Techniques and Tools for Formalising and Analysing the Resource Reservation Protocol: A Coloured Petri Net Approach

María E. Villapol
School of Computer Sciences
Central University of Venezuela
Av. Los Ilustres, Los Chaguaramos, Caracas, Venezuela
Tel: +212 58 605 1132, Fax: +212 58 605 1131
Email: mvillap@strix.ciens.ucv.ve

and

Jonathan Billington
Computer Systems Engineering Centre
University of South Australia
Mawson Lakes, Adelaide, SA, 5095,Australia
Tel:+618 8302 3940, Fax:+618 8302 3384
Email: j.billington@unisa.edu.au

Abstract

The goal of the *Resource Reservation Protocol* (RSVP) is to establish *Quality of Service* information in the form of resource reservations (such as buffers and bandwidth) within routers and host computers of the Internet. It is intended to support emerging Internet applications that require performance guarantees. Currently, Internet protocols are not formally specified when they are developed. Instead they are described in a narrative way in documents called *Request for Comments* (RFCs). This is the case for RSVP. To increase confidence in RSVP we have formalised and analysed its narrative specification using *Coloured Petri Nets* (CPNs). This paper demonstrates how CPNs can be used for modelling and analysing RSVP. Among the several beneficial features of CPNs are: graphical facilities for specification; support for different levels of abstraction; hierarchical structuring mechanisms; and verification and validation methods, such querying the state space to investigate properties, and language equivalence to check the consistency of different levels of abstraction. These facilities allow us to create a model, that provides a clear, unambiguous and precise definition of RSVP, and to analyse the protocol for functional correctness. The paper concentrates on the approach and the tools used in this investigation.

Keywords: RSVP, Quality of Service (QoS), Coloured Petri Nets, State Spaces.

1 Introduction

The *Internet Integrated Service Model (IntServ)* [3] is one of the proposals for providing the desired *Quality of Service* (QoS) for applications operating over the Internet. QoS guarantees are required for multimedia and real-time applications. The *Resource Reservation Protocol (RSVP)* [4][5] is part of the IntServ model. RSVP is a signalling protocol developed to create and maintain resource reservations (eg buffer and data rate allocations) in Internet routers and host computers, to provide the desired QoS. RSVP uses a soft-state approach, where state information concerning traffic characteristics and resource reservations must be periodically refreshed, or else face automatic removal. For the desired QoS to be guaranteed it is essential that RSVP works correctly.

Formal methods provide techniques to support the design and maintenance of communication protocols [1][14]. We have found that formal techniques have been seldom applied to Internet protocols. Some of this work has been focused on TCP [9][15], Internet Open Trading Protocol (IOTP) [12] and RSVP (which includes our own work [18]). *Coloured Petri Nets (CPNs)* [8] are a formal technique used for modelling many systems, particularly communication protocols [2]. The work presented here is based on an extensive investigation of RSVP mechanisms using CPNs [20]. Part of the contribution of this work has been a clear, unambiguous and precise definition of the major features of the protocol, which is missing in the current specification of RSVP [4]. The aims of this paper are to show how CPNs can be used to create a model of RSVP and to analyse the model using *state spaces* [8] and their associated *Strongly Connected Component (SCC)* graphs [8]. The paper also explains the standard behavioural properties of CPNs and their application to the analysis of RSVP. Our investigation is supported by a software tool called Design/CPN [11].

The paper has been organised as follows. Section 2 presents an overview of RSVP, which includes its characteristics and operation. Section 3 summarises the major steps of the methodology we use for the specification and analysis of RSVP. The methodology requires the definition of a set of RSVP Service Primitives, and these are given in Section 4. Section 5 gives an informal introduction to the components, dynamic behaviour and hierarchical structuring mechanism of CPNs, illustrated by part of the CPN model of RSVP. We assume the reader is familiar with basic Petri Net notation. Section 6 describes the state space method and its application to the analysis of the model of RSVP. Section 7 describes the standard properties of CPNs and how they can be used to demonstrate that the protocol works as expected. Finally, Section 8 concludes the paper.

2 Resource Reservation Protocol (RSVP) Overview

RSVP is designed to be run on network routers and in end hosts to support QoS applications. It reserves resources for a *data flow* from the sender to one or more destinations (i.e. a multicast destination). A data flow is a distinguishable packet stream, which results from using a single application (such as video conferencing) requiring a certain QoS. A packet stream includes all packets that travel from the same source to the same destination. Unlike other signalling protocols [5], RSVP destinations (receivers) request resource reservations. Those requests travel on the reverse path of the data flow by following the pre-established route setup by RSVP [4]. RSVP is also responsible for maintaining reservations on each node associated with the data flow. RSVP uses a soft-state approach where the reservation states must be refreshed periodically; otherwise they are automatically removed. The approach accommodates dynamic route changes, dynamic multicast group membership and dynamic QoS changes [4]. RSVP reserves resources for a *session*. A session includes all data flows from one or more senders to the same unicast (one receiver) or multicast destination (multiple receivers).

RSVP reservation requests are defined in terms of a *filter specification (filter spec)* and a *flow specification* (*flow spec*) [4][5]. A filter spec is used to identify the data flow that is to receive the QoS specified in a flow specification. A flow spec defines the desired QoS in terms of a service class, which comprises a *Reservation Specification (RSpec),* and a *Traffic Specification (TSpec).* A *RSpec* defines the reservation (i.e.. desired QoS) characteristics of the flow, for example, the service rate the application requests. A *TSpec* defines the traffic characteristics of the flow, for example, the peak data rate.

RSVP uses several messages in order to create, maintain, and release state information for a session between one or more senders and one or more receivers as shown in Figure 1. We now describe the main RSVP features, structured to facilitate the description to the model.

