

# Closed Form Expressions for the State Space of TCP's Data Transfer Service Operating over Unbounded Channels

Jonathan Billington and Bing Han

Computer Systems Engineering Centre  
University of South Australia  
Mawson Lakes, SA 5095, Australia  
Email: Jonathan.Billington@unisa.edu.au  
Bing.Han@postgrads.unisa.edu.au

## Abstract

The Internet is a very complex system, comprising a dynamically changing network of networks. The Internet's original designers created the Transmission Control Protocol (TCP) to provide a reliable end to end data service to its users, that would operate correctly in the face of failures in the network infrastructure. TCP handles the vast majority of the ever increasing traffic over the Internet and it is therefore of utmost importance that it operates correctly as the Internet grows and as speeds on the Internet increase. This paper provides a step towards its formal verification by providing a general formalisation of TCP requirements in the knowledge that it operates over a medium of very large and unknown capacity. These requirements are expressed in what is termed a service specification. Once the intent of the service that TCP is to provide to its users is defined in the service specification, TCP can be verified against this intent. A central part of the service specification is the definition of the sequences of user observable events (known as service primitives) that can occur at the user/TCP boundary. This is known as the service language. An important verification task is then to prove that TCP complies with this service language. The formal verification of a protocol as complex as TCP is an ambitious undertaking. To simplify the task, we consider the connection management and data transfer parts of the protocol and service separately. In this paper, we are concerned with the data transfer part of the service. Unfortunately the TCP Data Transfer Service language grows exponentially with the size of the medium over which it operates, which in general is unbounded. This implies that the automaton that represents the Data Transfer Service language is infinite. To tackle this problem, we parameterise the Data Transfer Service by the size of the medium. We provide a Coloured Petri Net model of the Data Transfer Service based on previous work and establish closed form expressions for its state space parameterised by the medium capacity. The state space is the automaton that represents the Data Transfer Service language. These expressions eliminate the need for reachability analysis and determine a parameterised automaton that embodies the Data Transfer Service language for arbitrary medium capacity.

*Key words:* Networking, Internet, TCP, Unbounded Systems, Modelling and Tools.

Copyright ©2004, Australian Computer Society, Inc. This paper appeared at the 27th Australasian Computer Science Conference, The University of Otago, Dunedin, New Zealand. Conferences in Research and Practice in Information Technology, Vol. 26. V. Estivill-Castro, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

## 1 Introduction

The Internet is a network of networks that are heterogeneous, dynamic and distributed in nature. The Internet is thus a complex system of systems which is constantly growing, both in terms of the number of nodes (host and routers) but also in terms of the amount of traffic being carried and the speed of its links. It is also vitally important to modern economies world-wide, with new applications being developed constantly such as banking and electronic commerce facilities. It is therefore imperative that the Internet provides a reliable service to its global community of users.

The TCP/IP protocol suite (Tanenbaum 2003) is at the heart of the Internet. It defines the rules that govern how information is moved between users. A major component of the TCP/IP protocol suite is the Transmission Control Protocol (TCP) [19]. TCP plays a pivotal role in ensuring that data is transferred reliably between Internet applications such as the World Wide Web and Email. The vast majority of traffic on the Internet uses TCP to ensure it is relayed successfully between users. With the global economy becoming more dependent on the Internet, it behoves us to thoroughly understand the operation of TCP, and to verify that it will behave according to its original designer's intent, in the face of growing traffic and increased data rates.

TCP divides the stream of application data into segments and passes them down to the Internet Protocol (IP) (Postel 1981a) for transmission. Because IP offers a best effort service, packets may be lost or duplicated, and can arrive out of order. Offering a reliable data transfer service, TCP ensures that data is delivered without loss or duplication and in the correct order. The Request for Comments (RFC) document, RFC 793 (Postel 1981b), provides an informal description of TCP and its user interface calls. The intent of TCP, i.e., the service that TCP intends to provide to its users, however, is not clearly defined in RFC 793. To verify that TCP offers a reliable data transfer service, we must firstly define the service TCP provides in a formal way. This has been achieved in (Billington & Han 2003b) and (Billington & Han 2003a), where the TCP service comprising the connection management service, the data transfer service and the abort service is defined. Related work on specifying the TCP service appears in (Smith 1996, Murphy & Shankar 1991, Pouffary & Young 1997). Our work is different in two significant ways: firstly we do not include purely local interactions in the service definition, but only consider those that have *end-to-end* significance; and secondly we have aimed at providing a complete service specification that includes all TCP services, in particular, the simultaneous establishment of connections and urgent

transfer of data.

We use coloured Petri nets (CPNs) (Jensen 1997a) to model TCP’s service, with a view to generating the *service language* for protocol verification (Billington, Wilbur-Ham & Bearman 1986). The service language comprises the set of global sequences of *service primitives* that convey information between the *service user* and the *service provider*. An important part of protocol verification is to decide if the *protocol language* (the set of sequences of service primitives generated by the protocol) is equivalent to the service language.

Coloured Petri nets have been used extensively in modelling many complex systems including communication protocols and services (Billington, Diaz & Rozenberg 1999, Jensen 1997c, Gordon 2001, Liu & Billington 2002, Villapol 2003, Ouyang, Kristensen & Billington 2002). They combine the power of Petri nets (Peterson 1981) for modelling concurrent and distributed systems, with the strength of a programming language (i.e., Standard ML (Ullman 1998)) for expressing and manipulating data.

A CPN model may be analysed using reachability analysis (Jensen 1997b), which involves constructing a directed graph known as the Occurrence Graph (OG) with vertices representing states (markings in CPNs) and edges (arcs) representing state changes. The OG (state space) of a service model can be regarded as an automaton (Hopcroft, Motwani & Ullman 2001) that accepts the service language. Thus obtaining the state space of TCP is an important step towards its verification. The state space can be generated automatically using software tools such as Design/CPN (University of Aarhus 1996), but this is only possible for moderately large state spaces (e.g.,  $10^6$  states).

