Validating DCCP Simultaneous-Open and Feature Negotiation Procedures

Somsak VANIT-ANUNCHAI^{†a)}, Member

SUMMARY This paper presents the formal analysis of the feature negotiation and connection management procedures of the Datagram Congestion Control Protocol (DCCP). Using state space analysis we discover an error in the DCCP specification, that result in both ends of the connection having different agreed feature values. The error occurs when the client ignores an unexpected Response packet in the OPEN state that carries a valid Confirm option. This provides an evidence that the connection management procedure and feature negotiation procedures interact. We also propose solutions to rectify the problem.

key words: coloured Petri nets, state space analysis, prioritized transitions, pseudo-code, RFC 4340, RFC 5596

1. Introduction

An Internet standard, called the Datagram Congestion Control Protocol (DCCP) [13], [15], has been introduced by the Internet Engineering Task Force (IETF) as RFC4340 in 2006. It 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 [7]. 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.

Experience with the Transmission Control Protocol (TCP) taught us that it is very difficult to design a protocol that works correctly. Because DCCP is a lot more complicated than TCP, it is very likely to have errors. During the development of RFC4340 we used Coloured Petri Nets (CPNs) to model and analyse DCCP connection management [23]. These CPN models helped us to understand the protocol's behaviour and discovered several bugs in the draft

specifications [5], [13], [14], [16]. The solutions to those bugs have already been incorporated into RFC4340 [15].

Apart from the reliable connection management, both sides must choose congestion control mechanisms and agree upon the same CCID. This requires a reliable negotiation procedure called *Feature Negotiation* (FN) which is also specified in RFC4340. If both sides are not aware of reaching an agreement with different CCIDs, the situation will be very harmful and currently there is no recovery mechanisms. Our work in [19] revealed that without DCCP connection management model, the simultaneous negotiation procedure could be broken even on a FIFO lossless channel. We also suggested two solutions to fix the problem. After applying our suggest changes, the analysis of the revised models showed that the undesired errors were eliminated.

While the DCCP connection management and the feature negotiation were separately modelled and validated in [17]–[19], [22], this paper pursues a further step by combining both procedures together. Thus, the complexity of the model and the probability of errors increase significantly. This provides the motivation for validating both procedures working together. Because our research lays on the interplay of model readability, state explosion and finding errors, combining both procedures worsens both the model readability and the state explosion problem. To alleviate the state explosion, a number of techniques have been proposed such as symmetry and equivalence [11], partial order [3], and sweep-line methods [22]. This paper proposes to reuse the CPN models from the previous work [17], [19], [22] to ensure the model correctness and adopt the prioritized transitions to make the model analyzable and alleviate the state explosion.

This paper provides three contributions. Firstly, we discuss decision rationale on the choice of the model structure and the conditions of underlying channels. Secondly, we discover another error in RFC4340 that in the OPEN state the client and the server have different agreed feature values. We believe that these unexpected different agreed feature values, especially of Congestion Control Identification (CCID), are very harmful and currently there is no recovery mechanism. Thirdly, we discover an evidence of interactions between connection management and feature negotiation procedures. Thus, both procedures cannot be separately validated.

This paper is organised as follows. Section 2 summarizes the overview of DCCP connection management. Section 3 describes the DCCP feature negotiation procedure.

Manuscript received August 29, 2016.

Manuscript revised December 18, 2016.

Manuscript publicized March 7, 2017.

[†]The author is with the School of Telecommunication Engineering, Suranaree University of Technology, Nakhon Ratchasima, 30000, Thailand.

a) E-mail: somsav@sut.ac.th

DOI: 10.1587/transinf.2016FOP0012

Fig. 1 DCCP state diagram.

Related work is discussed in Sect. 4. Section 5 describes our previous work. Section 6 discusses the desired properties of the protocol. Section 7 presents the modelling decision and the overview of our CPN model. Section 8 presents initial configurations and discusses analysis result. Section 9 discusses the conclusion and future work.

2. DCCP Connection Management Procedures

The Connection Management (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 1 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. 1 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 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 ad-

dress 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. 1), we suggest[†] to merge these two states. For more details of these procedures, see [6], [15].

3. Feature Negotiation Procedure

DCCP allows both the client and the server to negotiate their parameters called *features* using the options field. The feature negotiation can happen any time but is typically done during connection establishment. Each entity can initiate the negotiation of two kinds of feature numbers: *local features* (L)-the initiator's features and *remote features* (R)-the other side's features. Four particular options are dedicated to feature negotiations; Change L, Confirm L, Change R and Confirm R.

Currently, RFC 4340 defines two reconciliation rules: server priority and non-negotiable. Figure 2 and 3 show typical message sequence charts of each rule.