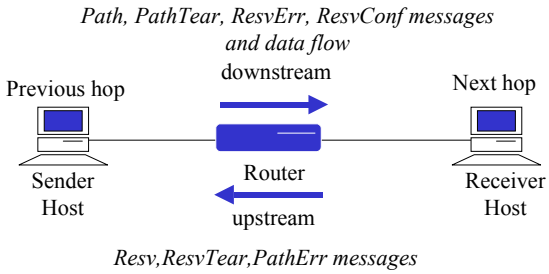

Figure 1: RSVP messages

a. Path Setup: In RSVP, reservation requests travel from receivers to the sender(s), in the opposite direction to the user data flow for which such reservations are being requested. Thus, *Path messages* are used by the sender to set up a route to be followed by the reservation requests, which uses the same routers as the corresponding data flow. These messages travel downstream and set up and maintain path state information (eg the Internet Protocol address of the previous router and the data flow's traffic characteristics).

b. Path Refresh: Path and reservation states have two timers associated with them: a *refresh timer* and a *cleanup timer*. A refresh timer determines when a path or reservation refresh message will be generated. The cleanup timer determines the maximum period of time that a node (i.e. router or host) can wait to receive a path or reservation refresh message, before it removes the associated state information. A path refresh is the result of either a path refresh timeout or a user request to modify the path state. Once a path is established, a node sends path refresh messages (i.e. *Path messages*) periodically (i.e. every refresh timeout period [4]).

c. Path Error: A node that detects an error in a Path message, generates and sends a *PathErr message* upstream towards the sender that created the error. It travels hop-by-hop to the sender and does not modify any path state at the nodes through which it passes. Once the sender receives the *PathErr message*, it can report the error to the application, which may take corrective action.

d. Path Release: RSVP tear down messages are intended to speed up the removal of path and reservation state information from the nodes. They may be triggered because a cleanup timeout occurs or an application wishes to finish a session. A *PathTear message* travels downstream from a sender to the receiver(s) and deletes any path state information and dependent resource reservation associated with the session and sender.

e. Reservation Setup: *Resv messages* carry reservation requests (eg for bandwidth and buffers) used to set up reservation state information along the route of the data flow. They travel upstream from the receiver(s) to the sender(s). Reservation requests, which arrive at a router, may be merged. The aim of merging is to control the overhead of reservation messages by making them carry more than one flow and filter specification [4]. Thus, the effective filter and flow specifications, which are carried in a reservation message, are the result of merging reservations from several requests.

f. Reservation Refresh: A reservation refresh is the result of either a reservation state refresh timeout or a receiver request to modify the reservation. Like path states, reservation states need to be refreshed. Thus, a receiver periodically sends reservation refresh messages (i.e. *Resv messages*) to the sender.

g. Reservation Release: *ResvTear messages* travel from the receiver(s) to the sender and remove any reservation state information associated with the receiver's data flow.

h. Reservation Error: If a node detects an error in a Resv message, it sends a *ResvErr message* downstream to the receiver that generated the failed Resv message. Processing ResvErr messages will not result in the removal of any reservation state.

i. Reservation Confirmation: Optionally, a receiver may ask for confirmation of its reservation. A *ResvConf message* is used to notify the receiver that the reservation request was successful. In the simplest case, a ResvConf message is generated by the sender (Figure 1).

3 Protocol Verification Methodology

Verification of RSVP firstly requires a formal specification of the protocol and the service it is intended to supply to its users [1]. This is set in the context of a protocol architecture, which in the case of RSVP, is given by the IntServ

architecture [3]. A *service specification* [7] defines a set of events, known as *service primitives* (see Section 4), and their possible sequences at the interfaces between the users (an application or higher level protocol entity) and the *service provider* (comprising the protocol entities concerned and their communication mechanisms). RSVP [4] does not include an explicit service specification, but it does include an application interface from which the RSVP service specification [19][20] has been derived, taking into account the features of the protocol specification. A *Protocol Specification* formally describes the features of the protocol that provide the required service. RFC 2205 [4] provides the source document from which the RSVP formal model is derived. Both the service and protocol specifications are defined using Coloured Petri Nets.

Once the service and protocol specifications are defined, the protocol can be analysed for general properties and specific properties defined for RSVP. It can also be compared with the service specification, to see if the sequences of primitives defined in the service specification are preserved in the protocol specification.

4 RSVP Service Primitives

Service primitives [7] provide an abstract way to describe the interaction between the RSVP service user (i.e. QoS-aware application) and the RSVP service provider. A QoS-aware application interacts with RSVP to request reservation services. Since the RSVP specification [4] does not define the RSVP service, the authors [19][20] recently defined a set of service primitives for RSVP. They are used in the CPN model and in the definition of the desired RSVP properties.

Each primitive can be either a request or an indication. A request (Req) is used by the application to ask for a service from RSVP. An indication (Ind) is used by RSVP to notify the application of the invocation of a request primitive by its peer or to notify the user that the RSVP service provider detected an error or to confirm reservations. We have defined the following service primitives [20].

a. **RSVP-Sender (Req/Ind):** a sender application uses this primitive to establish or update the traffic characteristics of a data flow for a RSVP session.
b. **RSVP-Reserve (Req/Ind):** a receiver application uses this primitive to establish or to modify a resource reservation during a session.
c. **RSVP-SenderRel (Req/Ind):** a sender application uses this primitive to close a session. This means that the user data flow will eventually not have any QoS reserved.
d. **RSVP-ReceiverRel (Req/Ind):** a receiver application uses this primitive to close a quality controlled session.
e. **RSVP-ResvConf (Ind):** is used by the service provider to confirm that a reservation has been made.
f. **RSVP-SenderError (Ind):** is used by the service provider to report an error in propagation or installation of the Sender's traffic characteristics.
g. **RSVP-ResvError (Ind):** is used by the service provider to report a reservation failure inside the network.

5 Coloured Petri Net Modelling

Coloured Petri Nets (CPNs) [8][10] provide compact descriptions of concurrent systems by including abstract data types within the basic Petri net framework. We describe CPNs and how they can be used for modelling RSVP through an example [20]. The example is based on RSVP's path management features (i.e. path setup, refresh, error and release) for a sending protocol entity, as shown in Figure 2.

