Analysis of Two-Layer Protocols: DCCP Simultaneous-Open and Hole Punching Procedures*

Somsak Vanit-Anunchai

School of Telecommunication Engineering
Institute of Engineering
Suranaree University of Technology
Muang, Nakhon Ratchasima, Thailand
email:somsav@sut.ac.th

Abstract

The simultaneous-open procedure of the Datagram Congestion Control Protocol (DCCP), RFC 5596, was published in September 2009. Its design aims to overcome DCCP weaknesses when the Server is behind a middle box, such as Network Address Translators or firewalls. The original DCCP specification, RFC 4340, only allows the Client to initiate the call. The call request cannot reach the Server behind the middle box. A widely used solution to address this problem is called the "hole punching" technique. This technique requires the Server to initiate sending packets. Using Coloured Petri Nets (CPN) this paper models and analyses the DCCP procedure specified in RFC 5596. However, the difficulty is that detailed modelling of the address translation is also required. This causes state space explosion. We alleviate the state explosion using prioritized transitions and the sweep-line technique. Modelling and analysis approaches are discussed in the hope that it is helpful for others who wish to analyse similar protocols. Analysis results are also obtained for the simultaneous-open procedure specified in RFC 5596.

1998 ACM Subject Classification C.2.2 Network Protocols, D.2.2 Design Tools and Techniques, D.2.4 Software/Program Verification

Keywords and phrases Network Address Translators, Coloured Petri Nets, Sweep-line Method, Prioritized Transitions.

Digital Object Identifier 10.4230/OASIcs.FSFMA.2013.3

1 Introduction

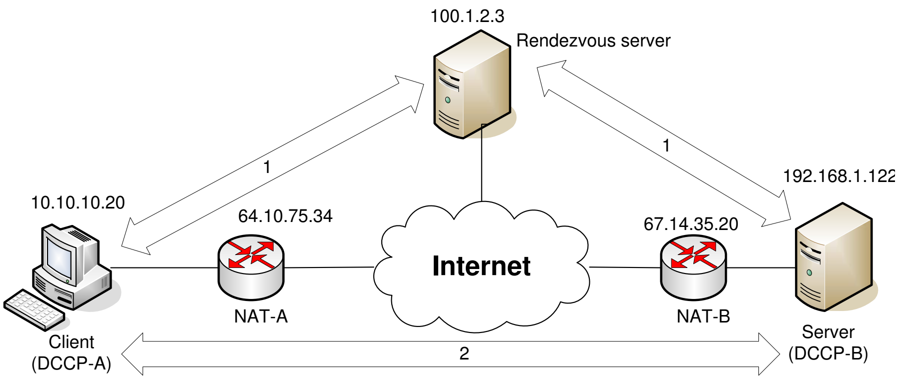
The Datagram Congestion Control Protocol (DCCP) [18] is a transport protocol that provides bidirectional flow of data for applications that prefer timeliness to reliability. It is a connection-oriented protocol operating over the Internet between two entities, the Client and the Server. Originally specified in RFC 4340, only the Client can initiate the connection while the Server passively listens to the incoming request. When the Server is located in a private network or behind a Network Address Translator (NAT[1]), the first incoming packet cannot reach the Server because address mapping in the NAT does not exist yet. To overcome this problem, a simple solution widely used with other transport protocols (UDP, TCP and SCTP) is known as the "hole punching" technique.

---

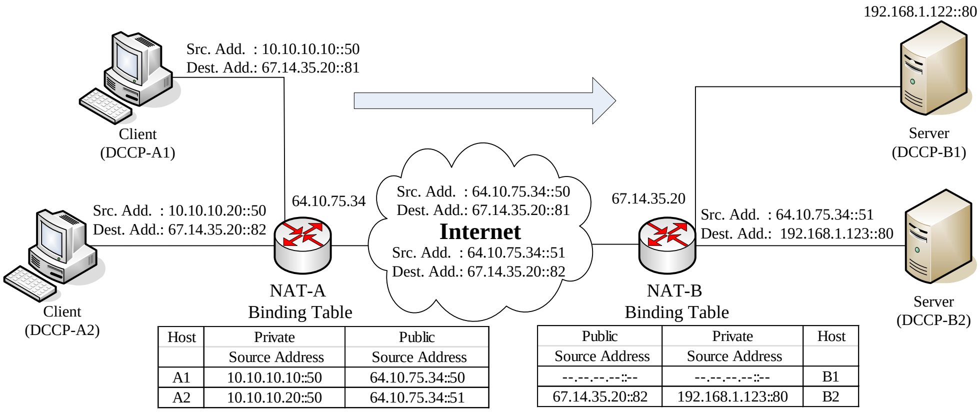
* This work is supported by Research Grant from the Thai Network Information Center Foundation and the Thailand Research Fund.

[1] NAT is a middlebox that maps private (IP addresses - port number) to public (IP addresses - port number) and allows many hosts behind NAT to share the same public IPv4 address.

1st French Singaporean Workshop on Formal Methods and Applications 2013 (FSFMA'13).
Editors: Christine Choppy and Jun Sun; pp. 3–17
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Peer-to-peer communication with rendezvous server.



■ **Figure 2** Binding tables in NAT-A and NAT-B.

The basic idea of hole punching consists of two phases, labeled 1 and 2, as shown in Fig. 1. Firstly, DCCP-A and DCCP-B, which are located behind NAT-A and NAT-B respectively, establish connections with a rendezvous server at a well-known public IP address (100.1.2.3). Because DCCP entities initiate the session via their NAT to the rendezvous server, the server can observe the public IP addresses and port numbers assigned for both sessions. The server then informs each entity of the public IP address and port number of its peer. After receiving this information, the connection between DCCP-A and DCCP-B can start. Illustrated in Fig. 2, when DCCP-A1 sends a packet to DCCP-B1, a "binding table" (or a hole) in NAT-A is created. However the packet is blocked by NAT-B because NAT-B has no binding for DCCP-B1 yet. Thus DCCP-B1 needs to send its packet to DCCP-A1 in order to create a binding table in NAT-B. After a hole is punched in NAT-B, the public address (67.14.35.20::81) associated with DCCP-B1 in the incoming packet will be translated to the private address of DCCP-B (192.168.1.122::80) so that the packet can be locally forwarded to DCCP-B1. In the hole punching scenarios, the Client and the Server initiate sending a packet at about the same time. This requires a new simultaneous open procedure as described in RFC 5596 [9].

