication and Authorisation phase of PANA considered in this paper).

The model focuses on the components of the protocol important for functional verification, i.e. the ordering of exchange of messages. Where possible, abstraction is used so that details of message content and format can be omitted. This makes analysis easier, but at the expense of a complete protocol specification. The structure of the model fol-

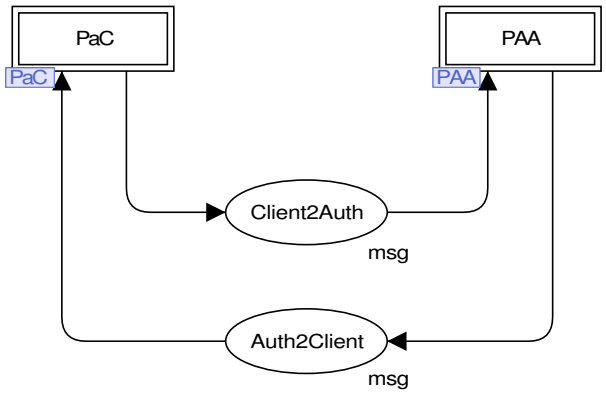


Figure 5: Top-level page of PANA CPN Model

lows a state-based approach, where transitions are used to model actions (each row) in the PANA state-tables. Further discussion of this approach is given in Section 5.

The PANA CPN is hierarchical, with the PaC and PAA modelled on separate pages, and then each state of the PaC/PAA modelled on separate pages. This is achieved using substitution transitions and port/socket places. At the highest level (Figure 5) there are two transitions (PaC and PAA) and two places modelling the communication channel between PaC and PAA (and vice versa). The channel is assumed to be reliable (no message loss), but allows re-ordering and delay of messages. As UDP is used as the lower layer by PANA, the assumption of no message loss is not always valid. However assuming no message loss is a useful starting point for the analysis, since modelling loss may hide deadlocks in the protocol. Studying the impact of message loss is part of future work.

Both the PaC and PAA transitions at the top-level contain detailed models on their respective sub-pages. These sub-pages (Figures 6 and 7) contain transitions that model the events at each state and a place to model the current state. For example, the place Client stores the current state of PaC, C_INITIAL, and state variables such as eap_piggyback=false. Places LastClientMsg and LastAuthMsg are used to store the previous message sent in case a retransmission is necessary.

3.2 Modelling State Tables

Each transition on the PAA/PaC sub-pages is further decomposed to individual pages that model the events, conditions, actions and next states as presented in the state-tables. To explain, the case of the PAA in the INITIAL state will be used as an example. The PANA state-table from [7] for the PAA INITIAL state is given in Figure 8.

For a given state, the state-tables in [7] specify:

- An exit condition, i.e. the conditions that must occur. For example, in Figure 8 there is an exit condition called EAP REQUEST, which indicates a request is received from the higher layer EAP entity.

- Exit actions, i.e. the actions that will be executed upon the conditions being met. For the EAP REQUEST condition, the actions are to transmit a PANA AuthRequest message (`Tx:PAR[S]`) and then start the retransmission timer (`RtxTimerStart()`. Note that the contents of the AuthRequest message depends on the type of EAP method received from the higher layer.

- The exit state, i.e. the next state of the entity. After the EAP REQUEST and corresponding actions, the PAA will re-enter (i.e. remain in) the INITIAL state.

