RSVP/ns: An Implementation of RSVP for the Network Simulator ns-2

Marc Greis
greis@cs.uni-bonn.de
Computer Science Department IV
University of Bonn

Abstract

The purpose of the Integrated Services Architecture as defined by the IETF is to provide Quality of Service (QoS) to applications which demand more from the network than IP's best-effort service, such as multimedia applications. The Resource ReSerVation Protocol RSVP [RFC2205], a part of the Integrated Services framework allows applications to perform receiver-initiated reservations for unicast or multicast flows, thus requesting resources (e.g. bandwidth or a bounded delay) from the network nodes on the path to the sender. However, to this date RSVP has not yet been deployed in larger portions of the Internet, so that large-scale research concerning RSVP has to rely on simulation results. "RSVP/ns" is an implementation of RSVP for the network simulator ns-2 [NS] which is a part of the VINT project [VINT]. This article serves both as internal and external documentation for RSVP/ns version 0.5.

1 Introduction

This article is meant for users of ns-2 who want to use this implementation of RSVP in ns-2 (from now on referred to as "RSVP/ns") in their own simulations and for those who want to understand and extend the underlying code. Section 2 describes the basic features of RSVP/ns and also the differences between RSVP/ns and the Functional Specification [RFC2205] and the Message Processing Rules [RFC2209]. Section 3 explains the implementation itself, assuming that the reader is familiar both with RSVP and the basic internal structure of ns-2. Section 4 describes how to add RSVP/ns to ns-2 and how to use RSVP/ns in simulation scenarios. Readers who only want to create their own simulation scenarios with RSVP/ns can skip sections 2 and 3. Section 5 gives a few examples to validate the correctness of RSVP/ns.

2 Features of RSVP/ns

This section describes which parts of the relevant RFCs have been implemented in RSVP/ns and where RSVP/ns differs from these RFCs. There are various reasons why an implementation of RSVP in a simulation scenario would be different from "real" implementations:

- The simulation environment imposes certain limitations on the implementation.
- Certain things can be easier to implement in the simulation environment.
- The original purpose of the implementation limits the necessary level of detail.
- The available memory and CPU power limits the possible level of detail.

The last point was the reason for most of the current differences between RSVP/ns and the relevant RFCs. However, it has been one of the main objectives during the implementation of RSVP/ns to make it a subset of RSVP that can easily be extended to RSVP's full functionality in the future.

2.1 Features and non-features

RSVP/ns currently includes the following features:

- Controlled Load bandwidth guarantees based on WFQ (or WF^2Q) scheduling in the links.
- Soft state with freely adjustable refresh intervals
- FF reservation style
- Gathering of statistics in the links and nodes
- Interfaces to parameter based and measurement based admission control
- A simple API with configurable upcalls for events
- The possibility of reserving a portion of a link's bandwidth for RSVP messages to avoid the loss of RSVP messages.

The following features have not been implemented in RSVP/ns yet:

- WF and SE reservation styles
- Blockade state
- ADSPEC objects
- INTEGRITY objects
- Policy control mechanisms

2.2 RSVP objects

Table 1 shows which RSVP objects have been implemented in RSVP/ns. All of these objects have been implemented to reflect both the object lengths for IPv4 and IPv6. Even though there is no explicit implementation of IPv6 in ns-2 it has to be assumed that users may want to simulate IPv6 networks in the future, in which case IPv6 message lengths should be simulated correctly.

Object	Class-Num
SESSION	1
RSVP_HOP	3
TIME_VALUES	5
ERROR_SPEC	6
STYLE	8
FLOWSPEC	9
FILTER_SPEC	10
SENDER_TEMPLATE	11
SENDER_TSPEC	12
RESV_CONFIRM	15

Table 1: RSVP objects in RSVP/ns

Object	Class-Num
INTEGRITY	4
SCOPE	8
ADSPEC	13
POLICY_DATA	14

Table 2: RSVP objects not implemented in RSVP/ns

There are various small differences between the objects in RSVP/ns and RSVP. One of the most important differences is the omission of the DstPort (destination port) and protocol id field in the SESSION object. In RSVP/ns, sessions are defined only by their their flow id. The destination address was included in the SESSION object, but only to simplify the RSVP message processing. The reason for that is the way the mechanism for classifying packets into service classes was implemented. This mechanism is based on a packet's flow id and the sender address. However, this is not really a difference between RSVP and RSVP/ns (even though there is no flow id in IPv4), since a function can be defined which maps the source port, the destination port and the protocol id onto a flow id and vice versa[1]. To be exact, this is a possibly useful extension of RSVP, as it is possible in RSVP/ns to send two flows to different agents on different nodes with the same flow id. In most cases this will not be desirable though, and then it will be the user's responsibility to assure that all flow id's used in a simulation scenario are unique at any given time.