1.1 Previous Work

Since 2003 we have constructed, refined and analysed Coloured Petri Net (CPN) [16] models of DCCP's connection management procedure according to RFC 4340, using Design/CPN [8].

In [24], we reported our experience with the incremental enhancement and iterative modelling of the connection management procedures as the DCCP specification was developed. Insight into the decisions behind the modelling choices can also be found in [24]. The full CPN specification of the connection management procedures can be found in Section 2 of [25]. Section 4 of [25] also explains the development of progress mappings for sweep-line state space analysis [21] of DCCP. We have published an enhanced version of [24] which also discusses a procedure-based model of DCCP's connection management procedures [5]. In [6], we discuss how to embed a parameterised channel into CPN models of protocols, using DCCP as an example.

1.2 Contributions

The contribution of this paper is two-fold. Firstly, we extend the Coloured Petri Net model and analysis of the DCCP connection management procedure (RFC 4340) in [5,23] to include the simultaneous open procedure (RFC 5596). Secondly, since embedding NAT with the hole punching procedure as a channel module [6] leads to significant state explosion, we demonstrate methods to circumvent the problem using prioritized transitions and the sweep-line technique [7].

1.3 Organisation

This paper is organised as follows. Section 2 provides an overview of the DCCP simultaneous open procedure. Modelling approach is discussed in Section 3. A description of the CPN model of DCCP's simultaneous open procedure is given in Section 4. Section 5 discusses analysis approach. Section 6 presents the experimental results, with Section 7 providing conclusions and future work.

2 DCCP Overview

2.1 Connection Management Procedures

The Datagram Congestion Control Protocol [17, 18] is a point-to-point transport protocol operating over the Internet between two DCCP entities, the Client and Server. It provides a bidirectional flow of data for applications, such as voice and video, that prefer timeliness to reliability. DCCP is designed to provide congestion control for these applications [11]. Its congestion control algorithms require statistics on packet loss because loss is related to the level of congestion in the network. DCCP uses sequence and acknowledgement numbers in packets to detect and report loss, and includes state variables in each protocol entity to keep track of these numbers. State variables on both sides must be synchronised, otherwise DCCP may misinterpret loss information. Thus DCCP needs mechanisms to set up, synchronise and clear state variables in both the Client and Server. We refer to these mechanisms in general as connection management (CM) procedures.

The CM procedures require packets to be exchanged between the Client and Server. RFC 4340 defines 10 different packet types for this purpose: Request, Response, Data, DataAck, Ack, CloseReq, Close, Reset, Sync and SyncAck. Figure 3 is a state diagram illustrating DCCP's connection establishment and release procedures for both the Client and Server. It is derived by combining the state diagrams in RFCs 4340 and 5596, with the dashed parts of the diagram being added by RFC 5596. Ellipses in Fig. 3 represent states while arrows represent state transitions. CLOSED is both an initial and a final state. The inscription on each arrow describes the input and output actions, if any. For instance, the inscription

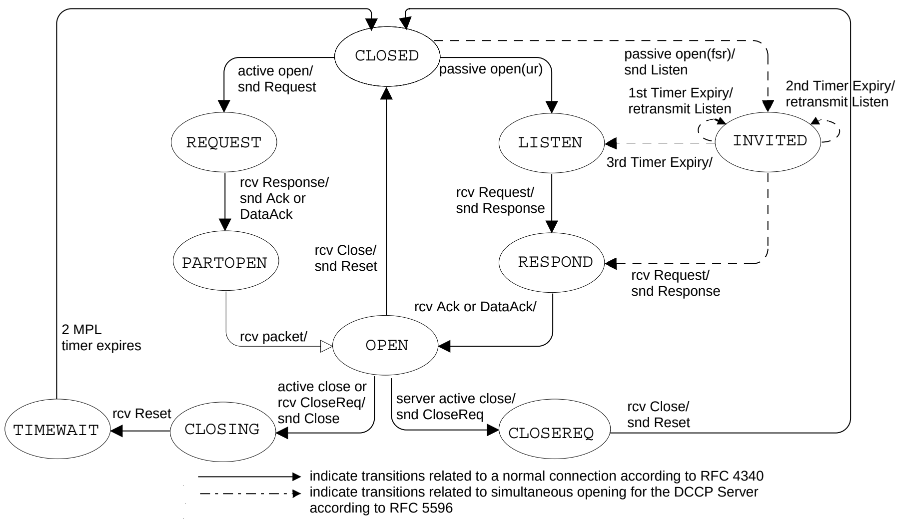


Figure 3 DCCP state diagram.

on the arc from REQUEST to PARTOPEN is "rcv Response snd Ack or DataAck". This means that when the Client receives a DCCP-Response while in the REQUEST state, it returns a DCCP-Ack or DataAck (if it has data to send) and moves to the PARTOPEN state. The Client is identified by an "active open" from its application and passing through the REQUEST state. On the other hand, the Server always receives a "passive open" and passes through the LISTEN state. Applications on both sides can issue an "active close" command but only the Server's application can issue the "server active close" command.

RFC 5596 defines a new packet type called DCCP-Listen and two new states called INVITED and LISTEN1. RFC 5596 differentiates between the cases when the Server connection end point is partially specified (the remote address and port number are unknown) and when it is fully specified. This corresponds to the commands "passive open(ur)" and "passive open(fsr)" respectively, where ur is for 'unspecified remote' and fsr stands for 'fully specified remote'. After receiving a passive open(fsr), a DCCP-Listen packet is sent, a timer is set and the Server transitions from CLOSED to INVITED. If a DCCP-request is not received in time, the DCCP-Listen packet can be retransmitted up to two times before moving to the LISTEN1 state. If the Server receives a DCCP-Request (in INVITED or LISTEN1), it sends a DCCP-Response and transitions to RESPOND. Because the behaviour of DCCPs in the LISTEN and LISTEN1 states are the same, to simplify the state diagram (Fig. 3), we suggest[2] to merge these two states. For more details of these procedures, see [9, 18].