(1) The server priority rule

This rule is applied when the feature value is a fixed-length byte string. During negotiation, DCCP entity keeps an ordered preference list of the feature values. The initiator

[†]This was originally suggested by Professor Jonathan Billington.

Fig. 2 An example of the feature negotiation: the server priority.

Fig. 3 An example of the feature negotiation: the non-negotiable.

Fig. 4 DCCP feature negotiation state diagram - redrawn from [15].

sends a Change option containing its preference list. The receiver responds with the Confirm option containing an agreed value followed by its preference list. Thus the agreed value will appear twice in the Confirm option. The agreed value is defined as the first element in the server's list that matches any element in the client's list. If there is no match, the agreed value remains the existing feature value.

(2) Non-negotiable rule

The Change and Confirm options under this rule contain only one feature value which is a byte string. After receiving the Change L from the feature local, the feature remote must accept the valid value and reply with Confirm R containing this value. If the received feature value is invalid, the feature remote must send an empty Confirm R. This non-negotiable rule must not be used with Change R and Confirm L options.

3.1 Finite State Machines

The feature negotiation procedures are represented by state diagrams. Figure 4 shows the state diagram for *feature local*. It comprises three states: STABLE; CHANGING; and UNSTABLE. The entity in the STABLE state always knows its feature value and expects the other to agree on the same value. When the local receives Change R, it calculates a new agreed value and replies Confirm L. On the other hand the Confirm R received will be discarded.

After the entity in STABLE sends the first Change L command, it enters the CHANGING state and goes back to the STABLE state upon receiving a Confirm R or a empty

Confirm R. If the local in CHANGING does not get a reply from the other side, it keeps retransmitting the Change L option.

When the preference list is changed by its user while the entity is in the CHANGING state, it enters the UNSTABLE state. Here it ignores the on-going negotiation but starts a new negotiation by sending a Change command with the new preference list before going back to the CHANGING state.

The state diagram for *feature remote* can be obtained by interchanging *Ls* and *Rs* in Fig. 4. Thus each entity consists of three state machines working together: connection management, feature local and feature remote. It is possible that one side initiates Change L while the other side initiates Change R of the same feature. According to Fig. 4 when the local in CHANGING receives Change R, it computes a new agreed value and replies with a Confirm L. This situation is called *simultaneous negotiation*. The specification also allows the preferences to be changed any time.

3.2 Important Rules of Feature Negotiation

Although the feature negotiation procedures explained in the previous section sound simple, the real situation may be very complex when packets are reordered and lost. Moreover, the negotiation for the same feature can be simultaneously initiated by both sides and the preference lists can be changed any time. To cope with this, the RFC specifies some rules intended to provide reliable signalling so that both sides reach agreement on the feature value.

3.2.1 Non-Reordered Change and Confirm Options

The RFC specifies that any Change and Confirm options in packets that do not arrive in strictly increasing order must be ignored. According to the related pseudo-code and algorithms, the strictly increasing order rule is only enforced for packets that contain the Change and Confirm options. An ordered packet with the Change and/or Confirm options may have a sequence number less than GSR if the later packets do not contain any Change or Confirm options.

In order to check the order of arrival, the RFC specifies another two variables: Feature Greatest Sequence Number Received (FGSR) and Feature Greatest Sequence Number Sent (FGSS). If the received packet's sequence number is less than or equal to FGSR, Change or Confirm options received must be ignored. If the acknowledgement number is less than FGSS or the packet contains no acknowledgement, the Confirm option received must be ignored.

Because DCCP-Data with short sequence numbers is vulnerable to attack, any option attached to DCCP-Data that might cause the connection to be reset shall be ignored. Thus both Change and Confirm options received in DCCP-Data must be ignored in all circumstances. FGSR is updated when the entity receives a valid packet containing non-reordered Change or Confirm options. FGSS is updated when the entity sends a Change option during a transition

from STABLE or UNSTABLE to CHANGING.

3.2.2 Retransmission

Change options must be retransmitted when the sender does not receive a non-reordered Confirm option within a specific period. The Confirm option must be generated only when a non-reordered Change option is received. Retransmission of options may be achieved by either generating a new packet (DCCP-Ack or DCCP-Sync) or by including the appropriate option field in a packet that is about to be transmitted. Retransmission continues until a non-reordered Confirm option is received or the connection is closed.

4. Related Work

Regarding the work on the formal analysis of DCCP done by others, As far as we know there is only one unpublished formal analysis of DCCP draft specification version 5 by Handley [5] using Label Transition System Analyser (LTSA). The rest is done by us [1], [8], [17]–[19], [22], [23]. Although our work is influenced by [9], they are quite different. DCCP is a transport protocol but it is designed for unreliable datagram and delay sensitive applications. Thus the protocol prefers the corrupted or loss rather than delayed messages. Due to this requirement, the DCCP's mechanism is very different from others'. Comparing with TCP, DCCP is much more complex. For instance, when the connection establishment with one retransmission without feature negotiation, DCCP's state space size is 14,782 nodes while TCP has only 438 nodes. This complexity mainly comes from the re-synchronization process and the role of Reset packets.

