Nordic Journal of Computing

FAST Verification of the Class of Stop-and-Wait Protocols modelled by Coloured Petri Nets *

Jonathan Billington and Guy Edward Gallasch
Computer Systems Engineering Centre
School of Electrical and Information Engineering
University of South Australia
Mawson Lakes, SA, 5095, AUSTRALIA
Jonathan.Billington@unisa.edu.au
Guy.Gallasch@postgrads.unisa.edu.au

Laure Petrucci
LIPN, CNRS UMR 7030, Université Paris XIII
99, avenue Jean-Baptiste Clément
F-93430 Villetaneuse, FRANCE
petrucci@lipn.univ-paris13.fr

Abstract. Most protocols contain parameters, such as the maximum number of retransmissions in an error recovery protocol. These parameters are instantiated with values that depend on the operating environment of the protocol. We would therefore like our formal specification or model of the system to include these parameters symbolically, where in general each parameter will have an arbitrary upper limit. The inclusion of parameters produces infinite family of finite state systems, which makes verification difficult. However, techniques and tools are being developed for the verification of parametric and infinite state systems. We explore the use of one such tool, FAST, for automatically verifying several properties (such as channel bounds and the stop-and-wait property of alternating sends and receives) of the stop-and-wait class of protocols, where the maximum number of retransmissions and the maximum sequence number are considered as unbounded parameters. Coloured Petri nets (CPNs), an expressive language for representing protocols, are used to model this stop-and-wait class. However, FAST's foundation is counter systems, automata where states are a vector of non-negative integers and with operations limited to Presburger arithmetic. We therefore also present some first steps in transforming CPNs to counter systems in the context of stop-and-wait protocols operating over unbounded FIFO channels.

ACM CCS Categories and Subject Descriptors: C.2.2, C.4, F.1.1.

Key words: Protocols, Automated Parametric Verification, Coloured Petri Nets, Counter Systems, FAST

*This work is supported by the French-Australian Science and Technology programme, FR040062, an Australian Research Council Discovery Grant, DP0559927, and a University of South Australia Divisional Small Grant, SP04.

Received September 1, 2005;;

1. Introduction

1.1 Background and Motivation

The design and development of computer communication protocols is central to the development of embedded and pervasive computing systems which nearly always involve the co-operation of distributed components. It is important that protocols behave according to their requirements, since their failure can have serious consequences particularly for life critical or financially sensitive applications. Being able to verify that protocols behave correctly is a significant challenge since they usually include a number of parameters (such as a maximum sequence number, flow and congestion control window sizes, and the maximum number of retransmissions) that may be chosen to suit the operating environment, and may vary widely. Thus we would like to consider a class of protocols where the parameters can take any value within their range, and verify their correctness for all values of the parameters. Sometimes the ranges for these parameters are unbounded, giving rise to an infinite family of state spaces, one for each value of the parameter. At other times no limit may be placed on the value of a parameter (e.g. the number of times a packet can be retransmitted) which may result in an infinite state space.

The approach we use to tackle this problem is that of model driven development. The first step is to develop a formal model of our system which we then analyse either using tools or if they fail then by hand or a combination of both. The model is normally at the design level and the proofs are intended to show that the design satisfies the requirements of the system. This is rather important because removing errors at the design stage is very cost effective in the development of systems compared with removing errors in the implementation using testing. The effect is even more pronounced if the errors are discovered after the product has been released. The development of the model and its analysis can also increase the level of understanding of the requirements. Further, if the model is executable it can be used in fast prototyping of system specifications. This also increases the designer's and customer's understanding of requirements, which is widely acknowledged as a problematic area in software development.

In previous work [Billington *et al.* 2004] we summarised a protocol verification methodology based on Coloured Petri nets [Jensen 1997] and finite state automata [Hopcroft *et al.* 2001]. Coloured Petri Nets (CPNs) are an executable modelling language with a formal semantics, based on Petri nets and the ML functional programming language. The verification methodology uses state space methods and has been applied successfully for finite state systems, for small values of parameters. Techniques such as partial orders and Binary Decision Diagrams for alleviating the state explosion problem [Valmari 1998] (and more recently the sweepline method [Kristensen and Mailund 2002]) help to extend the method to larger ranges of parameters, but cannot handle large or unbounded values.

In [Billington *et al.* 2004], the methodology is illustrated using a stop-and-wait Protocol (SWP) [Tanenbaum 2003][Stallings 2004] which involves two parameters: the maximum sequence number, MaxSeqNb; and the maximum number of retransmissions, MaxRetrans. In general, the values of these parameters may be

chosen arbitrarily. We would thus like to prove that the SWP class is correct for any values of MaxSeqNb ≥ 1 and MaxRetrans ≥ 0. This becomes impossible using finite state techniques, as we need to consider an infinite number of increasingly larger finite state spaces. For FIFO channels (either lossy or lossless), a hand proof is given in [Billington *et al.* 2004] that shows that the number of messages in the message channel (and the number of acks in the acknowledgement channel) has a least upper bound of 2MaxRetrans + 1, for any positive value of MaxSeqNb, and any non-negative value of MaxRetrans. For other properties, such as verifying that the protocol conforms to its service of alternating send and receive events, the standard methodology was used for a range of parameter values ($0 <$ MaxSeqNb < 1024, $0 \leq$ MaxRetrans ≤ 4), but no general result was obtained. This has motivated us to search for methods that will handle unbounded parameters and provide some degree of automation.

This paper addresses the unbounded parameter problem by using a tool called FAST (Fast Acceleration of Symbolic Transition systems) [Bardin *et al.* 2003], based on counter systems [Finkel and Leroux 2002]. FAST performs symbolic analysis of infinite state systems by using *accelerations* (*meta-transitions*) to encode an arbitrary number of iterations of sequences of actions within the system. Parameters can be input as variables that are not constrained, and hence automated parametric verification of systems may be possible. However, we face two difficulties using this tool. Firstly, FAST's input language is based on counter systems (CS), whereas we would like to use the much more expressive language of CPNs. CS can model Place/Transition nets augmented with special arc types such as inhibitors [Esparza *et al.* 1999], but as far as we are aware no attempt has previously been made to translate CPNs to CS. Secondly, FAST provides a semi-algorithm, which is not guaranteed to terminate. Hence we can never be sure the verification will succeed.

The purpose of our work is thus to explore the potential of FAST for the parametric verification of communication protocols which have been previously modelled using CPNs. This paper investigates the class of stop-and-wait protocols. This is because they require parametric verification and are the simplest representative example of the class of protocols which provide flow control and bit error recovery that are used in practice, such as in the data link and transport layers of communication protocol architectures. We slightly revise our CPN model in [Billington *et al.* 2004] to make it easier to translate to a counter system. We find that translating CPN places representing states, stored sequence numbers and the retransmission counter is straightforward, but queues are more of a challenge. We are able to use 4 integer variables to represent the FIFO queue, due to the operation of the SWP. The conditions that are required for the queue model to be valid are checked using FAST, as well as the following properties: channel bounds; deadlocks; the stop-and-wait property; in-sequence delivery; and absence of message loss and duplication; for both lossless and lossy FIFO channels.