2.2 Hole Punching Procedures

The message sequence chart in Fig. 4 provides an example of the hole punching procedure. Prior to connection establishment, we assume that both the Client and Server know each other's public address via a well-known rendezvous server (Fig. 1) using another signalling protocol such as the Session Description Protocol (SDP) [14]. As shown in Fig. 4, when the Client sends the first DCCP-Request packet via NAT-A, NAT-A creates a binding table (a

[2] This was suggested by Professor Jonathan Billington.

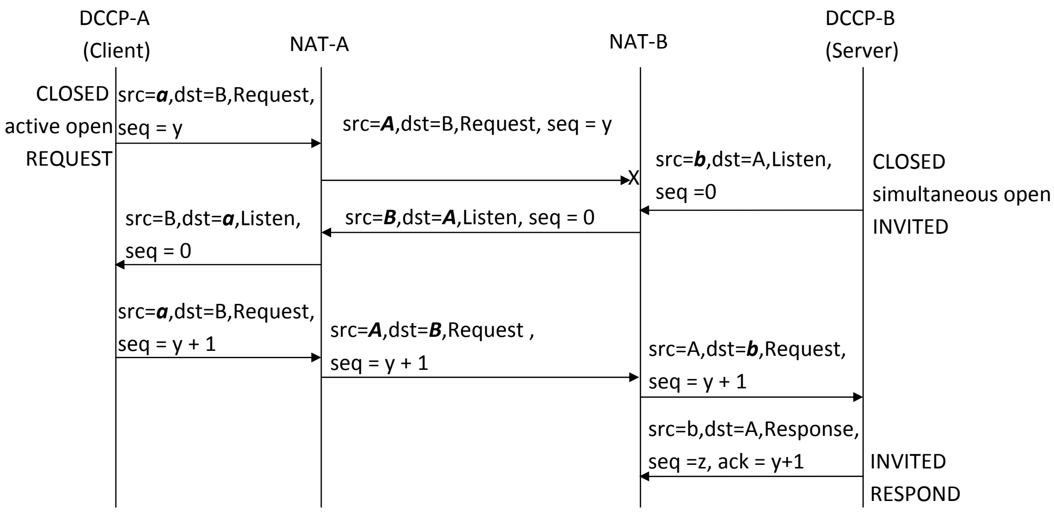


Figure 4 The hole punching procedure.

hole) and replaces the private source address "a" in the DCCP-Request with public source address "A". In this paper, a lower-case letter represents a private address and an upper-case letter represents a public address. The private source address,"a", in every outgoing packet from NAT-A is replaced by the public source address, "A". Similarly, the public destination address "A" in every incoming packet is replaced by the private address "a". However, since no binding for DCCP-B exists in NAT-B, the DCCP-Request packet is blocked by NAT-B, and discarded. In order to allow incoming packets to pass NAT-B and be delivered to DCCP-B, another hole (binding table) is required at NAT-B. As a consequence of prior signalling sessions via the Rendezvous server, DCCP-B sends a DCCP-Listen packet to indicate its willingness to set up a connection with public destination address "A". On receipt of the Listen packet, NAT-B creates a binding table so that the private source address,"b", in every outgoing packet from NAT-B will be replaced by the public source address, "B", for every packet destined for "A". Similarly, the public destination address "B" in every incoming packet from public source address, "A", will be replaced by the private address "b". Because the hole at NAT-A is already punched by the previous DCCP-Request packet, the DCCP-Listen packet can get through NAT-A and arrives at DCCP-A. When DCCP-A, in REQUEST, receives a DCCP-Listen packet, it retransmits the previous DCCP-Request with its sequence number incremented by one. The DCCP-Request is now accepted by NAT-B because "A" has the required entry in its binding table. NAT-B provides the address translation to the private address. The DCCP-request arrives at DCCP-B which sends a DCCP-Response packet and enters the RESPOND state. After that, the connection is established according to the normal connection set up procedure described in RFC 4340. Other scenarios are possible. For example, if the Listen packet is lost, DCCP-A will resend its Request packet after a timeout. Thus it is not essential for Listen packet to be received by DCCP-A, it just provides a speed-up if it gets through before the timeout occurs. It is possible for the DCCP-Listen packet to be sent before the DCCP-Request packet. In this case, the Listen packet will be blocked by NAT-A until it receives the Request packet from DCCP-A.

3 Modelling Approach

3.1 Layer Architecture

Protocols are often organized into a layered structure. Each layer represents a protocol which provides a standard interface to the lower and higher layers. From it own point of view, a specific layer may only observe the interaction at its interface so that the details of the underlying network infrastructure are hidden. Despite the fact that data flows vertically between layers at each end, we can consider that a specific layer horizontally conveys the data between the peer entities at the same layer. Thus each protocol specification at each layer needs to defines only its peer-to-peer behaviour. This peer-to-peer or end-to-end principle[3] abstracts away all lower layers and merges them into an underlying channel. We observe that almost all CPN models of the Internet protocol e.g. [1, 3–5, 12, 13, 19, 20, 22], implicitly use the end-to-end principle and hide all other underlying layers into two channel places. However, there are a few researchers who have investigated multi-layer protocols. For example, [10] modelled and validated connection establishment in the Generic Access Network which involves multiple layers of the protocol stacks. [10] suggested that studying multi-layer protocols provide us insights and understanding how protocol components interact to each other.

As the Internet technology has advanced considerably over recent years, we discover that the end-to-end principle is often violated. For example, the cross-layer design modifies interfaces to higher layers in order to provide performance optimization across layers. NAT is another example that violates the end-to-end principle. Thus NAT can not be abstracted away and its detailed model is required.

3.2 Embedding the NAT Functions in the CPN Models

Two approaches for embedding the underlying channel into a CPN protocol model have been discussed in [6]. The first approach integrates the channel model with the protocol entities. Applying this to our work, the channel model is the NAT functions that are implemented on the output arc inscriptions of the protocol entities. Although this approach helps to reduce the state space size, the model is subtle and tedious. The second approach embeds the channel model or the NAT functions as a module implemented by a substitution transition [15]. This is an elegant way of including NAT devices in the model. This modular approach requires two more substitution transition instances (NAT) and four more buffer places than the integrate approach does. As discussed in [6], from the analysis perspective, this modular approach significantly suffers from state explosion. However for sake of modelling clarity we have selected the modular approach.