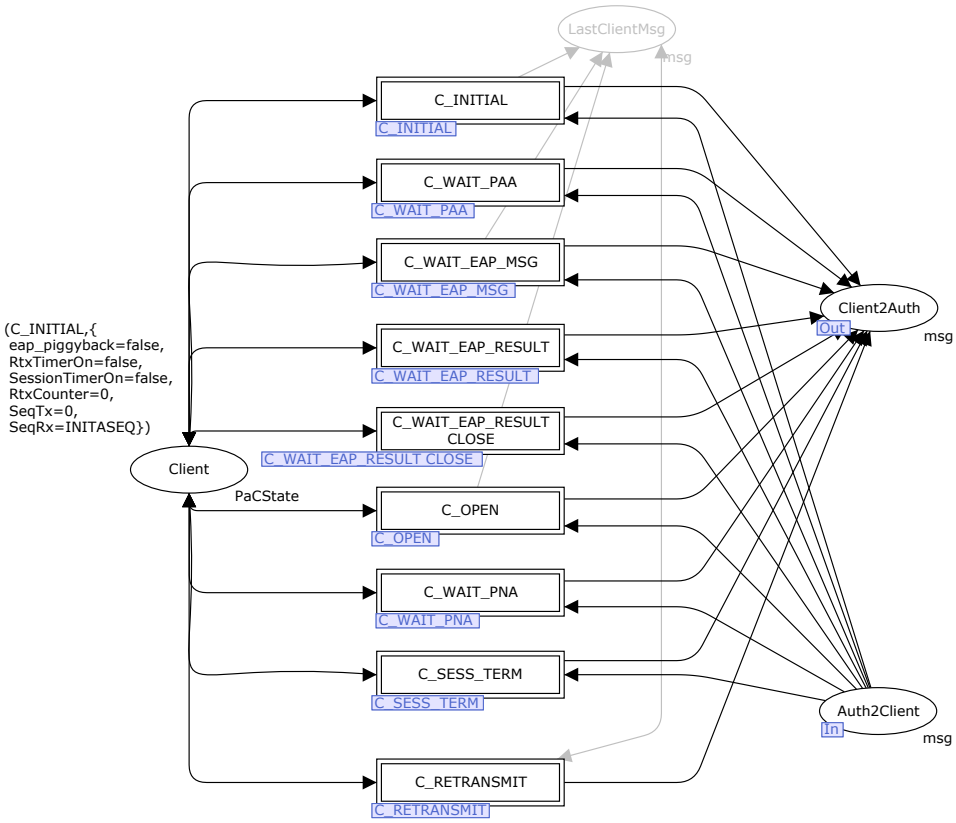


Figure 6: CPN Model of PaC

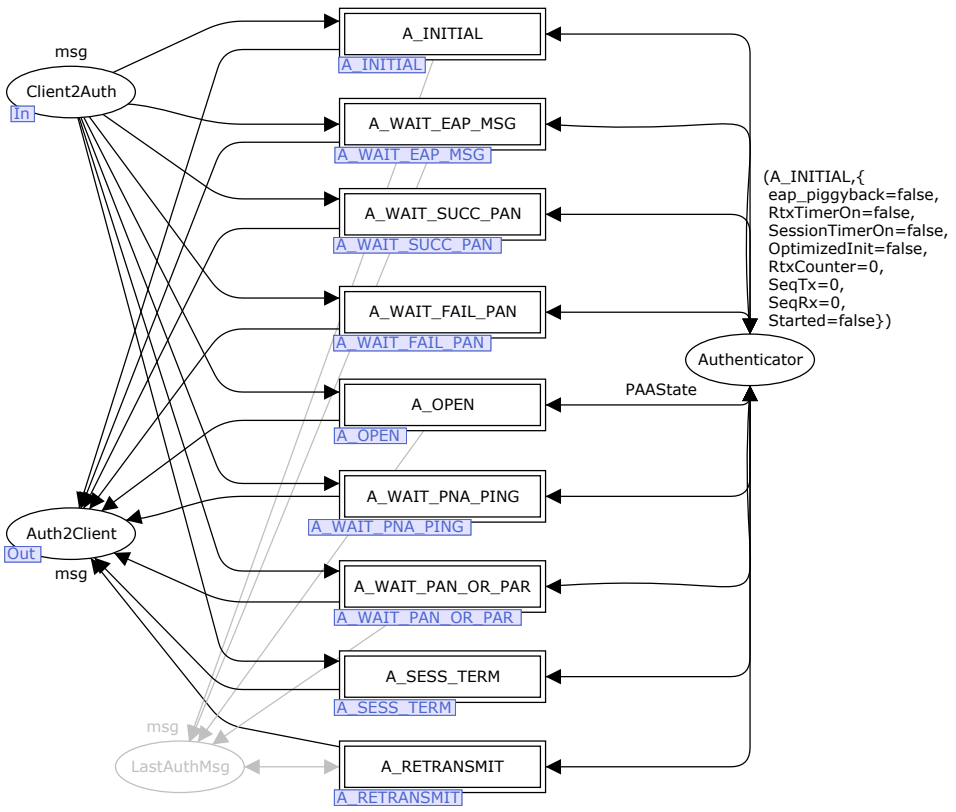


Figure 7: CPN Model of PAA

Ideally, each entry in the state table could be modelled by a single transition in the CPN. However for some entries multiple transitions are used. This is because the exit actions in the state table sometimes contain *conditions*. Consider the action for the exit condition `(Rx:PCI[] || PAC_FOUND)` in Figure 8. If `OPTIMIZED_INIT` is set then one sequence of actions are taken, and if not set another sequence of actions are taken. Hence this entry in the state table is modelled as two transitions. Furthermore, there are in fact two distinct exit conditions: `Rx:PCI[]` and `PAC_FOUND`. These are therefore modelled as separate transitions (this is necessary as the two conditions correspond to different interactions with the higher layer). As a result, the one state table entry in this case is modelled with four transitions. In Figure 9 the four transitions are: PAC_FOUND Optimum; PAC_FOUND NoOptimum; RxPCI Optimum; and RxPCI NoOptimum.

```
------------------------------
State: INITIAL (Initial State)
------------------------------

Initialization Action:

  OPTIMIZED_INIT=Set|Unset;
  NONCE_SENT=Unset;
  RTX_COUNTER=0;
  RtxTimerStop();

Exit Condition           Exit Action                Exit State
-----------------------+--------------------------+------------
 - - - - - - - - (PCI and PAA initiated PANA) - - - - - - - - -

(Rx:PCI[] ||             if (OPTIMIZED_INIT ==      INITIAL
 PAC_FOUND)                  Set) {
                           EAP_Restart();
                           SessionTimerReStart
                            (FAILED_SESS_TIMEOUT);
                         }
                         else {
                           if (generate_pana_sa())
                                Tx:PAR[S]("PRF-Algorithm",
                                   "Integrity-Algorithm");
                           else
                                Tx:PAR[S]();
                         }

EAP_REQUEST              if (generate_pana_sa())    INITIAL
                             Tx:PAR[S]("EAP-Payload",
                                "PRF-Algorithm",
                                "Integrity-Algorithm");
                         else