As the size of the medium over which TCP’s service operates is in general unbounded (we do not know the storage capacity that will be used for the connection in the Internet), the state space that represents the service language is infinite for the data transfer service (Billington & Han 2003b). To tackle this problem, we parameterise the data transfer service by the capacity of the medium. In this paper, we derive closed form expressions for the states and arcs of the state space of the TCP data transfer service in terms of the capacity of the medium. This eliminates the need for (computationally expensive) reachability analysis which would be required for each value of the capacity of the medium. It also provides a symbolic automaton that encodes TCP’s data transfer service language that is required for the verification of TCP’s data transfer protocol for arbitrary medium capacity.

We claim that this is the first time that closed form expressions have been defined for the automaton that represents the data transfer service language for arbitrary medium capacity, which is required for TCP operating over the Internet. This is an essential first step for our overall goal of verifying TCP’s behaviour is consistent with its requirements, expressed in the service specification.

This paper is organised as follows. Section 2 provides an informal description of the TCP data transfer service. Section 3 presents and explains the CPN model of the service. We propose closed form expressions for calculating the state space and prove them correct in Section 4. Finally, Section 5 provides some conclusions.

## 2 TCP Data Transfer Service

The TCP service comprises a connection management service, a data transfer service and an abort

service. We defined the TCP service (Billington & Han 2003b, Billington & Han 2003a) in accordance with the established Open Systems Interconnection (OSI) Service Conventions (ITU-T 1993). A service definition describes the behaviour of a service provider observed by its service users. A service definition comprises two main parts: a set of service primitives that represent the abstract interactions between the service user and provider; and a specification of their sequences at the service/provider interface.

A service primitive name comprises three components: an initial or initials, indicating the layer that provides the service; a *service type*; and a *primitive type*, i.e., request, indication, response and confirm (abbreviated to req, ind, res, and cnf). Service primitives are submitted and delivered at what is known as a *service access point* (SAP) at the service/provider interface, which is identified by an address. Primitives submitted by the service user are request and response type primitives. Primitives delivered by the service provider are indication and confirm type primitives.

Figure 1 depicts the bidirectional TCP data transfer service. It is concerned with delivering normal and urgent data from one application to another application. The service user is an application protocol such as HTTP (Fielding, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999) or FTP (Bhushan 1972), and the provider is TCP and its underlying protocols. It is a bidirectional service in that both ends can send and receive data. We define four service primitives: TCP-DATA.req and TCP-DATA.ind for normal data delivery, and TCP-UDATA.req and TCP-UDATA.ind for urgent data delivery.

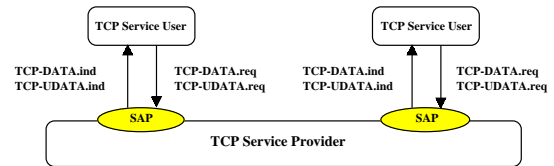


Figure 1: TCP data transfer service block diagram

A sequence of service primitives observed at a local SAP is called a *local sequence*. The set of local sequences of service primitives can be specified formally using a state table or finite state automaton (FSA). The global view of a sequence of primitives taking place at both SAPs is called a *global sequence*. The set of global sequences constitutes the *service language*. To obtain the complete set of global sequences, we need to relate the occurrence of primitives at one end to the occurrence of primitives at the other end. This can be achieved by creating a coloured Petri net (CPN) model which connects the two local FSAs using a queue and generating the state space of the model that encapsulates the global sequences (Billington et al. 1986).

TCP provides an orderly delivery service. That is, data is delivered to the application in the same order it is transmitted. When data arrives at the other end, it is queued in the receive buffer for processing. However, there is a special type of data called urgent data, (e.g., an interrupt), which needs immediate processing upon arrival, regardless of its position in the receive buffer. In this case, urgent data can overtake any amount of normal data ahead of it.

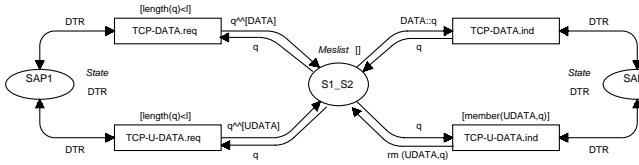


Figure 2: CPN graph of the Data Transfer Service,  $CPN_{DT}$

### 3 CPN Model of TCP Data Transfer Service

We model the TCP service at a high level of abstraction in that we only consider service primitive sequences and are not concerned with their parameters. A model of the bi-directional data transfer service was created in (Billington & Han 2003b). However, no attempt was made to generate the state space of the Data Transfer Service to obtain the service language, because the state space is infinite. To deal with this problem, in this paper we introduce a parameter that models the maximum medium capacity. The medium is modelled as a queue of maximum length  $l$ . The state space is therefore finite for finite  $l$ . Without loss of generality we also only consider the *one-way* data transfer service (between service users) as the data transfer service is independent and identical for each direction.

A CPN model is made up of three components: *places*, *transitions* and *arcs*. A place, typed by a *colour set*, contains collections (multisets) of data items called *tokens* which are values taken from the place's colour set. A transition represents an event and may have a boolean expression, called a *guard*, associated with it. Arcs connect places to transitions and vice versa. Based on (Billington & Han 2003b), we use Design/CPN (University of Aarhus 1996) to create a CPN model (Figure 2) for the one-way data transfer service which comprises 3 places (represented by ellipses) and 4 transitions (rectangles). Annotations associated with places, transitions and arcs are written with CPN ML, a variant of Standard ML (Ullman 1998), as are the declarations (i.e., types, variables, constants and functions) shown in Figure 3.

In Figure 2, places SAP1 and SAP2, typed by colour set STATE (written on one side of each place), model the state of the interface at each service access point. The collection of tokens in a place is known as the *marking* of that place. The initial marking for each SAP place is state DTR (data transfer ready). Place S1\_S2, typed by colour set Meslist (a list type, see line 4 in Figure 3), models communication as a queue where messages DATA (representing normal data) and UDATA (urgent data) are stored in the order they are sent. Place S1\_S2 is initially empty, as indicated by the empty list, [], written on the top of the place.

