Analysing Properties of the Resource Reservation Protocol

María E. Villapol¹ and Jonathan Billington²

¹ School of Computer Sciences, Central University of Venezuela, Av. Los Ilustres, Los Chaguaramos, Caracas, Venezuela

mvillap@strix.ciens.ucv.ve

² Computer Systems Engineering Centre, University of South Australia, Mawson Lakes, Adelaide, SA, 5095, Australia

j.billington@unisa.edu.au

Abstract. The goal of the *Resource Reservation Protocol* (RSVP) is to establish *Quality of Service* information within routers and host computers of the Internet. This paper describes a model of RSVP and presents the analysis approach and results. A large part of RSVP is modelled using *Coloured Petri Nets*. The model provides a clear, unambiguous and precise definition of the considered features of RSVP, which is missing in the current protocol specification. The model is analysed for a set of general properties, such as correct termination, and a set of RSVP specific properties defined in this paper. The properties are checked by querying the state graph and its associated strongly connected component graph. As a first step, we analyse RSVP under the assumption of a perfect medium to ensure that protocol errors are not hidden by rare events of the medium. The results show that the RSVP model satisfies the defined properties.

1 Introduction

The *Resource Reservation Protocol* (RSVP) [2][5] is a signalling protocol developed to create and maintain resource reservations (e.g. buffer and data rate allocations) in Internet routers and host computers, to provide *Quality of Service (QoS)* guarantees for multimedia and real-time applications. For the desired QoS to be guaranteed it is essential that RSVP works correctly. The aim of our work is thus to provide a step towards the verification of the correctness of RSVP's mechanisms.

Formal methods provide techniques to support the design and maintenance of communication protocols [14]. It is uncommon for formal techniques to be applied to Internet protocols [6][12][16] and we have found little work related to a detailed study of RSVP except for our initial work [18][19]. In two similar papers, Creese and Reed [4] and Reed et al. [15] present an induction technique for proving properties of arbitrary configurations of nodes and illustrate their technique using one aspect of RSVP. This is that the reservation requests are acknowledged by the node where merging (of reservation requests) does not cause resulting state changes [2]. Our work is complementary, in that it does not address merging but instead covers a wide range of RSVP

features including the soft state mechanisms, which have not been considered in [4][15]. Also, we define an extensive set of properties against which RSVP is analysed.

Coloured Petri Nets (CPNs) [9] are a formal technique used for modelling many systems, particularly communication protocols [1]. In this paper, RSVP mechanisms are modelled using CPNs and analysed with a software tool called Design/CPN [11]. We define a set of desired properties for RSVP based on the specification [2]. The properties are checked by querying the state space using powerful ML [13] functions suited to the Design/CPN environment.

Our first attempt at modelling and analysis of RSVP was presented in [18]. Since then, the model has been significantly revised, restructured and refined to include more features (e.g. confirmation of reservations). The new model has been developed using an incremental modelling and validation methodology [20] to increase our confidence in its validity. The contribution of this paper is the presentation of the new model, the definition of RSVP's desired properties, and their verification.

The paper has been organised as follows. Sect. 2 presents an overview of RSVP, which includes its characteristics and operation. In order to define RSVP's properties, we summarise RSVP's application interface interactions, known as Service Primitives [19], in Sect. 3. Starting with a discussion of scope, Sect. 4 describes the RSVP CPN model. Desired RSVP properties are defined in Sect. 5 and verified in Sect. 6. Sect. 7 concludes and presents future research directions.

2 Resource Reservation Protocol Overview

RSVP is designed to be run on network routers and in end hosts to support a QoS application. It reserves resources for a data flow from the sender to one or more destinations (i.e. 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 [2]. 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 [2]. 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)* [2][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 (i.e. the data

rate that a data flow can use). A *TSpec* defines the traffic characteristics of the flow, for example, the peak data rate (i.e. the maximum rate at which the sender is intended to send packets).

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 Fig. 1. We now describe the main RSVP features and their associated messages, structured to facilitate the definition of the desired properties of the protocol in Sect. 5.

Fig. 1. RSVP messages

Path Setup. *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 set up and maintain path state information (e.g. the Internet Protocol address of the previous router and the data flow's traffic characteristics).

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 periodically (i.e. every refresh timeout period [2]) sends path refresh messages (i.e. *Path* messages).

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.

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.

Reservation Setup. *Resv* messages carry reservation requests (e.g. 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 [2]. Thus, the effective filter and flow specifications, which are carried in a reservation message, are the result of merging reservations from several requests.

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.

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.

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.

Reservation Confirmation. 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 (Fig. 1).