5. Previous Work

In [21], we reported our experience with the incremental enhancement and iterative modelling of the DCCP connection management procedures as the DCCP specification was developed. Insight into the decisions behind the modelling choices can also be found in [21]. The full CPN specification of the connection management procedures can be found in [22]. We have published an enhanced version of [21] which also discusses a procedure-based model of DCCP's connection management procedures in [1]. In [2], we discuss how to embed a parameterised channel into CPN models of protocols, using DCCP as an example. In [17], we extended the work in [18], [22] by including DCCP simultaneous open procedure (RFC 5596) and Network Address Translators (NAT) into the model. This work involves protocols both network and transport layers. The feature negotiation procedure of DCCP has been modelled and analysed in [19].

6. Desired Safety Properties for DCCP Connection Management and Feature Negotiation

This section specifies the desired safety properties for the

DCCP feature negotiation and connection management procedures. Suggested by [10] three important safety properties are: proper terminal states and absence of deadlocks; absence of livelocks.

Property 1. The proper terminal states of DCCP feature negotiation and connection management procedures are

(I) Unsuccessful Establishment- After the client requests a connection be opened and the server accepts the connection, at the terminal states the client is in CLOSED and the server is in either CLOSED or LISTEN states.

(II) Successful Establishment but Unfinished Negotiation- Because of the unspecified number of retransmission for the Change option, it may be possible that after the connection has established, either side is in the CHANGING or UNSTABLE at terminal states.

(III) Successful Establishment and Negotiation- After the client requests a connection be opened and the server accepts the connection, both sides are in OPEN and STABLE at terminal states. Both sides must have the same agreed feature value.

(IV) Successful Abort- After either side requests a connection be closed, at terminal states both sides are in CLOSED.

Property 2. (Absence of Deadlocks) There must be no deadlocks. There must be no other terminal states other than described in Property 1.

Property 3. (Absence of Livelocks) There must be no Livelocks. The livelocks is when the protocol repeatedly executes without effective progress with respect to the aim of the procedure.

As a result of the violation of any above safety properties, the protocol can fail. By intuition or manual inspection, it is very difficult to be detected the violation. However the facilities in the occurrence graph tools in CPN Tools can easily detect the errors. The livelocks can be easily detected by generating the Strong Connected Component (SCC) graph of the state space using occurrence graph tool. If the state space has the same number of nodes and arcs as its SCC graph, the state space is isomorphic to its SCC and contain no livelocks.

The main focus of this paper is to validate the feature negotiation procedure. We do not expect to have any negotiation during the connection closing. Thus in this paper we analyse only the connection establishment and property 1.(IV) does not involve here.

7. The Coloured Petri Net Model

Our specification models, DCCP connection Management and feature negotiation procedures (DCCP-CM-FN) have been created and maintained using CPN Tools [4]. It is a software package for the creation, editing, simulation and state space analysis of CPNs. It supports the hierarchical construction of CPN models [12], using constructs called *substitution transitions*. These transitions hide the details of subnets and allow further nesting of substitution transitions. This allows a complex specification to be managed as

a series of hierarchically related pages which are visualised in a hierarchy page, automatically generated.

7.1 Decision Rationale

Our previous CPN model captures DCCP's procedures associated with connection set up and synchronisation as specified in RFC 4340 and RFC 5596 (Simultaneous-Open). The model processes a packet by one transition occurrence in order to keep the state space size minimum. Adding the feature negotiation into the CPN model increases the number of the state variables. This increases the number of possible scenarios and the state space size. We attempt to model and analyse them using two approaches.

The first one attempts to maintain processing a packet in one transition occurrence by using ML functions when modelling the FSMs of the feature negotiation (Fig. 4). The state transition and the sending option field in Fig. 4 are implemented at the output arc inscriptions. However the drawback of this approach is that the arc expressions, written in ML, are complicated and incomprehensible to readers. It is very difficult not only to understand the model but also to convince that the model works correctly. To ensure that the model works correctly and is comprehensible, this paper reuses and combines two existing models: the feature negotiation procedure in [19] and the simultaneous-open procedure in [17]. This second approach requires another six buffer places to interface between the CPN models. Thus, an incoming packet will be processed by three occurrences of transitions: extracting the option fields; negotiating the feature; and setting up the connection. Adding buffer places and increasing the number of transition occurrences increase the state space size enormously.