5.1 Places

In Figure 2, there are four *places* drawn as ellipses. The Sender place stores the state of the sending protocol entity. The places SOutgoingMsgs and SIncomingMsgs store RSVP messages travelling downstream and upstream, respectively. Finally, the place SenderUser represents the sending application which requires RSVP services. Each place has an associated *type* which is written in italics at the top right of the place. Place types are defined in a set of declarations shown in Figure 3. For example, place Sender is typed by *SenderState*, defined at the top of Figure 3.

A *Marking* of a place defines a collection of data values, known as *tokens*, that are associated with that place. The value is taken from the type of the place. This collection of tokens is a multi-set, since it may contain several tokens of the same value. CPNs also include the initial state of the system, called the *initial marking*. The initial marking (if not the empty multi-set) is written near each place. In the initial marking of the example, the place SenderUser contains the requested traffic characteristics, in this case, 1'Ta. Each communication place is empty (hence there is no inscription). The state place Sender contains a triple comprising null entries for path and reservation information and SESSION for the status of the sender.

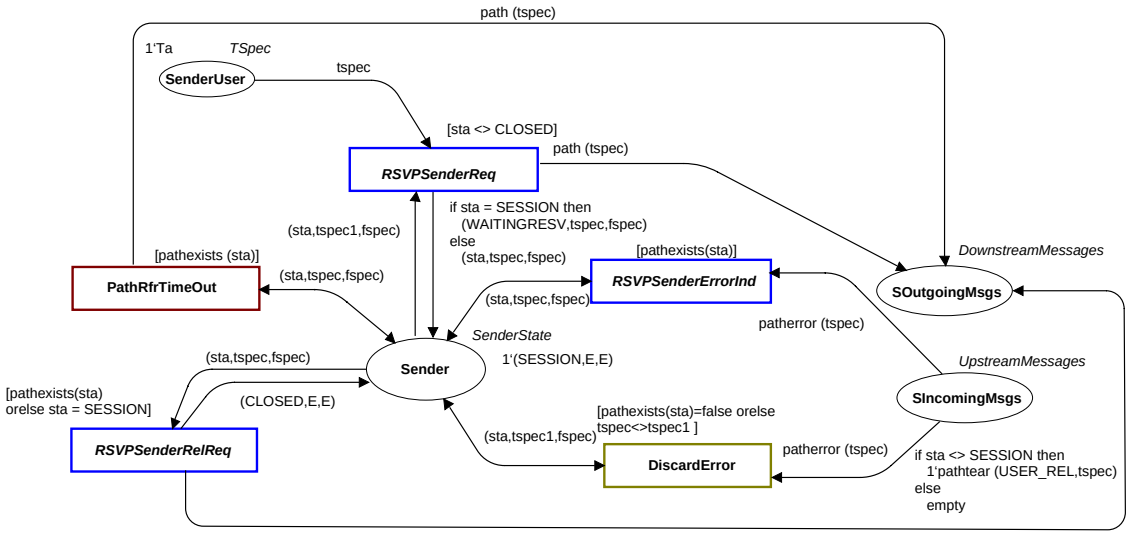

Figure 2: Path management.

5.2 Transitions

Transitions represent atomic events of the system. They are drawn as rectangles in Figure 2. They may also have *guards* associated with them, which are included in square brackets. Guards are Boolean expressions which are important for describing CPN dynamics and are discussed later in this section. There are five transitions in the example. The transition RSVPSenderReq models the establishment or updating of path state information and the occurrence of the service primitive RSVP-Sender.Req. The transition PathRfrTimeOut models periodic path refreshes required to maintain path state information. The RSVPSenderRelReq transition models the action taken when the sender user leaves the session on the occurrence of the service primitive RSVP-SenderRel.Req. The transition RSVPSenderErrorInd represents an occurrence of the service primitive RSVP-SenderError.Ind. Finally, DiscardError removes path error messages that are not reported to the application.

5.3 Arcs

Arcs connect transitions and places and are represented by arrows. A transition may have input places connected by incoming arcs and output places connected by outgoing arcs. Arcs have expressions associated with them. The expressions are built from constants, variables and functions and are written next to their associated arcs. The functions are defined, and the constants and variables declared, in the set of declarations (see Figure 3).

5.4 Declarations

Type definitions, variables, and functions are defined in what is called the *global declaration node* of a CPN. Parts of the global declaration node, including the definitions that are relevant to Figure 2, are shown in Figure 3. They are written in the functional programming language ML [13]. The variant used in Design/CPN, known as CPN ML, has some special key words. Here *colour* is used to denote a type. The declarations are divided into 4 sections.

States of RSVP Entities. ParameterValues is an enumeration type, which represents abstract values for both the *traffic specification* (*tspec*) and *flow specification* (*fspec*) parameters. The possible values are Ta, Tb, Fa, Fb and E (empty). The types, STSpec and SFSpec, are subsets of ParameterValues and represent the traffic specification stored as part of the path state information and the flow specification stored as part of the reservation state information, respectively. Status is an enumerated type, which defines the states of the RSVP entities as follows:

a. **SESSION:** the sender or receiver has opened a session, but no path or reservation has yet been established.
b. **IDLE:** there exists neither path nor reservation information at the router.
c. **WAITINGRESV:** a request with the Sender's traffic information has been accepted by the entity (i.e. sender, router or receiver) and sent (if the entity is not the receiver) but as yet no reservation request has been received.
d. **RESVREADY:** a reservation request has been accepted and sent (if the entity is not the Sender).
e. **RESVCONFIRMED:** a reservation has been established and a confirmation has been received.
f. **CLOSED:** the sender or the receiver has left the session.

SenderStatus only requires four of these states and is a subset of Status. SenderState represents the states of the sender and is the product of SenderStatus, STSpec and SFSpec. The Sender place has the type SenderState.

RSVP Messages. TSpec and FSpec, represent the parameters that may be carried in RSVP messages [4] and are the traffic specification and flow specification respectively. TearMsgType is intended to distinguish between: a PathTear message generated as a result of the sender or receiver leaving the session (REL); or a path or reservation cleanup time-out (TEARDOWN). The other seven RSVP messages defined in Section 2 are represented by UpstreamMessages and DownstreamMessages. The places SOutgoingMsgs and SIncomingMsgs are typed by DownstreamMessages and UpstreamMessages, respectively.