The use of the flow id for distinguishing between flows also makes the SrcPort (source port) field in the FILTER_SPEC and SENDER_TEMPLATE objects obsolete, and the C-Type 2 in the FILTER_SPEC and SENDER_TEMPLATE objects was be omitted, since one of the C-Types 2 and 3 has become redundant due to the omission of the SrcPort field.

The LIH (logical interface handle) field in the RSVP_HOP object could be omitted since there are no logical interfaces in ns-2. The FLOWSPEC object only contains the token bucket rate and the token bucket size to simplify the traffic control mechanisms. The minimum policed unit and the maximum packet size have been omitted, because it has been decided that these parameters are unnecessary details in the current implementation. It should be possible for the user to implement mechanisms which assign different flow id's to packets of certain sizes to exclude them from the reservation for a flow.

Table 2 shows which objects have not been implemented in RSVP/ns yet. INTEGRITY objects are not necessary in the simulation since there are no corrupted or spoofed messages in a simulation. SCOPE objects are not needed, since wildcard filter reservation styles have not yet been implemented in RSVP/ns. The ADSPEC object was left out of the implementation for simplicity's sake and POLICY_DATA objects were omitted since policy control was out of the scope of the thesis RSVP/ns was implemented for. All of these objects may be implemented in later versions of RSVP/ns.

2.3 RSVP messages

All RSVP message types except the PathErr message have been implemented in RSVP/ns. PathErr messages are not needed since they only report errors based on port number mismatches which can not occur in RSVP/ns. The message formats in RSVP/ns are the same as defined in section 3.1 of RFC 2205 except that the RSVP objects which have not been implemented in RSVP/ns (see 2.1) were omitted from the messages. It should be noted that in RSVP/ns, messages are not parsed for correctness. This is only needed in environments where various RSVP implementations have to interoperate, but in ns, the messages are always processed by the same RSVP process which generated these messages.

2.4 Error codes

Only a subset of the possible RSVP error codes is necessary for RSVP/ns since many of the RSVP error codes indicate corrupted messages, port conflicts or policy control failures, all of which can not occur in RSVP/ns Table 3 shows a list of these error codes.

[1]It should be noted that the sender address can not be obsoleted by the flow id, because there can be several senders for the same session and it is possible to perform distinct reservation for each of these senders in RSVP.

Error	Error code
Confirmation	0
Admission Control failure	1
No path information for this Resv message	3
No sender information for this Resv message	4
Unknown reservation style	6

Table 3: Error codes in RSVP/ns

The error codes 3, 4 and 6 can occur directly as the result of an API call (i.e. when the user tries to reserve resources for a flow without path state, or with a non-supported reservation style). In fact, error code 6 can only originate directly from an API call, because no non-supported reservation styles will ever be sent in a reservation message.

2.5 Integrated Services

Currently, only controlled-load service is supported by RSVP/ns. The bandwidth guarantees are enforced by WFQ [Dem89] (or WF^2Q [Ben96]) in the links. One traffic class is assigned to each incoming flow. Furthermore, the FLOWSPEC objects only contain the flow's rate and token bucket size, but not the peak rate in conformance with RFC 2211.

It should be noted that the flowspec for reservations is not computed as described in RFC 2205 as the minimum of the TSPECs from the path messages and the FLOWSPECs from the reservation messages. The TSPECs are only "suggestions". It might be interesting for certain simulation scenarios to perform over-reservations which are greater than the TSPECs.

3 Internals of RSVP/ns

This section describes the basic internal structure of the RSVP/ns implementation. One important design goal for RSVP/ns was to keep it as independent from the rest of ns as possible and to not change any files belonging to ns unless it is absolutely necessary, to achieve compatibility with later versions of ns.

3.1 RSVP objects and messages

The classes RSVPobject and RSVPmessage in RSVP/ns are similar in the way that they both have a "contents" field which is a buffer which is later copied into the "data_" buffer of the RSVP packets. The contents buffer of an RSVPmessage is put together from the contents buffers of the RSVPobjects it contains. At the same time the (simulated) message size is calculated as the sum of the object sizes, and later the appropriate IP header size (for IPv4 or IPv6) is added to this size. The RSVPobject and RSVPmessage classes both maintain "length" and "conlength" fields which store the simulated size of the object and the real size of the contents buffer respectively. Note that the "length" values can vary between objects which were created for IPv4 or IPv6 simulations, while the "conlength" is always the same.