We notice that RFC 4340 also specifies event processing as the sequential pseudo-code. In order to make the model analyzable, we adopt the prioritized transitions into the model. Although this choice provides us only a portion of the whole state space, we argue that it still complies to the pseudo-code specified in RFC 4340. We have already conducted experiments on both approaches. Because of readability and to convince the readers that the model is correct, this paper selects to present the CPN model of the second approach.

7.2 The Conditions of Underlying Channels

We incrementally study our CPN model with the following channel characteristics accordingly: FIFO without loss, reordering without loss, FIFO with loss, and reordering with loss. Due to space limitations, we shall confine our presentation to some particular interesting cases. An interesting case is when the protocol misbehaves and reaches the undesired terminal states. The error occurring when the protocol operates over FIFO without loss [19], [20] is the most important because of two reasons. First, due to the simplicity of FIFO without loss models, it is the clear proof of the defect in the protocol design. Second, the probability of the error

is very high. For the functional or qualitative measure, the lossy models are not so interesting because of two reasons. First, according to our experience, under the lossy channels, transport protocols often fail into undesired terminal states. Second, the lossy model often leads to state explosion.

In [19] we suggested two solutions to fix the discovered problem. Solution A is whenever the preference list is changed, the protocol entity sends a Change option to the other side and enters the CHANGING state. Solution B is that the DCCP does not discard non-reordered Confirm options in the STABLE state. Without the suggested solutions, the DCCP-CM-FN CPN model still exhibits the same defects discussed in [19]. This paper selects to use the Solution B because the Solution A involves retransmission which leads to the state explosion. After incorporating Solution B into the DCCP-CM-FN model, under the FIFO without loss condition we have not found any harmful error. Thus, this paper proceeds to focus on the condition of reordering channels without loss.

7.3 Modelling Assumptions and Specification Interpretation

We make the following assumptions regarding DCCP feature negotiation when creating our model.

1. We assume the medium to be reordering channels with no loss.
2. Without loss of generality, at each side, we model only two FSMs: connection management together with either feature local or feature remote (Fig. 4). In particular we assign the feature local's FSM to the Client and the feature remote's FSM to the Server.
3. A DCCP packet is modelled by a packet type, sequence and acknowledgement numbers, an option type and a list of feature values (preference list). Other fields are omitted because they do not affect the protocol operation.
4. RFC 4340 allows many options to be sent in one packet and many features to be negotiated at the same time. We consider the negotiation of *Congestion Control Identification* (CCID) that uses the server-priority reconciliation rule because the ability to negotiate the suitable congestion control mechanism is the main objective of DCCP.
5. Our model does not include the mandatory options, invalid options and unknown feature numbers.
6. We only consider a single connection instance and do not consider malicious attacks.

7.4 Model Overview

Our CPN model comprises five hierarchical levels. The complete model comprises 24 place instances, 123 transition instances, 556 arc instances and 33 CPN page. The top level page named DCCP shown Fig. 5 comprises two substitution transitions (for the Client and Server) and six places. Substitution transitions, DCCP_A and DCCP_B, represent the DCCP procedures in the Client and the Server, respectively. Two places allow communication with the applications that

Fig. 5 The top level CPN page: DCCP.

use DCCP, one for the Client, named **App_Client**, and the other for the Server, named **App_Server**. The states of the Client and Server protocol entities are stored in the places **Client_State** and **Server_State**.

The Client and Server communicate via two channel places, **A2B** and **B2A**, shown at the bottom of Fig. 5. Tokens inside these channel places are a multi-set. A multi-set is similar to a set but allows multiple appearances of any of its elements. One token represents one packet. When multiple packets are in the channel place, DCCP entity can take any packet nondeterministically. Thus, each channel place models a unidirectional reordering channel without loss, from the Client to the Server and the Server to the Client respectively. In the case of the FIFO channels DCCP can take only the first packet (token) of the queue (List) from the channel place. Comparing to the FIFO channel model, the reordering channel will increase the state space size.

Each place has a type (or colour set) which is defined in a set of declarations. The declarations define the data structures, any associated variables and functions used in the model. The declarations are written in CPN ML, a variant of Standard ML. For more details of the declarations of our CPN models, see [1], [17], [19].

7.5 Procedure_Client Page

Substitution transitions **DCCP_A** in Fig. 5 links to the second level page named **Procedure_Client** page. It comprises 6 substitution transitions[†] shown in Fig. 6. We attempt to arrange these transitions according to the pseudo-code in Sect. 8.5 of RFC 4340. The relationship between our CPN model and the pseudo-code is listed in Table 1. The pseudo-code describes 16 sequential steps for processing a received packet. We consider that the step 1 of the pseudo-code is irrelevant to our model because it involves the other packet fields that we do not model.