Each transition models a service primitive. When a transition occurs a message is deposited into or removed from place S1\_S2. Because the data storage capacity in the Internet is unbounded, we model it as a queue of arbitrary length  $l$  ( $l = 0, 1, 2, \dots$ ) by adding the guard  $length(q) < l$  to transitions TCP-DATA.req and TCP-U-DATA.req. Thus messages can only be sent into the queue (via the occurrences of the two transitions) if the queue length is less than  $l$ .

When the transition labelled TCP-DATA.req occurs, message DATA is added to the end of the queue, modelled by concatenating the current queue (represented by variable  $q$ ) with DATA (represented by list

---

```

1  color State=with DTR;
2  color Message=with DATA | UDATA;
3  var mes: Message;
4  color Meslist=list Message;
5  var q: Meslist;
6  val l=3;
7  fun member(mes,[]) = false
8    | member(mes,h::t) = mes=h orelse member(mes,t);
9  fun rm(mes,q) =
10     let val p=fn h => h=mes
11         fun addhd(p,[]) = []
12           | addhd(p,h::t) = if p h then t else h::addhd(p,t)
13     in addhd(p,q)
14     end;

```

---

Figure 3: Declarations of  $CPN_{DT}$

[DATA]), i.e., arc inscription  $q \wedge [DATA]$ . Transition TCP-DATA.ind can only occur if DATA is at the head of the queue, that is, when the expression  $DATA::q$  (a non-empty list with DATA at the head and  $q$  as the tail) evaluates to a list that matches the token in place S1\_S2. When TCP-DATA.ind occurs, DATA is removed from the head of the queue, modelled by removing the current queue and replacing it by the tail.

The occurrence of transition TCP-UDATA.req results in UDATA being put into the queue, modelled by concatenating the current queue with list [UDATA]. As described in Section 2, it is possible for urgent data to be processed before any of the normal data ahead of it in the queue. This situation is modelled by using the guard,  $member(UDATA,q)$ , on transition TCP-UDATA.ind and the inscription,  $rm(UDATA,q)$ , on the arc from transition TCP-UDATA.ind to place S1\_S2. The function,  $member()$ , is defined recursively in Figure 3 (lines 7 and 8) and the guard,  $member(UDATA,q)$ , requires that UDATA must be in the queue to enable transition TCP-UDATA.ind. However, it does not require that UDATA must be at the head of the queue. (This is in contrast with the rule for enabling transition TCP-DATA.ind, which requires that DATA must be at the head of the queue.) When TCP-UDATA.ind occurs, the old queue is replaced by a new queue, which has the first occurrence of UDATA removed, showing that UDATA has been received. This is realised by the recursive function  $rm()$ , which is defined in lines 9-14 in Figure 3. When DATA is at the head of the queue and UDATA also appears in the queue, the occurrence of TCP-DATA.ind or TCP-UDATA.ind is chosen arbitrarily. Note that the token in each SAP place remains the same (i.e., DTR) after each service primitive is submitted or delivered. This can be seen from the bidirectional arc labelled by DTR between a SAP place and a transition. Places SAP1 and SAP2 are redundant with respect to the state space of the model, however we retain them as they indicate the environment in which our results are obtained. That is they are only valid when both ends have interfaces in the DTR state.

### 4 Closed Form Expressions for the State Space for $CPN_{DT}$

Previous work (Han & Billington 2003) has shown that the number of markings ( $V_l$ ) and arcs ( $A_l$ ) increase exponentially in the capacity of the queue,  $l$ . Experiments indicate that it takes a significant amount of time to calculate the state space for  $l \geq 15$  using Design/CPN. For example, for  $l = 12$ , it takes

less than 4 minutes to compute the state space on a machine with a Celeron 800 MHz CPU and 128 MB RAM, whereas, for  $l = 15$ , it takes over 4 hours. Time increases by about a factor of 4 for each increment in  $l$ , making reachability analysis intractable for  $l \geq 20$ . Closed form expressions for the set of markings and arcs of the state space overcome this problem. Moreover, they provide the state space symbolically for arbitrary values of  $l$ . This section establishes theorems for the set of markings and arcs comprising the state space, also known as the occurrence graph (OG). We start by giving the formal definition of an occurrence graph.

**Definition 1** *The occurrence graph of a CPN is a labelled directed graph  $OG=(V,A)$  where*

- (i)  $V = [M_0]$ , is the set of markings of the CPN that are reachable from the initial marking  $M_0$ ;
- (ii)  $A = \{(M_1, b, M_2) \in V \times BE \times V \mid M_1[b]M_2\}$  is a set of labelled arcs, where  $BE$  is the set of binding elements of the CPN. A binding element is a pair consisting of a transition and the assignment of values to the variables associated with that transition.  $M_1[b]M_2$  denotes that the marking of the CPN changes from  $M_1$  to  $M_2$  on the occurrence of binding element  $b \in BE$ .

The OG of  $CPN_{DT}$  for a maximum queue length  $l = \{0, 1, 2, 3, \dots\}$ , is denoted  $OG_l = (V_l, A_l)$ , where  $V_l$  is the set of markings and  $A_l$  is the set of arcs. Figure 4 shows the OGs for  $l = 1, 2$  and 3, which are generated by Design/CPN. Markings are represented by rounded squares called nodes. Each marking is identified by a unique positive integer. Below this integer are written the numbers of incoming arcs and outgoing arcs associated with that marking, separated by a colon. Markings are arranged in layers. Markings with the same queue length of place S1\_S2 are located in the same layer. For example, in Figure 4 (b), the queue length of place S1\_S2 for markings on the second layer is 1, containing one message, either DATA or UDATA, and markings in the third layer have a queue length of 2. The value for each of the markings of place S1\_S2 is indicated by the list of items in the queue identified by each marking number and written near the marking to which they apply (where space permits). The markings of places SAP1 and SAP2 are always the same, given by 1'DTR, and are therefore not included. Each arc is labelled by the transition of a binding element, which is abbreviated to four letters: DREQ, dind, UREQ, or uind, standing for TCP-DATA.req, TCP-DATA.ind, TCP-UDATA.req and TCP-UDATA.ind respectively. Because we are interested in sequences of service primitives (the names of the transitions) we do not need to include the full binding element, as the value of the variable  $q$  is not relevant. The occurrence of DREQ and UREQ result in two new two successors. To facilitate characterising the state space, we lay out the OG so that the left successor is reached via DREQ and the right successor via UREQ.