                             Tx:PAR[S]("EAP-Payload");
                         RtxTimerStart();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - (PAN Handling)  - - - - - - - - - -
Rx:PAN[S] &&             if (PAN.exist_avp          WAIT_EAP_MSG
((OPTIMIZED_INIT ==         ("EAP-Payload"))
  Unset) ||                TxEAP();
PAN.exist_avp            else {
  ("EAP-Payload"))         EAP_Restart();
                           SessionTimerReStart
                            (FAILED_SESS_TIMEOUT);
                         }

Rx:PAN[S] &&             None();                    WAIT_PAN_OR_PAR
(OPTIMIZED_INIT ==
  Set) &&
! PAN.exist_avp
 ("EAP-Payload")
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 8: State Table for PAA in INITIAL state from [7]

Each transition has or may have:

- An input arc from a place containing the current state, and related state information, e.g. sequence numbers, flags. Consider the bottom transition in Figure 9 as an example. The transition is only enabled when PAA is in the A_INITIAL state.
- An input arc from the communication places (Client2Auth or Auth2Client, depending on the entity) if the event involves receiving a message. (The example transition is only enabled when a AuthAnswer has been sent by the PaC).
- A guard for the conditions related to the event. (AuthAnswer message must have the Start flag set, not contain EAPPayload, and PAA must be using OptimizedInit)
- An output arc to the communication places if the action involves sending a message. (No message is sent for the example transition).
- An output arc to the place containing state information, where the next state is stored. (The new state is A_WAIT_PAN_OR_PAA for the example transition—there are no changes to the state variables).

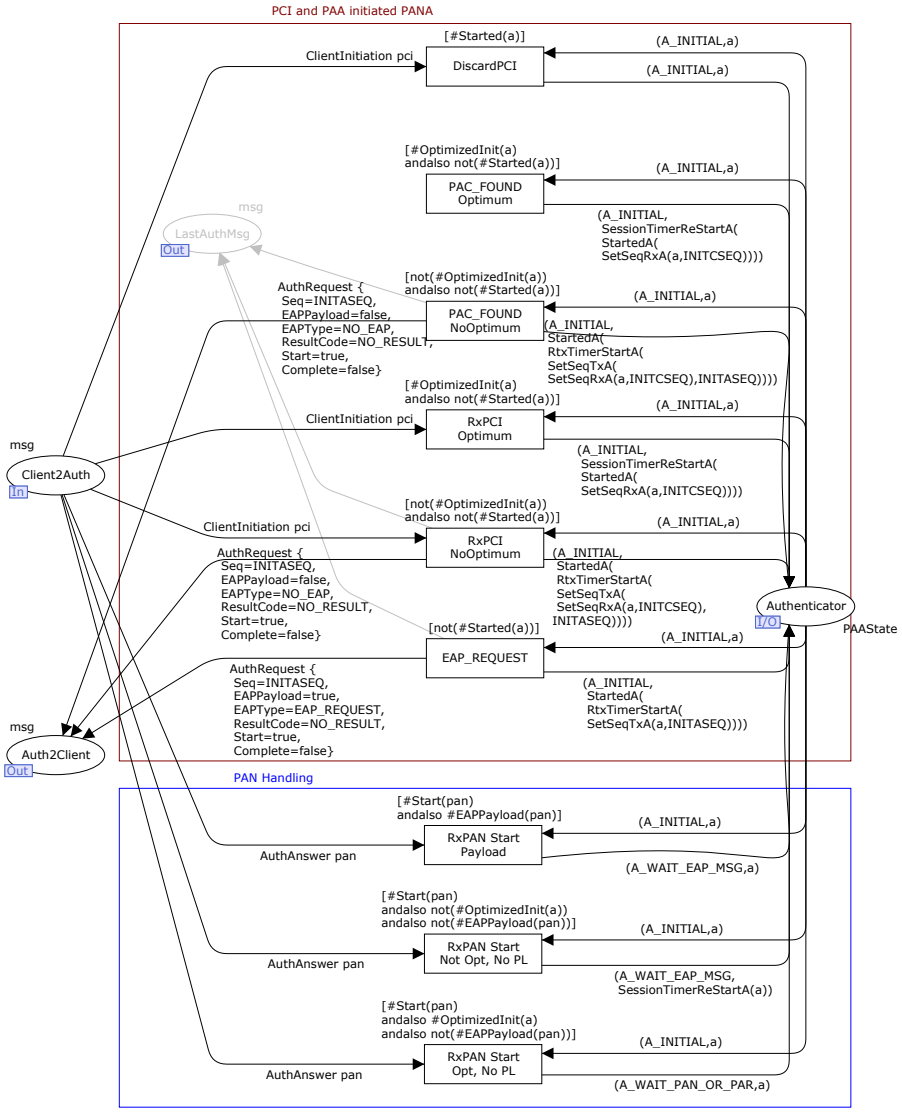


Figure 9: CPN Model of A_INITIAL state

3.3 Message Sequence Charts

To illustrate the message flow between PANA and EAP entities, BRITNeY [21] and the message sequence chart library are used in conjunction with CPN Tools. Each transition modelling state table entries has a code segment indicating the transfer of messages or special events (if any). For clarity, the code segments are omitted from the figures in this paper. The code segment for the bottom transition in Figure 9 is:

```
input (pan); output (); action (if mscOn then (
msc.addEvent("Network","PAA",concat ["AuthAnswer[S,"
  Int.toString (#Seq(pan)),")"])) else ());
```

The firing of the transition will draw an arrow indicating an AuthAnswer message being sent from the *Network* to the PAA. Message sequence charts, such as that illustrated in Figure 3, are useful for testing during model development, and exploring the protocol behaviour under specific conditions.

4 Analysis Results

4.1 Approach and Assumptions

The state space is generated with CPN Tools and examined to determine the presence of unexpected terminal states (deadlocks) in PANA, as well as the integer bounds on the communication places.

To investigate the protocol language (i.e sequence of interactions between PANA and higher layer), CPN Tools was used to translate the state space into a FSA, where states were mapped to their node number and each binding element was mapped to an integer depending on the service primitive it represented. The integer mappings are given in Table 1. All other binding elements map to 0. In addition, terminal states in the state space mapped to halt states. Then using AT&T's FSM Library[1] and GraphViz[2], the minimised deterministic FSA is created, representing the PANA protocol language. The protocol language is studied in order to identify unexpected sequences of events in PANA.

In this paper four protocol parameters are of interest:

1. Piggybacking (PB_{PaC} and PB_{PAA}): this can be either on (true) or off (false). If on, the PaC/PAA may combine two messages into one, thereby reducing the number of messages sent over the network.

2. Optimised Initiation ($OptInit$): this can be either on or off. If on, the PAA can send an EAP Request in the initial AuthRequest message.

3. PaC Maximum Retransmission Count (MRC_{PaC}): an integer indicating the maximum number of retransmissions of request messages by PaC.

4. PAA Maximum Retransmission Count (MRC_{PAA}): an integer indicating the maximum number of retransmissions of request messages by PAA.

The analysis assumes only a single EAP Request is sent by the Authenticator. In addition, 4-bit sequence numbers are used (instead of 32-bit), and the initial sequence numbers are randomly set at 3 and 8 for PaC and PAA, respectively.

In Section 4.2 analysis results for a simple configuration with no retransmissions are presented. Then the effect of retransmissions is considered in Section 4.3.

[1] http://www.research.att.com/~fsmtools/fsm/

[2] http://www.graphviz.org/

Table 2: State Space Analysis of PANA CPN with No Retransmissions

Piggyback	OptInit	States	Arcs	Terminal States	Client2Auth	Auth2Client
Off	Off	15531	34047	6866	5	3
On	Off	3436	7212	1265	3	2
Off	On	12079	26360	5292	5	3
On	On	2085	4233	775	3	2

4.2 Analysis of Simple Configuration: No Retransmissions

Statistics from the PANA state space are shown in Table 2. In all cases, $MRC_{PaC} = MRC_{PAA} = 0$. Also, for simplicity piggybacking is either on for both PaC and PAA *or* off for both PaC and PAA. The integer bounds of the communication places, Client2Auth and Auth2Client, are reported.

The state space is significantly larger when *Piggyback* is off. This is because without piggybacking, both the PaC and PAA must send a separate AuthRequest message to carry EAP responses. As illustrated in Figure 3, after receiving an EAP_REQUEST, the PaC sends an AuthAnswer(9) acknowledging the receipt of the EAP_REQUEST, and then sends a AuthRequest(3) containing the EAP_RESPONSE. With piggybacking, the EAP_RESPONSE can be sent in the AuthAnswer(9), omitting the need for the AuthRequest(3). Figure 10 shows the partial state space illustrating the sequences from Figure 3.

The integer bound of the communication places Client2Auth and Auth2Client indicates the number of messages an entity can send before it has to wait for a response from the peer entity. Normally an entity sends a AuthRequest and then waits for an AuthAnswer before sending another message. However, with piggybacking off for example, the PAA can send an AuthRequest message containing an EAP Request, and then if the EAP Authenticator returns a result to PAA, the PAA can send anotherAuthRequest message containing the EAP result. After this the PAA must wait for an answer. Hence the upper bound on the messages in Auth2Client is 2. Similar scenarios occur for the other configurations.

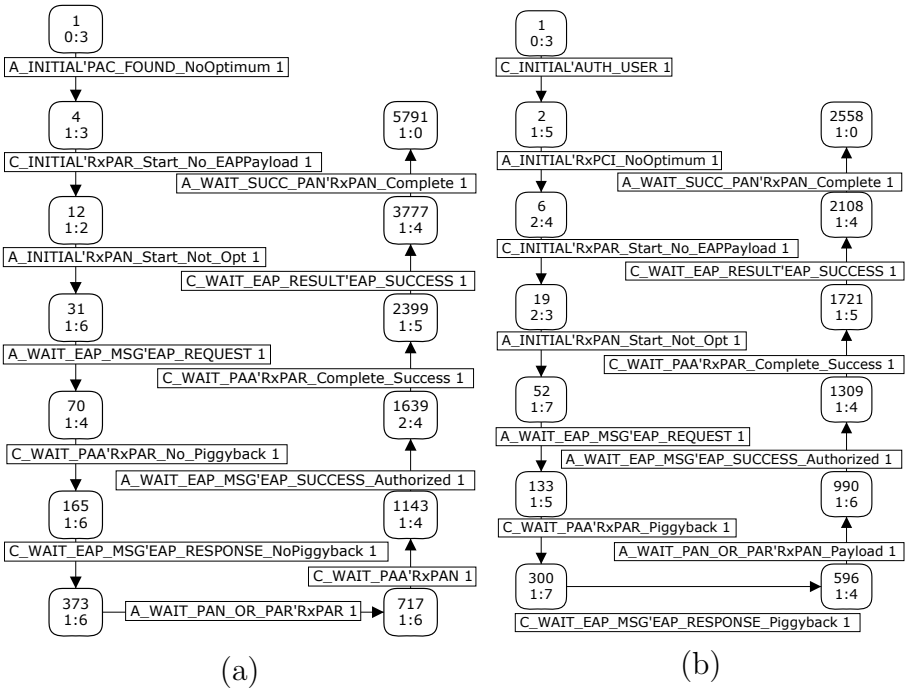


Figure 10: Selected state space showing sequence corresponding to those in Figure 3

4.2.1 Terminal States

Closer inspection of the terminal states is necessary to determine if any are unexpected. However first the expected terminal states must be defined. From previous analysis of PANA [13], the expected terminal states are classified by the state that the PaC and PAA finish in. Recall that only the Authentication and Authorisation phase of PANA is analysed: if this phase is successful the PANA session will be OPEN, and both PaC and PAA enter the Access phase. If unsuccessful, both PaC and PAA should enter the CLOSED state (PANA session is closed). Therefore expected terminal states are defined as those where the PaC/PAA entity is in a valid state. The four expected groups of terminal states are:

1. C_OPEN and A_OPEN: both PaC and PAA have opened a PANA session, i.e. authentication was successful.

2. C_CLOSED and A_CLOSED: both PaC and PAA have closed a PANA session, e.g. after a failed authentication attempt or abort due to too many retransmissions.

3. C_OPEN and A_CLOSED: PaC successfully opens a session, however the PAA aborts before opening the session thereby leaving it in the CLOSED state. This is considered valid because the PaC will enter the Access phase and eventually the PANA session will timeout (and close) after receiving no responses from the PAA. Rather than explicitly modelling the timeout event which is part of the Access phase, these terminal markings are considered valid.

4. C_CLOSED and A_OPEN: PAA successfully opens a session, however the PaC aborts. Following the same reasoning as above, this is a valid terminal state.

In [13] a fifth, unexpected group of terminal states was discovered. After the PaC responded to the initial AuthRequest message (with Start bit set), it entered the C_WAIT_PAA state. If the PAA aborted before receiving the AuthAnswer from PaC, then the PAA entered A_CLOSED. This was an invalid terminal state because as the PANA session had not yet started by the PaC, the session timer would never expire, leaving PaC in C_WAIT_PAA. This problem arose because, with no explicit abort messages, an entity only knows the peer has aborted if a time out occurs. In specific cases, such as described above, a timer is not started, and hence no timeout will occur.

In the updates of the PANA state table from version 6 to version 13 this behaviour has been fixed. That is, the session timer is started after PaC sends the initial AuthRequest message (rather than after entering the C_OPEN state). Therefore if PaC is waiting in the C_WAIT_PAA state, while the PAA has aborted, eventually PaC session will time-out. Similar behaviour can lead to a valid terminal state with PaC in C_WAIT_PAA and PAA in A_OPEN.

Finally, there is another special terminal state when optimised initiation is used (this case was not analysed in [13]). Note that both PaC and PAA may initiate the Authentication and Authorisation phase. If both entities initiate before receiving a message from the peer, then the PAA takes precedence. That is, the PAA will discard any ClientInitiation messages received from PaC. Similar to the above issues, if the PaC then aborts, the PAA may remain in the A_INITIAL start. However, as the session timer has been started, this is considered a valid terminal state (as eventually, the PAA will timeout and close the session).

Therefore, we define three more valid groups of terminal states: (5) C_WAIT_PAA and A_OPEN; (6) C_WAIT_PAA and A_CLOSED; and (7) C_CLOSED and A_INITIAL.

Using CPN Tools queries, the terminal states of each state space were inspected to determine if they matched any of the above 7 valid groups. Table 3 shows the count of terminal states for each group. No unexpected terminal states were discovered. Further discussion on the terminal states is given in Section 5.2.

Table 3: Terminal States with No Retransmissions

Piggyback	OptInit	1	2	3	4	5	6	7	Unexpected	Total
Off	Off	61	6090	164	456	11	84	0	0	6866
On	Off	41	871	98	61	8	186	0	0	1265
Off	On	59	4660	164	333	7	68	1	0	5292
On	On	23	534	59	38	4	116	1	0	775

4.2.2 Language Analysis

There is currently no formally defined service language for PANA. Table 1 lists the possible service primitives to be exchanged between PANA and the higher layer, but does not define any ordering. Hence language analysis of PANA is not used for verification (i.e. comparing a protocol language to a service language), but instead to gain confidence in the correct operation of the protocol and to investigate a possible service language. However, as will be shown shortly, the size of the PANA protocol language is too large for manual inspection. Therefore an abstraction is applied where the protocol languages for PaC and PAA are produced separately. That is, the *PaC Only* protocol language shows the sequence of primitives exchanged between the PaC and the higher layer. This is useful in validating that at least the PaC and PAA are operating in a normal manner.

Table 4 gives the number of states/arcs/final states in the minimised deterministic FSA, as well as the number of sequences in the language. Results are shown for the complete PANA protocol language, PaC only and PAA only. Figures 11 and 12 show the PaC only and PAA only protocol languages when piggybacking and optimised initiation are both on. Visual inspection of the PaC and PAA languages reveal no obvious unexpected sequences: the ordering of Requests then Responses is as expected; there are no Requests or Responses after a Success or Failure; and the PAA only sends a single Request. The sequence of primitives as seen by the PaC shown in Figure 3 are captured in the sequence from states 0–2–5–8–12–19 in Figure 11. Unfortunately, the full PANA protocol language is too large for inspection. Analysis of the full PANA language, and eventually determining a PANA service language, are left for future work.