Substitution transition **Rcv_Invalid_Pkt** checks the validity of the incoming packet according to the step 6 of the pseudo-code. Substitution transition **Rcv_Unexpected_Pkt** models the protocol's behaviour in the **TIME-WAIT** state and checks the unexpected packet types according to the step 2 and step 7 of the pseudo-code respectively.

An incoming packet with an option filed in the

[†]We try to use the same name for both CPN page and substitution transition in Fig. 6.

Fig. 6 The second level CPN page: Procedure_Client.

Table 1 Correspondence between the substitution transitions in Procedure_Client page and the relevant steps in the pseudo-code (RFC 4340).

Substitution transitions/Transitions	Priority	Relevant steps in Pseudo-code
1.Rcv.Invalid_Pkt	3	step 6
2.Rcv.Unexpected_Pkt	3	step 2,7
3.Process.Options_Client		
3.1 Separate	4	step 8
3.2 FN (Local)	1	step 8
4.Exception_Handling	4	Not specified
5.Connection_Setup	2	step 4,5,9,10,12,15
6.Data_Transfer	2	step 16

channel place **Pkt_in** is consumed by substitution transition **Process_Options**. This transition processes the option field and puts the output tokens into two places: **Option_out** (for outgoing feature options) and **Pkt_intoNextStage** (for DCCP packet). These two places are used to be input buffers for the connection setup (substitution transition **Connection_Setup**) and the data transfer procedures (substitution transition **Data_Transfer**).

7.6 Process_Option_Client Page

Substitution transition **Process_Options** links to the third level page named **Process_Option_Client** shown in Fig. 7. It comprises 2 transitions. Transition **Separate** extracts the option field from the incoming packet and forwards to the substitution transition **FN** (linked to **FN_Local** page). The CPN diagram of **FN_Local** is reused from [19]. The arc expression on the outgoing arc toward place **Opt_In** checks the validity of the option field according to the non-reordered rule in Sect. 3.2.1.

Fig. 7 The third level CPN page: Process.Option.Client.

7.7 Priority Assignment

Because substitution transitions `Connection.Setup` and `Data.Transfer` require the option field (place `Option.out`) from the feature negotiation procedure, substitution transition `FN` (Fig. 7) has a higher priority (No.1) than `Connection.Setup` and `Data.Transfer` have.

The pseudo-code processes one packet at a time. Mimicking the pseudo-code, the `Procedure.Client` page processes one packet at a time as well. Before consuming a new incoming DCCP packet from the channel place `Pkt.in`, the three transitions (`Rcv.Invalid.Pkt`, `Rcv.Unexpected.Pkt` and `Separate`) have to wait until no packet left in place `Pkt.intoNextStage`. Thus, substitution transitions `Connection.Setup` and `Data.Transfer` have a higher priority (No.2) than the three transitions have.

Suggested by steps 6, 7 and 8 of the pseudo-code, the `Rcv.Invalid.Pkt` and `Rcv.Unexpected.Pkt` in Fig. 6 screen out the invalid or unexpected incoming packets before the incoming packet is taken by transition `Separate`. Thus, substitution transitions `Rcv.Invalid.Pkt` and `Rcv.Unexpected.Pkt` have a higher priority (No.3) than transition `Separate` has.

Substitute transition `Exception.Handling` does not require the incoming packet and involves timer expiry. Although it is not specified in the pseudo-code, we suggest that DCCP entity should either receive a valid packet (transition `Separate`) or retransmit (substitute transition `Exception.Handling`) at the same level of priority (No.4).

The priorities of all transitions are summarized in Table 1. Note that in Fig. 6 the input place of an incoming DCCP packet for transitions `Connection.Setup` and `Data.Transfer` is different from the others. Although both transitions have a higher priority than the others, they cannot be executed without a packet token in place `Pkt.intoNextStage`.

8. Analysis

In this paper we investigate functional behaviour of DCCP feature negotiation working together with connection man-

Table 2 An agreed feature value before and after preference lists have been changed.

		Client (Location)	
		before	after
Server (Remote)	before	[8,7,6,5]	[3,4,5,6,7]
	after	[4,9,6,7]	[8,6,7]

agement using full state space generation. This technique generates a directed graph called state space and stores the whole state space in computer memory. Nodes of the graph represent the state variables called markings and arcs represents the state transitions or events.

8.1 Initial Configurations

We analyse our model using using CPN Tools [4], [12] version 4.0.1 on an Intel i5-4300U 1.90GHz with 4 GB RAM. To analyse a particular scenario, the CPN model needs to be initialised by distributing initial tokens to places. The channel places `A2B` and `B2A` initially are empty. Both Client's and Server's feature negotiation states initially are in `STABLE` with the default agreed values and their initial preference lists. Both Client's and Server's states initially are in `CLOSED`. The Client issues the active open command and the Server issues the passive open (fully specified remote) command.