1.2 Related Work

The simplest SWP restricts its sequence numbers to 0 and 1 and is known as the Alternating Bit protocol (ABP) [Bartlett *et al.* 1969]. The ABP and its extensions

(e.g. [CCITT 1984]) have been used extensively in the literature as case studies (e.g. [Reisig 1998]). Often such papers demonstrate in various ways whether the ABP works as expected over (lossy or lossless) FIFO channels [Billington *et al.* 1988] [Suzuki 1990], investigate performance [Steggles and Kosiuczenko 1998][Marsan *et al.* 1994], demonstrate new tools [Billington *et al.* 1988], or illustrate verification methodologies [Diaz 1982], the application of formal description techniques [Turner 1993], new modelling languages or derivatives of existing languages [Suzuki 1990][Steggles and Kosiuczenko 1998][Marsan *et al.* 1994]. However, none of these papers address the issue of parametric verification of the ABP (i.e. for arbitrary values of MaxRetrans.)

More recently there has been work in the area of symbolic verification of the ABP. Valmari and his co-workers (e.g. [Valmari and Kokkarinen 1998]) use a behavioural fixed point method and compositional techniques for the verification of parametric systems. In [Valmari and Kokkarinen 1998] a variant of the ABP using limited retransmission, i.e. where there is an arbitrary bound (e.g. MaxRetrans) on the number of retransmissions, is verified using Valmari's CFFD equivalence. There are several differences with our work. Perhaps the most significant is that the channels are limited to holding only one message or acknowledgement at a time, whereas ours are unbounded FIFO queues. Valmari [Valmari and Kokkarinen 1998] considers this to be a more difficult problem. Valmari's method relies on defining a separate counter process which needs to be synchronised (using parallel composition) with the sender logic, which has 18 states. The counter itself is a recursive parallel composition of counter cells. The receiver is a relatively straightforward 6 state process. The ack channel is given as a 3 state process, but the data channel is more complex and not given explicitly in the paper. To obtain the model, all these processes need to be synchronised with parallel composition. In contrast our CPN model integrates all these aspects in the one model, and extends the model to include unbounded FIFO queues and sequence numbers with an arbitrary maximum sequence number as a parameter. However, our model does not have explicit communication with the users (but relies on the send and non-duplicate receive transitions to be considered as synchronised communication with the user) and does not consider reporting errors to the user. We see no problem in extending our model to include these features, however, our aim is to illustrate the use of FAST in analysing parameterised CPN models, rather than a direct comparison with a particular ABP variant.

The ABP and another variant called the Bounded Retransmission Protocol are used in [Abdulla *et al.* 2004] to demonstrate a symbolic verification methodology [Abdulla *et al.* 1999]. TReX (Tool for Reachability Analysis of Complex Systems) [trex 2003] was used to implement this methodology in [Abdulla *et al.* 2004]. The content of unbounded lossy FIFO channels is modelled by (a restricted class of) regular expressions thus providing a symbolic representation of the channels. TReX also uses an acceleration technique similar to FAST. This allows a small symbolic state space to be calculated based on the states of the sender and receiver ABP processes. They verify that the ABP conforms to its service of alternating sends and receives, using the Aldebaran tool [cadp 2005]. The maximum number of retransmissions was considered to be unlimited giving rise to a single, infinite-

state model. This differs from our approach of modelling MaxRetrans explicitly as a parameter and thus having an infinite number of finite-state models, one for each parameter value. As mentioned above, we also model an arbitrary maximum sequence number, rather than being limited to a maximum sequence number of 1.

Recently, a new approach [Gallasch and Billington 2005b] to parametric verification of the SWP has been introduced. An algebraic expression is obtained for the infinite family of state spaces for the SWP when MaxRetrans = 0. An induction proof is used to prove the expression is correct. Automata reduction techniques based on language equivalence are then used to prove that the SWP operating over lossy FIFO channels does conform to its service of alternating sends and receives. However, all proofs are done manually.

1.3 Contribution and Organisation

This paper[1] provides three contributions. Firstly, we believe it is the first time that automatic parametric verification of the stop-and-wait protocol class operating over unbounded (lossy) FIFO channels has been undertaken where MaxSeqNb has been included as a parameter. We are able to verify the SWP for arbitrary values of MaxRetrans for small values of MaxSeqNb (i.e. 1 to 5), and when there is no retransmission (MaxRetrans = 0) for arbitrary values of MaxSeqNb, for an extensive range of safety properties. Secondly, we confirm the validity of the algebraic expression (derived in [Gallasch and Billington 2005b]) for the states of the system when MaxRetrans = 0. Finally, we provide some insight into how CPNs may be translated into counter systems.

The rest of the paper is organised as follows. Section 2 describes the stop-and-wait protocol using a Coloured Petri net model. Counter Systems are introduced in Section 3 which also describes a methodology for translating a CPN model into a CS. This methodology is applied in Section 4 to the SWP CPN of Section 2. The expected properties of the SWP are described in Section 5. After introducing FAST, we analyse the SWP CS in Section 6. Section 7 provides concluding remarks and identifies areas of future work.

2. Stop-and-Wait Class of Protocols: A CPN Model

We explain the class of stop-and-wait protocols by providing a parameterised Coloured Petri Net (CPN) model of it as shown in Figs. 1 and 2, which were created using Design/CPN [CPN 2004].

2.1 Coloured Petri Nets

A CPN diagram such as in Fig. 1 comprises a bipartite directed graph, consisting of two types of nodes: places (ellipses) and transitions (rectangles). The names given to places and transitions are, usually, written inside the place or transition.

[1] This paper is a revised version of [Billington *et al.* 2005]. It provides further information on how to translate from the CPN model to its CS equivalent, and also includes new results for the case when MaxSeqNb is arbitrary and MaxRetrans = 0.

Directed arcs link places to transitions (*input* arcs) and transitions to places (*output* arcs). Places connected to transitions via incoming arcs are called input places and places connected to transitions via outgoing arcs are called output places.

Places may contain *tokens*. In CPNs tokens are arbitrarily complex data values. Accordingly, each place in a CPN must be *typed* by a non-empty set of data values, called a *colour set*. The colour set defines all possible data values that tokens residing on the corresponding place are able to take. The colour set is usually written in italics below the place, to the left or right as room permits. For example, in Fig. 1, the place sender_state is typed by the colour set Sender.

The *marking* of a place, denoted M(place name), is a *multiset* of tokens over the colour set of that place. Each place may be given an initial marking, denoted M_0(place name), written by convention above the place and usually on the same side as the colour set inscription. The sender_state place in Fig. 1 has an initial marking of a single s_ready token. This is interpreted as 1‘s_ready (read as “a multi-set comprising one token of colour s_ready”) but when the multiplicity of the token colour is 1, it is usual to omit the 1‘ in the inscription. The initial marking of the CPN is denoted M_0 and the set of all states (markings) of the CPN reachable from the initial state M_0 is denoted $[M_0\rangle$.

Arcs may be annotated with arbitrarily complex expressions involving constants, variables and functions, provided that the arc expressions evaluate to a multiset over the colour set of the place associated with the arc. For example, we can see that the arc from transition receive_mess to place recv_seq_no has an expression containing a conditional statement, variables sn and rn and a call to the function NextSeq (defined in Fig. 2). This expression will, for any valid *binding* of values to the variables sn and rn, always evaluate to a single token from the colour set Seq. Again we note that when the multiplicity of this token is one, the 1‘ is omitted from the graphical representation.

Each transition in a CPN has a boolean expression called a *guard* associated with it. The guard may use constants, any of the variables present in the arc expressions of the incoming arcs of the transition in question, and may also introduce new variables, local to the guard. The guard is usually written near the transition and enclosed in square brackets. For example, the transition timeout_retrans has the guard [rc < MaxRetrans] with the variable rc and the constant MaxRetrans as defined in Fig. 2. The default guard expression ([true]) is omitted from the graphical representation.