4 DCCP Simultaneous Open CPN Model

DCCP simultaneous open CPN models have been developed using both CPN Tools and Design/CPN. Prioritized transitions play an important role in this paper so this section only examines the CPN Tools model. Since our model is extended from [5], this section emphasises on the extension part of the model. For more details of the declarations and the explanation of the previous work, see [5, 25]

[3] "End-to-end principle is an assumption of the Internet property that all nodes can send packets to other nodes of the network, without requiring intermediate network elements to further interpret them."

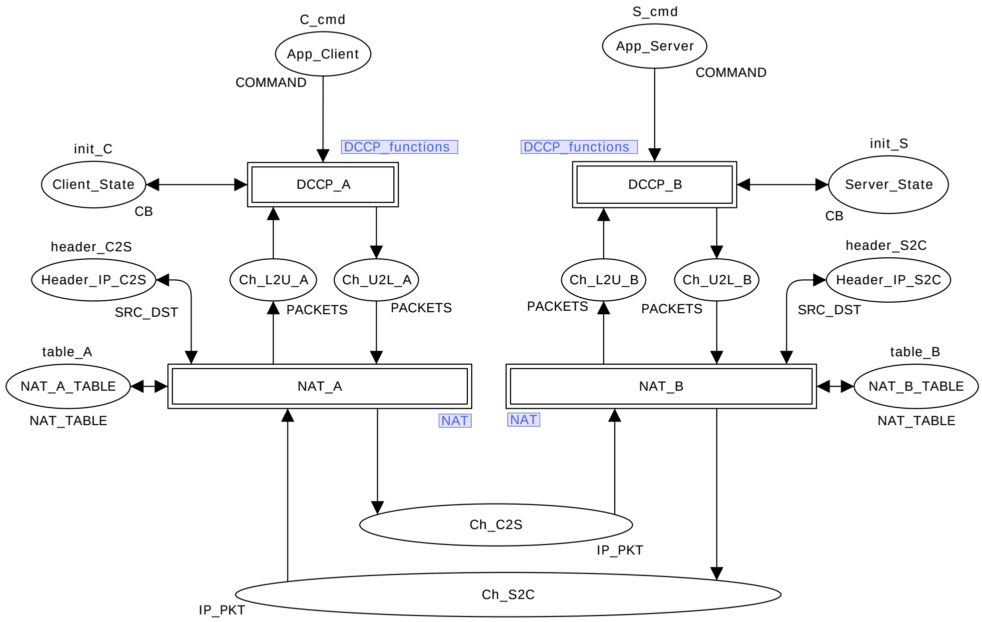


Figure 5 DCCP Top level.

4.1 Model Overview

Our procedure based DCCP-CPN model from Section 4 of [5] has been extended to incorporate the network layer comprising two Network Address Translators (NATs). In spite of the existence of many types of NAT, this paper investigates only "Address and Port-Dependent Mapping[4]". Because the NATs are embedded as a module (substitution transition), another type or combination of different types can be easily integrated in our model. Our procedure based CPN model comprises five hierarchical levels. The complete model comprises 14 places, 68 executable transitions and 25 ML functions.

Figure 5 shows the top level of our CPN model. Two places, App_Client and App_Server, typed by COMMAND (line 15 of Fig. 6), store tokens representing user commands. Substitution transitions, DCCP_A and DCCP_B, represent the DCCP procedures in the Client and the Server, respectively. Substitution transitions, NAT_A and NAT_B, which link to the second level CPN subpage, NAT, models the IP-Port address mapping procedure. Strictly speaking, we do not actually model the hole punching procedure because the hole punching behaviour automatically emerges from interactions among four component in the network: DCCP-A, NAT-A, NAT-B and DCCP-B.

4.2 Declaration of State Variables

DCCP states and variables are stored in Places Client_State and Server_State typed by CB (Control Block). Two new states: LISTEN1 and INVITED are specified by RFC 5596. Figure 6 defines CB (line 10) as the union of four colour sets: IDLE (for CLOSED, LISTEN, LISTEN1 and TIMEWAIT states), RCNT (for INVITED state), RCNTxGSSxISSxlisten_flag

[4] " The NAT reuses the port mapping for subsequent packets sent from the same internal IP address and port to the same external IP address and port" [2]

```
 1: (*         Retransmit Counter          *)
 2: colset RCNT         = int;
 3: colset ACTIVE_STATE = with  RESPOND | PARTOPEN | S_OPEN | C_OPEN
 4:                           | CLOSEREQ | C_CLOSING |S_CLOSING;
 5: colset IDLE         = with CLOSED_I | LISTEN | TIMEWAIT | CLOSED_F | LISTEN1;
 6: colset RCNTxGSSxISSxlisten_flag = product RCNT*SN48*SN48*BOOL;
 7: colset GS                       = record GSS:SN48*GSR:SN48*GAR:SN48;
 8: colset ISN                      = record ISS:SN48*ISR:SN48;
 9: colset ActiveStatexRCNTxGSxISN  = product ACTIVE_STATE*RCNT*GS*ISN;
10: colset CB = union IdleState:IDLE
11:                           + INVITED:RCNT
12:                           + ReqState:RCNTxGSSxISSxlisten_flag
13:                           + ActiveState:ActiveStatexRCNTxGSxISN;
14: (*        User Command        *)
15: colset COMMAND = with simu_Open | p_Open | a_Open | server_a_Close | a_Close;
```

Figure 6 The definition of CB (Control Block) and COMMAND.

(for REQUEST state), and ActiveStatexRCNTxGSxISN (for RESPOND, PARTOPEN, OPEN, CLOSEREQ and CLOSING states). INVITED in the union coloured set CB (line 10) is distinguished from others because this state stores only a retransmission counter. LISTEN1 is declared in the colour set IDLE (line 5). The Client's action, in the REQUEST state, depends whether it has ever received a DCCP-Listen or not. Thus a boolean flag is added in the state variables (line 6).

4.3 Declaration of DCCP and IP Packets

DCCP entities communicate with NATs via buffer places, Ch_L2U_A, Ch_U2L_A, Ch_L2U_B and Ch_U2L_B typed by PACKETS. Two substitution transitions, NAT_A and NAT_B, exchange IP packets via two buffer places, Ch_S2C and Ch_C2S, typed by IP_PKT. Figure 7 declares PACKETS (line 22) as the union of four colour sets: SN48 (for DCCP-Request), SN48 (for DCCP-Listen), SN (for DCCP-Data), Ack_DataAckPacket and OtherPackets. The new packet type defined by RFC 5596 is DCCP-Listen which always has the sequence number equal to zero. The Request, Listen and Data packets are distinguished from the others by ML selectors of the same name as defined in line 22. Figure 7 declares IP_PKT (line 28) as a record of three colour sets: IP (for source address), IP (for designation address) and PACKETS (for DCCP packets). IP are defined as a product of five integers instead of four integers because the port address is also included.