3 RSVP Service Primitives

Service primitives [8] 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 [2] does not define the RSVP service, the authors [19] 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 event. We have defined the following service primitives [19]. A sender application uses the *RSVP-Sender (Req/Ind)* primitive to establish or update the traffic characteristics of a data flow for a RSVP session. A receiver application uses the *RSVP-Reserve (Req/Ind)* primitive to establish or to modify a resource reservation during a session. A sender application uses the *RSVP-SenderRel (Req/Ind)* primitive to close a session. This means that the user data flow will eventually not have any QoS reserved. A receiver application uses the *RSVP-ReceiverRel (Req/Ind)* primitive to close a quality controlled session. The *RSVP-ResvConf (Ind)* primitive is used by the service provider to confirm that a reservation has been made. The *RSVP-SenderError (Ind)* primitive is used by the service provider to report an error in propagation or installation of the Sender's traffic characteristics. The *RSVP-ResvError (Ind)* primitive is used by the service provider to report a reservation failure inside the network.

4 CPN Model of RSVP

A model of RSVP is created with Coloured Petri Nets (CPNs) [9] using Design/CPN [11].

4.1 Scope

Since RSVP is a complex protocol, the scope of the model is limited to make analysis tractable. The CPN model has been developed based on the protocol specification [2] and includes most of RSVP's features. The network topology comprises two hosts (sender and receiver) and a single router between them (Fig. 1). This allows us to consider router behaviour in RSVP's operation. Although multiple RSVP sessions can be open simultaneously, without loss of generality, just one session is modelled to study the functional behaviour of RSVP, since RSVP treats each session independently. Following an incremental methodology, it is important to firstly consider a unicast network. The multicast case will be considered in future work.

4.2 Assumptions

Although the Internet Protocol (IP) over which RSVP operates, does not guarantee that the order of sending messages is maintained at the receiver, message overtaking is not included in the model. This is because we want to be sure that RSVP will work under normal conditions (re-ordering is a rare event), before considering unusual events. Similarly IP may lose messages. As mentioned before, RSVP Path and Resv refresh messages deal with occasional loss of RSVP messages. Although, message loss is not considered in the model presented here, the mechanisms for dealing with loss are modelled.

4.3 Model Hierarchy

We deal with RSVP's complexity by using the hierarchical constructs of CPNs [9]. 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, the *Global Declaration node*.

The hierarchy page of the RSVP CPN model consists of eleven (11) pages as illustrated in Fig. 2. The top-level page is called RSVPNetwork (Fig. 4), which describes the network topology (Fig. 1) and interaction with the applications that use RSVP. This page 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 to the major functions of RSVP described Sect. 2. The Path and Resv management pages include transitions that model the establishment, refreshment, release and error control of paths and reservations, respectively. Also in-

cluded is the Global Declaration node (Page 11). Each page at the lower level of the hierarchy uses transitions to model service primitives and protocol actions, which include implementing RSVP functions (e.g. path refresh) or discarding messages that cannot be processed.

Fig. 2. Hierarchy page of the RSVP CPN model

4.4 Global Declaration

Fig. 3 shows the colour sets (types), variables, and functions from the global declaration node. The colour set `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). These values allow us to test RSVP for changes to both the traffic specification and the reservations. The empty value (`E`) is used for initialisation. The colour sets, `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. The colour set `Status` is an enumeration type, which defines the states of the RSVP entities. The values of this colour set have the following meanings:

```

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 ReceiverStatus  = subset Status with [SESSION,WAITINGRESV,RESVREADY,
                        RESVCONFIRMED,CLOSED];
color RouterStatus    = subset Status with [IDLE,WAITINGRESV,RESVREADY];
color SenderState     = product SenderStatus * STSpec * SFSpec;
color ReceiverState   = product ReceiverStatus * STSpec * SFSpec;
color RouterState     = product RouterStatus * STSpec * SFSpec;
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;
color UserInd         = with USR|NOUSR;
color Flag             = int with 0..1;
color FSpecXFlag       = product FSpec * Flag;
color TSpecXFlag       = product TSpec * Flag;
(* ===== Application ===== *)

color TSpecList = list TSpec;
color FSpecList = list FSpec;
(* ===== Variables ===== *)

var sta: Status;
var tspec,tspec1: STSpec;
var fspec,fspec1: SFSpec;
var flag: Flag;
var rmtusrind: UserInd;
var ttype: TearMsgType;
var tspeclist: TSpecList;
var fspeclist: FSpecList;

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

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

```

Fig. 3. Global declarations

1. SESSION: indicates that the sender or receiver has opened a session, but no path information or reservation has yet been established.
2. IDLE: there exists neither path nor reservation information at the router.
3. WAITINGRESV: means that 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.
4. RESVREADY: means that a reservation request has been accepted and sent (if the entity is not the Sender).
5. RESVCONFIRMED: means that a reservation has been established and a confirmation has been received.
6. CLOSED: indicates that the sender or the receiver has left the session.