The execution of a CPN consists of a sequence of *occurrences* (or firings) of transitions. A transition can *occur* if it is *enabled*. For a particular marking of the CPN, a transition is enabled in a *mode*, determined by the binding of its variables, when the following conditions are true:

(1) Each input place of the transition contains at least the tokens obtained when its input arc expression is evaluated for the specific binding; and

(2) The guard of the transition evaluates to true for that binding.

Guards thus provide an additional constraint on the execution of the CPN.

When a transition occurs, the following two actions occur atomically:

(1) The multiset of tokens corresponding to the evaluation of each input arc expression is removed from the corresponding input place; and

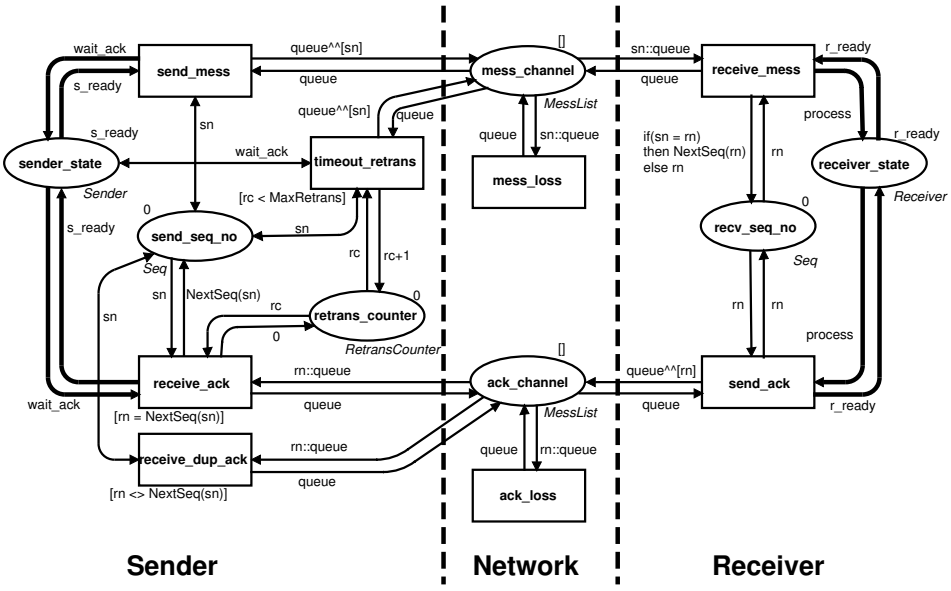

Fig. 1: CPN of the SWP operating over a lossy FIFO channel.

(2) The multiset of tokens corresponding to the evaluation of each output arc expression is added to the corresponding output place.

Transition modes may be *concurrently enabled* with one another, and with themselves, by a linear extension of the above enabling conditions and occurrence rules.

Every CPN model contains a set of declarations, comprising declarations for variables, and definitions for colour sets, functions and constants as illustrated in Fig. 2. Colour sets (types) are defined using the keyword color and a number of set constructors like with (for enumerated types), product and union. Variables are declared by the keyword var and are typed by a colour set, e.g. the variables sn and rn of type Seq (sequence number). Functions are defined using the keyword fun and values (e.g. constants) are defined using the keyword val.

2.2 The SWP CPN Model

Essentially three changes are made to the CPN model presented in [Billington and Gallasch 2004][Billington *et al.* 2004]:

- in the sender, one place (instead of two) is used to store its states, so that the colour set Sender comprises two states: s_ready and wait_ack;
- one place (receiver_state) is used in the receiver to store its states;
- a new place in the receiver stores its current sequence number; and
- arc inscriptions are revised accordingly.

This makes the CPN diagram more compact and provides a consistent modelling style. Control flow is indicated by bold arcs.

```
val MaxRetrans = 1;
val MaxSeqNb = 1;

color Sender = with s_ready | wait_ack;
color Receiver = with r_ready | process;
color Seq = int with 0..MaxSeqNb;
color RetransCounter = int with 0..MaxRetrans;
color Message = Seq;
color MessList = list Message;

var sn,rn      : Seq;
var rc         : RetransCounter;
var queue      : MessList;

fun NextSeq(n) = if(n = MaxSeqNb) then 0 else n+1;
```

Fig. 2: Global Declarations for the Stop-and-Wait Protocol CPN.

The protocol operates between a sender, shown on the left of Fig. 1 and a receiver shown on the right. The communication medium (Network) is represented by two lossy FIFO queues, one for each direction of message flow. The queues are modelled by using a list type for places mess_channel and ack_channel, adding messages to the end of the queue and removing messages from the head of the queue (the operator ^^ concatenates two lists while the operator :: appends an element to a list). Loss is represented by arbitrarily removing the head of the queue.

The protocol is implemented by the sender and receiver procedures. The sender has two states: s_ready and wait_ack, with the current state stored in place sender_state. When ready, it sends a message (transition send_mess) and waits for an acknowledgement before sending the next message (hence, stop-and-wait). To recover from the possibility of message loss, the sender sets a timer running on sending a message, and if it expires before receiving the acknowledgement (receive_ack), the message is retransmitted (timeout_retrans) and the timer is set running again. This works well if the message is lost. However, it is possible that the message has been received correctly but the acknowledgement is lost or delayed, causing retransmission of the original message, now a duplicate. Thus the receiver needs to detect and discard duplicate messages. To do this, a sequence number is included with each message. The sequence number space is finite, but allows for any range of consecutive integers, starting from zero. In our model we use the parameter MaxSeqNb to represent the maximum sequence number. Messages are represented by their sequence number only, as data is not used in the procedures. To detect duplicates both the sender (place send_seq_no) and the receiver (place recv_seq_no) store a sequence number, initially synchronised to zero. If a new message arrives at the receiver, its sequence number is the same as that stored in the receiver. The message is received (transition receive_mess with sn=rn) and the sequence number is incremented, modulo MaxSeqNb+1 (NextSeq(rn) see Fig. 2). Because the sequence number has been incremented in the receiver, duplicates will have a different number to that stored by the receiver (sn≠rn). In this case the sequence number remains the same. The sequence number in the receiver represents the next message to be

received, and this is the value that is sent back to the sender, as an acknowledgement. Acknowledgements are returned on receipt of each message, whether or not it is a duplicate. (This is required to recover from loss of acknowledgements.) To model flow control, we have two states in the receiver (r_ready and process), which allow the sending of the acknowledgement to be asynchronous with the receipt of a message. The current state is stored in place receiver_state.

While waiting for an acknowledgement, the sender may continue to retransmit messages, until it reaches a preset limit (MaxRetrans). It then gives up hope of the message getting through and passes control to a management entity for higher level recovery (not modelled). If an acknowledgement arrives before this, indicating that the message has been received (rn=NextSeq(sn)) transition receive_ack increments the send sequence number and returns the sender to ready, allowing the next message to be sent. Duplicate acknowledgements, which are generated by the receiver on receiving a duplicate message, are discarded by the sender (receive_dup_ack) at any time.

3. Mapping the CPN Model to a Counter System

As we aim at obtaining an extensive set of analysis results on the stop-and-wait protocol, parametric analysis is desirable. It can be achieved using tools such as FAST [Bardin *et al.* 2003]. FAST operates on counter systems (CS), so it is necessary to transform our CPN model into a CS. This is straightforward for Petri nets, even with extended arcs [Bérard and Fribourg 1999][Bardin and Petrucci 2004] but requires enhancement for CPNs.