Variables and Functions. The variables used in CPN inscriptions are typed in the declarations. As an example, the variable *sta* represents the status of the RSVP entity (e.g. the sender). The function *pathexists* is used to simplify guard inscriptions and returns true if the sender has path state information available.

```
(* ============= States of RSVP entities =========== *)

color ParameterValues     = with E|Ta|Tb|Fa|Fb;
color STSpec              = subset ParameterValues with [E,Ta,Tb];
color SFSpec              = subset ParameterValues with [E,Fa,Fb];
color Status              = with SESSION|IDLE|WAITINGRESV|RESVREADY|
                                 RESVCONFIRMED|CLOSED;
color SenderStatus        = subset Status with [SESSION,WAITINGRESV,RESVREADY,CLOSED];
color SenderState         = product SenderStatus * STSpec * SFSpec;

(* ================= RSVP Messages ================= *)

color TSpec               = subset ParameterValues with [Ta,Tb];
color FSpec               = subset ParameterValues with [Fa,Fb];
color TearMsgType         = with TEARDOWN|REL;
color ResvTear            = product TearMsgType * FSpec;
color PathTear            = product TearMsgType * TSpec;
color UpstreamMessages    = union patherror: TSpec + resvtear: ResvTear + resv: FSpec;
color DownstreamMessages  = union path: TSpec + resverror: FSpec + resvconf: FSpec +
                                  pathtear: PathTear;

(* ================    Variables    ================= *)

var sta: Status;
var tspec,tspec1: STSpec;
var fspec,fspec1: SFSpec;

(* =================   Functions ==================== *)

fun pathexists (sn) = (sn= WAITINGRESV orelse sn=RESVREADY orelse sn = RESVCONFIRMED);
```

Figure 3: Part of the global declaration of the RSVP model.

5.5 Enabling and Occurrence of Transitions

Arcs are inscribed with *expressions* written in ML. Transitions can be *enabled* and can then *occur*. A transition is enabled if its input places have the required tokens and its *guard* is true. These enabling requirements are determined by *binding* the transition's variables to values taken from their types. The choice of binding value is arbitrary. The required tokens are defined by evaluating the input arc expressions for a particular binding of the variables. This same binding is used for evaluating the guard. The occurrence of a transition removes tokens from the input places and adds tokens to the output places. The removed tokens are defined by the evaluated expressions on the corresponding incoming arcs for this binding of variables, while the values of the added tokens are determined by evaluating the arc expressions on the corresponding outgoing arcs for the same binding. Hence transitions can occur in different modes, depending on the bindings of the variables.

For example, in Figure 2, RSVP-Sender.Req is enabled when the Sender is not CLOSED (see the guard), and a new *tspec* is waiting to be sent in SenderUser. An occurrence of the transition RSVP-Sender.Req updates the *tspec* to the one requested by the user (ie to Ta). If the status of the sender is equal to SESSION, it is also updated so that it indicates that the sender is ready to receive a reservation request (WAITINGRESV). If not, the state of the sender is only updated by the new *tspec* value Ta. In addition, Ta is removed from the place SenderUser and a Path message carrying the corresponding *tspec* = Ta, is added to the SOutgoingMsgs place.

5.6 Hierarchical CPNs

We deal with RSVP's complexity by using the hierarchical constructs of CPNs [8][10]. Hierarchies are built using the notion of a *substitution transition*, which may be considered a macro expansion. The model starts with a top-level CPN diagram, which provides an overview of the system being modelled and its environment. In hierarchical CPNs, this top-level diagram will contain a number of substitution transitions. Each of these substitution transitions is then refined by another CPN diagram, which may also contain substitution transitions. The top-level diagram and each of the substitution transitions is defined by a module, called a *page*. The relationships between the different pages are defined by a *hierarchy page.* The hierarchy page also includes the name of the page that defines the declarations required for the CPN inscriptions, called the *Global Declaration node* (see Section 5.4).

The hierarchy page of the RSVP CPN model consists of eleven (11) pages as illustrated in Figure 4. The top-level page is called RSVPNetwork, which describes the network topology and interaction with the applications that use RSVP. The network topology consists of one sender and one receiver connected by a router (Figure 1). This page is the main one (i.e. prime page) and includes substitution transitions for the *Sender*, *Router* and *Receiver*, which are defined by their own pages. These in turn also comprise substitution transitions for Path and Reservation management, which are defined at the lowest level of the hierarchy. These correspond with the major functions of RSVP described in Section 2. The Path and Resv management pages include transitions that model the establishment, refreshment, release and error control of paths and reservations, respectively. Also included is the Global Declaration node (Page 11). Each page at the lowest level of the hierarchy uses transitions to model service primitives and protocol actions, which include implementing RSVP functions (eg path refresh) or discarding messages that cannot be processed. The Sender Path Management page is given in Figure 2. The description of the other pages can be found in [20].

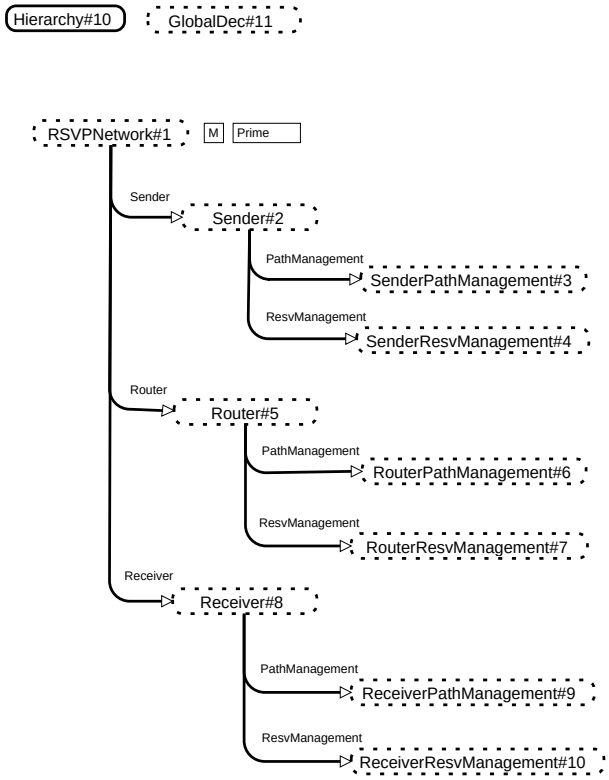

Figure 4: Hierarchy page of the RSVP CPN model.