Three subsets of the colour set Status have been defined: the SenderStatus, RouterStatus and ReceiverStatus. The colour sets SenderState, RouterState and Receiver-

State represent the states of the sender, router and receiver entities respectively. Each of them is the product of the status of the corresponding entity (i.e. SenderStatus, RouterStatus or ReceiverStatus), STSpec and SFSpec.

The colour sets, TSpec and Fspec, represent the parameters that may be carried in RSVP messages (see [2]) and are the traffic specification and flow specification, respectively. The colour set TearMsgType is intended to distinguish between a PathTear or ResvTear 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 Sect. 2 are represented by the colour sets UpstreamMessages and DownstreamMessages.

Colour set UserInd indicates whether the user at the other end has left the session (NOUSR) or not (USR). It is used to control the service primitive sequences allowed for each service user (see [20]).

Colour set FSpecXFlag indicates whether the RSVP-Reserve.Ind service primitive, which includes the requested FSpec, has occurred (flag is ON) or not (OFF). It is used to avoid multiple occurrences of this primitive with the same value of the flow specification. Similarly, colour set TSpecXFlag indicates whether the RSVPSender.Ind service primitive has occurred or not, avoiding multiple occurrences [20]. TSpecList and FSpecList are lists of traffic and flow specifications, respectively, that are requested by users.

The variables used in CPN inscriptions are typed in the declarations. As an example, the variable *rmtusrind* represents the state of a remote user. Two functions are used to simplify guard inscriptions. *Pathexists* returns true if the sender has path state information available, and similarly *resvexists* returns true if the receiver's state includes reservation information.

4.5 RSVP Network Page

The top-level CPN page, *RSVPNetwork*, is shown in Fig. 4. It shows the network topology including interaction with RSVP's users (applications). The three shaded rectangles (Sender, Router and Receiver) represent RSVP entities and are *substitution transitions*. A substitution transition is identified by a HS-tag in the lower left corner of the rectangle representing the transition. The ellipse, SenderUser, is a CPN *place* and is typed by TSpecList defined in Fig. 3. It represents a queue of possible traffic characteristics of the data flow requested by the sender user. Similarly the place ReceiverUser is typed by FSpecList and represents a queue of flow specifications requested by the receiver user. A *marking* of a place comprises a (multi) set of values (known as *tokens*) taken from the place's type. For example, SenderUser has the initial marking of a list of two traffic specifications [Ta,Tb]. The places SOutgoingMsgs and RIncomingMsgs have markings that represent RSVP messages travelling downstream, while ROutgoingMsgs and SIncomingMsgs have markings that represent RSVP messages travelling upstream. Places and substitution transitions are connected by *arcs*, which indicate the type of data required or produced by the substitution transitions.

Fig. 4. RSVP Network page

The complete RSVP model [20] cannot be described in this paper due to space limitations. We therefore only describe the Sender and Sender-Path-Management pages (see Fig. 2), which are representative of the operation of the CPN model.

4.6 RSVP Sender Page

The *RSVP-Sender* page (Fig. 5) includes two substitution transitions, PathManagement and ResvManagement and a new place Sender. PathManagement is described below. ResvManagement models the establishment, refreshment, error control and release of reservation state information. The place Sender is typed by SenderState and models the status of the sender together with path and reservation information. The other places were described in the previous section (see Fig. 4).

Fig. 5. Sender page

4.7 Path Management Page

The Sender-Path-Management page is shown in Fig. 6 and comprises five transitions and four places that have already been described. In Fig. 6, transition `RSVPSenderReq` models the establishment or updating of path state information and the occurrence of the service primitive `RSVP-Sender.Req`. `RSVPSenderReq` is enabled when the Sender is not `CLOSED` and a new *tspec* is waiting to be sent in `SenderUser`. Its occurrence updates the *tspec* to the one requested by the user, initially (i.e. if the sender status is `SESSION`) also updates the sender status to `WAITINGRESV` (i.e. ready to receive a reservation request), removes the *tspec* from the place `SenderUser` and sends it in a Path message by adding the corresponding token to `SOutgoingMsgs`.

The transition `PathRfrTimeout` models periodic path refreshes required to maintain path state information. It is enabled after the initial sender request has occurred, but the session isn't closed. An occurrence of the transition does not change the state of the sender but adds a Path message carrying the corresponding *tspec*, to the place `SOutgoingMsgs`. The other three transitions model the sender processing errors or leaving the session.

