Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 99 (2004) 245–266
www.elsevier.com/locate/entcs

Automatic Analysis of the NRL Pump ⋆

Ruggero Lanotte[a], Andrea Maggiolo-Schettini[b], Simone Tini[a], Angelo Troina[b] and Enrico Tronci[c]

[a] *Dipartimento di Scienze della Cultura, Politiche e dell'Informazione, Università dell'Insubria, Via Valleggio 11, 22100 Como, Italy*

[b] *Dipartimento di Informatica, Università di Pisa, Via Buonarroti 2, 56127 Pisa, Italy*

[c] *Dipartimento di Informatica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy*

Abstract

We define a probabilistic model for the NRL Pump and using FHP-murφ show experimentally that there exists a *probabilistic* covert channel whose capacity depends on various NRL Pump parameters (e.g. buffer size, number of samples in the moving average, etc).

1 Introduction

A computer system may store and process information with a range of classification levels and provide services to users with a range of clearances. The system is said to be *multilevel secure* [3] if users have access to information classified at or below their clearance, and are prevented from accessing information classified above their clearance.

In a distributed framework, multilevel security can be achieved by using multilevel secure components to connect single-level systems at different security levels, thus creating a *multiple single-level security* architecture [10,11].

The role of multilevel secure components is to minimize leaks of high level information from high level systems to lower level systems. A proposal for this

⋆ Research partially supported by progetto cofinanziato "Metodi Formali per la Sicurezza e il Tempo" (MEFISTO)

1571-0661/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.entcs.2004.02.011

component is the NRL Pump [13,14,12], which is a trusted device that acts as a router forwarding messages from a low-level system to a high-level system, by monitoring the timing of the acknowledgments in the opposite way. More precisely, the NRL Pump lets information be securely sent from a system at a lower security level to one at a higher security level. Now, if the high system passed acknowledgments directly to the low system, then, the high system, by altering the acknowledgment timing, could pass information to the low system. To avoid such *covert channel*, while maintaining acknowledgments to have a reliable communication, the pump decouples the acknowledgment stream. As, however, for performance reasons, the long-term high-system-to-pump behavior should be reflected in the long-term low-system-to-pump behavior, the pump uses statistically modulated acknowledgments.

In [12] a NRL Pump assurance strategy is proposed. A logical view of the pump is refined using a combination of Statemate activity charts and state charts. The activities and behavior of the logical design are mapped to an implementation described in terms of Statemate module charts. A Verdi specification expresses the requirements to which the implementation must be shown to conform. Verification is done through proof using the EVES verification system, and by testing using WhiteBoxDeepCover tool.

In this paper we model the NRL Pump by Probabilistic Timed Automata [2,15,1]. These automata and requirements on their behavior are translated into specifications for the FHP-murφ tool [5,6,17]. Using this tool, we show experimentally that there exists a probabilistic covert channel whose capacity depends on various NRL Pump parameters (e.g. buffer size, number of samples in the high-system-to-pump moving average, number of samples in the pump-to-low-system moving average, ...).

In section 2 we describe the NRL Pump and its security problem. In section 3 we introduce Probabilistic Timed Automata. In section 4 we use the automata to model the NRL Pump. In section 5 we discuss the NRL Pump probabilistic ack delay modeling. In section 6 we give the FHP-Murφ model of the NRL Pump. In section 7 we show our experimental results.

2 The NRL Pump

The NRL Pump is a special purpose-device that forwards data from low level agents to high level agents, but not conversely. More precisely, the pump works as follows:

(i) A low agent sends a message to some high agent through the pump.

(ii) The pump stores the message in a buffer and sends an acknowledgment to the low agent, in order to make the communication reliable. The time

of the acknowledgment is probabilistically based on a moving average of acknowledgment times from the high agents to the pump. In this way, the high agent cannot alter the acknowledgment timing to send information to the low agent. Moreover, the long-term high-agents-to-pump behavior is reflected by the long-term pump-to-low-agents behavior, so that performance is not degraded.

(iii) The low agent cannot send any new message until the acknowledgment of the previous message is received.

(iv) The pump stores the message until the high agent is able to receive it.

(v) The high agent receives the message and, then, sends an acknowledgment to the pump.

(vi) If the high agent does not acknowledge some received message before a fixed timeout expires[1], the pump stops the communication.

The validation protocol that models the initialization of the communication between the pump and the low and high systems is defined as follows (LS represents the low system, P represents the pump, HS represents the high system and $A \rightarrow B : msg$ represents the message msg sent from A to B):

$LS \rightarrow P : req_L$	the low system requests to the pump to start a communication with the high system
$P \rightarrow LS : valid_L$	the pump checks if the low system is a valid process, and, then, it acknowledges its request
$P \rightarrow HS : req_H$	the pump requests to the high system to start a communication with the low system
$HS \rightarrow P : valid_H$	the high system checks if the pump is a valid process, and, then, it acknowledges its request
$P \rightarrow HS : grant_H$	the pump communicates to the high system that the communication can start
$P \rightarrow LS : grant_L$	the pump communicates to the low system that the communication can start

[1] Such a timeout is defined by the pump-administrator.

Data communication can be modeled as follows:

$LS \rightarrow P : send_L$ the low system sends to the pump data to deliver to the high system

$P \rightarrow LS : ack_L$ the pump acknowledges to the low system with a random delay

$P \rightarrow HS : send_H$ the pump sends the data to the high system

$HS \rightarrow P : ack_H$ the high system acknowledges to the pump

When the low system receives the ack related to the last message, it ends the communication by sending a closure message:

$$LS \rightarrow P : close$$

When the pump does not receive any ack from the high system before the expiration of the timeout, it closes the connection by sending to the low system an exit message:

$$P \rightarrow LS : exit$$

When the low system tries to restore a communication stopped by an exit message from the pump, the pump forgets all the messages in the buffer undelivered to the high system.

3 Probabilistic Timed Automata

We give a definition of Probabilistic Timed Automata which differs slightly from those in [2,15,1].

Let us assume a set X of *integer variables*, with a subset Y of variables called *clocks*. A *valuation* over X is a mapping $v : X \rightarrow \mathbb{Z}$ assigning natural values to clocks and integer values to variables in $X \setminus Y$. For a valuation v and a time value $t \in \mathbb{N}$, let $v+t$ denote the valuation such that $(v+t)(x) = v(x)$, for each integer variable $x \in X \setminus Y$, and $(v+t)(y) = v(y)+t$, for each clock $y \in Y$.