#### 4.1 The Set of Reachable Markings

Every marking  $M \in V_l$  comprises the markings of the three places: SAP1, S1\_S2 and SAP2. From  $CPN_{DT}$  (Figure 2) it is clear that the markings of places SAP1 and SAP2 always comprise the simple token DTR:

$$\forall M \in V_l, M(SAP1) = M(SAP2) = 1'DTR$$

**Theorem 1** *The set of markings of  $CPN_{DT}$   $V_l = \{DTR\} \times V_l^{S1-S2} \times \{DTR\}$ , where  $V_l^{S1-S2}$  is the*

*set of markings of place S1\_S2 and is expressed as follows.  $V_l^{S1-S2} = \{DATA, UDATA\}^{l*}$  where  $X^{l*}$  represents the set of strings over  $X$  of length  $l$  or less including the empty string ( $\epsilon$ ).*

**Proof.** Initially the queue is empty so the empty string (represented by the empty list in Design/CPN) is a marking of S1\_S2. Only DATA and UDATA can be deposited into the queue by transitions DREQ and UREQ. DREQ and UREQ are both enabled for all markings, so long as the queue has less than  $l$  items in it. Thus markings can be obtained for S1\_S2 which comprise arbitrary strings of DATA and UDATA up to length  $l$ , by arbitrary firings of DREQ and UREQ until the queue is full. The transitions dind and uind do not create any new markings, as they just delete an item from the queue, giving a marking that has already been taken into account. Hence  $V_l^{S1-S2} = \{DATA, UDATA\}^{l*}$ . The two SAP places contain a DTR token for all markings, due to the bi-directional arc from these places to each of the four transitions.  $\square$

#### 4.2 The Set of Arcs

This section gives a theorem for the set of arcs. Firstly, it introduces the notation for markings. Secondly, it partitions the set of arcs into the four sets  $A_l^{DREQ}$ ,  $A_l^{UREQ}$ ,  $A_l^{dind}$  and  $A_l^{uind}$ , according to the 4 transitions of  $CPN_{DT}$  and specifies each of these in a proposition which is then proved. Finally, a theorem is given which specifies  $A_l$  as the union of these four sets.

##### 4.2.1 Notation for Markings

To reflect the structure of  $OG_l$ , we renumber the markings as shown in Figure 5. Each marking is identified as an ordered pair  $(i, n_i)$  where the queue length  $i = 0, \dots, l$  is the number of items in the queue corresponding to a layer in the OG for that marking and  $n_i = 1, 2, 3, \dots, 2^i$  is a number that uniquely identifies a marking that has  $i$  items in the queue. We use  $M_i^{n_i}$  to denote such a marking.

In order to relate the numbering used for markings to the marking of place S1\_S2 we use the bits 0 and 1 to represent DATA and UDATA respectively. Every occurrence of DREQ places a 0 (DATA) into the queue and likewise, an occurrence of UREQ places a 1 (UDATA) into the queue. Figure 6 illustrates this situation. It is obtained from Figure 5 (c) by deleting the arcs labelled by dind and uind, and including the marking of place S1\_S2 in its binary form below each marking. Because the OG is laid out with the follower marking resulting from DREQ being placed on the left and that resulting from UREQ being positioned to its right, it follows from Theorem 1 that the value of the binary number will be incremented by one as we move from the left most marking to the rightmost marking on any layer  $i$ , from 0 to  $2^i - 1$ . For  $M_i^{n_i}$  ( $1 \leq i \leq l$ ), the decimal number associated with the marking of place S1\_S2, corresponds to the number  $n_i - 1$ . To obtain the contents of the queue corresponding to marking  $M_i^{n_i}$  we convert  $n_i - 1$  to its binary representation,  $(n_i - 1)_2$ , which must have  $i$  digits. If  $n_i - 1$  does not have  $i$  digits when converted to binary, leading 0s are added to obtain a binary number that is  $i$  digits long. For example, if  $n_i = 3$  and  $i = 3$ ,  $(n_i - 1)_2 = 010$ . The position of each digit 0 or 1 in the binary number is the same as that of DATA or UDATA it represents in the queue. Therefore, converting 0 to DATA and 1 to UDATA gives us  $M_i^{n_i}(S1_S2)$ .