Fig. 6. Path Management page

5 Desired Properties of RSVP

We wish to verify RSVP against a set of desirable properties. They include standard protocol properties such as freedom from deadlock and absence of livelocks and RSVP specific properties concerning the setup, maintenance and release of both paths

and reservations. They are formalised in the following subsections. In this description, the term *node* refers to a marking in the occurrence graph (OG); while *RSVP node* is a network device (e.g. a router), which supports RSVP.

Firstly, we summarise the notation that we need, some of which is standard CPN notation [9]:

- $B(t)$ denotes the set of all bindings of the variables associated with transition t .
- $M(p)$ denotes the marking of place p .
- $[M >$ denotes the set of markings that are reachable from marking M , and with $[M_0 >$ denoting the set of markings reachable from the initial marking M_0 .
- DM denotes the set of all dead markings.
- SCC_T denotes the set of all terminal SCCs.
- $M \xrightarrow{(t,b)} M'$ denotes that marking M' is directly reachable from marking M by the occurrence of the binding element (t,b) , $b \in B(t)$.
- $Proj_1(x,y,z) = x$ is the first component projection function for a triple (x,y,z) .
- $Proj_2(x,y,z) = y$ is the second component projection function for a triple (x,y,z) .
- $Proj_3(x,y,z) = z$ is the third component projection function for a triple (x,y,z) .
- $\emptyset$ is the empty (multi)set (depending on context).

5.1 Termination Property

RSVP must finish in a state where: all the message queues are empty; any path or reservation state information has been removed from the RSVP nodes; and the sender and receiver have left the session (i.e. are in the CLOSED state).

Definition 1. *The RSVP CPN model terminates correctly iff*

$$\forall M \in DM, Proj_1(M(Sender)) = Proj_1(M(Receiver)) = CLOSED \wedge Proj_1(M(Router)) = IDLE \wedge (M(SOutgoingMsgs) = M(SIncomingMsgs) = M(ROutgoingMsgs) = M(RIncomingMsgs) = \emptyset)$$

where the marking of a state place is converted from a singleton multiset comprising one token to its token (e.g. we replace $1'(a,b,c)$ by (a,b,c)) before the projection function is applied.

5.2 Livelock Property

Livelock [7] occurs when protocol sequences are executed indefinitely without the possibility of making effective progress. Thus it is important to check that the RSVP model doesn't contain livelocks. Livelocks will be revealed as terminal strongly connected components of the OG that include cycles (and hence are not dead markings). We define a CPN to be livelock free if it has no terminal SCC (SCC_T) that contains cycles.

Definition 2. *A CPN is livelock free if $|SCC_T| = |DM|$*

This is equivalent to checking that the terminal markings form a *home space*. 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 [9].

5.3 Path Setup Property

Once the RSVP-SenderReq service primitive occurs, RSVP should be able to establish or update the path state in all RSVP nodes on the route of the data flow, unless the receiver has closed the session before the RSVP-SenderReq service primitive occurs. Let tsr be the transition $RSVP\text{SenderReq}$. The property is defined as follows:

Definition 3. *The RSVP CPN model satisfies the path set-up property iff*
For $M, M' \in [M_0 >$,

$$Proj_1(M(Receiver)) \neq CLOSED \wedge M \xrightarrow{(tsr, tspec=rtspec)} M' \Rightarrow \exists M'' \in [M' > : \\ Proj_2(M''(Sender)) = Proj_2(M''(Receiver)) = Proj_2(M''(Router)) = \\ rt-spec$$

5.4 Path Maintenance Property

If the path state information, for a traffic specification that has been established along all RSVP nodes on the route of the data flow, is removed in any of those nodes as a result of the expiration of the path cleanup timer, it is possible for it to be established again. This cannot occur if either the sender or receiver user closes the session before the timer expires.

Let $\mathcal{T}_{pc} = \{\text{RouterPathCleanup}, \text{ReceiverPathCleanup}\}$ be the set of path cleanup transitions and $tpc \in \mathcal{T}_{pc}$. Let $\mathcal{M}_{PE}$ be the set of markings where a path has just been established in all RSVP nodes:

$$\mathcal{M}_{PE} = \{M \in [M_0 > | \\ Proj_1(M(Sender)) = Proj_1(M(Receiver)) = Proj_1(M(Router)) = WAITINGRESV\}$$

Definition 4. *The RSVP CPN model satisfies the path maintenance property iff*
For $M \in [M_{pe} >$, where $M_{pe} \in \mathcal{M}_{PE}$, and $b \in B(tpc)$

$$Proj_1(M(Sender)) \neq CLOSED \wedge Proj_1(M(Receiver)) \neq CLOSED \wedge M \xrightarrow{(tpc, b)} M' \\ \Rightarrow \exists M'' \in [M' > : Proj_1(M''(Sender)) = Proj_1(M''(Receiver)) = Proj_1(M''(Router)) = \\ WAITINGRESV$$