The set of *constraints* over X, denoted $\Phi(X)$, is defined by the following grammar, where ϕ ranges over $\Phi(X)$, $x \in X$, $c \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \neq, >, \geq\}$:

$$\phi ::= x \sim c \,|\, \phi \wedge \phi \,|\, \neg\phi \,|\, \phi \vee \phi \,|\, true$$

We write $v \models \phi$ when *the valuation v satisfies the constraint ϕ*. Formally, $v \models x \sim c$ iff $v(x) \sim c$, $v \models \phi_1 \wedge \phi_2$ iff $v \models \phi_1$ and $v \models \phi_2$, $v \models \neg\phi$ iff $v \not\models \phi$,

$e_0 = (q_0, b, true, \emptyset, q_1)$ $\quad \pi_{q_0,b}(e_0) = 1$

$e_1 = (q_0, a, true, \emptyset, q_2)$ $\quad \pi_{q_0,a}(e_1) = \frac{1}{3}$

$e_2 = (q_0, a, true, \emptyset, q_3)$ $\quad \pi_{q_0,a}(e_2) = \frac{2}{3}$

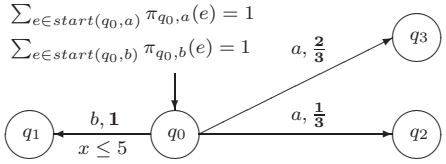

Fig. 1. Example of Probabilistic Timed Automaton

$v \models \phi_1 \vee \phi_2$ iff $v \models \phi_1$ or $v \models \phi_2$, and $v \models true$.

An *assignment* over X is a set of expressions either of the form $x' = c$ or of the form $x' = y + c$, where $x, y \in X$ and $c \in \mathbb{Z}$.

With $Ass(X)$ we denote the set of sets of assignments $\{x'_1 = e_1, \ldots, x'_n = e_n\}$ such that $x_i \in X$ and $x_i \neq x_j$, for any $i \neq j$.

Let $B \in Ass(X)$; with $v[B]$ we denote the valuation resulting after the assignments in B. More precisely, $v[B](x) = c$ if $x' = c$ is in B, $v[B](x) = v(y) + c$ if $x' = y + c$ is in B, and $v[B](x) = v(x)$, otherwise.

Definition 3.1 *A* Probabilistic Timed Automaton *(PTA for short) is a 6-tuple $A = (\Sigma, X, Q, q_0, \delta, \pi)$, where:*

- Σ *is a finite alphabet of* actions.
- X *is a finite set of* variables *with a subset Y of* clocks.
- Q *is a finite set of* states *and $q_0 \in Q$ is the* initial state.
- $\delta \subseteq Q \times \Sigma \cup \{\tau\} \times \Phi(X) \times Ass(X) \times Q$ *is a finite set of* transitions. *The symbol τ represents the silent or internal move. For a state q, we denote with $start(q)$ the set of transitions with q as source state, i.e. the set $\{(q_1, a, \phi, B, q_2) \in \delta \,|\, q_1 = q\}$; and we denote with $start(q, a)$ the set of transitions with q as source state and a as action, i.e. the set $\{(q_1, \alpha, \phi, B, q_2) \in \delta \,|\, q_1 = q$ and $\alpha = a\}$.*
- $\pi = \{\pi_{q,a} : start(q, a) \to (0, 1] \,|\, q \in Q, a \in \Sigma \cup \{\tau\}\}$ *is a family of* probability functions $\pi_{q,a}$ *such that $\pi_{q,a}(e)$ is the probability of performing the transition e in state q if action a is chosen. We require that, for each probability function $\pi_{q,a}$, it holds that $\sum_{e \in start(q,a)} \pi_{q,a}(e) \in \{0, 1\}$. Intuitively, an automaton chooses non-deterministically the kind of action it wants to take in a state, and then makes a probabilistic choice among the actions of that type it can perform in that state. See Figure 1 for an example.*

Let us explain now the behavior of a PTA A.

A *configuration* of A is a pair $s = (q, v)$, where $q \in Q$ is a state of A, and

v is a valuation over X.

The set of all the configurations of A is denoted with $\mathcal{S}_A$.

There is a *step* from a configuration $s_1 = (q_1, v_1)$ to a configuration $s_2 = (q_2, v_2)$ through action $a \in \Sigma \cup \{\tau\}$, after time $t \in \mathbb{N}$, and with probability p, written $s_1 \xrightarrow{(a,t,p)} s_2$, if there is a transition $e = (q_1, a, \phi, B, q_2) \in \delta$ such that $(v_1 + t) \models \phi$, $\pi_{q_1,a}(e) = p > 0$ and $v_2 = (v_1 + t)[B]$.

Given a configuration s, with $Adm(s)$ we denote the set of pairs (a, t) such that from s there is a step $s \xrightarrow{(a,t,p)} s'$ to some configuration s'.

A configuration $s = (q_i, v_i)$ is called *terminal* iff $Adm(s) = \emptyset$; we denote with S_t the set of the terminal configurations.

An *execution fragment* starting from s_0 is a finite sequence of steps $\sigma = s_0 \xrightarrow{(a_1,t_1,p_1)} s_1 \xrightarrow{(a_2,t_2,p_2)} s_2 \xrightarrow{(a_3,t_3,p_3)} \ldots \xrightarrow{(a_k,t_k,p_k)} s_k$. We define $last(\sigma) = s_k$, $|\sigma| = k$, and $step(\sigma, k) = (a_k, t_k)$. For any $j < k$, with σ^j we define the sequence of steps $s_0 \xrightarrow{(a_1,t_1,p_1)} s_1 \xrightarrow{\ldots} \xrightarrow{(a_j,t_j,p_j)} s_j$.

If $|\sigma| = 0$ we put $P(\sigma) = 1$, else, if $|\sigma| = k \geq 1$, we define $P(\sigma) = p_1 \cdot \ldots \cdot p_k$.

The execution fragment σ is called *maximal* iff $last(\sigma) \in S_t$. We denote with $ExecFrag(s)$ the set of execution fragments starting from s.

An *execution* is either a maximal execution fragment or an infinite sequence $s_0 \xrightarrow{(a_1,t_1,p_1)} s_1 \xrightarrow{(a_2,t_2,p_2)} \ldots$. We denote with $Exec(s)$ the set of executions starting from s. Finally, let $\sigma\uparrow$ denote the set of executions σ' such that $\sigma \leq_{prefix} \sigma'$, where *prefix* is the usual prefix relation over sequences.