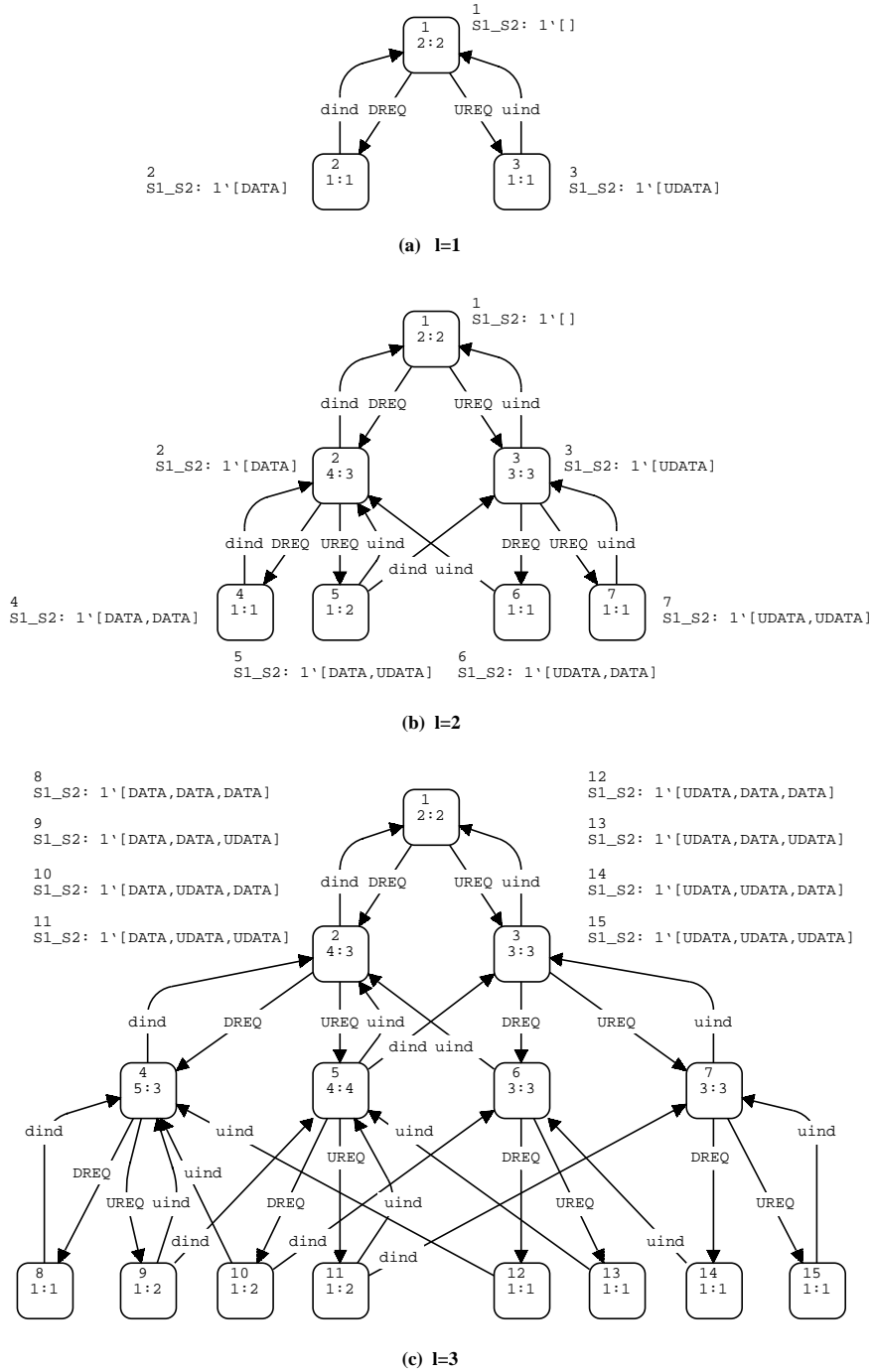


Figure 4: Occurrence graphs for  $l=1,2,3$

To formalise this relationship we define a family of functions  $f_i$  for  $i \in \{1, 2, \dots, l\}$ :

$$f_i : \{0, 1\}^i \rightarrow \{DATA, UDATA\}^i$$

where  $X^i$  is the set of strings of length  $i$  over set  $X$  and  $f_1(0) = DATA$ ,  $f_1(1) = UDATA$  and for  $a_1, a_2, \dots, a_i \in \{0, 1\}$

$$f_i(a_1 a_2 \dots a_i) = f_1(a_1) f_1(a_2) \dots f_1(a_i)$$

where juxtaposition is used for concatenation.

$M_i^{n_i}(S1\_S2)$  is given in the following theorem.

**Theorem 2** *The marking of  $S1\_S2$  of  $CPN_{DT}$  can be represented as follows.*

$$M_i^{n_i}(S1\_S2) = \begin{cases} \epsilon & i = 0 \\ f_i((n_i - 1)_2) & i \in \{1, 2, \dots, l\} \end{cases}$$

where  $n_i \in \{1, 2, \dots, 2^i\}$  and  $(n_i - 1)_2$  comprises a string of  $i$  bits.

**Proof.** For  $i=0$ , the queue is empty, represented by the empty list in Design/CPN. Therefore we have  $M_0^1(S1\_S2) = \epsilon$ . For  $1 \leq i \leq l$ , the proof follows from the above arguments.  $\square$

#### 4.2.2 Specifying the Four Arc Subsets

As indicated in the introduction to this section, the set of arcs can be partitioned into 4 sets of arcs, each related to a particular transition of  $CPN_{DT}$ . This subsection specifies these sets in four propositions.