3.2 RSVP agents

The RSVP agents maintain path and reservation state on all RSVP nodes, generate RSVP messages and process incoming RSVP messages. They also provide the API for RSVP which is necessary on

the end nodes. Section 3.2.1 describes the stuctures used in RSVP agents such as path state blocks and reservation state blocks. Section 3.2.2 describes the session list in which all sessions with their corresponding path state blocks and reservation state blocks are stored and the timer list which is used for scheduling the sessions for sending refresh messages and the section also describes how the scheduling in the RSVP agents is performed.

3.2.1 Basic structures

There is one "session" structure for each session in an RSVP agent. It contains various fields, including pointers to its PSB list and RSB list with all relevant path states and reservation states for this session. It also contains a "ref_flag", which is set if the session is currently scheduled for a refresh, and a "local" flag which is set if the session was created by a local API call. It also contains a "confirm" flag which is only relevant for sessions on end nodes, since it determines if a RESV_CONFIRM object will be sent with the next reservation message to request a confirmation for this reservation from the network.

For each path state in each session, a "psb" structure is added to the session's PSB list. It contains various objects as described in RFC 2209 and also a pointer to the RSVPChecker object (see section 3.3) which handed the RSVP message to the agent that caused the creation of this state block, as well as a "timeout" field which stores the time at which the path state is going to expire unless it is refreshed by path messages. The "rsb" structure is mostly identical to the RSB structure as described in RFC 2209, except that it also contains a "timeout" field. In the TCSBs for a session, the current reservations for each outgoing link are stored.

3.2.2 Session list and timer list

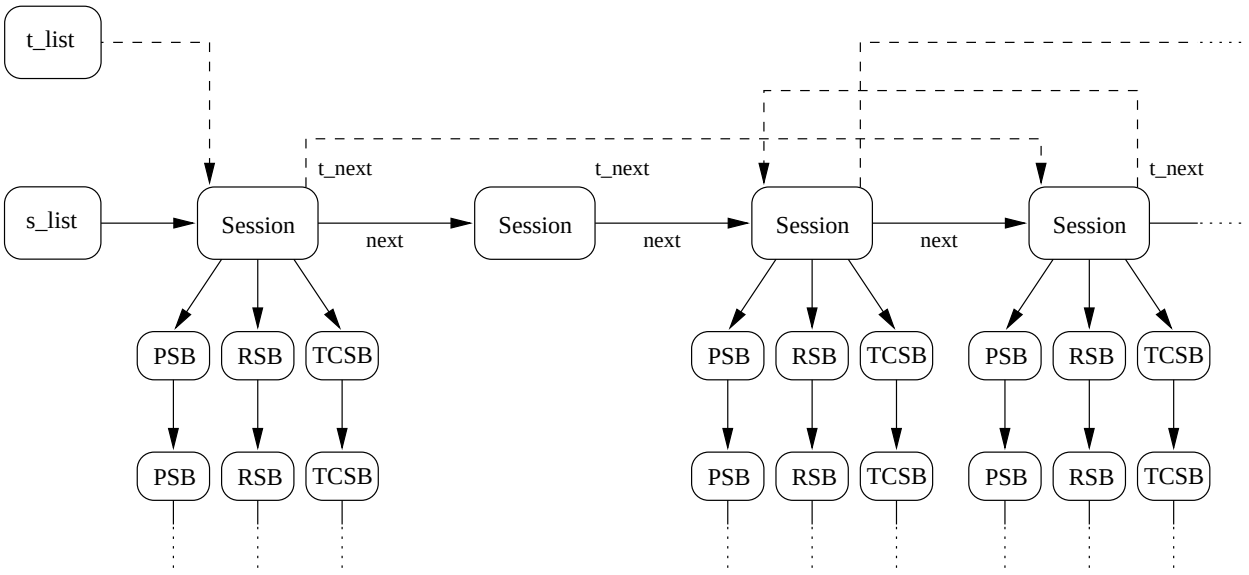

Figure 1: The session and timer lists

Figure 1 shows the structure of the two lists which are being maintained for the sessions. All sessions are in the session list ("s_list") and all sessions with state blocks (i.e. sessions which have to be refreshed) are also in the timer list ("t_list") which is sorted in ascending order by the next refresh

time for this session[2].

Usually it would be possible to schedule all reservation and path refreshes for all sessions as single events in ns (with the "at" command), but that would be a large waste of memory. For this reason, RSVP/ns only adds the first element in the timer list to the scheduler. It is still possible that new sessions are sorted into the timer list before the already scheduled element at its head, but that is not very likely.