Our analysis focuses on negotiating the feature during the connection establishment with no retransmission of each packet type but they may retransmit the Change option once. After generating the whole state space, we use the ML query language to investigate if the feature negotiation procedure works correctly and has behaviour conformed to the desire properties defined in Sect. 6.

We choose to model and analyse the negotiation of the feature `CCID`. This feature uses the reconciliation rule: server priority. The default feature value is 2 which represents TCP-like congestion control. Although currently the standard specifies only `CCID2` (RFC4341), `CCID3` (RFC4342) and `CCID4` (RFC5622), we make up `CCID` numbers in each preference list for the purpose of validating the feature negotiation procedure. Table 2 shows the values in preference lists we used in our experiment before and after the preference has been changed. The resolved values before and after the preference changed under the server-priority reconciliation rule are shown in Table 2 as well.

Table 3 shows the initial configurations of twelve possible scenarios. They are classified according to which sides are allowed to initiate the negotiation and which sides change their preference lists. Our CPN model allows simultaneous negotiation and both sides can change their preference lists (Case 12). In all cases Initial Sequence Number Sent (ISS) of the Client are set to 5 and ISS of the Server are set to 15.

8.2 Analysis Result

The analysis results of DCCP feature negotiation CPN

Table 4 Analysis results of the DCCP-CM-FN CPN model.

Case (1)	nodes (2)	arcs (3)	time hh:mm:ss (4)	Terminal Markings					
				Type I-a (5)	Type I-b (6)	Type II (7)	Type III-a (8)	Type III-b (9)	Type III-c (10)
1(1,1)	10,808	29,225	00:58:59	42	1	11	74	0	0
2(1,1)	13,393	33,417	01:07:58	52	1	17	106	0	0
3(1,1)	105,745	348,737	06:43:17	93	1	0	665	0	0
4(0,0)	12,056	31,701	01:32:15	63	1	11	71	5	0
5(0,1)	2,936	5,913	00:21:45	30	1	6	12	0	0
6(0,0)	-	-	-	-	-	-	-	-	-
7(1,0)	33,724	91,828	03:03:44	95	1	22	148	0	2
8(0,0)	15,008	36,932	01:18:52	84	1	24	88	20	0
9(1,0)	166,678	532,150	15:31:19	130	1	469	350	50	2
10(0,0)	32,102	84,457	02:40:06	102	1	13	77	5	0
11(1,1)	39,353	97,802	02:25:39	128	1	30	111	20	0
12(0,0)	-	-	-	-	-	-	-	-	-

Table 3 Initial configurations of twelve possible scenarios.

Case	FN_Command		Change of Preference List	
	Local	Remote	Local	Remote
1	l'e	empty	disable	disable
2	empty	l'e	disable	disable
3	l'e	l'e	disable	disable
4	l'e	empty	enable	disable
5	empty	l'e	enable	disable
6	l'e	l'e	enable	disable
7	l'e	empty	disable	enable
8	empty	l'e	disable	enable
9	l'e	l'e	disable	enable
10	l'e	empty	enable	enable
11	empty	l'e	enable	enable
12	l'e	l'e	enable	enable

model using the initial configurations described in previous section are shown in Table 4. Note that our CPN model has already included our suggested solution B from [19]. Column (1) shows initial configurations listed in Table 3 together with the maximum number of retransmission allowed for Change options. For example 7(1,0) means Case 7 in Table 3. The maximum number of retransmitting Change option from the Client is 1. The maximum number of retransmitting Change option from the Server is 0 (no retransmission). The total number of states, arcs in each case is shown in the second and third columns. The state space generation time is given in the fourth column.

8.3 Termination Markings

Table 4 also shows the terminal markings of each scenario. All terminal markings have no packets left in the channels or buffers and hence there are no unspecified receptions. Due to the state explosion we are unable to obtain the results of case 6 and 12. The terminal markings are classified into 3 types.

1. Type-I-a and Type-I-b terminal markings arise in situations when the connection attempt fails because the back-off timer[†] expires. Both terminal markings are acceptable.

[†]After retrying for a period (measured by a “back-off” timer), the Client will send a DCCP-Reset and will “back off” to the CLOSED state [15].

In Type I-a terminal markings, both sides are CLOSED. For Type I-b terminal markings, the Client is CLOSED, but the Server is in the LISTEN state. This can happen when the Server is initially CLOSED (down for maintenance or busy) and rejects the connection request. The Server then recovers and moves to the LISTEN state. Some cases have more than one terminal markings depending on the state variables of the feature negotiation procedures.

2. Type-II terminal markings arise when the negotiation is unfinished. It is possible that at terminal states after both the Client and Server are in the OPEN state, either side is still in the CHANGING or UNSTABLE.