5.5 Path Release Property

Once the RSVP-SenderRelReq service primitive occurs, the path and corresponding reservation state must be removed from all the nodes along the route of the data flow. Let tpr denote the transition $RSVP\text{SenderRelReq}$. The property is defined as follows:

Definition 5. The RSVP CPN model satisfies the path release property iff
For $M \in [M_0 >$, and $b \in B(tpr)$
 $M \xrightarrow{(tpr,b)} M' \Rightarrow \forall M'' \in [M' >, \exists M''' \in [M'' > \mid Proj_1(M''(Sender)) = CLOSED$
 $\wedge Proj_1(M''(Router)) = IDLE \wedge (Proj_1(M''(Receiver)) = CLOSED \vee$
 $Proj_1(M''(Receiver)) = SESSION)$

5.6 Resv Setup Property

If the RSVP-ReserveReq service primitive occurs, RSVP should be able to establish or update the requested reservation along the route of the data flow, unless the sender has left the session before the reservation request is sent. Also, path state information must exist at any RSVP node before the reservation is established.

Let $Trs = \{SenderRSVPReserveInd, SenderReserve, RouterResvEstablished, ReceiverReserveReq, ReceiverReserve\}$ be the set of reservation setup transitions. $trs \in Trs$.

Definition 6. The RSVP CPN model satisfies the path set-up condition (i.e. path state information must exist before any trs transition occurs) iff

For $b \in B(trs)$
 $\forall M \in [M'] \mid M' \xrightarrow{(trs,b)} M'', M' \in [M_0 > \mid Proj_1(M(Sender)), Proj_1(M(Router)) \in \{WAITINGRESV, RESVREADY\} \wedge Proj_1(M(Receiver)) \in \{WAITINGRESV, RESVREADY, RESVCONFIRMED\}$

Let trr denote the transition RSVPReserveReq. The property is defined as follows:

Definition 7. The RSVP CPN model satisfies the reservation set-up condition iff

For $M, M' \in [M_0 >$,
 $Proj_1(M(Sender)) \neq CLOSED \wedge M \xrightarrow{(trr, fspec=rfspec)} M' \Rightarrow \exists M'' \in [M' > :$
 $Proj_3(M''(Sender)) = Proj_3(M''(Receiver)) = Proj_3(M''(Router)) = rfspec$

Definition 8. The RSVP CPN model satisfies the reservation set-up property if Definition 6 and Definition 7 are satisfied.

5.7 Resv Maintenance Property

If the reservation state information for a traffic specification that has been established in all RSVP nodes on the route of the data flow is removed in any of those nodes as a result of expiration of the path or reservation cleanup timer, it can be established again. This cannot occur if either the sender or receiver user closes the session before the timer expires.

Let $Trc = \{RouterPathCleanup, ReceiverPathCleanup, RouterResvCleanup, SenderResvCleanup\}$ be the set of cleanup transitions and $trc \in Trc$. Let M_{RS} be the

set of markings where a reservation has just been established in all RSVP nodes and defined by:

$$\mathcal{M}_{RS} = \{ M \in [M_0 > \mid Proj_I(M(Sender))=Proj_I(M(Receiver))=Proj_I(M(Router))=RESVREADY \}$$

Definition 9. *The RSVP CPN model satisfies the resv maintenance property iff*

For $M \in [M_{rs} >$, $M_{rs} \in \mathcal{M}_{RS}$, $b \in B(trc)$

$$Proj_I(M(Sender)) \neq CLOSED \wedge Proj_I(M(Receiver)) \neq CLOSED \wedge M \xrightarrow{(trc,b)} M' \\ \Rightarrow \exists M'' \in [M' > : Proj_I(M''(Sender))=Proj_I(M''(Receiver))=Proj_I(M''(Router))=RESVREADY$$

5.8 Resv Release Property

If the RSVP-ReceiverRelReq service primitive occurs, the reservation must be removed from all RSVP nodes.