Executions and execution fragments of a PTA arise by resolving both the nondeterministic and the probabilistic choices [15]. We introduce now *schedulers* of PTAs as functions that resolve all the nondeterministic choices of the model.

A *scheduler* of a PTA $A = (\Sigma, X, Q, q_0, \delta, \pi)$ is a function F mapping every execution fragment σ of A to a pair (a, t), where $a \in \Sigma$ and $t \in \mathbb{N}$, such that $(a, t) \in Adm(last(\sigma))$. With $\mathcal{F}$ we denote the set of all schedulers of A.

For a scheduler F of a PTA A we define $ExecFrag^F$ as the set of execution fragments σ such that $step(\sigma, k) = F(\sigma^j)$ for all $1 \leq j \leq |\sigma|$, and $Exec^F$ as the set of executions σ such that $step(\sigma, k) = F(\sigma^j)$ for all $1 \leq j \leq |\sigma|$ if σ is terminal, or for all $j \in \mathbb{N}$ if σ is infinite.

Assuming the basic notions of probability theory (see e.g. [8]) we define the probability space on the executions starting in a given configuration $s \in \mathcal{S}_A$ as follows. Given a scheduler F, let $Exec^F(s)$ be the set of executions starting in s, $ExecFrag^F(s)$ be the set of execution fragments starting in s, and $\Sigma_{Field}(s)$ be the smallest sigma field on $Exec^F(s)$ that contains the basic cylinders $\sigma\uparrow$,

where $\sigma \in ExecFrag^F(s)$.

The probability measure $Prob$ is the unique measure on $\Sigma_{Field}(s)$ such that $Prob(\sigma\uparrow) = P(\sigma)$.

3.1 Parallel composition

Given two PTAs A_1 and A_2 with the same integer variables X, we define the parallel composition of A_1 and A_2, denoted $A_1||^p_L A_2$, where $p \in [0,1]$ and $L \subseteq \Sigma$. The set of states of $A_1||^p_L A_2$ is given by the cartesian product of the states of the two automata A_1 and A_2. Intuitively, A_1 and A_2 synchronize on actions in L, and, if some action not in L is performed, then A_1 computes with probability p, and A_2 computes with probability $1-p$. Formally, given a state (r,q) of $A_1||^p_L A_2$, the set of transitions starting from (r,q) is obtained by the following rules:

- If from state r the PTA A_1 has a transition (r,a,ϕ,B,r') with action $a \notin$ Ł and probability p', and A_2 does not have in state q any transition with action a, then $A_1||^p_L A_2$ has a transition $((r,q),a,\phi,B,(r',q))$ with probability p'.
- If from state q the PTA A_2 has a transition (q,a,ϕ,B,q') with action $a \notin$ Ł and probability p', and A_1 does not have in state r any transition with action a, then $A_1||^p_L A_2$ has a transition $((r,q),a,\phi,B,(r,q'))$ with probability p'.
- If from state r the PTA A_1 has a transition (r,a,ϕ,B,r') with action $a \notin$ Ł and probability p', and also A_2 can perform in state q a transition with action a, then $A_1||^p_L A_2$ has a transition $((r,q),a,\phi,B,(r',q))$ with probability $p' \cdot p$.
- If from state q the PTA A_2 has a transition (q,a,ϕ,B,q') with action $a \notin$ Ł and probability p', and also A_1 can perform in state r a transition with action a, then $A_1||^p_L A_2$ has a transition $((r,q),a,\phi,B,(r,q'))$ with probability $p' \cdot (1-p)$.
- If from state r the PTA A_1 has a transition (r,a,ϕ_1,B_1,r') with action $a \in$ Ł and probability p', and from state q the PTA A_2 has a transition (q,a,ϕ_2,B_2,q') with probability p'' and $B_1 \cup B_2 \in Ass(X)$, then A_1 and A_2 synchronize and therefore $A_1||^p_L A_2$ has a transition $((r,q),a,\phi_1 \wedge \phi_2, B_1 \cup B_2,(r,' q'))$ with probability $p' \cdot p''$.

4 Modeling the NRL Pump with PTAs

In this section we show how the NRL Pump presented in Section 2 can be modeled by PTAs. We model the pump as the parallel composition of the PTAs $Pump_1$ and $Pump_2$ depicted in Figure 4. The Low and High Systems are depicted in Figures 2 and 3. For simplicity, we have omitted constraints

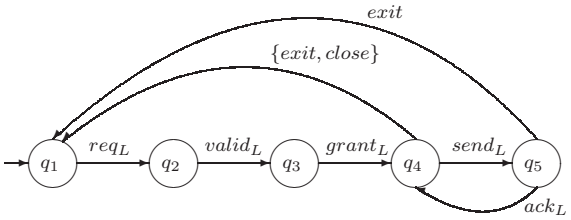

Fig. 2. LS: The Low System

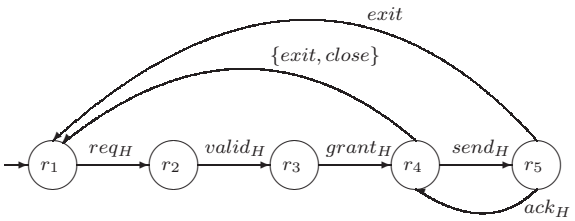

Fig. 3. HS: The High System

true and the probabilities when they are equal to 1.

The low system starts the communication by sending a request to the pump (req_L), waits for its validation ($valid_L$) and the start message ($grant_L$). Now, in state q_4, the low system starts sending its data to the pump ($send_L$) and receiving the related acknowledgment (ack_L). If the communication ends in a correct way, it stops the communication by sending a *close* message to the pump, if the communication ends in an incorrect way (for instance if the high system does not acknowledge to the pump), the pump stops the connection by executing an *exit* action.

The high system enters the communication process by receiving a request from the pump (req_H), sending its validation message ($valid_H$) and waiting for the start message ($grant_H$). Now, in state r_4, the high system starts receiving data from the pump ($send_H$) and sending back the related acknowledgment (ack_H). If the communication ends in a correct way, it receives the *close* message from the low system through the pump, if the communication ends in an incorrect way (for instance if it does not acknowledge to the pump before the timeout expires), the pump stops the connection by executing an *exit* action.

In states $g_1, \ldots, g_7$ the automaton $Pump_1$ initializes the connection between the high and the low system.