4.4 CPN Subpage NAT

Apart from input and output buffer places, subpage NAT comprises two places and two transitions. Place src_dst typed by SRC_DST stores a record of private source address and public designation address. Transition NAT_TX views the token {src=a, dst=B} together with the token packet forming an incoming IP packet from the private network. Transitions NAT_TX and NAT_RX the priority value, P_HIGH = 100, while P_NORMAL is equal to 1000. Place TABLE typed by NAT_TABLE stores binding tables used for address translations. NAT_TABLE is defined in Fig. 7 (line 26) as a record of three tuples: private source address, public source address and public designation address. When creating a binding table, function put(a) is used to set up the public source address.

4.5 Connection Establishment Pages

This section illustrates two CPN subpages which model connection establishment, the Server and Client pages. Initially, both entities are CLOSED with a simultaneous open command

```
(*       Sequence and Acknowledgement Numbers      *)
colset SN48       = int with 0..MaxSeqNo48;
colset SN24       = int with 0..max_seq_no24;
colset SN48_AN48 = record SEQ:SN48*ACK:SN48;
colset SN24_AN24 = record SEQ:SN24*ACK:SN24;
colset SN         = union longSN:SN48  + shortSN:SN24
colset SN_AN      = union longSA:SN48_AN48 + shortSA:SN24_AN24
(*      Sequence and Acknowledgement Variables     *)
var sn:SN;        var sn48:SN48;           var sn24:SN24;
var sn_an:SN_AN; var sn48_an48:SN48_AN48; var  sn24_an24:SN24_AN24;
(*  Define the DCCP Packet Structure *)
colset Ack_DataAckPktTypes = with Ack | DataAck;
var    ack_dataack:Ack_DataAckPktTypes;
colset OtherPktTypes       = with Sync | SyncAck | Response | CloseReq | Close | Rst;
var p_type:OtherPktTypes;
colset Ack_DataAckPacket = product Ack_DataAckPktTypes*SN_AN;
colset OtherPackets       = product OtherPktTypes*SN48_AN48;
colset PACKETS            = union Request:SN48 + Listen:SN48 +  Data:SN
               + Ack_DataAck:Ack_DataAckPacket + PKT:OtherPacket
(*  Define the IP Packet Structure *)
colset IP         = product INT*INT*INT*INT*INT;
colset NAT_TABLE = record local_src:IP*global_src:IP*global_dst:IP;
colset SRC_DST    = record src:IP*dst:IP;
colset IP_PKT     = record src_add:IP*dst_add:IP*dccp:PACKETS;
var packet:PACKETS;
var a, A, B, gb_src:IP;
```

Figure 7 The definition of DCCP PACKETS and IP_PKT.

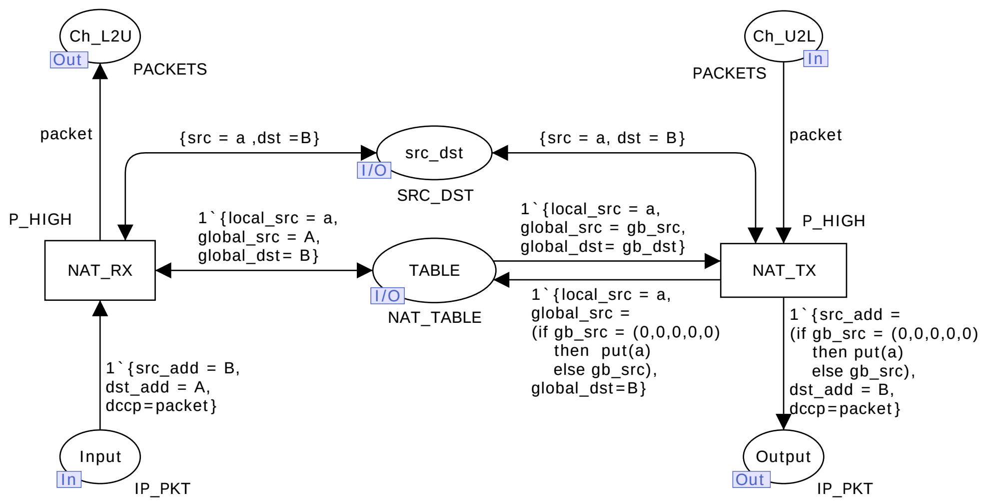


Figure 8 CPN Subpage NAT.

(1‘simu_Open) in Place App_Server and an active open command (1‘a_Open) in Place App_Client.

4.5.1 Server Page

The part of Fig. 9 below App_Server, is the normal connection establishment specified in RFC 4340. The upper part is the standard simultaneous open procedure specified in RFC 5596. With reference to Fig. 3, the occurrence of transition simuOpen (Fig. 9) transmits DCCP-Listen and puts the Server in the INVITED state, waiting for DCCP-Request from the Client.