Let trr denote the transition RSVPReceiverRelReq. The property is defined as follows:

Definition 10. *The RSVP CPN model satisfies the resv release property iff*

For $M \in [M_0 >$, and $b \in B(trr)$

$$M \xrightarrow{(trr,b)} M' \Rightarrow \forall M'' \in [M' >, \exists M''' \in [M'' > \mid (Proj_I(M'''(Sender))=WAITINGRESV \vee Proj_I(M'''(Sender))=CLOSED) \wedge (Proj_I(M'''(Router))=WAITINGRESV \vee Proj_I(M'''(Router))=IDLE) \wedge Proj_I(M'''(Receiver))=CLOSED \}$$

6 Analysis and Verification

Design/CPN [11] is used to simulate and analyse the CPN model. The model is analysed by generating the OG [9] and its corresponding *Strongly Connected Component* (SCC) graph. Design/CPN was run on a 1.6 GHz Linux Pentium 4 PC with 1 GB RAM.

The RSVP model was developed and analysed incrementally by adding the features described in Sect. 2 (e.g. path refresh, path release) one at a time. This facilitates the validation task and increases confidence in the model. The model analysed in this paper includes all of these features.

The CPN model of Sect. 4 can generate an infinite state space due to periodic refreshes and unbounded communication places (e.g. see transition PathRfrTimeOut in Fig. 6). We thus modify the model so that the communication places have finite capacity using a standard approach [10].

The properties are checked by implementing the definitions in Sect. 5 as OG query functions in ML [13]. To illustrate the approach, this section describes the ML query for one of the Path Setup property and how livelocks are detected. The description of

the queries for the others can be found in [20]. All properties, except for the termination and livelock properties, are verified by examining the nodes of the OG and checking reachability among multiple nodes by using the SCC graph (instead of the OG) because it is more efficient.

6.1 Initialisation

The RSVP model is initialised by distributing tokens to one or more places of the model to create the *initial marking*. Each of the places SenderUser and ReceiverUser contains a list of the requested traffic or reservation characteristics, respectively, as specified in Fig. 4. Each communication place contains a single empty slot token (*uslot(N)* or *dslot(N)*) indicating that there is space for one message. Each state place (Sender, Router and Receiver) contains a triple comprising null entries for path and reservation information and SESSION for the status of the sender and receiver and IDLE for the router status.

6.2 Occurrence Graph and SCC Graph Statistics

In the course of our investigations [20], a series of state spaces was generated for different models, that were incrementally developed to include more features, and for different initial markings. This was to gain further confidence in the model as it was developed. For example, in the simplest case, we just considered one sender traffic spec (Ta) and reservation user request (Fa) over a channel of capacity one. Here we only present the results for the fully featured model, for the initial marking given above.

The size of the OG and its SCC graph and the corresponding generation times using Design/CPN are shown in Table 1. The table also includes the number of dead markings and terminal SCC nodes. Design/CPN output shows that each terminal SCC node contains one marking, corresponding to a dead marking. Thus there are no livelocks. The size of the SCC graph is smaller than the OG indicating that there are cycles. This is expected due to the soft-state mechanisms.

Table 1. State space results

Statistical Information	OG	SCC graph
Number of Nodes	154986	56962
Number of Arcs	769880	428644
Number of Dead Markings	85	-
Number of Terminal SCC nodes	-	85
Calculation Time (hh:mm:ss)	3:27:20	00:05:57

A large number of dead markings (85) occurs because of the possible markings of places other than those concerned with communication and RSVP entity states, such as user places (e.g. SenderUser) and some control places that exist in the model.

6.3 General Properties

Boundedness and *liveness* [9] were investigated to validate and debug the model and to provide insight into RSVP's behaviour. This information is included in the standard report for the state space generated by Design/CPN [11].

Boundedness. Integer and multi-set bounds were analysed for the places of the model. For example, Table 2 lists the upper integer and multi-set bounds for the communication places. Upper Integer bounds describe the maximum number of tokens that can occur in a place, while multi-set bounds indicate which tokens can occur in a place [10].

Table 2. Upper bounds for communication places

Place	Integer Bounds	Multi-set Bounds
SOutgoingMsgs	1	1`path(Ta)++ 1`path(Tb)++ 1`resverror(Fa)++ 1`resverror(Fb)++ 1`resvconf(Fa)++ 1`resvconf(Fb)++ 1`pathtear((REL,Ta))++ 1`pathtear((REL,Tb))++ 1`dslot(N)
RIncomingMsgs	1	1`path(Ta)++ 1`path(Tb)++ 1`resverror(Fa)++ 1`resverror(Fb)++ 1`resvconf(Fa)++ 1`resvconf(Fb)++ 1`pathtear((TEARDOWN,Ta))++ 1`pathtear((TEARDOWN,Tb))++ 1`pathtear((REL,Ta))++ 1`pathtear((REL,Tb))++ 1`dslot(N)
SIncomingMsgs	1	1`patherror(Ta)++ 1`patherror(Tb)++ 1`resvtcar((TEARDOWN,Fa))++ 1`resvtcar((TEARDOWN,Fb))++ 1`resvtcar((REL,Fa))++ 1`resvtcar((REL,Fb))++ 1`resv(Fa)++ 1`resv(Fb)++ 1`uslot(N)
ROutgoingMsgs	1	1`patherror(Ta)++ 1`patherror(Tb)++ 1`resvtcar((REL,Fa))++ 1`resvtcar((REL,Fb))++ 1`resv(Fa)++ 1`resv(Fb)++ 1`uslot(N)