3.3 RSVP links

The duplex-rsvp-links are based on duplex-intserv-links to which an RSVPChecker object and a WFQ queue are added. The purpose of the RSVPChecker object is to intercept RSVP packets on the link and hand them to the RSVP agent on the link's destination node. The interaction between RSVP agents and RSVPChecker objects is illustrated in figure 2. To send Path and PathTear messages, the messages are handed to the RSVPChecker object from which the path message originated which set up the corresponding path state. The packet header for these messages is changed to make it look as if the packet came from the previous hop. With this simple "spoofing" mechanism, full independence from the underlying routing protocol is achieved. This is not necessary however with Resv, ResvTear and ResvErr messages, since they are sent hop by hop from RSVP agent to RSVP agent, using the PHOP information from the path states, while path messages have to rely on the routing protocol, since they have to be routed through the same hops as the data packets.

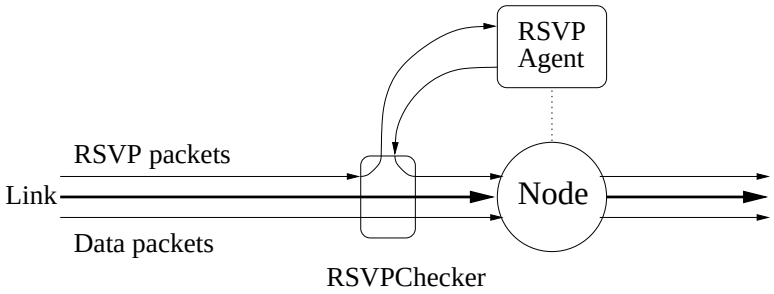

Figure 2: An RSVP link

3.4 WFQ

To enforce the bandwidth guarantees, WFQ [Dem89] and WF^{2}Q [Ben96] algorithms have been implemented for ns. For RSVP/ns, the option "best-effort" is used, which means that a queue is installed for all traffic that the classifier does not recognize (otherwise, an error message would be returned for each packet with an unknown flow id/sender combination). The option "borrow" is also used, which means that packets which are over-limit for a certain class are put into the best-effort queue.

The WFQ queues use their own hash tables to classify the packets to avoid possibly slow upcalls to Tcl to access the ns hash classifiers.

4 How to use RSVP/ns

In section 4.1 a short description of the installation procedures necessary for adding RSVP/ns to ns-2 is given. Section 4.2 describes the Tcl commands which are needed for creating simulation scenarios

[2]That is either the next time for a path refresh or a reservation refresh, whichever is smaller

with RSVP/ns.

4.1 Installation

First, the file "rsvpns.tar.gz" has to be copied into the ns directory (the directory which holds the source code for ns and the ns binary). The source files for RSVP/ns are extracted into the ns directory, the file "ns-rsvp.tcl" is added to the tcl/lib/ directory and several example scripts (some of which are described in section 5) are extracted into the directory tcl/ex/rsvp/. The file "README.RSVP" is also extracted into the ns directory. It contains information about which ns source files have to be edited and which entries have to be added to the Makefile.

Note: To use RSVP/ns, ns version 2.1b3 or higher is needed.

4.2 Tcl commands

This section describes how to use RSVP/ns in simulation scripts, assuming that the reader already has a basic experience with writing Tcl simulation scripts for ns.

A new link type has been added for RSVP/ns. This link, called "duplex-rsvp-link" is described in section 4.2.1. Section 4.2.2 describes how to add RSVP agents to nodes. The API commands for RSVP agents are described in section 4.2.3 and section 4.2.4 lists the API upcalls which can be triggered by RSVP events. Section 4.2.5 describes how some basic statistical information can be read from the RSVP agents and RSVP links.

4.2.1 Setting up an RSVP link

An RSVP link between the nodes n1 and n2 is created with the command

```
ns duplex-rsvp-link <n1> <n2> <bw> <delay> <reservable> <rsvp> <queue> <adc> <est>
```

where "ns" is an instance of the simulator. The other arguments are:

- bw : The bandwidth for this link
- delay : The link's delay
- reservable : The portion of the link's bandwidth that can be used by RSVP
- rsvp : The bandwidth (in bits per second) which is reserved for RSVP messages
- queue : The size of the queue for the best-effort class (in bytes)
- adc : The admission control algorithm
- est : The estimator used by measurement based admission control algorithms

If the "rsvp" argument is set to zero, all RSVP messages will be transmitted as best-effort packets. Otherwise, a WFQ class for RSVP messages with the rate specified by the "rsvp" argument will be added to the link to avoid RSVP message loss. The simple formula $n * s * 8/30$ (where n is the number of sessions which are going to traverse a link and s is the expected average message size, usually a value close to 100 bytes) should yield a good approximation of the necessary bandwidth, though this value will have to be higher for example if frequent reservation changes occur. The user has to ensure that the sum of the bandwidth which is reserved for RSVP messages and the reservable bandwidth does

not exceed the link's bandwidth (e.g. if 100% of a link's bandwidth are reservable, then no bandwidth should be reserved for RSVP messages).