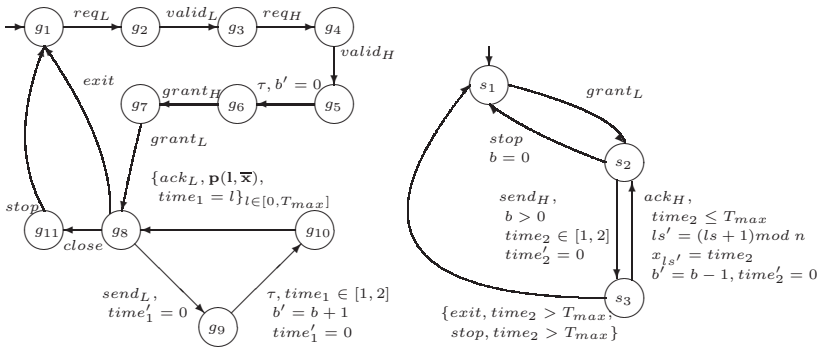

Fig. 4. $Pump_1$ and $Pump_2$

The variable b represents the number of messages in the buffer for the high system. In state g_5, before giving grants, the $Pump_1$ empties the buffer.

In state g_8 $Pump_1$ waits for a message from the low system. When the message is received, the automaton $Pump_1$ employs a time enclosed in $[1, 2]$ to store the message (transition for state g_9 to state g_{10}).

Now, $Pump_1$ must send an acknowledgment to the low system. The time in which the acknowledgment is sent is probabilistically computed. With $\overline{x}$ we denote the average time $\frac{\sum_{i=1}^{n} x_i}{n}$ of the last n acknowledgments sent by the high system, where the management of variables $\{x_1, \ldots, x_n\}$ is owned by $Pump_2$. We denote with $p(l, \overline{x})$ the probability that the NRL Pump sends an acknowledgment to the low system at the time l when the average acknowledgment delay time is $\overline{x}$. Of course $p(l, \overline{x})$ can be defined in many ways each of which may yield different performances as of security. We shall examine possible choices for $p(l, \overline{x})$ in Section 5. Moreover with the time T_{max} we denote the NRL Pump timeout.

When the time is expired, $Pump_1$ enters state g_1 and waits for a new connection. If the connection is closed by the low system, then $Pump_1$ waits that $Pump_2$ has finished its work.

Automaton $Pump_2$ starts its execution when the initialization is finished (symbol $grant_L$). $Pump_2$ can forward a message to the high system when at least one message is in the buffer (condition $b > 0$). After the message has been forwarded (symbol $send_H$), $Pump_2$ receives an acknowledgment from the high system (symbol ack_H). The time of this acknowledgment is stored in the variables $x_1 \ldots, x_n$ which are used as an array. The index ls represents the position in which the new time must be stored. If the time is over T_{max}, then the connection is expired. If the connection is closed by the low system and either the buffer is emptied (transition from s_2 to s_1) or the time is expired (transition from s_3 to s_1), then $Pump_2$ enters state s_1 and waits for a new

connection.

All components can perform actions with the same label. Hence, the whole system $PUMP$ is the system

$$PUMP = (Pump_1 ||^{0.5}_{\{grant_L, exit, stop\}} Pump_2).$$

The system including also the low system is

$$LS \text{ plus } PUMP = LS ||^{\frac{1}{3}}_{A} PUMP$$

where $A = \{exit, close, req_L, valid_L, grant_L\}$.

Finally, the system including also the High system is

$$LS \text{ plus } PUMP ||^{\frac{3}{4}}_{B} HS$$

where $B = \{exit, close, req_H, valid_H, grant_H\}$.

5 NRL Pump probabilistic ack delay modeling

The probability $p(l, \overline{x})$, as well as the range of l, may be defined in many ways, each of which yields a *probabilistic ack schema*. In the following we consider a few possible scenarios.

5.1 Uniform distribution

We may think of computing the NRL Pump ack delay $\mathbf{l}$ as follows: $\mathbf{l} = \overline{x} + \mathbf{d}$, where $\mathbf{d}$ is a uniformly distributed random variable with range $[-\Lambda, +\Lambda]$.

Since the expected value of $\mathbf{d}$ (notation $E(\mathbf{d})$) is 0, we have $E(\mathbf{l}) = \overline{x} + E(\mathbf{d}) = \overline{x}$, as required by the NRL Pump specification. As regards $p(l, \overline{x})$ we have: $p(l, \overline{x}) =$ **if** $(l \in [\overline{x} - \Lambda, \overline{x} + \Lambda])$ **then** $1/(2\Lambda + 1)$ **else** 0. Of course Λ must be chosen small enough so that $\overline{x} - \Lambda \geq 1$.

The drawback of the above approach is that, if the schema is known to the low and high systems, than the following *deterministic* covert channel can be established. To transmit bit b $(b = 0, 1)$ to the low system, the high system sends h consecutive ack's to the pump with delay H_b. If h is large enough, from the law of large numbers we know that we will have $\overline{x} \sim H_b$, and $l \in [H_b - \Lambda, H_b + \Lambda]$. Without loss of generality, let us assume $H_0 < H_1$. Then, whenever the low system *sees* an ack time $l \in [H_0 - \Lambda, H_1 - \Lambda)$ (resp. $l \in (H_0 + \Lambda, H_1 + \Lambda]$), the low system knows (with certainty) that a bit 0 (bit 1) has been transmitted from the high system. Of course, if the ack time is in the interval $[H_1 - \Lambda, H_0 + \Lambda]$ the low system does not know which bit is being

transmitted from the high system. However, if the high system is sending ack's with delay H_0 (H_1) and h is large enough then we know that (with high probability) the low system will observe an ack delay in $[H_0-\Lambda, H_1-\Lambda)$ (resp. $(H_0+\Lambda, H_1+\Lambda]$). Note that once the low system receives an ack with delay $[H_0-\Lambda, H_1-\Lambda)$ (resp. $(H_0+\Lambda, H_1+\Lambda]$) then it knows *with certainty* that the high system has sent a bit 0 (bit 1). On the other hand the fact that when the high system is sending ack's with delay H_0 (H_1) the low system will receive an ack with delay $[H_0-\Lambda, H_1-\Lambda)$ (resp. $(H_0+\Lambda, H_1+\Lambda]$) is highly likely, but not certain.

5.2 Binomial distribution

The deterministic covert channel described in the previous section arises since the range of l depends on $\overline{x}$. We overcome this problem by using a *binomial distribution*.