Each of the communication places may have a maximum number of one token. This corresponds with the maximum capacity of the message buffer (i.e. one). The multi-set bounds show that all RSVP messages can be generated by the protocol. For example, the place RIncomingMsgs can contain a token representing a Path message, which carries the traffic characteristics of the data flow (Ta). It may also contain a token representing a ResvErr message, which indicates an error. The message carries the reservation characteristics (Fa) for which the request failed. Similar information can be derived from the multi-set bounds for the other places.

The boundedness results are as expected and further confirm that the model is valid.

Dead Transitions. A *dead transition* is one that is not enabled in any reachable marking [9]. The Design/CPN standard report showed that there are no dead transitions. This is expected as there should be no 'dead code' in the specification.

6.4 ML Query for Path Set-up

The Path Setup property is checked using the following ML function:

```
1 fun PathSetup (des) = let
2   (* Returns the SCC nodes which include at least one
   path-setup node *)
3   val pathsetupasccs = PathSetupNodes (des);
4   val senderreqdasccs = SenderReqDestNodes(des);
5   fun PSAN x = ListMember (x,pathsetupasccs);
6   fun SRAN x = ListMember (x, senderreqdasccs);
7   (* Checks if all SenderReqDest Nodes can reach any
   PathSetup Node *)
8   val vl,r1,m1 =Reachable ([~1],SRAN,PSAN,[],[],0);
9 in
10  r1
11 end;
```

The function, *PathSetup*, returns each of the destination markings of an occurrence of the RSVPSenderReq transition that cannot reach a marking where a path state for a requested traffic specification (des, line 1) (see Fig. 3) has been established. The function *PathSetupNodes* (line 3) returns the SCC nodes that include at least one OG node where a path has been established. This means that the status of the RSVP entities is equal to WAITINGRESV, RESVREADY or RESVCONFIRMED and the requested *tspec* corresponds to the *tspec* in the state of the entity (see Fig. 6). The function *SenderReqDestNodes* (line 4) returns the SCC nodes that include at least one destination OG node of the occurrence of the RSVPSenderReq transition. Each of the functions *PSAN* and *SRAN* (lines 5,6) is a predicate that checks if a SCC node is a member of the list returned by the function *PathSetupNodes* or *SenderReqDestNodes*, respectively.

We developed the algorithm *Reachable* 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. In order to check multiple nodes, the Design/CPN function must be invoked several times. This is inefficient when there are many nodes. The *Reachable* algorithm is described in detail in [20]. In line 8, the function *Reachable* traverses the SCC graph. It returns the SCC nodes (*r1*), which belong to the list returned by the function *SenderReqDestNodes*, that cannot reach any SCC node, which belongs to the list returned by the function *PathSetupNodes*. In order to say that the property is satisfied, the *PathSetup* function must return an empty list (line 10).

6.5 RSVP Specific Properties

The CPN model is verified against the desired RSVP properties by running all the corresponding queries together in batch mode. Table 3 lists the analysis results, which indicate that all the properties are satisfied (OK). It also includes the approximate

time spent for running all the functions that implement the properties. The results show that the RSVP model works as expected under the assumptions we have made.

Table 3. Analysis results

Property	Result
Termination	OK
Livelock	OK
Path Setup	OK
Path Maintenance	OK
Path Release	OK
Resv Setup	OK
Resv Maintenance	OK
Resv Release	OK
Estimated Verification Time (hh:mm:ss)	8:54:10

7 Conclusions

Coloured Petri Nets have been used to model the main features of RSVP based on a number of assumptions. We use a simple representative network topology and a limited and reliable communication medium to verify that RSVP will operate correctly under ideal conditions. This is a necessary first step.

The main problem found during modelling was the lack of a well-defined specification of RSVP [2], where only a narrative description is provided. The resulting model provides a clear, unambiguous and precise definition of the considered features of RSVP. The model was developed incrementally and checked at each stage to reduce the possibility of modelling errors.

RSVP uses soft-state refresh and clean-up mechanisms, which are modelled in a non-deterministic way. Similar mechanisms are used in other Internet protocols (e.g. several routing protocols [5]). Therefore, the RSVP model can be used as a reference for modelling other protocols that use similar procedures.