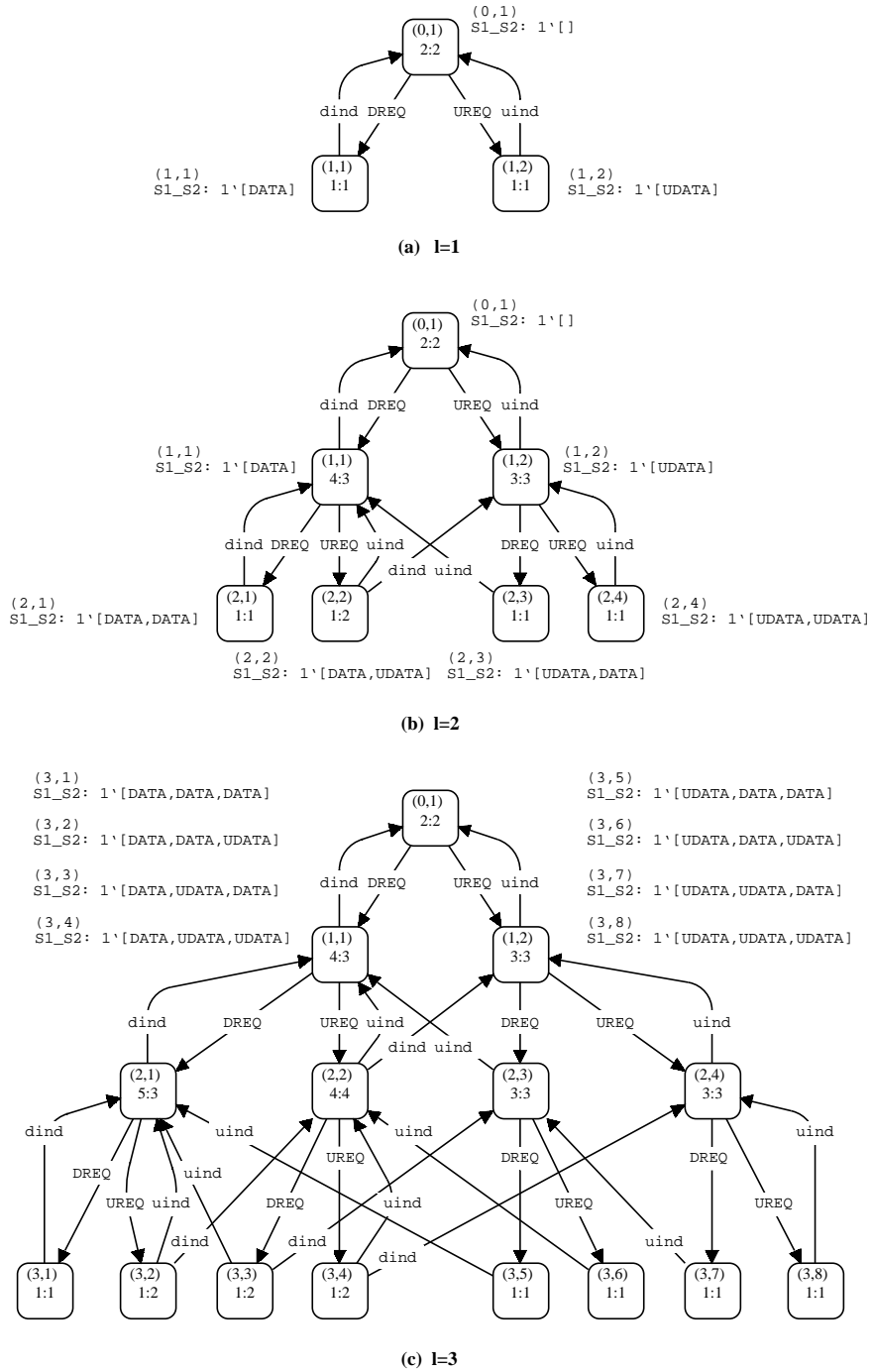


Figure 5: Renumbered OGs for  $l=1,2,3$

**Proposition 1** For  $l \in \mathbb{N}^+$ ,  $A_l^{DREQ} = \{(M_i^{n_i}, DREQ, M_{i+1}^{2n_i-1}) \mid i \in \{0, 1, \dots, l-1\} \text{ and } n_i \in \{1, 2, \dots, 2^i\}\}$  and  $A_0^{DREQ} = \emptyset$ .

**Proof.** The proof is in two parts. Firstly we prove that the occurrence of DREQ will always lead to a new marking that has one more item in the queue (Lemma 1) and secondly that the number of the new marking (for the new queue length) is always odd (Lemma 2).

**Lemma 1** For  $l \in \mathbb{N}^+$ ,  $A_l^{DREQ} = \{(M_i^{n_i}, DREQ, M_{i+1}^{n_i+1}) \mid i \in \{0, 1, \dots, l-1\} \text{ and } n_i \in \{1, 2, \dots, 2^i\}\}$

**Proof.** From Figure 2, we observe that DREQ is enabled in all markings for which  $i < l$ . This is because the input conditions are satisfied in all markings, but the queue length,  $i$ , must be less than its maximum value,  $l$ , to satisfy the guard. When DREQ occurs, it adds DATA to the queue, increasing its length,  $i$ , by one. Thus because of our numbering convention for markings (where  $i$  is the current size of the queue, and  $n_i$  is the number of the marking for a queue size of  $i$ ) the occurrence of DREQ in marking  $M_i^{n_i}$ , for  $i \in \{0, 1, \dots, l-1\}$ , results in marking  $M_{i+1}^{n_i+1}$ . This is true for any of the markings with a queue size of  $i$ , so long as  $i < l$ . The occurrence of DREQ, corresponds to an arc labelled by DREQ in the OG, and is thus in  $A_l^{DREQ}$ .  $\square$

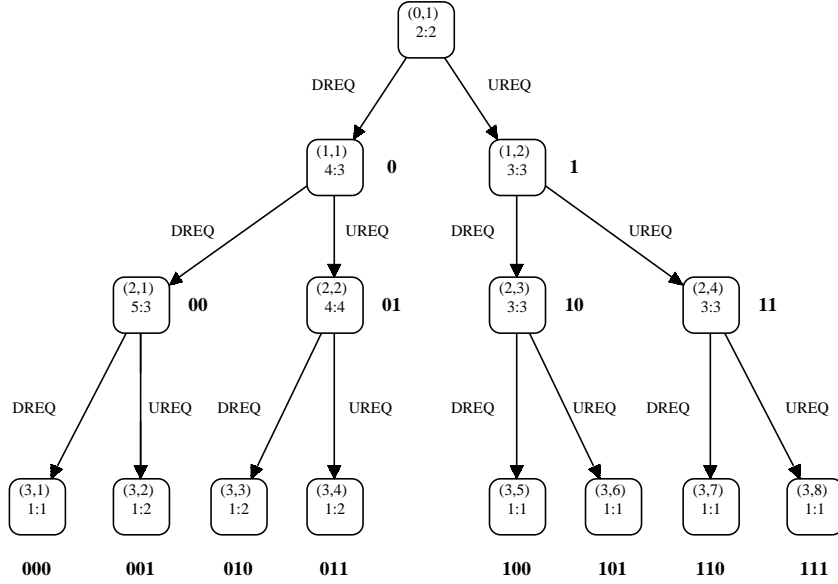


Figure 6: Binary tree for  $OG_3$

**Lemma 2** In Lemma 1, for  $i \in \{0, 1, \dots, l-1\}$  and  $n_i \in \{1, 2, \dots, 2^i\}$ ,  $n_{i+1} = 2n_i - 1$ .

**Proof.** Every marking  $M_i^{n_i}$  such that  $i \in \{0, 1, \dots, l-1\}$  has exactly two successor markings where the size of the queue is increased by one. These are obtained on the occurrence of DREQ and UREQ, as UREQ is also enabled in all markings for which  $i < l$ . Because of the convention for numbering markings (see Fig. 5(c) where we start numbering markings resulting from the occurrence of DREQ with 1 and alternate the numbering with the markings resulting from the occurrence of UREQ, each time incrementing by 1) then the marking resulting from the occurrence of DREQ in  $M_i^{n_i}$  always has an odd value given by  $n_{i+1} = 2n_i - 1$  for  $n_i \in \{1, 2, \dots, 2^i\}$  (e.g. in Fig. 5(c),  $M_2^1[DREQ]M_3^1$  and  $M_2^2[DREQ]M_3^3$  etc.)  $\square$