6 Analysis of CPNs

6.1 Modifying the CPN Model for Analysis

To analyse the CPN model we employ *state space* methods [8]. An *Occurrence Graph* can be generated using Design/CPN [16]. It includes all possible markings that can be reached from the initial marking and is represented by a directed graph where the nodes represent the markings and the edges the occurring *binding elements* comprising the CPN transition and the assignment to the transition's variables.

The CPN model of RSVP can generate an infinite state space when path and reservation refreshes are included (e.g. see transition PathRfrTimeOut in Figure 2), since an arbitrary number of RSVP messages can be in the communication places (e.g. SOutgoingMsgs). The first step towards analysing the CPN model is to ensure that its state space is finite. This can be achieved by limiting the number of refreshes that can occur (which we consider to be a severe limitation on the operation of RSVP) or by limiting the storage capacity of the communication network. Given that storage in the network is finite, this is a more realistic option. We thus modify the model so that the communication places have finite capacity.

The types, DownstreamMessages and UpstreamMessages, are modified to include a *no message* value. Figure 5 illustrates how the Path Management page (see Figure 2) has been modified to limit the capacities of the communication places using the notion of empty message slots. The Sender can send a message to the network, (e.g. see transition RSVPSenderReq) if SOutgoingMsgs contains a token indicating the presence of an empty slot (i.e. 1'nodmsg (N)). Also, when the sender receives a message (e.g. see transition RSVPSenderErrorInd) it adds a token representing an empty slot (i.e. 1`noumsg (N)) to SIncomingMsgs. We also modify the model to ensure that the sequence of requests for traffic specification changes by the sender user is maintained, by using a list type. In addition, we have incorporated the transition RSVPSenderRelReq2, which has the same functionality as the transition RSVPSenderRealReq, however its occurrence does not depend on the availability of a token on the SOutgoingMsgs place. In other words, when there is no state information at the Sender, it can leave the session without sending any PathTear message to the network.

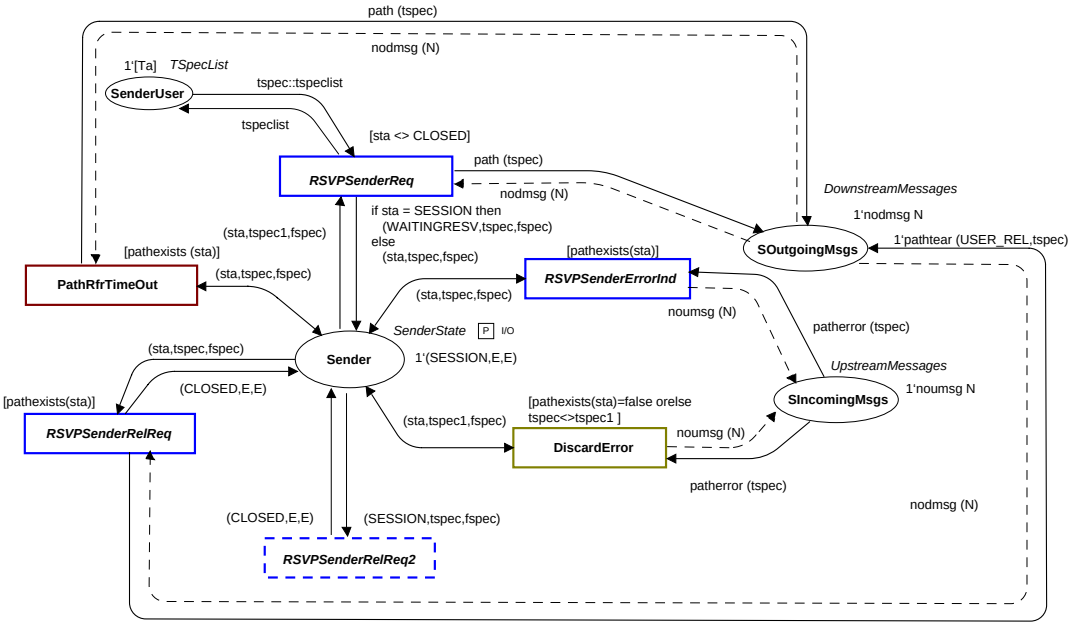

Figure 5: Modified Path Management page.

6.2 State Space Analysis

Figure 6 shows the state space for our model of RSVP when the above changes are incorporated into the Router and Receiver parts of the model. To obtain a small state space suitable for presentation we have restricted the model to only include the basic features of path and reservation establishment. Our aim here is to provide an illustration of the approach, rather than the full analysis, for which the reader is referred to [20].

There are nineteen (19) reachable markings (or nodes), represented by rounded boxes in the figure. Each marking has an identification number located at the top. Also, there are two numbers separated by a colon (":"), which represent the number of input and output arcs, respectively. The node at the top of the figure is the initial marking and has the identification number 1.

The details of each of the markings can be obtained easily from Design/CPN and are shown in the dashed boxes next to the nodes for the initial marking and for each of the terminal markings (leaf nodes). For example, in the initial marking (i.e. node 1): the Router is idle (IDLE,E,E) and the Sender and Receiver are in the same session (SESSION,E,E); each of the communication places (SIncomingMsgs, ROutgoingMsgs, SOutgoingMsgs and RIncomingMsgs) has an empty slot (i.e. 1`noumsgs (N) or 1`nodmsgs (N)); and the SenderUser is ready to provide its traffic characteristics (1`[Ta]) while the ReceiverUser desires that a reservation (1`[Fa]) be made.