Counter systems are automata extended with *unbounded integer variables*. FAST uses *accelerations* (sometimes called *meta-transitions*) to enable it to calculate the exact effect of iterating a behavioural loop an arbitrary number of times, and produces a symbolic occurrence graph representing the infinite state system. Details on counter systems and the theory behind FAST can be found in [Finkel and Leroux 2002][Boudet and Comon 1996][Wolper and Boigelot 2000][Leroux 2003].

The places of a CPN are transformed into a set of counter system variables and a single counter system state. This transformation is straightforward if the types of the places are or can be mapped to integers (e.g. enumerated types). If a place, p, has a type, $Type(p)$, that can be mapped to the natural numbers $\mathbb{N}$ by an injective mapping, $I_p : Type(p) \rightarrow \mathbb{N}$, and p always contains one token ($\forall M \in [M_0\rangle, |M(p)| = 1$), then we can create an integer variable, v_p, in the CS, that takes the values of the token in the place transformed by I_p for each marking. This is the case both for the places in the sender and those in the receiver of our SWP CPN model.

However, the stop-and-wait protocol uses two FIFO queues: one for messages and one for acknowledgements, represented by places mess_channel and ack_channel both typed by a message list, where messages are represented by sequence numbers (integers). These queues can be any size, depending on the maximum number of retransmissions [Billington *et al.* 2004]. The values of the sequence numbers depend on the MaxSeqNb parameter. Because the sequence numbers are integers

we can store the value of a queue item in a CS variable, and the number of queue items of that value in an associated CS variable. As long as sequence numbers do not wrap, we can always remove the item with the 'smallest' value from the queue and hence maintain FIFO order. However, this will require an unbounded number of variables in the general case, but not if the queue can only contain a finite number of values at any one time. For example, if the queue can contain only one message value at a time, then it can be represented by two variables: one storing the message value and a second storing the number of messages in the queue.

For the SWP operating over FIFO channels it turns out that there can be at most two different messages (represented by their sequence numbers) in the queue at any one time *and* that the messages of the same type are contiguous in the queue. Thus the queue is of the form mess1*mess2*. (Before doing the analysis, this property is a conjecture, so we must check that this property holds as a first step in validating the model.) Therefore, the queue can be modelled using four variables:

- Old is the smallest/oldest sequence number (modulo MaxSeqNb) that is in the queue;
- New is the latest sequence number that was put in the queue;
- NbOld is the number of messages with sequence number Old;
- NbNew is the number of messages with sequence number New.

Now, we will explain how to add messages to and remove messages from the queue. We will also show that this is done in a consistent manner.

The queue can contain:

(1) *no message*. Hence NbOld = NbNew = 0;

(2) *one type of message*. Then, Old = New and NbOld = NbNew $\neq$ 0;

(3) *two types of message*. Thus, Old $\neq$ New, NbOld $\neq$ 0 and NbNew $\neq$ 0.

In the following, a prime denotes the new value of the variable after an action has been performed. Variables that do not change are not mentioned.

Firstly, consider adding a message with sequence number SeqNb. If the queue is in state (numbered as above):

(1) The new message is the only one. Therefore, after adding the message: Old' = New' = SeqNb and NbOld' = NbNew' = 1. This is consistent with the above statement for a queue having a single message, hence containing only one type of message;

(2) The new message can be either:
   - of the same type as those already in the queue. Then, after adding the new message, we have: NbOld' = NbNew' = NbOld + 1(= NbNew + 1). This is consistent with the queue containing a single type of message;
   - of a new type. Thus, New' = SeqNb and NbNew' = 1. This is consistent with the queue now having two types of message.

(3) In this case, only a New message (i.e. a duplicate) can be added to the queue (we check this property when validating the model) and hence NbNew' = NbNew + 1. This is consistent with the queue containing exactly two types of message.

Now, we explain how to remove a message SeqNb. If the queue is in state:

(1) It is empty, so this case should never occur as there is nothing to consume;
(2) The message consumed is of the single type in the queue. Hence: SeqNb = Old = New and NbOld' = NbNew' = NbOld − 1 = NbNew − 1. Note that the resulting queue can either contain messages of the same single type or no message at all;
(3) The message consumed can be of type either New or Old. Both cases can be handled in a similar manner, but in this paper, the queues considered are FIFO. Therefore, the message consumed is the oldest in the queue, i.e. SeqNb = Old. Then two cases can be considered:
    - The message consumed is the last one of type Old, i.e. NbOld = 1. Then the resulting queue contains a single type of message, Old' = New and NbOld' = NbNew;
    - There is more than one message of type Old in the queue. Then, NbOld' = NbOld − 1.

4. The SWP CS Model

The CPN model of the stop-and-wait protocol can now be transformed into a counter system by application of the techniques from the previous section.

4.1 SWP CS Variables

We start by mapping the types of the places of the SWP CPN to CS variables. Our mapping technique relies on each non-queue place containing exactly one value in each reachable marking. Direct inspection of the CPN in Fig. 1 reveals this to be the case. For example, consider place sender_state. The marking of this place can only be changed by transitions send_mess, timeout_retrans and receive_ack. M_0(sender_state) = 1‘s_ready hence $|M_0(\text{sender\_state})| = 1$. The occurrence of these transitions just replaces one value by another (send_mess and receive_ack) or does not change the marking (timeout_retrans). Hence $|M(\text{sender\_state})| = 1$ for all $M \in [M_0\rangle$. This property also holds for the places send_seq_no, retrans_counter, receiver_state and recv_seq_no.

Table I shows the mapping to CS variables of the non-queue places sender_state, send_seq_no, retrans_counter, receiver_state and recv_seq_no. The columns give the place name, place type, the name of the corresponding CS variable and the mapping I_p from tokens to natural numbers.

The places mess_channel and ack_channel are modelled using four CS variables each, as described in Section 3. Place mess_channel is modelled using the CS variables MCOld, MCNew, NbMCOld and NbMCNew to represent, respectively, the sequence number of the oldest type of message in the channel, the sequence number of the newest type of message in the channel, and the numbers of both types of message. The place ack_channel is modelled similarly, using the CS variables ACOld, ACNew, NbACOld and NbACNew.

TABLE I: Mapping from Non-queue Place Types to CS Variables

Place p	Place Type $Type(p)$	CS Variable v_p	I_p	
			$t \in Type(p)$	$I_p(t)$
sender_state	Sender	SState	s_ready	1
			wait_ack	0
send_seq_no	Seq	SSeqNb	ss	ss
retrans_counter	RetransCounter	Retrans	rc	rc
receiver_state	Receiver	RState	r_ready	1
			process	0
recv_seq_no	Seq	RSeqNb	rs	rs

Two other variables are needed for the parameters of the system: the maximum sequence number MaxSeqNb and the maximum number of retransmissions MaxRetrans.