The model is analysed based on general properties of the protocol and a set of eight desirable properties defined and formalised for the first time in this paper. The analysis of the protocol is carried out by querying the state space and SCC graph. For 6 of RSVP properties the use of the SCC graph greatly enhances the efficiency of the model checking process. The analysis of the model shows that RSVP works as expected under our simplifying assumptions.

Normally model checking uses a form of temporal logic (such as Computation Tree Logic (CTL)) to express properties. In our case, we have found the use of ML queries to be of immediate benefit with Design/CPN, without having to master another formalism. However, it is interesting to compare our current predicate logic/ML approach with using a temporal logic model checker. Design/CPN provides a temporal logic model checking library called ASK-CTL [3], which implements a CTL-like temporal logic. We translated our definitions of properties into ASK-CTL formulas (see [20]). Our current results indicate that the ML approach is significantly faster than the ASK-CTL approach. In the future, we intend to compare our model checking

approach with the ASK-CTL approach in terms of expressiveness and elegance of property definition. It will also be beneficial to compare our use of the SCC graph, with the use of SCCs for optimisation of ASK-CTL verification as discussed in [3].

Further work will extend the model to include multicast operation and merging, and to relax the assumptions on the communication channels. However, analysing this model will be a challenge, due to its inherent complexity. Thus it will be necessary to explore ways of making the model tractable for analysis, including the use state space reduction techniques [9] [17].

This work extends the application of formal methods to a new protocol that uses a soft-state approach, proposed for providing QoS guarantees over the Internet. The results presented here are being extended to the verification of RSVP against its service specification [19][20].

Acknowledgements

We would like to acknowledge Dr Lars Kristensen for his advice in the analysis of the protocol and for suggesting the use of SCC graphs to improve the performance of the ML queries. We would also like to thank Dr Laure Petrucci for her comments on the formalisation of RSVP's properties.

References

- [1] Billington J., Diaz M. and Rozenberg G. (eds), *Application of Petri nets to Communication Networks*, Advances in Petri Nets, LNCS, Vol. 1605, 1999.
- [2] Braden R., et al. *Resource Reservation Protocol (RSVP) -- Version 1: Functional Specification*. RFC 2205, IETF, September, 1997.
- [3] Cheng A. and Christensen S. and Mortensen K. *Model Checking Coloured Petri Nets Exploiting Strongly Connected Components*, in Proc. Of the International Workshop on Discrete Event Systems, Institute of Electrical Engineers, University of Edinburgh, UK, August 1996, pp 169-177.
- [4] Creese S and Reed J N, *Verifying End-to-End Protocols Using Induction with CSP/FDR*, Parallel and Distributed Processing, 13th International Parallel Processing Symposium and 10th International Symposium on Parallel and Distributed Processing, Puerto Rico, April 1999, LNCS 1586, Springer.
- [5] Durham D. and Yavatkar R. *Inside the Internet's Resource Reservation Protocol*. Wiley, USA, 1999.
- [6] Han B. and Billington J. *Validating TCP Connection Management*. Proceedings of the Workshops on Software Engineering and Formal Methods and Formal Methods Applied to Defense Systems, Adelaide, Australia, June 2002, pp 47-55.
- [7] Holzmann G. *Design and Validation of Computer Protocols*. Prentice Hall. 1991.
- [8] ITU-T *Convention for the Definition of OSI Services*. Recommendation X.210. 1994.
- [9] Jensen K. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Vol. 1, 2 and 3. Springer-Verlag, 2nd edition, April, 1997.
- [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 LM and Billington J. *A Formal and Executable Specification of Internet Open Trading Protocol*. Third International Conference, EC-Web 2002 Aix-en-Provence, France, September 2002, LNCS 2455, pp 377-387.
- [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), 1997-2000.
- [15] Reed JN, Jackson DM, Deianov B. and Reed GM. *Automated Formal Analysis of Networks: FDR Models of Arbitrary Topologies and Flow-Control Mechanisms*. Proceedings of Joint European Conferences on Theory and Practice of Software, ETAPS98, as part of Fundamental Approaches to Software Engineering, FASE98, Lisbon, LNCS 1382, Springer.
- [16] Smith M.A. *Formal Verification of Communication Protocols*. In Reinhard Gotzhein and Jan Brederke, editors Formal Description Techniques IX: Theory, Applications, and Tools FORTE/PSTV'96: Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification, Kaiserslautern, Germany, October 1996, pp 129-144. Chapman & Hall, 1996.
- [17] Valmari A. *The State Explosion Problem*. Lectures on Petri Nets I: Basic Models, Vol. 1491, 1998, pp 429-528.
- [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*. Doctoral Thesis, University of South Australia, March 2003.