Let $p \in [0,1]$ and T be an integer such that $T \geq T_{max}$. Let $\mathbf{d}$ be a random variable taking integer values in the range $[0,T]$ with probabilities: $P(\mathbf{d}=k) = \begin{pmatrix} T \\ k \end{pmatrix} p^k(1-p)^{T-k}$. We note that the range of $\mathbf{d}$ does not depend on p. Let p be $(\overline{x}-1)/T$. Since $\mathbf{d}$ has a binomial distribution we have $E(\mathbf{d}) = T \cdot p = T \cdot \frac{\overline{x}-1}{T} = (\overline{x}-1)$ and $Var(\mathbf{d}) = T \cdot p(1-p) = (\overline{x}-1)(1-\frac{\overline{x}-1}{T})$. We can define the NRL Pump ack delay $\mathbf{l}$ as follows: $\mathbf{l} = \mathbf{d}+1$, so that $\mathbf{l}$ does not depend on $\overline{x}$. Then we have $E(\mathbf{l}) = \overline{x}$, as required from the NRL Pump specification, and $Var(\mathbf{l}) = Var(\mathbf{d}) = (\overline{x}-1)(1-\frac{\overline{x}-1}{T})$.

Since the range of $\mathbf{l}$ does not depend on $\overline{x}$, the covert channel that we had with the schema used in Section 5.1 does not exist. However the high system can send information to the low system as follows. To transmit bit b ($b=0,1$) to the low system, the high system sends h consecutive ack's to the pump with delay H_b. If h is large enough, from the law of large numbers we know that we will have $\overline{x} \sim H_b$. The low system can compute a moving average $\overline{y}$ of the last m ack delays from the NRL Pump. If m is large enough we have $\overline{x} \sim \overline{y}$. Then, by comparing $\overline{y}$ with H_0 and H_1, the low system can estimate (with arbitrarily low error probability) the bit value sent by the high system.

Note that in this case, unlike that in Section 5.1, the low system knows the bit sent from the high system only in a probabilistic sense. The error probability depends on many parameters. Here are some of them: T (range of $\mathbf{l}$), h (high system *sustain time*), n (number of samples in NRL Pump moving average), m (number of samples in low system moving average), B (size NRL Pump buffer).

To study how the error probability depends on the above parameters we

```
const   -- constant declarations (i.e. model parameters)
BUFFER_SIZE :    5;   -- size of pump buffer
NRL_WINDOW_SIZE: 5;   -- size of nrl pump sliding window
LS_WINDOW_SIZE:  5;   -- size of low system sliding window
INIT_NRL_AVG :   4.0; -- init. value of high-syst-to-pump mobile average
INIT_LS_AVG :    0.0; -- init. value of pump-to-low-syst mobile average
DELTA :          10;  -- maximal pump-to-low-system ack delay
HS_DELAY :       4.0; -- constant added to high-system-to-pump ack delay
DECISION_THR :   1.0; -- decision threshold
```

Fig. 5. Constants (model parameters)

will model the NRL Pump using the probabilistic model checker FHP-Murφ [5].

6 FHP-Murφ Model of the NRL Pump

FHP-Murφ (*Finite Horizon Probabilistic Murφ*) [5,6,17] is a modified version of the Murφ verifier [7,16]. FHP-Murφ allows us to define *Finite State/Discrete Time Markov Chains* and to automatically verify that the probability of reaching in *at most* k steps a given error state is below a given threshold.

In this Section we describe our FHP-Murφ model of the NRL Pump. We restrict our attention on the NRL Pump features involved in the probabilistic covert channel described in Section 5.2. Our goal here is to compute the error probability of the low system when it tries to estimate the bit value sent from the high system. This error probability is a function of h (i.e. the number of consecutive high system ack's to the pump with delay H_b, Section 5.2) and some system parameters such as number of samples in high-system-to-pump moving average (registered in a NRL Pump sliding window), number of samples in the pump-to-low-system moving average (registered in a low system sliding window), size of NRL Pump buffer.

FHP-Murφ syntax is the same as that of Murφ. FHP-Murφ comment lines start with `--`. FHP-Murφ can also use C-like comments.

Figure 5 shows constants (model parameters) used in our model. FHP-Murφ can use (finite precision) real numbers and uses a C-like Csyntax for them.

Figure 6 shows types (data structures) used in our model. As to be expected type definition `A : B` defines `A` to be an alias for `B`. The type `real(6, 99)` denotes finite precision real numbers with `6` decimal digits and a mantissa with absolute value not greater than `99`.

Figure 7 shows declarations for the global variables of our model. These variables define the state of our model and, therefore, define also the number

```
type  -- type definitions (i.e. data structures)
real_type : real(6,99); -- 6 decimal digits, |mantissa| <= 99
BufSizeType : 0 .. BUFFER_SIZE; -- interval [0, BUFFER_SIZE]
NrlWindow : 0 .. (NRL_WINDOW_SIZE - 1); -- interval
LSWindow : 0 .. (LS_WINDOW_SIZE - 1); -- interval
AckDelay : 0 .. DELTA;  -- range of pump-to-low-system ack delay
```

Fig. 6. Types (data structures)

```
var -- declarations of global variables (i.e. model state variables)
b : BufSizeType;        -- number of msgs in pump buffer
nrl_avg : real_type;     -- high-system-to-pump moving average
ls_avg : real_type;    -- pump-to-low-system moving average
nrl_delays : array[NrlWindow] of real_type;
      -- pump sliding window: last high-system-to-pump ack times
ls_delays : array[LSWindow] of real_type;
      -- low syst sliding window: last pump-to-low-system ack times
nrl_first_index : NrlWindow;     -- cursor nrl sliding window
ls_first_index : LSWindow; -- cursor low system sliding window
nrl_ack : real_type;   -- pump-to-low-system ack timer
hs_wait : real_type;   -- high-system-to-pump ack timer
ls_decision: 0 .. 1; -- 0: hs sent 0, 1: hs sent 1
ls_decision_state: 0 .. 2; -- 0: to be decided;
     -- 1: decision taken; 2: decision taken and state reset done
```

Fig. 7. Global variables (state variables)

of bytes needed to represent each state (76 bytes in our case). Variable `b` represents the number of messages in the buffer. Variable `nrl_avg` represents the average of the last delays for the acks received by the pump from the high system. These delays are saved in array `nrl_delays`, where index `nrl_first_index` points to the eldest one. Variable `ls_avg` represents the average of the last delays of the acks sent by the pump and received by the low system. These delays are saved in array `ls_delays`, where index `ls_first_index` points to the eldest one. Variable `nrl_ack` represents the timer used by the pump for sending the next ack to the low system. Once decided the value of the timer, at each step the `nrl_ack` variable is decreased by 1. The pump sends the ack to the low system when `nrl_ack` ≤ 0. Similarly, variable `hs_wait` is the timer used by the high system for sending acks to the pump. Variables `ls_decision` and `ls_decision_state` are used for modeling the transmission of a bit on the covert channel between the high system and the low system.