4.2 SWP CS Transitions

Each CPN transition is mapped to a *set* of CS transitions as a result of the more complex representation of the message and acknowledgement queues (four CS variables each) and the need to explicitly represent sequence number wrap in CS transitions. For example, the action of the send_mess transition must be differentiated according to the four cases highlighted in Section 3:

- The message channel is empty;
- The message channel contains only one type of message with the same sequence number as the message to send ($\mathsf{NbMCOld} > 0 \wedge \mathsf{MCOld} = \mathsf{MCNew} \wedge \mathsf{MCOld} = \mathsf{SSeqNb}$);
- The message channel contains only one type of message with a different sequence number to the message to send ($\mathsf{NbMCOld} > 0 \wedge \mathsf{MCOld} = \mathsf{MCNew} \wedge \mathsf{MCOld} \neq \mathsf{SSeqNb}$); or
- The message channel contains two types of message ($\mathsf{MCNew} \neq \mathsf{MCOld} \wedge \mathsf{MCNew} = \mathsf{SSeqNo}$).

Each case represents a different manipulation of the four CS variables representing the message channel. However, this can lead to the inclusion of CS transitions that turn out to be dead. For example, the send_mess transition will never occur in the second or fourth cases, hence only the first and third cases need to be mapped to CS transitions. These are the first two transitions (sendM1 and sendM2) listed in the excerpt in Fig. 3 described below. It is important to only include firable transitions in the model, to reduce the number of CS transitions, as the number of CS transitions can increase the computation time significantly. Part of the soundness property discussed in Section 5 and verified in Section 6 is to ensure the completeness of the CS model with respect to capturing all allowable behaviours of the CPN model.

```
model SWP {
 var SState, SSeqNb, Retrans, RState, RSeqNb, MCOld, MCNew, NbMCOld, NbMCNew,
    ACOld, ACNew, NbACOld, NbACNew, MaxSeqNb, MaxRetrans;
 states marking;

// send1: new message with no message in queue
 transition sendM1 := {
   from      := marking;
   to        := marking;
   guard     := SState=1 && NbMCOld=0;
   action    := SState'=0, MCNew'=SSeqNb, NbMCNew'=1, MCOld'=SSeqNb, NbMCOld'=1;
};
// send2: new message with old duplicates in the queue
 transition sendM2 := {
   from      := marking;
   to        := marking;
   guard     := SState=1 && !(NbMCOld=0);
   action    := SState'=0, MCNew'=SSeqNb, NbMCNew'=1; };
// receive duplicate message with seq nb MCNew != MCOld, NbMCOld = 1
 transition receiveM3 := {
   from      := marking;
   to        := marking;
   guard     := RState=1 && NbMCOld=1 && !(MCOld=MCNew) && !(MCOld=RSeqNb);
   action    := RState'=0, MCOld'=MCNew, NbMCOld'=NbMCNew; };
// sendAck: new ack for empty ack queue
 transition sendACK1 := {
   from      := marking;
   to        := marking;
   guard     := RState=0 && NbACOld=0;
   action    := RState'=1, ACOld'=RSeqNb, NbACOld'=1, ACNew'=RSeqNb, NbACNew'=1;};
// receive expected ack
 transition recack := {
   from      := marking;
   to        := marking;
   guard     := NbACOld>0 && ACOld=ACNew && SState=0 && ((SSeqNb=MaxSeqNb
                && ACOld=0) | | (SSeqNb<MaxSeqNb && ACOld=SSeqNb+1));
   action    := NbACOld'=NbACOld-1, NbACNew'=NbACNew-1, SState'=1, Retrans'=0,
                SSeqNb'=ACOld; };
...
}
```

Fig. 3: An Excerpt of the SWP CS Model.

4.3 The Stop-and-Wait protocol CS Model

Figure 3 shows an excerpt of the SWP CS model, illustrating FAST model input. It comprises the integer variables identified above, a dummy state, marking, of the counter system, and specifications of the transitions. Comments are prefixed by the symbol //. Each transition is described by its source and destination states (from and to fields), which in this model is always the state marking. A guard is associated with each transition, giving an enabling condition on the values of the variables.

The effect of the transition is given in action, describing how the values of variables are changed when the transition occurs. The symbol && indicates logical AND, || represents logical OR, ! indicates negation, and the prime is as defined in Section 3.

Consider the declaration of transition sendM1. It starts and ends in the counter system state marking. The guard means that the sender sends a message only if it is ready to do so (SState=1) and the case handled by this transition is when the message channel is empty (NbMCOld=0). When these conditions are met, the transition can be fired and the action occurs, leading to a state where the sender is waiting for an acknowledgement (SState'=0), and the queue, containing only one message, is updated (MCNew'=MCOld'=SSeqNb, NbMCNew'=NbMCOld'=1).

5. Required Properties of the CS Model

The model of the SWP should satisfy several properties, which are of two kinds: the properties ensuring that our translation from the CPN model to the counter system is sound, i.e. all the assumptions made are valid; and the properties that the protocol itself should satisfy.

5.1 Model Soundness

For the model to be sound, we need to verify the modelling assumptions. Our model is correct if both the message and acknowledgement channels: contain no more than two different types of message, where the 'type' of the message refers to its sequence number (i.e. Old and New from Section 3); and all messages of the same type are contiguous in the queue (i.e. the contents of the queue is of the form Old*New*). To verify this, we check that if there are already two types of message in the queue (i.e. Old and New), no enabled transition can add a message with a sequence number different from New.

We also verify the completeness of the model, i.e. that all the relevant actions of the CPN model are taken into account by CS transitions. Hence, all cases that are not explicitly described by the guards of the CS transitions can never occur. This is done by verifying that there is no reachable marking that would enable a CS transition with a guard specifying conditions on the channel content not explicitly described by any of the existing guards.

5.2 SWP Properties

We wish to prove the following SWP properties:

Consecutive sequence numbers If there are different types of message in a channel, they have consecutive numbers. Hence:

$$\mathsf{MCOld} \neq \mathsf{MCNew} \Rightarrow (\mathsf{MCNew} = \mathsf{MCOld} + 1 \vee (\mathsf{MCNew} = 0 \wedge \mathsf{MCOld} = \mathsf{MaxSeqNb}))$$

$$\mathsf{ACOld} \neq \mathsf{ACNew} \Rightarrow (\mathsf{ACNew} = \mathsf{ACOld} + 1 \vee (\mathsf{ACNew} = 0 \wedge \mathsf{ACOld} = \mathsf{MaxSeqNb}))$$

Number of messages in channels The least upper bounds for the number of messages in both channels, and the least upper bound on the total number of messages (i.e. messages plus acknowledgements) is 2MaxRetrans + 1. This is checked by counting the messages in the channels. The number of messages in the message channel is:

$$Nb\_Messages = \text{if MCOld} \neq \text{MCNew then NbMCOld + NbMCNew else NbMCOld}$$

Hence, for the message channel:

$$Nb\_Messages \leq 2\text{MaxRetrans} + 1$$

should hold over all reachable markings, but

$$Nb\_Messages \leq 2\text{MaxRetrans}$$

should not. Similarly for the acknowledgement channel. We can also check the bound on the number of messages (or acknowledgements) of a particular type that can exist in the message (or acknowledgement) channel:

$$\begin{aligned}\text{if MCOld} \neq \text{MCNew then NbMCOld} \leq \text{MaxRetrans} \wedge \text{NbMCNew} \leq \text{MaxRetrans} + 1\\ \text{else NbMCOld} \leq \text{MaxRetrans} + 1\end{aligned}$$

Stop-and-Wait Property A sent message is received before the next (new) message is sent and after receiving a message, a new message is sent before the next message is received (i.e. alternating send and receive events).

No data loss Each original message (or a retransmission) is eventually received, except for the last message in case the original plus all retransmissions were lost and the maximum number of retransmissions is reached.