Currently it is possible to choose from five different admission control algorithms[3] [Jam97]:

- Param : A parameter-based "Simple Sum" algorithm
- MS : Measured Sum
- HB : Hoeffding Bounds
- ACTO : Acceptance Region-Tangent at Origin
- ACTP : Acceptance Region-Tangent at Peak

The most usual choice would be "Param". There are four different estimator modules for the admission control algorithms:

- Null (for Param)
- TimeWindow (for MS)
- ExpAvg (for HB)
- PointSample (for ACTO and ACTP)

So the command to set up an RSVP link between the nodes a and b with a capacity of 1MBit/s (half of which should be reservable by RSVP), a delay of 10ms with 100bit/s reserved for RSVP messages, a queue size of 50000 bytes for the best-effort class and parameter-based admission control would be:

```
[Simulator instance] duplex-rsvp-link $a $b 1Mb 10ms 0.5 100 50000 Param Null
```

4.2.2 Creating and setting up RSVP agents

An RSVP agent is added to a node with the command

```
<node> add-rsvp-agent
```

This command returns a handle for the agent. The following example adds an RSVP agent to the node "no" and stores the handle in "rsvpagent":

```
set rsvpagent [$no add-rsvp-agent]
```

The handle to a node's RSVP agent is also returned by the following command:

```
<node> get-rsvp-agent
```

Table 4 shows the options which can be set for RSVP agents and their default values. All of these options can either be set for each RSVP agent or globally with "`Agent/RSVP set <option> <value>`".

[3] The admission control modules were written by Sandeep Bajaj (formerly Xerox PARC, now Cisco) and Lee Breslau (Xerox PARC). Some documentation for these modules can be found in the IntServ test suite in the directory tcl/ex/.

Option	Default value	Description
noisy_	0	Defines which events create an upcall to the API (see section 4.2.4)
refresh_	30	The refresh value "R" as described in section 3.7 of RFC 2205
lifetime_factor_	3	The value "K" as described in section 3.7 of RFC 2205
ip6_	0	If set to "1", IPv6 message lengths will be simulated
nam_	1	If set to "1", all packets will appear in nam with their correct message types, otherwise all packets are "RSVP" packets

Table 4: Configurable options for RSVP agents

4.2.3 API commands

The following commands are defined in RSVP/ns ("rsvp-agent" is always a handle for an RSVP agent that was added to a node):

- Create a session:

  ```
  <rsvp-agent> session <destination> <flow-id>
  ```

  The argument "destination" can be either a node id, a node handle or a multicast address. This call returns an ID for the session which has to be used in all subsequent commands for this session. A session is created automatically by a path event.

- Release a session:

  ```
  <rsvp-agent> release <session-id>
  ```

  A session that was not created by a "session" command is released automatically when no path states for this session are left.

- Send path messages:

  ```
  <rsvp-agent> sender <session-id> <rate> <bucket> <ttl>
  ```

- Reserve bandwidth:

  ```
  <rsvp-agent> reserve <session-id> <style> <flow descriptor list>
  ```

  At the moment, the style can only be 'FF'. The format of the flow descriptor list is described below.

- Send a RESV_CONFIRM object with the next reservation message:

  ```
  <rsvp-agent> confirm <session-id>
  ```

- Get a list of all sessions in an agent:

  ```
  <rsvp-agent> sessions
  ```

- Set the status value for a session:

```
<rsvp-agent> set-status <session-id> <value>
```

- Get the status value for a session:

```
<rsvp-agent> get-status <session-id>
```

- Assign an object handle to a session:

```
<rsvp-agent> set-handle <session-id> <object>
```

- Get the object handle for a session:

```
<rsvp-agent> get-handle <session-id>
```

The status value does not actually represent a session's current status. It can be set freely by the user to any value. However, its purpose is to provide a possibility to keep track of a session's current status, usually with the help of the upcall functions (see section 4.2.4). For example, the "upcall-resv" function can be redefined to set the value to "1", and the "upcall-resv-tear" function would set it to "0" again. The object handle for a session has a similar purpose. It gives the user the possibility to assign an object to a session on an RSVP agent, for example the source object for the flow that corresponds to the session.

RSVP sends messages with the flow ID 46 to enable RSVP links to assign RSVP messages to their reserved class in the links. There are also some other protocol messages (e.g. Prune messages for multicast protocols) that are sent with a special flow ID, so the best way to avoid any problems is to use high flow IDs, for example IDs above 1000.