Table 4: Protocol Language of PANA CPN with No Retransmissions. (S = States; A = Arcs; FS = Final States; Seq = Sequences)

		PANA				PaC Only				PAA Only			
PB	OptInit	S	A	FS	Seq	S	A	FS	Seq	S	A	FS	Seq
Off	Off	95	369	7	17862	14	36	6	71	7	23	2	70
Off	On	100	383	6	14742	20	56	8	60	7	23	2	56
On	Off	88	350	6	13604	14	35	5	65	9	31	3	64
On	On	93	340	6	11468	20	47	6	48	9	28	3	54

4.3 The Effect of Retransmissions

The simple configuration assumed no retransmissions from either PaC or PAA. However in PANA the PaC (or PAA) may retransmit an AuthRequest (or ClientInitiation) message up to MRC_{PaC} (or MRC_{PAA}) times if a corresponding AuthAnswer has not been received.

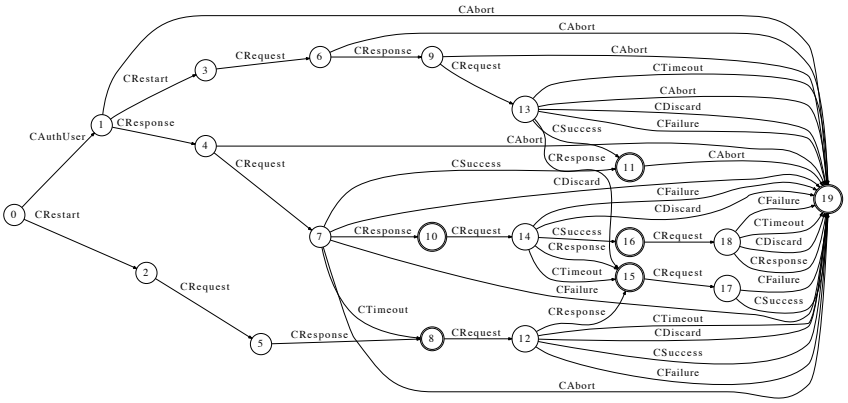


Figure 11: PaC language (piggybacking on, optimised initiation on)

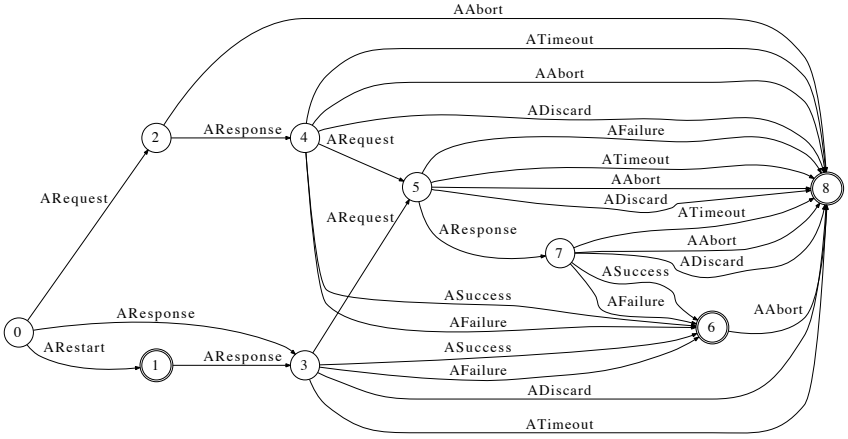


Figure 12: PAA language (piggybacking on, optimised initiation on)

Table 5 lists state space and language statistics for the PANA CPN for increasing values of MRC_{PaC}. In all cases, $MRC_{PAA} = 0$ [3].

First consider the results when piggybacking is on. For both cases of Optimised Initiation on and off, the increase in MRC_{PaC} leads to a quadratic increase in the size of the state space, as well as number of terminal markings. For values 0 to 5 of MRC_{PaC}, the number of states (S), arcs (A) and terminal markings (T) can be expressed as Equations (1) to (3) for the case of Optimised Initiation off:

$$S_{MRC_{PaC}} = 441(MRC_{PaC})^2 + 2958(MRC_{PaC}) + 3436 \quad (1)$$

$$A_{MRC_{PaC}} = 1190.5(MRC_{PaC})^2 + 7029.5(MRC_{PaC}) + 7212 \quad (2)$$

$$T_{MRC_{PaC}} = 71.5(MRC_{PaC})^2 + 813.5(MRC_{PaC}) + 1265 \quad (3)$$

and Equations (4) to (6) for the case of Optimised Initiation on:

$$S_{MRC_{PaC}} = 270.5(MRC_{PaC})^2 + 3721.5(MRC_{PaC}) + 2085 \quad (4)$$

$$A_{MRC_{PaC}} = 699(MRC_{PaC})^2 + 4150(MRC_{PaC}) + 4233 \quad (5)$$

$$T_{MRC_{PaC}} = 45.5(MRC_{PaC})^2 + 502.5(MRC_{PaC}) + 775 \quad (6)$$

[3]Initial results reveal the increase in state space size as MRC_{PAA} is increased is much larger than in the case of MRC_{PaC}. More detailed analysis of MRC_{PAA} is left for future work

Table 5: State Space Analysis of PANA CPN with PaC Retransmissions (Term = Terminal States; C2A = Client2Auth; A2C = Auth2Client; Seq = Sequences)

Parameters				State Space				Bounds		Language
PB	*OptInit*	MRC_{PaC}	MRC_{PAA}	States	Arcs	Term.	Time	C2A	A2C	Seq.
Off	Off	0	0	15531	34047	6866	91	5	3	17862
Off	Off	1	0	47046	112620	18943	1549	6	3	17862
Off	Off	2	0	101624	257908	38064	5129	7	3	17862
Off	Off	3	0	186485	494481	65729	15118	8	3	17862
Off	On	0	0	12079	26360	5292	52	5	3	13604
Off	On	1	0	38691	93267	15174	665	6	4	14675
Off	On	2	0	85589	219656	30983	2248	7	4	14675
Off	On	3	0	159568	428774	54100	11413	8	4	14675
Off	On	4	0	268212	747212	85947	33286	9	4	14675
On	Off	0	0	3436	7212	1265	5	3	2	14742
On	Off	1	0	6835	15432	2150	14	4	2	14742
On	Off	2	0	11116	26033	3178	24	5	2	14742
On	Off	3	0	16279	39015	4349	50	6	2	14742
On	Off	4	0	22324	54378	5663	127	7	2	14742
On	Off	5	0	29251	72122	7120	146	8	2	14742
On	Off	10	0	77116	196557	16550	997	13	2	14742
On	On	0	0	2085	4233	775	2	3	2	11468
On	On	1	0	4163	9082	1323	6	4	2	11468
On	On	2	0	6782	15329	1962	14	5	2	11468
On	On	3	0	9942	22974	2692	28	6	2	11468
On	On	4	0	13643	32017	3513	51	7	2	11468
On	On	5	0	17885	42458	4425	83	8	2	11468
On	On	10	0	47210	115633	10350	342	13	2	11468

The equations hold for $MRC_{PaC} = 10$, giving increased confidence that they are true for any value of MRC_{PaC}. Similar relationships between retransmission limits and state space size have been observed with other protocols [11, 12].

A much larger growth in the state space size is seen when considering no piggybacking (see Figure 13). Further state space results are necessary to determine the exact relationship between state space size and MRC_{PaC} when piggybacking is off.

When piggybacking is on, the PaC can send EAP responses in AuthAnswer messages. That is, the PaC does not send any AuthRequest messages (instead, the PaC sends AuthAnswer messages in response to AuthRequest messages sent by the PAA). Therefore the only message that can be retransmitted is the ClientInitiation which is used by the PaC to start the session.

When piggybacking is off, the PaC can retransmit a ClientInitiation message as well as an AuthRequest message that contains the EAP response (since the EAP response cannot be piggybacked in an AuthAnswer). The ability to retransmit the AuthRequest results in significant increase in the number of possible states, as illustrated in the partial state space in Figure 14.

The integer bound of the communication place Client2Auth increases linearly as the retransmission limit MRC_{PaC} increases. This is expected as the retransmission mechanism simply means an additional MRC_{PaC} messages can be sent, and stored in Client2Auth before the PaC must wait for the PAA to receive a message and respond.