No duplication When a duplicate message arrives, it is detected as such and discarded. No duplicate message is mistakenly accepted as a new one.

In-sequence delivery The messages are received in the order they are sent.

Deadlocks When using reliable channels, there should be no deadlock. When using unreliable channels, only expected deadlocks should exist, i.e. the maximum number of retransmissions is reached but the sender is stuck waiting for an acknowlegement, the receiver expects a message, and both message and acknowledgement channels are empty:

$$\begin{aligned}\text{Retrans} = \text{MaxRetrans}, \text{SState} = 0, \text{RState=1}\\ \text{MCOld} = \text{MCNew}, \text{NbMCOld} = \text{NbMCNew} = 0,\\ \text{ACOld} = \text{ACNew}, \text{NbACOld} = \text{NbACNew} = 0\end{aligned}$$

5.3 Instrumentation of the model

In order to check several of the properties, some instrumentation of the model is required. We add a variable SRprop, which is set to the sequence number plus 1,

when a new message is sent. When an expected message (i.e. not a duplicate) is received, this variable is set to 0. Checking the stop-and-wait property then amounts to verifying that there is no pending new message in the message channel when the sender is ready to send (a send cannot follow a send), and that the receiver cannot be ready to receive the next expected message when SRprop = 0 (a receive cannot follow a receive), i.e. no state such that:

$$\mathsf{SRprop} > 0 \wedge \mathsf{SState} = 1$$

and no state such that:

$$\mathsf{SRprop} = 0 \wedge \mathsf{RState} = 1 \wedge \mathsf{NbMCOId} > 0 \wedge \mathsf{MCOId} = \mathsf{RSeqNo}$$

When operating over a FIFO medium, because the stop-and-wait property holds (a new message can only be sent if the expected one was received) it follows that there is no loss of data (except possibly the last message as described in the previous subsection.)

To verify the no duplication property, we check that there is no state such that the receiver is ready to accept a new message with a sequence number other than that most recently sent by the sender, i.e. there is no state such that:

$$\mathsf{SRprop} = \mathsf{MCOId} + 1 \wedge \mathsf{RState} = 1 \wedge \mathsf{NbMCOId} > 0 \wedge \neg(\mathsf{MCOId} = \mathsf{RSeqNb})$$

When a duplicate is received the value of SRprop will be 0 (if no new message has yet been sent by the sender) or the sequence number plus 1 of the new message sent, which is different to the sequence number plus 1 of the duplicate being received.

Finally, to prove the in-sequence delivery property, we note that variable SRprop contains the number (plus one) of the last new message sent, and that it is not possible to receive an original message with a sequence number different from that of the last new sent message, i.e.:

$$\neg(\mathsf{SRprop} = \mathsf{MCOId} + 1) \wedge \mathsf{RState} = 1 \wedge \mathsf{NbMCOId} > 0 \wedge \mathsf{MCOId} = \mathsf{RSeqNb}$$

6. Analysis of SWP using FAST

6.1 Introduction to FAST

FAST [Bardin *et al.* 2003][Bardin and Petrucci 2004] is a tool dedicated to checking safety properties on counter systems. The main issue addressed by FAST is the symbolic computation of the (infinite) state space. Although FAST uses a semi-algorithm which is not guaranteed to terminate, experiments with its use on practical examples have been promising [Fast 2005].

6.1.1 Inputs and Outputs

FAST requires both a *model* and a *strategy* to be input for analysis. Outputs are messages indicating whether the system satisfies a property or not. The model input format was illustrated in Section 4 for an excerpt of our SWP model.

```
strategy SWP {
...
Region init := {SState=1 && SSeqNb=0 && Retrans=0 && MCOld=0
                && MCNew=0 && NbMCOld=0 && NbMCNew=0 && ...};

Region reach := post*(init, t, 2);

// Consecutive sequence numbers in Message channel
Region diffminmaxM := {(MCOld=MCNew) | | (MCNew=MCOld+1) | |
                       (MCOld=MaxSeqNb && MCNew=0)};

if (subSet(reach,diffminmaxM)) then
  print("Message channel consecutive seq numbers OK");
  else print("Message channel consecutive seq numbers NOK");
endif
...
}
```

Fig. 4: An Excerpt of the SWP CS Strategy.

The *strategy* is the sequence of computations to perform in order to check the validity of the system. The strategy language is a script language which operates on regions (sets of states), transitions and booleans. All the usual operators on sets are available and primitives to compute the reachability set (forward or backward) are provided. Checking a safety property involves declaring the initial states, computing the reachability set A, declaring the property to check (*good states*) B, and testing if $A \subseteq B$.

An excerpt of the SWP CS strategy, illustrating FAST strategy input, is shown in Fig. 4. Firstly, the region init is declared, which is used to describe the initial states. Then, the set of reachable states, reach, is computed from init, using the forward reachability function post*. Parameter t is the set of transitions to be used, i.e. here all transitions (the declaration of t is not included in the excerpt), and 2 is the length of cycles to accelerate. Region diffminmaxM characterises the set of states with consecutive sequence numbers in the message channel. If reach is a subset of diffminmaxM then the consecutive sequence numbers property is satisfied, otherwise it is not. An appropriate message is printed.

6.1.2 Architecture

The FAST computational engine can be used as a standalone application, or with a graphical user interface in a client-server architecture [Bardin and Petrucci 2004]:

- *the server* is the computation engine of FAST. It contains a Presburger library, the acceleration algorithm and the search heuristics;
- *the client* is a front-end which allows interaction with the server through a graphical user interface. This interface facilitates guided editing of models and strategies, with features such as pretty printing and predefined strategies. Once the computation starts, feedback is supplied through different measures and graphs (time elapsed, memory used, number of states, ...).

TABLE II: FAST Experimental Results with MaxRetrans as a Parameter

Channels	MaxSeqNb	Nb compositions	Nb accelerations	time (hh:mm:ss)	memory (MB)	Nb states
Reliable	1	324	70	00:11:38	28	88
Reliable	2	324	112	00:49:37	33	204
Reliable	3	324	116	00:51:54	40	201
Reliable	4	324	126	02:21:48	37	421
Reliable	5	324	150	02:31:18	37	448
Lossy	1	576	93	00:10:11	27	110
Lossy	2	576	144	00:57:12	31	259
Lossy	3	576	149	00:42:47	34	246
Lossy	4	576	249	03:51:25	43	554
Lossy	5	576	266	03:43:37	39	567

6.2 Analysis Results

Our aim is to investigate both the properties and structure of the state space. An attempt was made to analyse the SWP CS model from Section 4 using FAST with both MaxSeqNb and MaxRetrans as parameters. However, FAST ran out of memory, due either to the semi-algorithm not terminating, or a lack of available computer memory. The SWP CS model was thus analysed for the following restricted cases:

- Fixed values of MaxSeqNb (from 1 to 5) with MaxRetrans as a parameter; and
- MaxRetrans=0 with MaxSeqNb as a parameter.

Further, an additional experiment was carried out to prove automatically the form of the markings in the occurrence graph of the SWP CPN when MaxRetrans=0.

6.2.1 Fixed MaxSeqNb *with* MaxRetrans *as a Parameter*

With only MaxRetrans as a parameter, the SWP CS model was analysed with separate runs of FAST for fixed values of MaxSeqNb from 1 to 5. The results can be seen in Table II. The analysis was performed on the lossy channel model as well as on a model with reliable channels (where the loss transitions were removed).