The flow descriptor list in "reserve" commands is defined exactly like the flow descriptor list for FF style reservation messages in section 3.1.4 of RFC 2205 by the following BNF:

```
<flow descriptor list> ::=
         <flowspec> <sender> |
         <flow descriptor list> <FF flow descriptor>
<FF flow descriptor> ::=
         [ <flowspec> ] <sender>
```

This means that a (virtually) unlimited number of senders can follow after each flowspec. Senders can be either node handles or node IDs. Flowspecs are comprised of two numbers, the rate and the bucket size. To distinguish flowspecs from senders, flowspecs have to begin with a "+". For example, to reserve 200,000 bits per second with a bucket size of 10,000 bytes for the sender nodes n1 and n2, and 40,000 bits per second with a bucket size of 5,000 bytes for the sender node n3, the following command would be used:

```
$rsvp-agent reserve $session-id FF +200000 10000 $n1 $n2 +40000 5000 $n3
```

4.2.4 API upcalls

API upcalls are triggered by events, such as path events or reservation events. To determine which events trigger an upcall, one bit for each possible upcall can be set in the "noisy_" byte. Table 5 shows which bit value in the "noisy_" byte corresponds to which upcall and the parameters that are passed to the upcalls. In RSVP/ns, upcalls can occur on all nodes, not only on end nodes, making it possible to monitor the events on each node. To globally enable all upcalls on all RSVP agents, the command

Value	Function header	Arguments
1	`upcall-path`	session-id, rate, bucket, sender
2	`upcall-resv`	session-id, rate, bucket, sender
4	`upcall-resv-error`	session-id, error-code, error-value, error-node
8	`upcall-resv-confirm`	session-id, sender, node
16	`upcall-path-timeout`	session-id, sender
32	`upcall-path-tear`	session-id, sender
64	`upcall-resv-timeout`	session-id, sender
128	`upcall-resv-tear`	session-id, sender

Table 5: API upcalls and their bit values in the "noisy_" byte

```
Agent/RSVP set noisy_ 255
```

can be used at the start of a simulation script.

There are simple predefined functions for each possible upcall. However, it is possible to override these functions by redefining them. For example, it would be possible to define an "upcall-path" function which automatically triggers a reservation for the session.

4.2.5 Statistics

RSVP agents maintain the values "num_psb_" (the number of path states in the agent), "num_rsb_" (the number of reservation states) and "num_flows_" (the number of reservations, for distinct reservation the same as "num_rsb_"), which can be read from the agent at any time. Additionally, WFQ links maintain the values "num_classes_" (the number of WFQ classes), "num_flows_" (the number of flows), "num_drops_" (the number of packets from *reserved* classes which were dropped) and "size_drops_" (the sum of the packet sizes of the packets which were counted in "num_drops_"). The last two values can be reset to zero with the command "`<wfq> clear-stats`", where "wfq" is the handle for a WFQ queue which can be obtained from a link with the command "`<link> queue`".

5 Validation

In [Jain91], three possible sources for the validation of a simulation model are given:

- Expert intuition
- Real system measurements
- Theoretical results

In this section, expert intuition is used to validate the correctness of RSVP/ns, because no complex RSVP test bed was available that could have been used for measurements and it would probably take more time to develop a correct theoretical model for RSVP that could be used for comparisons with RSVP/ns than it took to implement RSVP/ns.

Several examples will be given in this section to prove for certain special situations that RSVP/ns functions properly. However, to quote Jain, "a 'fully validated model' is a myth", so the scope of this section is to cover the most important cases for RSVP, but not to fully prove the correctness of RSVP/ns.