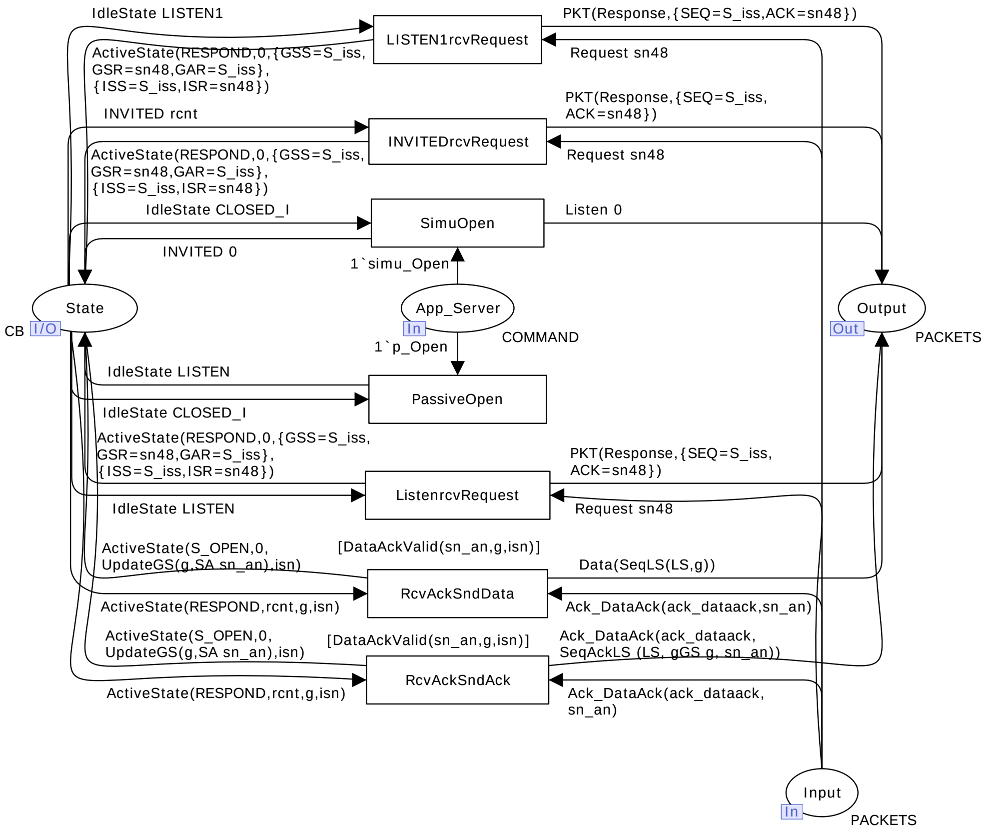


Figure 9 DCCP Server.

After retransmitting twice , the Server enters the LISTEN1 state. These actions are modelled in other CPN subpages: Retransmission and BackOffFails pages. When the Server, in either INVITED, LISTEN1 or LISTEN, receives a DCCP-Request (transition INVITEDrcvRequest, LISTEN1rcvRequest, LISTENrcvRequest) it replies with a DCCP-Response containing the Server's initial sequence number and an acknowledgement for the DCCP-Request. It enters the RESPOND state and appropriately initialises its state variables. These upper three transitions are directly related to the state diagram in Fig. 3.

4.5.2 Client Page

The transition RcvListen in Fig. 10 models actions specified by RFC 5596. On receipt of the DCCP-Listen(seq=0), if the Client has never received DCCP-Listen, it replies with DCCP-Request. If the Client has received DCCP-Listen before, it silently discards the DCCP-Listen.

5 Analysis Approach

A typical approach to alleviate the state explosion problem is to make the number of generated states more compact. We observe that *after writing* the address translation table, NAT in our *specification* model performs only two functions, reordering and forwarding the packets. Intuitively the CPN model of the underlying layer and NAT can be combined and reduced into two channel places. Thus, the outgoing packet from a DCCP entity is immediately the incoming packet to the other. However the NAT cannot be abstracted away because its behaviour *before writing* the address translation table is different.

By separating the actions before and after writing the address translation table, we suggest that transitions NAT_RX and NAT_TX in the NAT page should get the highest priority. When NAT reorders packets, many sequences of these actions (reorder) lead to the same markings. Analysis using *prioritized transitions* will keep one sequence but discard the rest. Thus, the number of total states is significantly reduced and the safety properties (terminal markings) are preserved.

5.1 CPN Tools versus Design/CPN

Previously, our model [25] was created and maintained using Design/CPN. Because Design/CPN does not support prioritized transitions, we switch to CPN Tools instead. Although using prioritized transitions can reduce the state space significantly, the CPN Tools can generate full state spaces of our model for only a few scenarios. To gain more confidence in the specification RFC 5596, analysis of more scenarios is required. A technique that was successfully used to analyse the DCCP connection management CPN models in [25] is the sweep-line technique. We also wish to apply the sweep-line technique to analyse the DCCP simultaneous open properties. Unfortunately CPN Tools, which support prioritized transitions, do not support sweep-line library. On the other hand, Design/CPN has the sweep-line library but does not support prioritized transitions.

5.2 Prioritized Transitions versus Timed Models

To circumvent the problem in Design/CPN, prioritized transitions are imitated using a timed token enabling all transitions in the DCCP layer. Enabling transitions in NAT layer does not require a timed token. When any transition in the DCCP layer is fired, the time stamp