The computation is done at a reasonable cost with respect to both time and memory usage, as shown by the experimental results in Table II. Column 1 in Table II indicates whether the channel is reliable or lossy. Column 2 gives the fixed value of MaxSeqNb for each run. The third and fourth columns give the number of transition compositions and acceleration compositions for each run. Columns 5 and 6 give the total computation time and the peak memory usage of FAST. Finally, column 7 gives the number of symbolic states at the end of computation.

The total computation time is divided into three steps (for technical details, see e.g. [Finkel and Leroux 2002]):

- transition compositions which take 1 minute 32 seconds in the reliable case (324 compositions) and 1 minute 58 seconds in the lossy case (576 compositions);
- accelerations computation which takes 1 minute in the reliable case and 41 seconds in the lossy case;
- applying the accelerations to construct the state space.

The computation time for compositions and accelerations are a constant for all 10 cases in Table II, as the same preliminary computations are performed. The results obtained from FAST confirm the expected properties from Section 5. When attempts were made to analyse the SWP CS model for values of MaxSeqNb > 5, FAST did not terminate in a reasonable amount of time.

6.2.2 *Fixed* MaxRetrans *with* MaxSeqNb *as a Parameter*

Analysis of the SWP CS model with MaxSeqNb as a parameter and MaxRetrans fixed to 0 was conducted next. The computation required the acceleration of cycles of length 4, which is consistent with a send_mess/receive_mess/send_ack/receive_ack loop. It terminated within 35 minutes. All properties were valid and thus proved regardless of the value of the maximum sequence number. The results also showed that some transitions were never enabled, which is consistent with having no retransmissions. Also, it was proved that there is at most 1 message in each queue at any one time. This means that for MaxRetrans $= 0$ we do not need to distinguish MCOld and MCNew, ACOld and ACNew, NbMCOld and NbMCNew, nor NbACOld and NbACNew. The variable Retrans is no longer needed as no retransmissions can occur.

6.2.3 *Automated Proof of the Form of the Markings of the SWP CPN*

The third experiment aimed to prove automatically the form of the markings in the occurrence graph of the SWP CPN when MaxRetrans $= 0$. This was manually proved in Gallasch and Billington [2005a, 2005b], and the different possible markings are described in Table III. They are grouped into 6 different sets, one per row of the table. Columns 2 through 8 of the table describe the values of the relevant CS variables. The variables Retrans, MCNew, NbMCNew, ACNew and NbACNew are not needed (as MaxRetrans=0) and the value of RSeqNb is specified in terms of SSeqNb.

TABLE III: Occurrence graph markings with MaxRetrans $= 0$

Set	SState	NbMCOld	MCOld	NbACOld	ACOld	RState	RSeqNb
1	1	0	–	0	–	1	SSeqNb
2	0	1	SSeqNb	0	–	1	SSeqNb
3	0	0	–	0	–	1	SSeqNb
4	0	0	–	0	–	0	SSeqNb $\oplus$ 1
5	0	0	–	1	RSeqNb	1	SSeqNb $\oplus$ 1
6	0	0	–	0	–	1	SSeqNb $\oplus$ 1

The symbol $\oplus$ denotes modulo MaxSeqNb+1 addition. Each row can be evaluated for each sender sequence number, $0 \leq \mathsf{SSeqNb} \leq \mathsf{MaxSeqNb}$ to obtain the set of markings for that particular SSeqNb. The union of all 6 rows, when evaluated over $0 \leq \mathsf{SSeqNb} \leq \mathsf{MaxSeqNb}$, is the full state space.

A predicate is created from Table III by deriving 6 regions, one for each of the 6 sets of markings defined in Table III, characterising the markings in each of the 6 sets. To check efficiently that the full state space is exactly the union of these sets of markings, we took advantage of the properties previously proved to reduce the size of the model. Specifically, the transitions that are never enabled were removed from the model. Since there is no retransmission, variable Retrans was removed. As there is at most one message in the channels, the variables NbMCNew, MCNew, NbACNew and ACNew were eliminated. Variable SRprop, which was used only as an instrument for proving particular properties, was also no longer needed. When using this reduced model, we proved *automatically* that the algebraic expression representing the states of the occurrence graph of the SWP CPN is correct, in less than 35 seconds.

7. Conclusions and Future Work

Finite state methods for protocol verification can fail due to state explosion when considering ranges of values for important parameters such as the maximum number of retransmissions or the size of the sequence number space. When considering these parameters, we would like to provide a general result that allows protocol properties to be proved for any value of each parameter. When arbitrary values are considered, we need to generate an infinite number of finite state spaces, one for each value of the parameter. (This is quite different from considering, for example, the specific case of no limit on the number of retransmissions, which gives rise to a single infinite state system.)

This paper has addressed this problem for the stop-and-wait class of protocols, where we modelled the parameters explicitly. We used a recently developed tool called FAST to facilitate parametric verification. FAST allows symbolic state spaces to be generated by taking advantage of encoding arbitrary iterations of sequences of events, known as accelerations. It is based on counter systems, which are automata where states are vectors of (unbounded) integers.

The stop-and-wait protocol (SWP) has two parameters: MaxRetrans representing the maximum number of retransmissions; and MaxSeqNb representing the maximum sequence number that can be used. In previous work [Billington *et al.* 2004] we modelled the SWP using Coloured Petri Nets and provided a hand proof that the bound on the number of messages in the FIFO communication channel was 2 MaxRetrans + 1. However, we were only able to prove other properties, such as the stop-and-wait property of alternating sends and receives, for up to 10 bit sequence numbers and with up to 4 retransmissions using automated finite state techniques.

In this paper we have overcome these limitations for the MaxRetrans parameter. Fully automatic proofs have been obtained for channel bounds (confirming the

previous hand proofs and including proving that the sum of the messages and acknowledgements in the channels does not exceed 2 MaxRetrans + 1), the stop-and-wait property, that there is no loss of messages (except for the last one when the maximum number of retransmissions is reached), no duplication and that messages are delivered in-sequence. This has been done for $1 \leq \text{MaxSeqNb} \leq 5$ by considering acceleration of cycles of length 2. Unfortunately, FAST does not terminate in a reasonable amount of time when both MaxSeqNb and MaxRetrans are considered as unbounded parameters, or for values of MaxSeqNb greater than 5.

We also considered the situation when the value of MaxRetrans was set to 0 and MaxSeqNb was a parameter. FAST was able to terminate when considering accelerations of cycles of length 4. In addition, we have used FAST to automatically confirm hand proofs of algebraic expressions for the occurrence graph of our CPN model when $\text{MaxRetrans} = 0$. However FAST did not terminate for $\text{MaxRetrans} \geq 1$. The authors suspect that memory constraints imposed by the MONA library [MONA 2004] used by FAST, which limits the size of the automata handled to 2^{24} nodes, is the cause of the non-termination problems encountered.

Further we have shown how to translate our CPN model into a counter system by using a novel approach to represent a FIFO queue by 4 integer variables. This is valid when the queue can hold only two types of message indicated by their sequence numbers and all messages of the same sequence number are adjacent. This condition is proved using FAST as part of model validation. Some general guidance has also been given for translating CPNs to counter systems.

Future challenges include generalising the method to channels that allow reordering of messages and formally incorporating data independence, which has been assumed in our work so far. Other ways of representing queues that are efficient and suit the FAST framework of Presburger arithmetic could be investigated. A more general and formal translation of CPNs into counter systems is also of interest, to allow models that have already been constructed in CPNs to be automatically translated and input to FAST. Automatically translating the properties formulated on the CPN model to those on the counter system and translating the results back is also an interesting issue. We would also like to investigate the use of other tools such as TReX and compare them with FAST.