An arc represents the occurrence of a *binding element*. The details of binding elements can be shown on the arcs, which we have done for several of them. Also included is the identification number of the arc (located in the upper left corner) and the related node numbers (*n1->n2*, indicates that the marking *n2* is reached from marking *n1* when the binding element occurs). For example, the binding element 1 (represented by the arc 1) is the only one enabled in the initial marking. It includes the transition RSVPSenderReq and the bindings: *tspeclist = []* (no more sender requests), tspec1= E (no traffic information installed at the sender), *tspec =Ta* (the sender informs RSVP of the required traffic characteristics equal to Ta), *sta = SESSION* (the status of the sender is SESSION), *fspec = E* (no reservation information is installed at the sender).

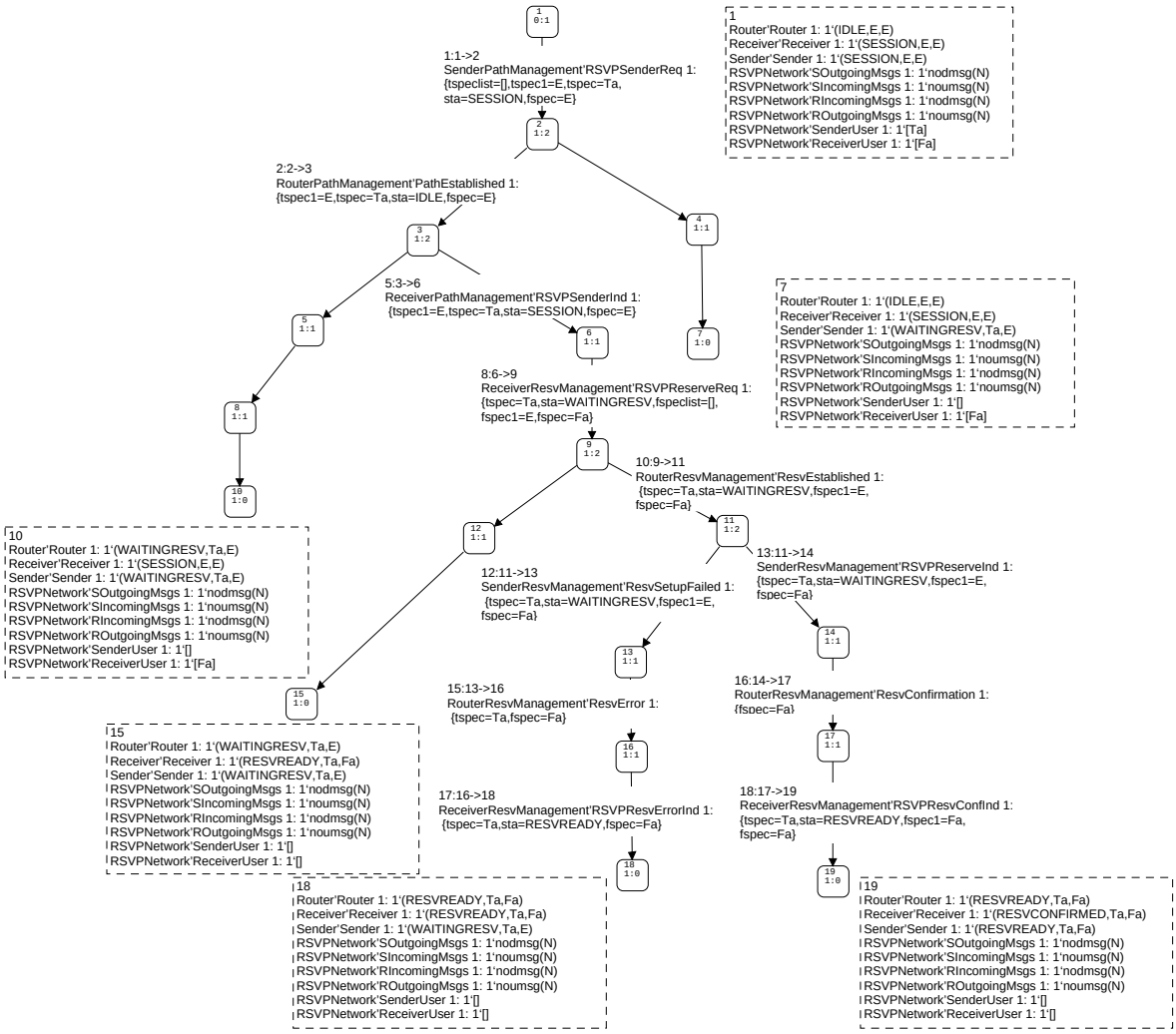

Figure 6: State space of the example.

In Figure 6, the path defined by the node sequence 1,2,3,6,9,11,14,17,19 shows the sequences of events leading to the successful establishment of a reservation, while the path 1,2,3,6,11,13,16,18 illustrates an event sequence where the reservation is not set up in the sender. The Sender couldn't establish the reservation (see arc 12) because, for example, it didn't have enough resources to satisfy the request. The other leaf nodes of the graph (7, 10 and 15) represent situations where the path is not established (7,10), or the reservation fails (15). The terminal nodes show that the completion of RSVP execution satisfies the following expected conditions: RSVP finishes in a state where all the message buffers are empty (represented by the tokens noumsgs (N) and nodmsgs (N)) and a path state has not been installed in at least one of the RSVP entities because the sender request has failed in one RSVP node, so no reservation exists (i.e. the RSVP entity, e.g. the Router, is not in the WAITINGRESV state) (e.g. see node 7). Alternatively, a reservation state has not been established in at least one of the RSVP entities (i.e. it is not in the RESVREADY or RESVCONFIRMED state) because the reservation request has failed in one RSVP node (e.g. see node 15); or a reservation state has been set up in all RSVP entities along the route of the data flow (i.e. they are in the RESVREADY or RESVCONFIRMED sate) (see node 19).