Murφ (and thus FHP-Murφ) programming language allows definition of functions and procedures using a Pascal-like syntax. Formal parameters declared "var" are passed by reference. Formals that are not declared "var" are passed by reference, but the function or procedure is not allowed to modify them.

```
procedure init();
begin
 b := 0;
 nrl_ack := 0;
 hs_wait := 0;
 nrl_avg := INIT_NRL_AVG;
 ls_avg := INIT_LS_AVG;
 nrl_first_index := 0;
 ls_first_index := 0;
 for i : NrlWindow do nrl_delays[i] := INIT_NRL_AVG; end;
 for i : LSWindow do ls_delays[i] := INIT_LS_AVG; end;
 ls_decision := 0;
 ls_decision_state := 0;
end;
```

Fig. 8. Function `init()` defines the initial state

Function `init()` (Figure 8) defines the initial state for our model (i.e. it defines the *initial distribution* for the Markov Chain defined by FHP-Murφ).

Each variable declared in Figure 7 holds the state of a finite state automaton. In principle the dynamics of such automata could be described using pictures. However this turns out to be quite complicated. For this reason we just describe such dynamics by giving the transition relation of the automata. Our model of the NRL Pump is organized as the synchronous parallel composition of many (i.e. those declared in Figure 7) automata. Each transition relation is defined using a procedure which compute the *next state* in the `var` formal parameters.

Function `nrlpump_ack()` (Figure 9) defines the transition relation for the automaton modeling the pump-to-low-system ack timer (`nrl_ack`). When the timer reaches 0, the low system gets an ack from the pump. If there is space in the buffer the low system sends a message, and the pump picks a delay `d` for sending an ack to such message. Delay `d` is an input of function `nrlpump_ack()`. The value of `d` is chosen probabilistically using FHP-Murφ rules.

Function `hs()` (Figure 10) defines the transition relation for the automaton modeling the high-system-to-pump ack timer (`hs_wait`).

Function `buffer()` (Figure 11) defines the transition function for the automaton modeling the NRL Pump buffer. As we can see in Figure 11, we have three cases: *i*) both variables `hs_wait` and `nrl_ack` are less than or equal to 0 (pump received ack from the high system and sent ack to the low system), at the same time the pump can send a message to the high system and receive a message from the low system; *ii*) only `hs_wait` is less than or equal to 0 (pump received ack from the high system), the pump can send a message to the high system; *iii*) only `nrl_ack` is less than or equal to 0 (pump sent an

```
procedure nrlpump_ack(Var nrl_ack_new : real_type; d : AckDelay); begin

-- ack timer expired, low system sends new msg, timer takes d
if  ((nrl_ack <= 0.0) & (b < BUFFER_SIZE))
then nrl_ack_new := d; return; endif;

-- ack timer not expired, waiting ack, timer decremented
if (nrl_ack > 0.0) then nrl_ack_new := nrl_ack - 1; return; endif;

end;
```

Fig. 9. Function `nrlpump_ack` defines pump-to-low-system ack times

```
procedure hs(Var hs_wait_new : real_type);
var last_index : NrlWindow;
begin

-- ack timer not expired, waiting ack, timer decremented
if (hs_wait > 0) then
hs_wait_new := hs_wait - 1; return; endif;  -- hs processing msg

-- ack timer expired, high syst receives new msg, timer takes HS_DELAY
if ((hs_wait <= 0.0) &  (b > 0))
then hs_wait_new := HS_DELAY; return;  endif;

-- ack timer expired, high system waiting for new msg
if ((hs_wait <= 0.0) & (b <= 0))
then hs_wait_new := -2.0; return; endif;

end;  -- hs()
```

Fig. 10. Function `hs` defines high-system-to-pump ack times

ack to the low system low system), the low system can send a message to the pump.

Function `nrlpump()` (Figure 12) updates the value of the moving average of the high system ack times. When the pump waits for the ack from the high system (`hs_wait` > 0) it updates the value of the last ack delay. When the `hs_wait` timer expires ($-1 \leq$ `hs_wait` ≤ 0), the pump updates the new average for the acks delays and its auxiliary variables.

Function `obs()` (Figure 13) defines the transition relation for the automaton modeling the low system computation to estimate the high system ack delay. Initially, the low system updates its information regarding the last received ack delay as done by the pump. When the high system wants to send a bit 0 (1) to the low system, it will use `HS_DELAY` (`HS_DELAY` $+ 2.0$) as ack time. The low system tries to estimate the high system ack time by computing a moving average of the NRL Pump to low system ack times. Such computation

```
procedure buffer(Var b_new : BufSizeType);  begin

if (((hs_wait <= 0.0) & (b > 0)) &
    ((nrl_ack <= 0.0) & (b < BUFFER_SIZE)))
then  return; endif; -- send and get at the same time, b does not change

if ((hs_wait <= 0.0) & (b > 0))
then b_new := b - 1; return; endif; -- high syst gets msg from buffer

if ((nrl_ack <= 0.0) & (b < BUFFER_SIZE))
then b_new := b + 1; return; endif;  -- low syst sends msg to buffer

end;
```

Fig. 11. Function `buffer` models the pump buffer

```
procedure nrlpump(Var avg_new : real_type);
var last_index : NrlWindow;
begin
last_index := (nrl_first_index + NRL_WINDOW_SIZE - 1)%NRL_WINDOW_SIZE;

-- high system processing message
if (hs_wait >  0)
then nrl_delays[last_index] := nrl_delays[last_index] + 1.0; endif;

-- high system sends ack to the pump
if ( (hs_wait >=  -1) & (hs_wait <= 0.0) )    then
avg_new := nrl_avg +
((nrl_delays[last_index] - nrl_delays[nrl_first_index])/NRL_WINDOW_SIZE);
nrl_first_index := (nrl_first_index + 1)%NRL_WINDOW_SIZE;
nrl_delays[(nrl_first_index + NRL_WINDOW_SIZE - 1)%NRL_WINDOW_SIZE] := 0;
endif

end;
```