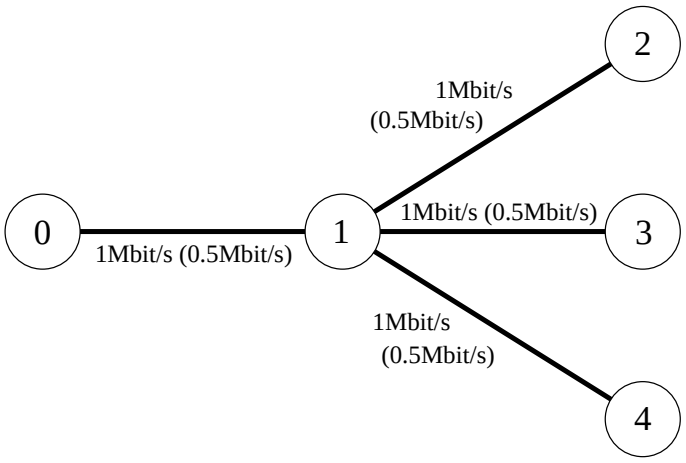

Figure 3: The topology for the examples in section 5.1 and 5.2

5.1 Merging of reservations

The purpose of this example is to show if RSVP/ns merges reservations correctly. Figure 3 shows the topology that is used for this example (the reservable bandwidth for each link is set in brackets). Node 0 sends three multicast flows with a constant bit rate of 0.5Mbit/s to the nodes 2, 3 and 4. These nodes successively perform increasing reservations and release them again. The following lines were taken from the simulation script:

```
$ns at 300.0 "$rsvp2 reserve 0 ff +100000 100000 $n0"
$ns at 600.0 "$rsvp3 reserve 0 ff +300000 100000 $n0"
$ns at 900.0 "$rsvp4 reserve 0 ff +500000 100000 $n0"
$ns at 1200.0 "$rsvp4 release 0"
$ns at 1500.0 "$rsvp3 release 0"
$ns at 1800.0 "$rsvp2 release 0"
```

The normal behaviour for RSVP would be to merge the reservation requests from the three nodes and to only forward a reservation for the lowest upper bound. During the simulation, the rate for the three flows was measured at one of the receiver nodes[4] and figure 4 shows the results.

It is obvious that at 300 seconds, the received rate for flow 1 increases. This is caused by the reservation for 0.1Mbit/s which gives the flow an advantage over the other flows. At 600 seconds, the reservation increases even more, then at 900 seconds it goes up to the flow's peak rate, to 0.5Mbit/s. The other reservations request have been merged into the request for 0.5Mbit/s (which represents the lowest upper bound). The other two flows share the remaining bandwidth. Without merging, the request for 0.5Mbit/s would have been rejected, because reservations for 0.1Mbit/s and 0.3Mbit/s were already in place. When the sessions are released on the receiver nodes, the reservations are removed correctly which can be seen in the measurements from the decreasing rate for flow 1.

The result is that at least for this scenario, merging of reservation requests for multicast flows is performed correctly by RSVP/ns. The example script can be found in the directory "tcl/ex/rsvp/". The filename is "rsvp_merge.tcl".

[4]It does not matter which one of the three nodes is chosen, because the link between node 0 and node 1 is the bottleneck, and the traffic downstream from node 1 is basically the same for all receiver nodes

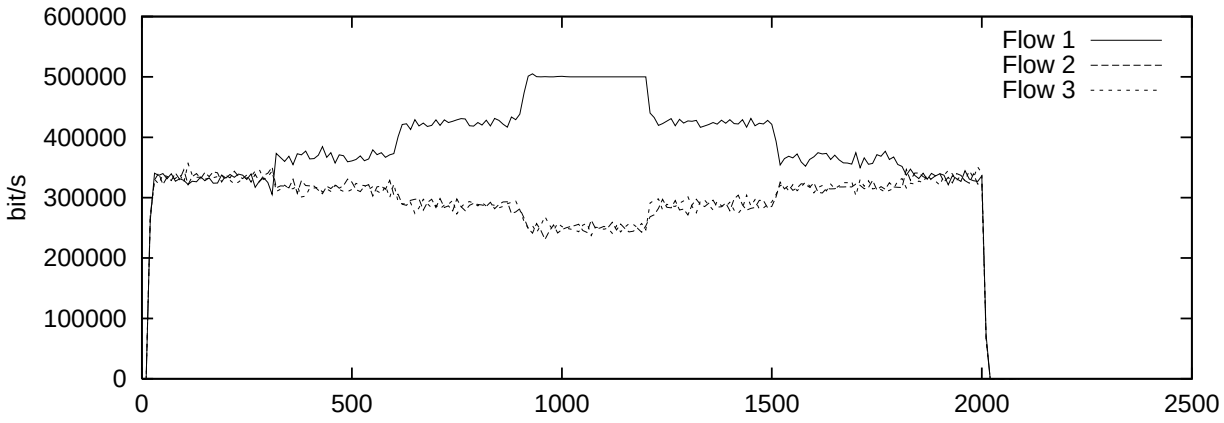

Figure 4: The measurements for the example in 5.1

5.2 Distinct reservations

For this example, the same topology is used as for the merging example. In this case, the nodes 2, 3 and 4 are the senders though. Each of them sends a data flows with a constant bit rate of 0.5Mbit/s. All of these flows are being sent for the same session (i.e. with the same flow id and the same receiver node). Now the receiver node 0 performs distinct reservations for the single senders. The following lines were taken from the simulation script:

```
$ns at 300.0 "$rsvp0 reserve 0 ff +100000 100000 $n2"
$ns at 600.0 "$rsvp0 reserve 0 ff +500000 100000 $n3"
$ns at 900.0 "$rsvp0 reserve 0 ff +400000 100000 $n3"
$ns at 1200.0 "$rsvp0 reserve 0 ff +100000 100000 $n3"
$ns at 1500.0 "$rsvp0 reserve 0 ff +300000 100000 $n4"
```