The size of the state space is increased as we increase the capacity of the buffers because the content of the communications places can have more than one message. In addition, the messages may be processed by the RSVP

entities in an arbitrary order. It is also increased when we include more features of RSVP. For example, when the Path and Resv refreshes are activated in the model, the state space has 281 nodes and 1049 arcs [20]. This is because both Path and Resv messages can be in the network simultaneously. This is not possible in the current model with the given initialisation because the Receiver must wait until it receives the first Path message to generate and send a reservation request, and only one Resv and one Path can be generated and sent to the network.

6.3 Strongly Connected Components

A *Strongly Connected Component (SCC)* of the occurrence graph is a maximal sub-graph, whose nodes are mutually reachable from each other [8]. A SCC graph has a node for each SCC and arcs that connect each SCC with other SCCs. A SCC without incoming arcs is called the *initial SCC*, and a SCC without outgoing arcs is called a *terminal SCC*. Each node in the state space belongs to only one SCC, so the SCC graph will never have more nodes than the corresponding occurrence graph (OG).

The SCC graph of the OG of Figure 6 will be the same as Figure 6, as it contains no cycles. To illustrate a non-trivial SCC graph we generate an OG for a RSVP model in which path refreshes are generated at the sender but no reservation request is sent by the receiver. The full state space has 20 nodes and 21 arcs, while the SCC graph has 6 nodes and 13 arcs as shown in Figure 7. Node number ~1 is the initial SCC. Terminal SCC nodes are indicated by solid boxes (i.e. nodes ~4 and ~6). Each SCC node has information about the number of nodes and arcs, which comprise the associated SCC. The number of OG arcs from one SCC node to another, is indicated next to the SCC graph arc. For example, node ~2 comprises four markings (OG nodes) and 5 OG arcs, one input arc and two output arcs to SCC node ~3.

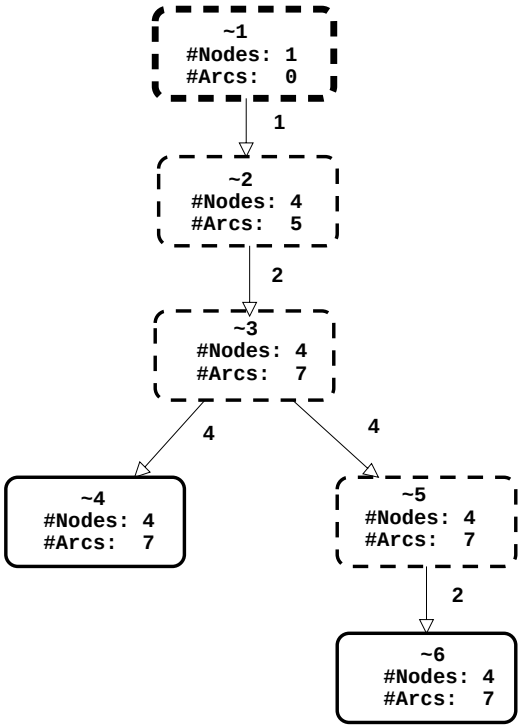

Figure 7: Example illustrating a SCC graph.

In the RSVP model the cycles indicated by the SCC graph are the result of refreshes. Terminal SCC nodes represent protocol sequences that can be executed indefinitely, possibly without making effective progress (i.e. livelocks exist). Thus they must be checked to see if these sequences are expected. In this case, these sequences must correspond to either successful path establishment or to a path establishment attempt that failed at the receiver. The terminal SCC sequences can be easily checked using standard Design/CPN functions [11] that are provided to explore OG nodes included in a SCC node. We have checked them (see [20]) and they are as expected.

7 Behavioural Properties of Coloured Petri Nets

This section introduces in an informal way the main behaviour or dynamic properties of CPNs and how they can be used for RSVP model analysis. They describe the expected behaviour of the model. More details about these properties and their formal definitions can be found in [8].

7.1 Reachability

By convention, Mn denotes the marking of node number n. Mn is reachable from M1 if there is an occurrence sequence from marking M1 to Mn. We have defined a set of desired properties that describe the expected behaviour of RSVP [20]. These are so called *safety properties*, which we have checked by establishing that the desired markings are reachable.

For example, an important property of RSVP is *Resv Setup*, which is defined as follows. If the RSVP-Reserve.Req service primitive occurs, RSVP should be able to establish or update a reservation along the route of the data flow, unless the sender has left the session before the reservation request is sent. Also, the corresponding path state information must exist at all RSVP nodes along the route of the data flow before the reservation is established. In Figure 6, we see that the reservation request primitive occurs in marking 6, and that the property is satisfied by marking 19. We see that marking 19 is reachable from marking 9 (the resultant marking on the occurrence of the Reservation request in marking 6) by the following sequence of markings: 9,11,14,17,19.

As the size of the state space increases, it is not possible to verify these safety properties using visual inspection. We developed the algorithm *Reachable* [20] to check if multiple nodes can reach at least one of the nodes in a list, since the in-built function provided by Design/CPN [11] only checks the reachability of one node from another. The algorithm is used to check RSVP's safety properties such as the Resv Setup property just explained.

7.2 Boundedness

The *upper* and *lower integer bounds* indicate the maximum and minimum number of tokens that can be located on each place in the reachable markings. For example, they are useful for detecting unbounded communication buffers. In Figure 6, the upper and lower bounds for the communication places are 1. This means that each of these places can either contain one message or an empty slot (represented by the token nodmsg (N)or noumsg (N)).

The other concept related to boundedness properties is *multi-set bounds*. They provide information about the value of the tokens that the places can carry. The *upper multi-set bound* of a place is defined as the smallest multi-set which is larger than or equal to all reachable markings of the place [16]. The *lower multi-set bound* of a place is defined as the largest multi-set which is smaller than or equal to all reachable markings of the place. For example, the best upper multi-set bound of the place Sender is: *1`(SESSION,E,E)++ 1`(WAITINGRESV,Ta,E)++ 1`(RESVREADY,Ta,Fa)*. This indicates that the Sender can be in all the defined states (see Section 5.4) except for CLOSED because the sender release feature (i.e. the transition RSVPSenderRel in Figure 2) has been switched off.