Finally, when  $l = 0$ , the queue cannot accept any messages, and hence DREQ cannot occur, so that  $A_0^{DREQ} = \emptyset$ .  $\square$

**Proposition 2** For  $l \in \mathbb{N}^+$ ,  $A_l^{UREQ} = \{(M_i^{n_i}, UREQ, M_{i+1}^{2n_i}) \mid i \in \{0, 1, \dots, l-1\} \text{ and } n_i \in \{1, 2, \dots, 2^i\}\}$  and  $A_0^{UREQ} = \emptyset$ .

**Proof.** This is essentially the same as the proof of Proposition 1, where the markings resulting from an occurrence of UREQ have even numbers ( $n_{i+1} = 2n_i$ ) according to the numbering convention, rather than the odd ones for DREQ.  $\square$

**Proposition 3** For  $l \in \mathbb{N}^+$ ,  $A_l^{dind} = \{(M_i^{n_i}, dind, M_{i-1}^{n_i}) \mid i \in \{1, \dots, l\} \text{ and } n_i \in \{1, 2, \dots, 2^{i-1}\}\}$  and  $A_0^{dind} = \emptyset$ .

**Proof.** The set of arcs that exist in  $OG_l$  that are labelled by dind result from the occurrence of transition TCP-DATA.ind (dind). TCP-DATA.ind can only occur if DATA is in the queue, and thus  $A_l^{dind}$  will be empty for  $l = 0$ .

For  $OG_l$ ,  $l > 0$ , we firstly determine the set of markings that enable dind as these are the source

nodes of the arcs. From  $CPN_{DT}$  (Fig. 2) the set of markings that enable dind are those that have DATA at the head of the queue, and is given by  $\{M_i^{n_i} \mid i \in \{1, \dots, l\} \text{ and } n_i \in \{1, 2, \dots, 2^{i-1}\}\}$ . This is due to our numbering convention, where the first half of all the markings, for a particular queue length, have DATA at the head of the queue. We then observe that an occurrence of dind in a marking with a queue length of  $i$  results in a marking with a queue length of  $i-1$ , as dind removes DATA from the head of the queue. Thus for a given maximum queue length,  $l$ , the set of arcs is given by  $A_l^{dind} = \{(M_i^{n_i}, dind, M_{i-1}^{n_i}) \mid i \in \{1, \dots, l\} \text{ and } n_i \in \{1, 2, \dots, 2^{i-1}\}\}$ .

We now need to prove that the number of the resulting marking  $n_{i-1}$  in layer  $i-1$  is identical to the number of the source marking in layer  $i$ , i.e.  $n_{i-1} = n_i$  for  $i \in \{1, \dots, l\}$ . If we represent  $(n_i - 1) \in \{0, 1, \dots, 2^{i-1} - 1\}$  in its binary form, we obtain a string of  $i$  bits starting with a zero as DATA is the first item in the queue. When TCP-DATA.ind (dind) occurs, the DATA item is removed from the head of the queue, thus removing the leading zero from the binary string representation to obtain another binary string of length  $i-1$ , which corresponds to  $n_{i-1} - 1$ . Hence the number is the same (deleting the most significant bit, when zero, does not alter the value of the number), and hence  $n_{i-1} = n_i$ .  $\square$

**Proposition 4** For  $l \in \mathbb{N}^+$ ,  $A_l^{uind} = \{(M_i^{n_i}, uind, M_{i-1}^{n_i}) \mid i \in \{1, \dots, l\} \text{ and } n_i \in \{2, \dots, 2^i\}\}$ , such that  $n_{i-1} = n_i - 2^k$  where  $k$  is the largest integer such that  $2^k < n_i$  and  $A_0^{uind} = \emptyset$

**Proof.** The proof is similar to the proof of Proposition 3. The arcs labelled with uind arise from the occurrence of transition TCP-UDATA.ind (uind). TCP-UDATA.ind can only occur if UDATA is in the queue, and thus  $A_l^{uind}$  will be empty for  $l = 0$ .

From  $CPN_{DT}$  (Fig. 2), the markings that enable uind are those where UDATA is in the queue and are given by  $\{M_i^{n_i} \mid i \in \{1, \dots, l\} \text{ and } n_i \in \{2, \dots, 2^i\}\}$ , since when  $n_i = 1$  there is no UDATA in the queue due to our numbering convention for markings. These are the source nodes of the arcs labelled by uind.

When und occurs in a marking with a queue length  $i > 0$  it removes UDATA from the queue. This gives a queue of length of  $i - 1$  and hence a resulting marking  $M_{i-1}^{n_i-1}$ . The occurrence of und removes UDATA from the queue, which deletes the first (most significant) '1' in the binary representation of  $n_i - 1$  to give  $n_{i-1} - 1$ . This is the same as subtracting the largest power of 2 from  $n_i$  to give a positive result. Thus  $n_{i-1} = n_i - 2^k$  where  $k$  is the largest integer such that  $2^k \leq n_i - 1$ , which implies  $2^k < n_i$ .  $\square$

Based on the above results we obtain the following theorem for the arcs.