Fig. 12. Function `nrlpump` updates value of moving average of high syst ack times

is carried out by function `obs()` which stores its estimation in the global state variable `ls_avg`. The low system goal is to decide if the high system sent a 0 or a 1 bit. This is done using the value held in `obs_avg`. More specifically, if `HS_DELAY` - 1.0 $<$ `ls_avg` $<$ `HS_DELAY` + 1.0 then `obs()` decides that the high system sent a bit 0; if `HS_DELAY` + 1.0 $<$ `ls_avg` $<$ `HS_DELAY` 3.0 then `obs()` decides that the high system sent a bit 1. To avoid being fooled by noise, `obs()` takes a decision only when `ls_avg` is stable `enough`, i.e. when the absolute value of the difference (stored in `lsdiff`) between the present value of `ls_avg` and the previous one is below a given threshold (`DECISION_THR`). As a result the observer may or may not take a decision which in turn may or may not be correct. By the law of large numbers, it is quite clear that waiting

```
procedure obs(d : AckDelay);
var ls_last_index : LSWindow;
var ackval : real_type;
var ls_old : real_type;
var lsdiff : real_type;
begin

if ((nrl_ack <= 0.0) & (b < BUFFER_SIZE)) then
ackval := d;
ls_last_index := (ls_first_index + LS_WINDOW_SIZE - 1)%LS_WINDOW_SIZE;
ls_delays[ls_last_index] := ackval;
ls_old := ls_avg;
ls_avg := ls_avg + ((ls_delays[ls_last_index] -
                     ls_delays[ls_first_index])/LS_WINDOW_SIZE);
ls_first_index := (ls_first_index + 1)%LS_WINDOW_SIZE;
ls_delays[(ls_first_index + LS_WINDOW_SIZE - 1)%LS_WINDOW_SIZE] := 0;

-- make decision
if (ls_decision_state = 0) then -- decision has not been taken yet
-- make decision only when ls_avg stable (i.e. lsdiff small)
lsdiff := fabs(ls_avg - ls_old);
if ( (lsdiff < DECISION_THR) & (HS_DELAY - 1.0 < ls_avg)  &
     (ls_avg < HS_DELAY + 1.0) )
then  -- decision taken
ls_decision := 0; ls_decision_state := 1; return; endif;

if ( (lsdiff < DECISION_THR) & (HS_DELAY + 1.0 < ls_avg)  &
     (ls_avg < HS_DELAY + 3.0) )
then -- decision taken
ls_decision := 1; ls_decision_state := 1; return; endif;

endif; endif; end;
```

Fig. 13. Function `obs` computes the low syst estimate of the high syst ack time

for a very long time before making a decision (e.g. by making `DECISION_THR` very small) the low system can be quite sure of making a correct decision. However, the more the observer waits for making a decision, the smaller the *bit-rate* of this *covert channel*. Our goal is to estimate the probability that the observer takes the correct decision within h time units.

Function `goto_stop_state()` (Figure 14) resets the NRL Pump state after a decision about the bit sent from the high system has been made by the low system. This function has nothing to do with the NRL Pump working. It is only used to easy our covert channel measures.

Function `main()` (Figure 15) updates the system state. Essentially `main()` triggers the computation of the *next state* of all the automata composing our model.

Function `prob_delay_bin(m, d)` (Figure 16) returns the probability that

```
-- reset all, but obs_decision_state
procedure goto_stop_state(); begin
init();
ls_decision_state := 2; -- decision taken and reset done
end;
```

Fig. 14. Function `goto_stop_state` resets system states after a decision on bit sent has been taken

```
procedure main(d : AckDelay);
Var b_new : BufSizeType;
Var nrl_ack_new : real_type;
Var hs_wait_new : real_type;
Var avg_new : real_type;
begin
if (ls_decision_state = 2)
then return; endif; -- decision  and state reset done

if (ls_decision_state = 0) then -- decision not taken yet
 b_new := b;
 nrl_ack_new := nrl_ack;
 hs_wait_new := hs_wait;
 avg_new := nrl_avg;
 buffer(b_new);
 nrlpump_ack(nrl_ack_new, d);
 obs(d);
 nrlpump(avg_new);
 hs(hs_wait_new);
 b := b_new;
 nrl_ack := nrl_ack_new;
 hs_wait := hs_wait_new;
 nrl_avg := avg_new;
else -- decision taken, reset state
 goto_stop_state();
endif;  end;
```

Fig. 15. Function `main` updates system state

the NRL Pump ack time is `d` when the NRL Pump moving average value is `m`. Function `prob_delay_bin(m, d)` implements a binomial distribution with average value `m` on the interval `[0, DELTA]`.

Figure 17 shows the definition of the initial state and of the probabilistic transition rules for our model of the NRL Pump.

The invariant shown in Figure 18 states that no decision has been taken. This invariant is false on states in which a decision has been taken. Thus the error probability returned by FHP-Murφ allows us to compute the probability $P_{\mathrm{dec}}(h)$ of making a decision within h time units.

The invariant shown in Figure 19 states that no decision or the correct decision has been taken. This invariant is false on states in which the wrong

```
function binomial(n : real_type; k : real_type) : real_type;
var result : real_type;
var nn, kk : real_type;
begin
result := 1; nn := n; kk := k;
while (kk >= 1) do
  result := result*(nn/kk);
  nn := nn - 1; kk := kk - 1;
endwhile;
return (result);
end;

function prob_delay_bin(m : real_type; d : AckDelay) : real_type;
var p : real_type;
begin
p := m/DELTA;
return ( binomial(DELTA, d)*pow(p, d)*pow(1 - p, DELTA - d) );
end;
```

Fig. 16. Function `prob_delay_bin()` updates high sys ack timer

```
-- define init state
startstate "initstate" init(); end;

-- define transition rules
ruleset d : AckDelay do
rule "time step"
prob_delay_bin(nrlavg, d) ==> begin main(d) end;
end;
```

Fig. 17. Startstate and transition rules for NRL pump model

```
invariant "no_decision_taken" 0.0
(ls_decision_state = 0);
```

Fig. 18. Invariant stating that no decision has been taken

```
invariant "no-dec_or_right-dec" 0.0
(ls_decision_state = 0) | ((ls_decision_state>0) & (ls_decision=0));
```

Fig. 19. Invariant stating that no decision or the correct decision has been taken

decision has been taken. Thus the error probability returned by FHP-Murφ allows us to compute the probability $P_{\text{wrong}}(h)$ of making a wrong decision within h time units.