7.3 Home Markings

A *home marking* is a marking that can always be reached from all other reachable markings. In Figure 6, it can be seen that there is no home marking. A *home space* is a set of markings such that from each reachable marking, it is possible to reach at least one of these markings. Although there is no home marking in this model, the markings M7,M10,M15,M18 and M19 form a home space. This means that the protocol can always finish in an expected state (see Section 6.2).

7.4 Deadlock-Freeness

A dead marking is a marking with no enabled binding elements. In Figure 6, M7,M10,M15,M18 and M19 are dead markings, however they are expected (see Section 6.2) and hence are not deadlocks from an RSVP perspective.

7.5 Dead Transitions

A *dead transition* is not enabled in any reachable marking. When generating the state space shown in Figure 6, we have deactivated several transitions to make the state space small. For example, the refresh transitions, such as PathRfrTimeOut (see Figure 5), do not occur in the example, so they are dead transitions. Dead transitions represent code of the protocol, which is never executed. This situation is not acceptable, so the dead transitions must be analysed to see if they are the result of a modelling problem or a protocol specification problem.

8 Conclusions

In this paper, we have shown how RSVP can be formalised and analysed using CPNs. It summarises part of our effort for formally specifying and analysing RSVP [20]. We have described the protocol in a graphical way using the features of CPNs. The hierarchical structuring mechanism of CPNs allows us to identify the major entities

involved in the architecture (i.e. a sender, receiver and router), to describe the major modules that form the protocol (e.g. path and reservation management) and to describe the network topology considered in the example (i.e. one sender and one receiver connected by a router). Since CPN models are executable, we could investigate the behaviour of RSVP using simulations. The state space method of CPNs is used to verify and validate the model against a set of behavioural properties. The facilities provided by CPNs (and Design/CPN) allow us to create a model that provides a clear, unambiguous and precise definition of RSVP, and to check the protocol for correctness.

The main limitation found during the analysis of RSVP is that of state space explosion [17]. The problem with very large state spaces is that they cannot be generated with limited computer memory and if they can, they may be difficult to analyse. To make the analysis feasible we have limited the scope of the model, for example, we have only considered a very simple network topology and excluded some of RSVP's functionality such as merging.

We have managed to analyse RSVP under set of simplifying assumptions [20]. The analysis results have shown that the protocol works as expected under these assumptions. Future work in this area will attempt to relax some of these assumptions and may include modelling other features of RSVP (eg merging), more complex network topologies and inclusion of network imperfections such as message loss and/or duplication.

References

[1] Billington J. *Formal Specification of Protocols: Protocol Engineering.* Encyclopaedia of Microcomputers, Marcel Dekker, New York, 1991, Vol. 7, pp 299-314.

[2] Billington J., Diaz M. and Rozenberg G. (eds), Application of Petri nets to Communication Networks, Advances in Petri Nets, Lecture Notes in Computer Science, Vol. 1605, Springer-Verlag, 1999.

[3] Braden R., Clark D., and Shenker S. *Integrated Services in the Internet Architecture: an Overview.* RFC 1633, IETF, June, 1994.

[4] Braden R., et al. *Resource Reservation Protocol (RSVP) -- Version 1: Functional Specification*. RFC 2205, IETF, September, 1997.

[5] Durham D. and Yavatkar R. *Inside the Internet's Resource Reservation Protocol*. Wiley, USA, 1999.

[6] Holzmann G. *Design and Validation of Computer Protocols.* Prentice Hall. 1991.

[7] ITU-T *Convention for the Definition of OSI Services*. Recommendation X.210. 1994.

[8] Jensen K. Coloured Petri Nets: *Basic Concepts, Analysis Methods and Practical* Use. Vol. 1, 2 and 3. Springer-Verlag, 2nd edition, April, 1997.

[9] Figueiredo J. and Kristensen L.M. *Using Coloured Petri Nets to Investigate Behavioural Performance Issues of TCP Protocols*. Proceedings of Second Workshop on Practical use of Coloured Petri Nets, Aarhus, Denmark, 1999.

[10] Kristensen L.M., Christensen S., and Jensen K. *The practitioner's guide to coloured Petri nets.* International Journal on Software Tools for Technology Transfer, Springer, 1998, Vol. 2, Number 2, pp 98-132.

[11] Meta Software Corporation. *Design/CPN Reference Manual for X-Windows*, Version 2, Meta Software Corporation, Cambridge, 1993.

[12] Ouyang C., Kristensen L. and Billington J., *Towards Modelling and Analysis of Internet Open Trading Protocol Transactions using Coloured Petri Nets*. In Proc. 11th Annual International Symposium of the International Council of Systems Engineering (INCOSE 2001), Melbourne, Australia, 1-5 July 2001.

[13] Paulson L. *ML for the Working Programmer*. Cambridge University Press.1991.

[14] Proceedings of the (Joint) International Conferences on Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE) and Protocol Specification, Testing & Verification (PSTV), 1997-2001.

[15] Smith M. and Ramakrishnan K. K. *Formal Verification of Safety and Performance Properties of TCP Selective Acknowledgment.* In Proceedings of 1998 International Conference on Network Protocols, Austin, Texas, October 1998.

[16] University of Aarhus, Computer Science Department. *Design/CPN Occurrence Graph Manual*. Version 3.0. 1996.

[17] Valmari A. *The State Explosion Problem.* Lecture Notes in Computer Science, Vol. 1491, pp 429-528, Springer-Verlag, 1998,.

[18] Villapol M.E. and Billington J. *Modelling and Initial Analysis of the Resource Reservation Protocol using Coloured Petri Nets*, Proceedings of the Workshop on Practical Use of High-Level Petri Nets, Aarhus, Denmark, June 27, 2000, pp 91-110.

[19] Villapol M.E. and Billington J. *Generation of a Service Language for the Resource Reservation Protocol Using Formal Methods*, Proceedings of Eleventh Annual International Symposium of the International Council On Systems Engineering (INCOSE), 1-5 July 2001, on CD-ROM.

[20] Villapol M.E. *Modelling and Analysis of the Resource Reservation Protocol Using Coloured Petri Nets*. Draft Version of the Doctoral Thesis, University of South Australia, April 2002.