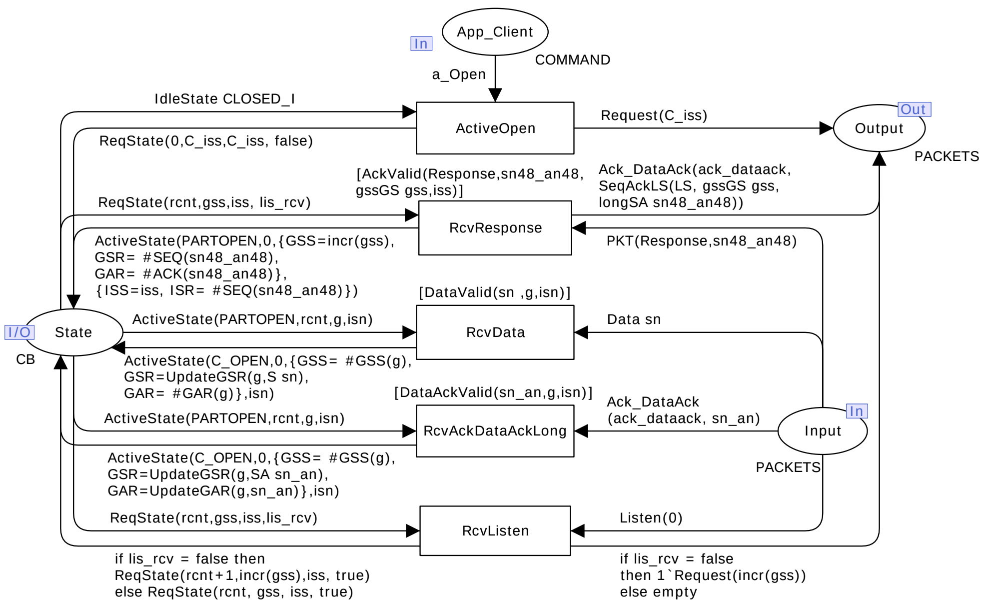


Figure 10 DCCP Client.

```
(*  The Initial State of NAT_A and NAT_B  *)
val header_C2S = 1‘{src=(10,0,0,1,4321), dst=(138,76,29,7,31000)};
val header_S2C = 1‘{src=(10,1,1,3,4321), dst=(155,99,25,11,62000)};
val table_A=
  1‘{global_src=(0,0,0,0,0), global_dst=(0,0,0,0,0), local_src=(10,1,2,3,4322)}
++1‘{global_src=(0,0,0,0,0), global_dst=(0,0,0,0,0), local_src=(10,0,9,1,4361)}
++1‘{global_src=(0,0,0,0,0), global_dst=(0,0,0,0,0), local_src=(10,0,0,1,4321)};
val table_B=
  1‘{global_src=(0,0,0,0,0), global_dst=(0,0,0,0,0), local_src=(10,1,1,3,4321)}
++1‘{global_src=(0,0,0,0,0), global_dst=(0,0,0,0,0), local_src=(10,2,9,1,5321)}
++1‘{global_src=(0,0,0,0,0), global_dst=(0,0,0,0,0), local_src=(10,0,6,1,4341)};
```

Figure 11 The Initial State of NAT_A and NAT_B.

in the token advances one step. Because the global clock is less than the time stamp by one step, all transitions in NAT layer (if any) have to finish firing before the global clock advances and the transitions in the DCCP layer can be enable. Thus the transitions in the NAT layer have higher firing priority than every transition in the DCCP layer. This imitated method has a drawback that the timed state space is always larger because the global clock and time stamps contribute to the presence of new states. Increasing state space sizes seems to be the wrong path because it encourages state explosion. However [25] demonstrated that if a new additional variable, such as time stamp, is used as progress measure for the sweep-line analysis, in spite of a larger state space size, the peak memory used and exploration time can be significantly reduced. Finally we analyse the augmented model similar to the Sweep-line analysis in [25]. The experimental results are discussed in section 6.2.

6 Experimental Results

This section contains analysis results for the DCCP simultaneous open procedures when operating over *reordering channels without loss*. In contrast to the previous work that considers various cases according the combination of user commands. This paper investigates only the simultaneous open scenario when the Client user issues an “active open” and the Server user issues a “simultaneous open” command. The initial markings of all buffer and channel places are empty. The initial state of both side are CLOSED and the initial send sequence number (ISS) on both sides is set to 10. The initial markings in Places Header_IP_C2S, Header_IP_S2C, NAT_A_TABLE and NAT_B_TABLE are specified in Fig. 11. Without loss of generality, only long sequence numbers are used. All experiments are conducted on a AMD 9650 2.31GHz PC with 4 GByte RAM. CPN Tools runs on Window XP while Design/CPN runs on Fedora Core version 6.

6.1 The Prioritized Transition Model

Table 1 illustrates the experimental results when we use prioritized transitions and analyse the model by CPN Tools. The first column (*Config.*) in this table defines the configuration being analysed, where the 3-tuple represents the maximum number of retransmissions allowed for Request, Listen and Ack packets respectively. Columns *total nodes* and *total arcs* record the total number of markings and arcs in the state space, respectively. The time (hours:minutes:seconds) to generate the full state space is given in Column *time*. The next two columns (*DMs*) records the number of dead markings. Dead markings are classified into type I and type II. Type I dead markings are desirable and correspond to successful connection establishment where both the Client and Server are in the OPEN state. In Type II dead markings both the Client and Server are in still CLOSED state. Both types are

Table 1 DCCP simultaneous open using Prioritized Transitions.

Config.	total nodes	total arcs	time	DMs I	DMs II	Bounds Ch L2U_B	Bounds Ch L2U_A
(0,0,0)	16,441	28,308	00:00:43	13	1	3	5
(0,0,1)	78,360	141,749	00:10:36	33	1	4	5
(0,1,0)	24,579	43,612	00:01:23	13	1	3	6
(0,1,1)	117,264	217,964	00:22:00	33	1	4	6
(0,2,0)	32,736	58,952	00:01:58	13	1	3	7
(0,2,1)	156,187	294,215	00:37:47	33	1	4	7

Table 2 DCCP simultaneous open using the sweep-line method with the augmented model.

Config.	Sweep-line with the augmented model: total nodes	total arcs	peak nodes	time	DMs I	DMs II	Bounds Ch L2U _B	Bounds Ch L2U _A	% space
(0,0,0)	40,984	65,463	288	00:00:29	26	14	3	5	1.75
(0,0,1)	279,581	469,298	1,080	00:02:33	66	19	4	5	1.38
(0,1,0)	81,531	135,246	496	00:00:50	39	16	3	6	2.02
(0,1,1)	557,615	967,911	2,059	00:07:34	99	21	4	6	1.76
(1,0,0)	2,896,471	4,921,848	3,142	00:38:27	148	24	4	6	-
(1,0,1)	34,412,454	60,468,592	17,908	09:29:13	360	30	5	6	-
(1,1,0)	5,770,971	10,105,648	5,810	01:22:53	222	26	4	7	-
(1,1,1)	68,581,787	123,703,372	34,892	20:57:25	540	32	5	7	-
(0,2,0)	135,454	229,717	794	00:01:14	52	18	3	7	2.43
(0,2,1)	927,819	1,642,398	3,347	00:09:04	132	23	4	7	2.14
(0,2,2)	6,719,017	12,943,167	16,034	00:01:51	236	29	5	8	-
(1,2,0)	9,596,365	17,103,716	9,486	01:42:07	296	28	4	8	-
(1,2,1)	114,060,085	208,918,444	57,427	36:58:00	720	34	5	8	-

expected dead markings. All dead markings have no packets left in all buffers and channels. The last two columns, *Bounds*, record the maximum number of packets that can occur in the channel places Ch_L2U_B and Ch_L2U_A.

6.2 Analyses the Timed Model using the Sweep-line Method

Using prioritized transitions reduces the state space sizes significantly but we can analyse only six scenarios. When we attempt to analyse the scenarios (1,0,0) , (0,2,2) and (0,1,2), the available memory is exhausted. As discuss in Section 5, we turn to the sweep-line technique (with the augmented model). Table 2 illustrates the experimental results when the sweep-line is applied to the timed CPN model. We use the progress vector suggested in Section 4.5 of [25] together with the time stamp. Conducting search experiments, we discover that the best position of the time stamp in the progress vector is at the end of the list.