Closer inspection of the PANA protocol language reveals the language is independent of MRC_{PaC} (with one exception, explained shortly). In other words, retransmissions by the PaC do not result in additional interactions between PAA and EAP Authenticator, nor between PaC and EAP Peer. The reason is that retransmissions are only used in two possible instances by the PaC: ClientInitiation in the C_INITIAL state and AuthRequest

carrying EAP response in the C_WAIT_EAP_MSG state. In both cases, when the PAA receives one of these messages (either original or retransmitted) it will process that message and ignore any subsequent duplicates. An exception is the protocol language with piggybacking off, and optimised initiation on. With no retransmissions there are less sequences than with retransmissions. The reason for this is retransmitted messages in this case can cause a different ordering of interactions between PAA and EAP Authenticator.

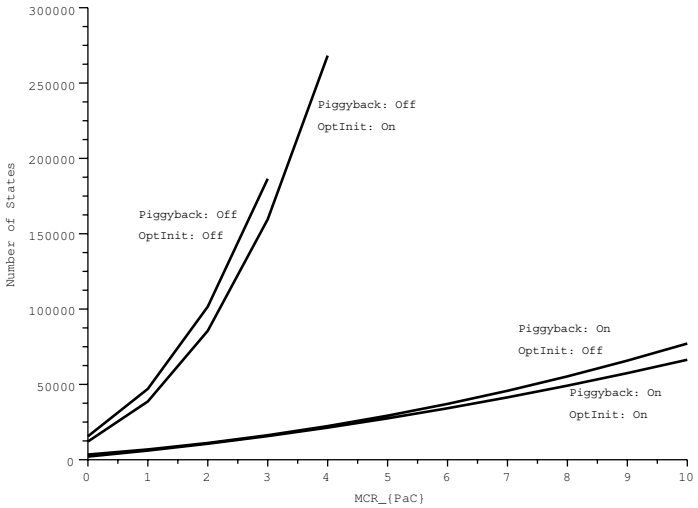


Figure 13: Impact of PaC retransmissions on state space size (in number of states)

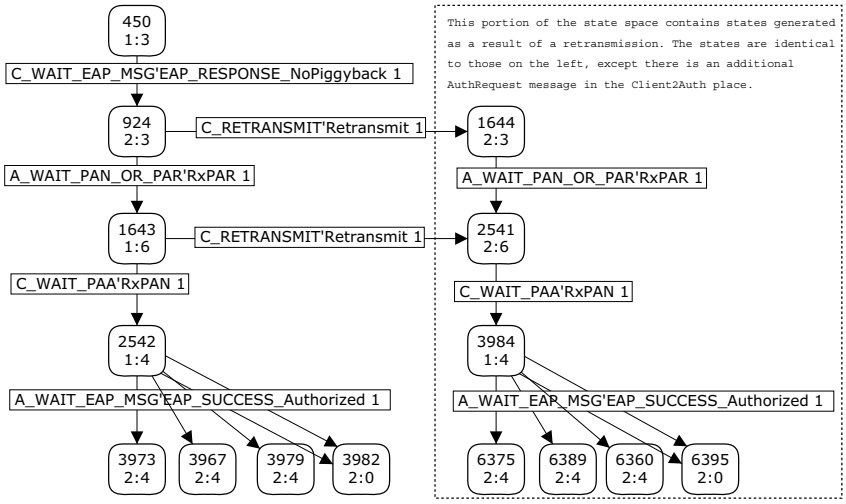


Figure 14: Partial state space showing effect of retransmission on state space size

In summary, the analysis with increasing MRC_{PaC} shows that retransmissions by the PaC do not adversely affect the operation of PANA. However, currently it is assumed only a single EAP Request is sent and messages cannot be lost. Further work is needed to analyse the impact of relaxing these assumptions.

5 Discussion

Steps from a protocol engineering methodology utilising Coloured Petri nets have been applied to PANA, a protocol for carrying authentication information between clients and servers. In previous work, initial CPN modelling and analysis of an earlier version of

PANA revealed an undesirable state when the authenticator aborted a session. This paper modelled and analysed the latest version of PANA, showing the undesirable state is now avoided. In addition, new PANA configurations have been analysed, in particular when Optimised Initiation is used by the authenticator, and when the client can retransmit messages. The results so far show no unexpected behaviour in the latest version of PANA.

The PANA CPN model has been developed over a period of about two years, over which time the PANA protocol specification and state tables have progressed from versions 14 to 18 and 5 to 13, respectively. The modelling, maintenance and analysis has amounted to 3-4 months of effort from a single person new to CPN Tools (but experienced with CPNs and Design/CPN). The following discusses lessons learned in the current work, as well as ideas for future work.

5.1 Modelling Approach

A state-based approach is used for modelling PANA as a CPN. The aim is for the CPN model to closely follow the PANA state tables, which is advantageous during the development of the protocol (and corresponding CPN model). In many cases there is a direct mapping from a state table row to a CPN transition. However the approach has limitations [3]. For example in the PANA CPN there are many transitions that model the same behaviour (but in different states) and could be folded together. In addition, by closely following the protocol state tables, little consideration is given in optimising the model for state space analysis.

Despite the close relationship between the original state tables and CPN model, as changes to PANA are made within IETF it is still time consuming to ensure those changes are accurately reflected in the CPN model. This was especially difficult with PANA, as there was an official PANA RFC with informal description of the protocol, as well as a separately maintained (and often out-of-date) Internet Draft for the state table description. With the IETF protocol descriptions, the difference between versions can be visually highlighted using *diff*-based tools. Similar functionality would be useful in CPN Tools: for example, highlighting CPN elements that have been changed since a previous model. Another method to assist in validating the PANA CPN model is to generate state tables directly from the model. Using the state-based modelling approach this is possible and is currently work-in-progress.

5.2 State Space Analysis

A property of interest for many communication protocols is the absence of deadlocks. With state space analysis of a CPN model, this requires specifying the expected terminal states. In this paper, the simplest possible definition of an expected terminal state is used: the state name of the PaC/PAA is expected (e.g. PaC and PAA both in the OPEN state). A more precise definition would also consider the state information stored by the PaC/PAA (e.g. value of session timer, current sequence number) as well as messages remaining in the communication places. Also, as only the Authentication and Authorisation phase of PANA is analysed in this work, the expected set of terminating conditions is quite large. As discussed in Section 4.2.1, there are 7 different valid states of the PaC/PAA, a number of which are valid only because it is expected in later phases a valid terminal state will be reached (because a time-out will occur). In the current CPN model the timeout in the subsequent phase is not explicitly modelled. This CPN model design decision was chosen to keep the analysis of PANA phases separate, however it leads to extra complexity when defining/analysing the terminal states. If all PANA phases were