3. Type-III terminal markings are the center of our attention in this paper. They arise when both the Client and Server are in the OPEN state indicating that the connection has been successfully established. Type-III also have both local and remote in the STABLE state. When an endpoint receives the valid Confirm option, it re-computes the agreed feature value using its preference list and the preference list in the Confirm option. If the result is equal to the confirmed value in the Confirm option, its agreed value is set to the confirmed value otherwise it is set to the default feature value (which is two). The Type III terminal markings are further divided into 3 subtypes.

Type III-a is the desired terminal state, where both Client and Server reach the same feature value.

Type III-b is the undesired terminal state, where both sides reach different feature values and one of the feature values is equal to the default confirmed value (two). In this case the endpoint holding the default confirmed value knows that the confirmed feature value in the received Confirm option is wrong. Thus, it should re-negotiate or reset the connection.

Type III-c is also the undesired terminal state where both sides reach different feature values and none of the feature values is equal to the default confirmed value (two). In this case both do not know that their feature values do not match. These Type III-c terminal states illustrate that the feature negotiation procedure still has serious errors.

Fig. 8 An undesired scenario caused by the unexpected respond packet received in OPEN.

8.3.1 Investigating the Error in Type-III Terminal Markings

After generating the state space shown in Table 4, we use the occurrence graph tools listing the shortest path from the initial marking to each undesired terminal marking (Type III-c) in Case 7. Both scenarios are very similar. We consider that it is sufficient to describe only one time sequence diagram in Fig 8.

After the Server replies the first Confirm R in the first Response (seq=15,ack=5) packet, it changes its preference list. However the first Response packet does not arrive in time. After the Client retransmits the second Request (seq=6) packet with Change L option, the Client receives the non-reordered Response (seq=15,ack=5) packet with Confirm R. Thus it accepts the confirmed value 7, changes its state to PARTOPEN and replies with Ack (seq=7,Ack=15).

Meanwhile the Server receives the second Request (seq=6) with Change L, it not only replies with the second Response (seq=16,ack=6) with Confirm R but also re-computes and accepts the new feature value 4. Unfortunately the second Response (seq=16,ack=6) packet with Confirm R is rejected by the Client because it is unexpected to be received in OPEN state. This behaviour conforms to the pseudo code of the connection management procedure, step 7 in Sect. 8.5, page 70 of [15]. It specifies that the Response packet receiving in OPEN be invalid and does not process its Options field. Thus the Client holds the old confirmed value (7) but the Server holds the new confirm value (4).

8.3.2 Interaction between the Two Protocol Procedures

The above evidence suggests that the connection manage-

ment procedure can affect the feature negotiation procedure. When the packets carrying the feature options are classified as sequence number invalid or unexpected packets (the pseudo code, step 6 and step 7 in Sect. 8.5 of [15]), the options in these packets will not be processed. This action may result the different feature values on both sides.

During model development we also discover Type III-b terminal markings of which the confirmed value in the received Confirm option could be wrong. Because before receiving the Confirm option, the preference list of the Change sender has been changed. When re-calculating the confirmed value, it could be different from the confirmed value in the Confirm option received. RFC4340 specifies this behaviour as the invalid option and requires the connection be reset immediately. This evidence shows that the feature negotiation procedure can affect the connection management procedure by resetting the connection. We modify the model by allowing the Respond packet received in OPEN be valid and process its option accordingly. The new analysis result shows no undesired terminal markings left.

8.4 Absence of Livelocks

We also use the occurrence graph tool generating the SCC graph of the state space in Cases 1-5 and Cases 7-11. In each case both state space and its SCC graph have the same number of nodes and arcs. Thus there is no presence of livelocks. Unfortunately in Case 6 and 12 the whole state space are too large. We are unable to check the absence of livelocks property in Case 6 and 12.

9. Conclusions and Future Work

We have expressed the motive to analyse the DCCP feature

negotiation working together with connection management procedures. After the desired properties for formal analysis are specified, we discuss our modelling decision. Using CPN Tools the CPN diagram in [19] and [17] are reused and combined together. The causes of the state explosion in our work are: the reordering channels; the state variables for the feature negotiation; six buffer places and three transitions. To alleviate the state explosion, we adopt the prioritized transitions by attempting to mimic the pseudo-code specified in RFC 4340. The state space analysis result of the interaction between connection management and feature negotiation procedures is discussed in this paper. The analysis result of the revised models proves that there is still an error in RFC 4340 specification. This error leads both side to undesired terminal states holding the different feature values in the OPEN and STABLE states.

Although the main procedures in RFC 4340 and RFC 5596 have been modelled and analysed, the next challenge is to incorporate the congestion control mechanism into the model. The study could focus on how three procedures affect each other and performance of the data transfer.