Acknowledgements

We are grateful to Antti Valmari for fruitful discussions regarding the contents of this paper. We also thank the anonymous referees for their useful comments which helped to improve the paper.

References

ABDULLA, P. AZIZ, ANNICHINI, A., AND BOUAJJANI, A. 1999. Symbolic Verification of Lossy Channel Systems: Application to the Bounded Retransmission Protocol. In *Proceedings of TACAS'99*, Volume 1579 of *Lecture Notes in Computer Science*. Springer-Verlag, 208–222.

ABDULLA, P. AZIZ, COLLOMB-ANNICHINI, A., BOUAJJANI, A., AND JONSSON, B. 2004. Using Forward Reachability Analysis for Verification of Lossy Channel Systems. *Formal Methods in System*

Design 25, 1, 39–65.

BARDIN, S., FINKEL, A., LEROUX, J., AND PETRUCCI, L. 2003. FAST: Fast Acceleration of Symbolic Transition systems. In *Proceedings of CAV'2003*, Volume 2725 of *Lecture Notes in Computer Science*. Springer, 118–121.

BARDIN, S. AND PETRUCCI, L. 2004. From PNML to counter systems for accelerating Petri Nets with FAST. In *Proc. of the Workshop on Interchange Formats for Petri Nets (at ICATPN 2004)*.

BARTLETT, K.A., SCANTLEBURY, R.A., AND WILKINSON, P.T. 1969. A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. *Communications of the ACM 12*, 5 (May), 260–261.

BÉRARD, B. AND FRIBOURG, L. 1999. Reachability analysis of (timed) Petri nets using real arithmetic. In *Proceedings of CONCUR'99*, Volume 1664 of *Lecture Notes in Computer Science*. Springer, 178–193.

BILLINGTON, J. AND GALLASCH, G. E. 2004. An Investigation of the Properties of Stop-and-Wait Protocols over Channels which can Re-order messages. Tech. Report CSEC-15, Computer Systems Engineering Centre Report Series, University of South Australia.

BILLINGTON, J., GALLASCH, G. E., AND HAN, B. 2004. A Coloured Petri Net Approach to Protocol Verification. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, Volume 3098 of *Lecture Notes in Computer Science*. Springer-Verlag, 210–290.

BILLINGTON, J., GALLASCH, G.E., AND PETRUCCI, L. 2005. Transforming Coloured Petri Nets to Counter Systems for Parametric Verification: A Stop-and-Wait Protocol Case Study. In *Proceedings of 2nd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'05), Rennes, France*, TUCS General Publication, No. 39, 37–55.

BILLINGTON, J., WHEELER, G.R., AND WILBUR-HAM, M.C. 1988. PROTEAN: A High-level Petri Net Tool for the Specification and Verification of Communication Protocols. *IEEE Transactions on Software Engineering 14*, 3 (March), 301–316.

BOUDET, A. AND COMON, H. 1996. Diophantine equations, Presburger arithmetic and finite automata. In *Proceedings of CAAP'96*, Volume 1059 of *Lecture Notes in Computer Science*. Springer, 30–43.

CADP. 2005. CADP *homepage*. `http://www.inrialpes.fr/vasy/cadp/`.

CCITT. 1984. ISDN user-network interface Data link layer specification. Draft Recommendation Q.921, Working Party XI/6, Issue 7.

CPN. 2004. DESIGN/CPN *online*. `http://www.daimi.au.dk/designCPN`.

DIAZ, M. 1982. Modelling and Analysis of Communication and Co-operation Protocols Using Petri Net Based Models. In *Protocol Specification, Testing and Verification*. North-Holland, 465–510.

ESPARZA, J., FINKEL, A., AND MAYR, R. 1999. On the verification of broadcast protocols. In *Proceedings of LICS'99*. IEEE CS Press, 352–359.

FAST. 2005. FAST *homepage*. `http://www.lsv.ens-cachan.fr/fast/`.

FINKEL, A. AND LEROUX, J. 2002. How To Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *Proceedings of FST&TCS'2002*, Volume 2556 of *Lecture Notes in Computer Science*. Springer, 145–156.

GALLASCH, G. E. AND BILLINGTON, J. 2005a. Towards the Parametric Verification of the Class of Stop-and-Wait Protocols over Ordered Channels. Tech. Report CSEC-21, Computer Systems Engineering Centre Report Series, University of South Australia.

GALLASCH, G. E. AND BILLINGTON, J. 2005b. Using Parametric Automata for the Verification of the Stop-and-Wait Class of Protocols. In *Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA 2005), Taipei, Taiwan*, Volume 3707 of *Lecture Notes in Computer Science*. Springer, 457–473.

HOPCROFT, J. E., MOTWANI, R., AND ULLMAN, J.D. 2001. *Introduction to Automata Theory, Languages, and Computation*, 2nd edition. Addison-Wesley.

JENSEN, K. 1997. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, second edition. Volume 1. Springer.

KRISTENSEN, L. M. AND MAILUND, T. 2002. A Generalised Sweep-Line Method for Safety Properties. In *Proceedings of FME'02*, Volume 2391 of *Lecture Notes in Computer Science*. Springer-Verlag, 549–567.

LEROUX, J. 2003. The affine hull of a binary automaton is computable in polynomial time. In *Proceedings of INFINITY'2003*, Volume 98 of *Electronic Notes in Theor. Comp. Sci.*. Elsevier

Science, 89–104.

MARSAN, M. A., BIANCO, A., CIMINIERA, L., SISTO, R., AND VALENZANO, A. 1994. A LOTOS Extension for the Performance Analysis of Distributed Systems. *IEEE Transactions on Networking 2*, 2, 151–165.

MONA. 2004. *The* MONA *project*. `http://www.brics.dk/mona/`.

REISIG, W. 1998. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer-Verlag.

STALLINGS, W. 2004. *Data and Computer Communications*, 7th edition. Prentice Hall.

STEGGLES, L.J. AND KOSIUCZENKO, P. 1998. A Timed Rewriting Logic Semantics for SDL: a case study of the Alternating Bit Protocol. Volume 15 of *Electronic Notes in Theor. Comp. Sci.*. Elsevier Science, 83–104.

SUZUKI, I. 1990. Formal Analysis of the Alternating Bit Protocol by Temporal Petri Nets. *IEEE Transactions on Software Engineering 16*, 11, 1273–1281.

TANENBAUM, A. 2003. *Computer Networks*, 4th edition. Prentice Hall.

TREX. 2003. TReX *homepage*. `http://www.liafa.jussieu.fr/~sighirea/trex`.

TURNER, K. J. 1993. *Using Formal Description Techniques: An Introduction to Estelle, Lotos and SDL*. Wiley Series in Communication and Distributed Systems. John Wiley & Sons.

VALMARI, A. 1998. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*. Volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, 429–528.

VALMARI, A. AND KOKKARINEN, I. 1998. Unbounded Verification Results by Finite-State Compositional Techniques: 10^{any} States and Beyond. In *Proceedings of ACSD'98*. IEEE CS Press, 75–85.

WOLPER, P. AND BOIGELOT, B. 2000. On the construction of automata from linear arithmetic constraints. In *Proceedings of TACAS'2000*, Volume 1785 of *Lecture Notes in Computer Science*. Springer, 1–19.