Column *peak nodes* in Table 2 lists the peak number of nodes stored in main memory at any one time. Column *time* records the time used to explore the state space. The last column (*% space*) of Table 2 shows the ratio of the number of peak states compared to the total number of states in Table 1. The smaller the number, the more efficient the sweep-line algorithm is. The number of peak states is reduced to only 1–2% of the full untimed state space. This analysis method has potential to explore more scenarios.

7 Conclusions and Future Work

This paper has presented a Coloured Petri Nets model and analysis of DCCP simultaneous open procedure. Our CPN model is developed based on both RFC 4340 and RFC 5596.

Because NATs with the hole punching procedure affect DCCP behaviour, they cannot be simply abstracted away using the layered architecture. We suggest to separate NAT operations into before and after writing the address translation table and remove some transition occurrences using prioritized transitions. It is possible to use the timed model to imitate prioritized transitions. Analysing the timed models using Sweep-line method is more efficient than generating full state space of the prioritized transitions models

In future, we are interested in modelling different types of NATs, and increasing the number of protocol entities. Instead of studying functional behaviour, we wish to investigate performance behaviour of each protocol entity as well.

Acknowledgments. This work is supported by Research Grant from the Thai Network Information Center Foundation and the Thailand Research Fund. The author is thankful to Professor Jonathan Billington, Professor Lar M. Kristensen and the anonymous reviewers. Their constructive comments have helped to improve the quality of this paper.

References

1 *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 1999.

2 F. Audet and C. Jennings. Network Address Translation (NAT) Behavioral Requirements for UNicast UDP RTP: A Transport Protocol for Real-Time Applications, RFC 4787. Available via http://www.rfc-editor.org/rfc/rfc4787.txt, January 2007.

3 J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 210–290. Springer, Heidelberg, 2004.

4 J. Billington and B. Han. Modelling and Analysing the Functional Behaviour of TCP's Connection Management Procedures. *International Journal on Software Tools for Technology Transfer*, 9(3-4):269–304, June 2007. Available via http://dx.doi.org/10.1007/s10009-007-0034-1.

5 J. Billington and S. Vanit-Anunchai. Coloured Petri Net Modelling of an Evolving Internet Standard: the Datagram Congestion Control Protocol. *Fundamenta Informaticae*, 88(3):357–385, 2008.

6 J. Billington, S. Vanit-Anunchai, and G. E. Gallasch. Parameterised Coloured Petri Nets Channel Models. In *Transactions on Petri Nets and Other Models of Concurrency*, volume 5800 of *Lecture Notes in Computer Science*, pages 71–97. Springer, Heidelberg, 2009.

7 S. Christensen, L.M. Kristensen, and T. Mailund. A Sweep-Line Method for State Space Exploration. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *Lecture Notes in Computer Science*, pages 450–464, Genova, Italy, 2-6 April 2001. Springer, Heidelberg.

8 Design/CPN Online. http://www.daimi.au.dk/designCPN/.

9 G. Fairhurst. Datagram Congestion Control Protocol (DCCP) Simultaneous-Open Technique to FacilitateNAT/Middlebox Traversal, RFC 5596. Available via http://www.rfc-editor.org/rfc/rfc5596.txt, September 2009.

10 P. Fleischer and L. M. Kristensen. Formal Specification and Validation of Secure Connnection Establishment in a Generic Access Network Scenario. In *Proceedings of ICATPN'08*, volume 5062 of *Lecture Notes in Computer Science*, pages 153–171. Springer, Heidelberg, 2008.

11 S. Floyd, M. Handley, and E. Kohler. Problem Statement for the Datagram Congestion Control Protocol (DCCP), RFC 4336. Available via http://www.rfc-editor.org/rfc/rfc4336.txt, March 2006.

12 S. Gordon. *Verification of the WAP Transaction Layer uisng Coloured Petri Nets.* PhD thesis, Institute for Telecommunications Research and Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia, November 2001.

13 B. Han. *Formal Specification of the TCP Service and Verification of TCP Connection Management.* PhD thesis, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia, December 2004.

14 M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol, RFC 4566. Available via http://www.rfc-editor.org/rfc/rfc4566.txt, July 2006.

15 K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer, Heidelberg, 2nd edition, 1997.

16 K. Jensen and L.M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems.* Springer, Heidelberg, 2009.

17 E. Kohler, M. Handley, and S. Floyd. Designing DCCP: Congestion Control Without Reliability. In *Proceedings of the 2006 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'06)*, pages 27–38, Pisa, Italy, 11-15 September 2006.

18 E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol, RFC 4340. Available via http://www.rfc-editor.org/rfc/rfc4340.txt, March 2006.

19 L. M. Kristensen and K. Jensen. Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad Hoc Networks. In *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 248–269. Springer, Heidelberg, 2004.

20 L. Liu. *Towards Parametric Verification of the Capability Exchange Signalling Protocol.* PhD thesis, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia, May 2006.

21 T. Mailund. *Sweeping the State Space - A Sweep-Line State Space Exploration Method.* PhD thesis, Department of Computer Science, University of Aarhus, February 2003.

22 C. Ouyang. *Formal Specification and Verification of the Internet Open Trading Protocol using Coloured Petri Nets.* PhD thesis, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia, June 2004.

23 S. Vanit-Anunchai. *An Investigation of the Datagram Congestion Control Protocol's Connection Management and Synchronisation Procedures.* PhD thesis, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia, November 2007.

24 S. Vanit-Anunchai and J. Billington. Modelling the Datagram Congestion Control Protocol's Connection Management and Synchronisation Procedures. In *Proceedings of the 28th International Conference on Application and Theory of Petri Nets and other models of concurrency (ICATPN'07)*, volume 4546 of *Lecture Notes in Computer Science*, pages 423–444, Siedlce, Poland, 25-29 June 2007. Springer, Heidelberg.

25 S. Vanit-Anunchai, J. Billington, and G.E. Gallasch. Analysis of the Datagram Congestion Control Protocol's Connection Management Procedures using the Sweep-line Method. *International Journal on Software Tools for Technology Transfer*, 10(1):29–56, 2008. Available via http://dx.doi.org/10.1007/s10009-007-0050-1.