Setting :	1	2
BUFFER_SIZE	2	5
NRL_WINDOW_SIZE	2	5
LS_WINDOW_SIZE	2	5
cpu time (sec)	40565.44	105866.59s

Table 1
NRL pump parameter settings and experiments CPU time.

7 Experimental Results

In this Section we show our experimental results.

We studied two issues: the probability of making a decision within h time units ($P_{\text{dec}}(h)$) and the probability of making the wrong decision within h time units ($P_{\text{wrong}}(h)$). From the above probabilities we can compute the probability of making the right decision within h time units ($P_{\text{right}}(h)$) as follows: $P_{\text{right}}(h) = P_{\text{dec}}(h)(1 - P_{\text{wrong}}(h))$.

We studied the above probabilities for various settings of our model parameters (defined in Figure 5). More precisely we changed the values of the following constants: `BUFFER_SIZE, NRL_WINDOW_SIZE, LS_WINDOW_SIZE`. In Table 1 we show some of the set of values we considered for the above constants. The last row of Table 1 gives the CPU time needed by FHP-Murφ to carry out the required analysis (on a 2GHz Intel Pentium PC with Linux OS and 500MB of RAM).

In all cases we considered the probability of making a wrong decision turns out to be identically 0. Thus $P_{\text{right}}(h) = P_{\text{dec}}(h)$.

Figure 20 shows $P_{\text{dec}}(h)$ (and thus $P_{\text{right}}(h)$) as a function of h for the parameter settings in Table 1.

As can be seen from Figure 20, within 10000 time units the low system with probability essentially 1 has decided in the right way the value of the bit sent by the high system.

Our time unit is about the time to transfer messages from/to the pump. Thus we may reasonably assume that a time unit is about 1ms. Then Figure 20 tells us that the high system can send bits to the low system at a rate of about 1 bit every 10 seconds, i.e. 0.1 bits/sec. This means that for many applications the NRL Pump can be considered *secure*, i.e. the covert channel has such a low capacity that it would take too long to the high system to send interesting messages to the low system. On the other hand it is not hard to conceive scenarios where also a few bits sent from the high system to the low system can be a security threat.

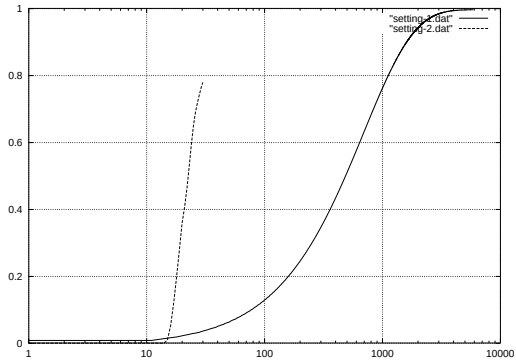

Fig. 20. Probability of taking a decision within h time units.

8 Conclusions

We have defined a probabilistic model for the NRL Pump and using FHP-murφ we have shown experimentally that there exists a *probabilistic* covert channel whose capacity depends on various NRL Pump parameters (e.g. buffer size, number of samples in the moving average, etc). The experiments show that the probabilistic covert channel exists from the high system to the low system at a rate such that the Pump can be considered secure for many applications.

References

[1] R. Alur, C. Courcoubetis, and D.L. Dill: Verifying Automata Specifications of Probabilistic Real-Time Systems. Proc. REX Workshop, Real-Time: Theory in Practice, Lecture Notes in Computer Science 600, Springer, Berlin, 1992, 28–44.

[2] D. Beauquier: On Probabilistic Timed Automata. Theoretical Computer Science 292, 2003, 65–84.

[3] D. Bell and L.J. La Padula: Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical report ESD-TR-75-301, MITRE MTR-2997, 1976.

[4] A. Bianco and L. de Alfaro: Model Checking of Probabilistic and Nondeterministic Systems. Proc. Int. Conf. on Foundation of Software Technologies and Theoretical Computer Science, Lecture Notes in Computer Science 1026, Springer, Berlin, 1995, 499–513.

[5] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M.V. Zilli: Finite Horizon Analysis of Markov Chains with the Murphi Verifier. Proc. IFIP WG 10.5 Advanced Research Working Conf. on Correct Hardware Design and Verification Methods, Lecture Notes in Computer Science 2841, Springer, Berlin, 2003, 58–71.

[6] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M.V. Zilli: Finite Horizon Analysis of Stochastic Systems with the Murphi Verifier. Proc. Italian Conf. on Theoretical Computer Science, Lecture Notes in Computer Science, Springer, Berlin, 2003.

[7] D.L. Dill, A.J. Drexler, A.J. Hu, and C.H. Yang: Protocol Verification as a Hardware Design Aid. Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors, IEEE Computer Society Press, Los Alamitos, California, 1992, 522–525.

[8] P.R. Halmos: Measure Theory. Springer, Berlin, 1950.

[9] H. Hansson and B. Jonsson: A Logic for Reasoning About Time and Probability. Formal Aspects of Computing 6, 1994, 512–535.

[10] M.H. Kang, J. Froscher, and I.S. Moskowitz: A Framework for MLS Interoperability. Proc. IEEE High Assurance Systems Engineering Workshop, IEEE Computer Society Press, Los Alamitos, California, 1996, 198–205.

[11] M.H. Kang, J. Froscher, and I.S. Moskowitz: An Architecture for Multilevel Security Interoperability. Proc. IEEE Computer Security Application Conf., IEEE Computer Society Press, Los Alamitos, California, 1997, 194–204.

[12] M.H. Kang, A.P. Moore, and I.S. Moskowitz: Design and Assurance Strategy for the NRL Pump. IEEE Computer 31, 1998, 56–64.

[13] M.H. Kang and I.S. Moskowitz: A Pump for Rapid, Reliable, Secure Communication. Proc. ACM Conf. on Computer and Communication Security, ACM Press, New York, 1993, 119–129.

[14] M.H. Kang, I.S. Moskowitz, and D. Lee: A Network Pump. IEEE Transactions on Software Engineering 22, 1996, 329–338.

[15] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston: Automatic Verification of Real-Time Systems with Discrete Probability Distribuitions. Proc. AMAST Workshop on Real-Time and Probabilistic Systems, Lecture Notes in Computer Science 1601, Springer, Berlin, 1999, 79-95.

[16] Murphi web page: `http://sprout.stanford.edu/dill/murphi.html`

[17] Cached Murphi web page: `http://www.dsi.uniroma1.it/~tronci/cached.murphi.html`