analysed at once, the number of valid terminal states would be less (e.g. only the case when PaC/PAA are both CLOSED).

The state space results as the number of PaC retransmissions increases suggests, if the PANA service language is completed, parametric verification may be applicable to analyse PANA properties [10]. In addition, the CPN model can be optimised for state space analysis, and Figure 14 indicates equivalence classes may be promising as a state space reduction technique.

Acknowledgements

The detailed comments from the reviewers are highly appreciated, not only for improving this paper, but also for providing interesting ideas for future work.

References

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol. IETF RFC 3748, June 2004.

[2] J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri net approach to protocol verification. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, pages 210–290. Springer-Verlag, 2004.

[3] J. Billington and S. Vanit-Anunchai. Coloured Petri nets modelling of an evolving internet standard: the Datagram Congestion Control Protocol. *Fundamenta Informaticae*, 88(3):357–385, 2008.

[4] P. Chamuczynski, O. Alfandi, H. Brosenne, C. Werner, and D. Hogrefe. Enabling pervasiveness by seamless inter-domain handover: Performance study of PANA preauthentication. In *Proc. Sixth Annual IEEE Intl. Conf. Pervasive Computing and Communications*, pages 372–376, Hong Kong, China, 17–21 Mar. 2008.

[5] V. Fajardo, Y. Ohba, and S. Das. Network service provider selection and security bootstrapping using PANA. In *Proc. of the 2nd Intl Conf. Testbeds and Research Infrastructures for the Development of Networks and Communities*, Barcelona, Spain, 1–3 Mar. 2006.

[6] V. Fajardo, Y. Ohba, and R. Lopez. State machines for protocol for carrying authentication for network access (PANA). IETF Internet Draft draft-ietf-pana-statemachine-06 (work in progress), Oct. 2007.

[7] V. Fajardo, Y. Ohba, and R. Lopez. State machines for protocol for carrying authentication for network access (PANA). IETF RFC 5609, Aug. 2009.

[8] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin. Protocol for carrying authentication for network access (PANA). IETF RFC 5191, May 2008.

[9] B. Gaabab, D. Binet, and J.-M. Bonnin. Authentication optimization for seamless handovers. In *Proc. 10th IFIP/IEEE Intl. Symp. Integrated Network Management*, pages 829–832, Munich, Germany, 21–25 May 2007.

[10] G. E. Gallasch and J. Billington. A parametric state space for the analysis of the infinite class of stop-and-wait protocols. In *Proc. 13th Intl. SPIN Workshop on Model Checking of Software*, pages 201–218, Vienna, Austria, 30 March - 1 April 2006.

[11] G. E. Gallasch and J. Billington. Parametric language analysis of the class of stop-and-wait protocols. In *Proc. 29th Intl. Conf. Application and Theory of Petri Nets and Other Models of Concurrency*, Xi'an, China, 25-27 June 2008.

[12] S. Gordon. *Verification of the WAP Transaction layer using Coloured Petri nets.* PhD thesis, Institute for Telecommunications Research, University of South Australia, Adelaide, Australia, Nov. 2001.

[13] S. Gordon. Formal analysis of PANA authentication and authorisation protocol. In *Proc. Ninth Intl. Conf. Parallel and Distributed Computing, Applications and Technologies*, Dunedin, NZ, 1-4 Dec. 2008.

[14] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1997.

[15] K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *Intl. J. Software Tools for Technology Transfer*, 9(3-4):213–254, June 2007.

[16] L. Liu and J. Billington. Verification of the Capability Exchange Signalling protocol. *Intl. J. Software Tools for Technology Transfer*, 9(3-4):305–326, June 2007.

[17] P. S. Pagliusi and C. J. Mitchell. PANA/GSM authentication for Internet access. In *Proc. Joint First Workshop on Mobile Future and Symposium on Trends in Communications*, pages 146–152, Bratislava, Slovakia, 26–28 Oct. 2003.

[18] T. Tanizawa, M. Goto, V. I. Fajardo, and Y. Ohba. A wireless LAN architecture using PANA for secure network selection. In *Proc. IEEE Intl. Conf. Wireless And Mobile Computing, Networking And Communications*, pages 111–118, Montreal, Canada, 22–24 Aug. 2005.

[19] S. Vanit-Anunchai. Towards formal modelling and analysis of SCTP connection management. In *Proc. Ninth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Aarhus, Denmark, 20-22 Oct. 2008.

[20] J. Vollbrecht, P. Eronen, N. Petroni, and Y. Ohba. State machines for Extensible Authentication Protocol (EAP) peer and authenticator. IETF RFC 4137, Aug. 2005.

[21] M. Westergaard. BRITNeY: Basic Real-time Interactive Tool for Net-based animation. URL: http://wiki.daimi.au.dk/britney/, Sept. 2006.