**Theorem 3** For  $l \in \mathbb{N}^+$ ,  $A_l = A_l^{DREQ} \cup A_l^{UREQ} \cup A_l^{dind} \cup A_l^{uind}$

## 5 Conclusions

This paper addresses the problem of the reliability of the Internet by considering the difficult problem of the verification of the Transmission Control Protocol (TCP), the protocol at the core of the Internet's operation. It provides a first step by deriving closed form expressions for the state space of the TCP data transfer service in terms of the capacity of the underlying medium. This overcomes the problem of the state space being infinite for the data transfer service when considering the unknown capacity of the Internet over which TCP operates. This was achieved by firstly modelling the one-way data transfer service of TCP using coloured Petri nets where the storage capacity of the Internet is represented as a queue of finite length  $l$ . We then derived closed form expressions for the set of reachable states in terms of the messages in the queue and its maximum length  $l$ , and for the set of arcs labelled by service primitives that also depend on  $l$ . These are expressed as two theorems together with their proofs. These results eliminate the need for reachability analysis and provide a parameterised result for arbitrary capacity.

The state space of the CPN model of the TCP service is required for verifying TCP against its service. It embodies the set of sequences of service primitives, known as the service language, against which the TCP data transfer protocol can be verified. Future work will require the development of a model for TCP's data transfer protocol that is also parameterised in the size of the medium over which it operates. This model will include events that represent service primitive occurrences. A significant challenge will be the generation of the occurrence graph that is also parameterised by  $l$  representing the protocol language. Techniques would then need to be developed to compare the parameterised automata representing the service and protocol languages. Failing this, verification of TCP's data transfer protocol could be achieved for various (small) values of  $l$ , by using reachability analysis, and language reduction and comparison techniques supported by tools such as FSM (AT&T Labs 2003).

Another direction for future work could explore generalising the results for an arbitrary number of message types, instead of just the two types (DATA and UDATA) considered for TCP. This may be appropriate for priority queueing systems, with several levels of priority.

## References

AT&T Labs (2003), *FSM Library*, Web site: <http://www.research.att.com/sw/tools/fsm>.

- Bhushan, A. (1972), The File Transfer Protocol, RFC 354, IETF.
- Billington, J., Diaz, M. & Rozenberg, G., eds (1999), *Application of Petri nets to Communication Networks: Advances in Petri nets*, Vol. 1605 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.
- Billington, J. & Han, B. (2003a), Formalising the TCP Symmetrical Connection Management Service, in 'Proceedings of the Design, Analysis and Simulation of Distributed Systems Conference, Orlando, Florida, USA', pp. 178–184.
- Billington, J. & Han, B. (2003b), On Defining the Service Provided by TCP, in 'Proceedings of the 26th Australasian Computer Science Conference, Adelaide, Australia', Vol. 16 of *Conferences in Research and Practice in Information Technology*, pp. 129–138.
- Billington, J., Wilbur-Ham, M. C. & Bearman, M. Y. (1986), 'Automated Protocol Verification', *Protocol Specification, Testing, and Verification V*, 59–70.
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999), Hypertext Transfer Protocol - HTTP/1.1, RFC 2616, IETF.
- Gordon, S. (2001), Verification of the WAP Transaction Layer using Coloured Petri Nets, PhD Thesis, University of South Australia, Australia.
- Han, B. & Billington, J. (2003), Formulas for the State Space Size of a Model of TCP's Data Transfer Service, Technical report, Computer Systems Engineering Centre, University of South Australia.
- Hopcroft, J. E., Motwani, R. & Ullman, J. D. (2001), *Introduction to Automata Theory, Languages and Computation*, 2nd edn, Addison-Wesley, U.S.A.
- ITU-T (1993), 'Information Technology - Open Systems Interconnection - Basic Reference Model: Conventions for the Definition of OSI Services'. ITU-T Recommendation X.210.
- Jensen, K. (1997a), *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 1, Basic Concepts*, Monographs in Theoretical Computer Science, Springer-Verlag, Berlin.
- Jensen, K. (1997b), *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 2, Analysis Methods*, Monographs in Theoretical Computer Science, Springer-Verlag, Berlin.
- Jensen, K. (1997c), *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 3, Practical Use*, Monographs in Theoretical Computer Science, Springer-Verlag, Berlin.
- Liu, L. & Billington, J. (2002), Tackling the Infinite State Space of a Multimedia Control Protocol Service Specification, in 'Proceedings of 23rd International Conference on Application and Theory of Petri Nets', LNCS 2360, Adelaide, Australia, pp. 273–293.

- Murphy, S. L. & Shankar, A. U. (1991), 'Connection Management for the Transport Layer: Service Specification and Protocol Verification', *IEEE Transactions on Communications* **39**(12), 1762–1775.
- Ouyang, C., Kristensen, L. M. & Billington, J. (2002), A Formal Service Specification for the Internet Open Trading Protocol, in 'Proceedings of 23rd International Conference on Application and Theory of Petri Nets', LNCS 2360, Adelaide, Australia, pp. 352–373.
- Peterson, J. L. (1981), *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ.
- Postel, J. (1981*a*), Internet Protocol - DARPA Internet Program Protocol Specification, RFC 791, IETF.
- Postel, J. (1981*b*), Transmission Control Protocol, RFC 793, IETF.
- Pouffary, Y. & Young, A. (1997), ISO Transport Service on top of TCP (ITOT), RFC 2126, IETF.
- Smith, M. A. (1996), 'Formal verification of communication protocols', *Formal Description Techniques IX: Theory, Applications and Tools* pp. 129–144.
- Tanenbaum, A. S. (2003), *Computer Networks*, 4th edn, Prentice Hall.
- Ullman, J. D. (1998), *Elements of ML Programming*, 2nd edn, Prentice Hall, Englewood Cliffs, NJ.
- University of Aarhus (1996), *Design/CPN Online*, Web site: <http://www.daimi.au.dk/designCPN>.
- Villapol, M. E. (2003), Modelling and Analysis of the Resource Reservation Protocol, PhD Thesis, University of South Australia, Australia.