Acknowledgments

This work is supported by Research Grant no. TRG5380023 from the Thai Network Information Center Foundation and the Thailand Research Fund.

References

- [1] J. Billington and S. Vanit-Anunchai, "Iterative modelling of an evolving Internet standard: the datagram congestion control protocol," *Fundamenta Informaticae*, vol.88, no.3, pp.357–385, 2008.
- [2] J. Billington, S. Vanit-Anunchai, and G.E. Gallasch, "Parameterised coloured petri net channel models," *Trans. Petri Nets and Other Models of Concurrency III*, vol.5800, pp.71–97, 2009.
- [3] E.M. Clarke, O. Grumberg, M. Minna, and D. Peled, "State Space Reduction using Partial Order Techniques," *International Journal on Software Tools for Technology Transfer*, vol.2, no.3, pp.279–287, 1999.
- [4] CPN Tools home page. <http://cpntools.org>
- [5] DCCP home page. <http://www.read.cs.ucla.edu/dccp/>
- [6] G. Fairhurst, "Datagram congestion control protocol (DCCP) simultaneous-open technique to Facilitate NAT/Middlebox traversal," RFC 5596, Sept. 2009.
- [7] S. Floyd, M. Handley, and E. Kohler, "Problem statement for the datagram congestion control protocol (DCCP)," RFC 4336, March 2006.
- [8] G.E. Gallasch, J. Billington, S. Vanit-Anunchai, and L.M. Kristensen, "Checking Safety Properties On-The-Fly with the Sweep-line Method," *International Journal on Software Tools for Technology Transfer*, vol.9, no.3-4, pp.374–391, June 2007.
- [9] 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, Dec. 2004.
- [10] G.J. Holzmann, Design and Validation of Computer Protocols, Prentice Hall, 1990.
- [11] K. Jensen, "Condensed State Spaces for Symmetrical Coloured Petri Nets," *Formal Methods in System Design*, vol.9, no.1/2, pp.7–40, 1996.
- [12] K. Jensen and L.M. Kristensen, Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer, Heidelberg, 2009.
- [13] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion Control Without Reliability," *Proceedings of the 2006 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'06)*, pp.27–38, Pisa, Italy, Sept. 2006.
- [14] E. Kohler, M. Handley, and S. Floyd, SUBSTANTIVE DIFFERENCES BETWEEN draft-ietf-dccp-spec-11 AND draft-ietf-dccp-spec-12, Dec. 2005.
- [15] E. Kohler, M. Handley, and S. Floyd, Datagram Congestion Control Protocol, RFC 4340, March 2006.
- [16] E. Kohler, M. Handley, and S. Floyd, SUBSTANTIVE DIFFERENCES BETWEEN draft-ietf-dccp-spec-13 AND RFC 4340, March 2006.
- [17] S. Vanit-Anunchai, "Analysis of two-layer protocols: DCCP simultaneous-open and hole punching procedures," *Proc. 1st French Singaporean Workshop on Formal Methods and Applications, (FSFMA 2013)*, pp.3–17, Singapore, July 2013.
- [18] 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, Nov. 2007.
- [19] S. Vanit-Anunchai, "Validating DCCP simultaneous feature negotiation procedure," *Trans. Petri Nets and Other Models of Concurrency XI*, vol.9930, pp.71–91, 2016.
- [20] S. Vanit-Anunchai, "Validating SCTP simultaneous open procedure," *Proceedings of the 5th International Conference on Fundamentals of Software Engineering, FSEN2013, Revised Selected Papers*, pp.233–249, Springer, Heidelberg, Tehran, Iran, April 2013.
- [21] S. Vanit-Anunchai and J. Billington, "Modelling the datagram congestion control protocol's connection management and synchronisation procedures," *Proc. 28th International Conference on Application and Theory of Petri Nets and other models of concurrency (ICATPN'07)*, pp.423–444, Springer, Heidelberg, Siedlce, Poland, June 2007.
- [22] 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*, vol.10, no.1, pp.29–56, 2008.
- [23] S. Vanit-Anunchai, J. Billington, and T. Kongprakaiwoot, "Discovering Chatter and Incompleteness in the Datagram Congestion Control Protocol," *Proceedings of the 25th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, vol.3731, pp.143–158, Springer, Heidelberg, Taipei, Taiwan, Oct. 2005.

Somsak Vanit-Anunchai received the B.Eng degree in Electrical Engineering from Chulalongkorn University in 1985, and the MSEE degree from University of South Florida in 1993, and the PhD degree in Computer Systems Engineering from University of South Australia in 2007. Since 1998 he has been working as a Lecturer with Suranaree University of Technology. His research interests lie in the area of formal model and verification of complex systems.