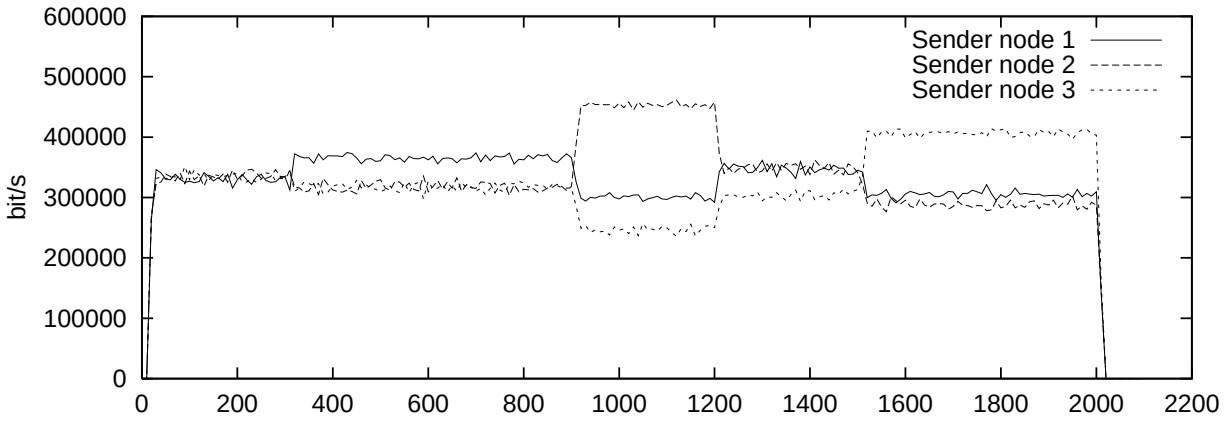

Figure 5: Measurements for distinct reservations

The normal behaviour for RSVP would be to add all FF reservation requests and to forward the sum of all flowspecs. This means that the request in the second line would have to be rejected, because

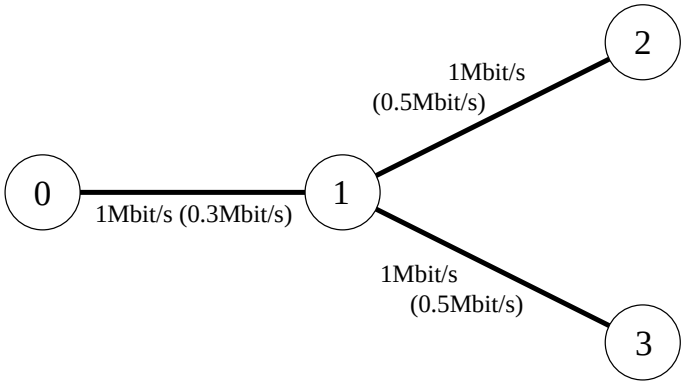

Figure 6: The scenario for the example in section 5.3

a reservation for 0.1Mbit/s is already in place at this point, and only 0.5Mbit/s are reservable on each link. All other reservation requests should be admitted though.

The received rate for each of the three flows was measured at node 0. The results are presented in figure 5. An obvious effect of the reservation at 300 seconds for the flow from node 2 can be seen. There is no effect at all at 600 seconds (as expected), but the reservation request at 900 seconds succeeds. At 1200 seconds, the reservation for the flow from node 3 is reduced to the reservation for node 2, so both flows receive the same bandwidth for the rest of the simulation. At 1500 seconds, a reservation request for node 4 with 0.3Mbit/s succeeds, which also causes a clearly visible effect in the measurements.

The result is that at least for this scenario, distinct reservations for different senders are performed correctly by RSVP/ns. The example script can be found in the directory "tcl/ex/rsvp/". The filename is "rsvp_distinct.tcl".

5.3 Confirmation messages

This section illustrates the possible problem with confirmation messages which is mentioned in section 2.6 of RFC 2205. Figure 5.3 shows the simple topology which is used for this example. Again, the reservable bandwidth for each link is set in brackets. Node 0 sends path messages for a multicast session to the nodes 2 and 3. Node 2 sends a reservation message for a rate of 500kbit/s. This reservation request is accepted at node 1. However, there is not enough reservable bandwidth on the link from node 0 to node 1. 0.7 seconds later, before the reservation request from node 2 has reached node 0, node 3 sends a reservation message with a RESV_CONFIRM object. When this message reaches node 1, it is merged with the existing reservation state from node 2, and a confirmation message is sent back to node 3. Then, 1.3 seconds later, node 3 receives an error message for the same reservation, because it could not be established on the link from node 0 to node 1. This is the output for node 3 from the example script.

```
3  PATH EVENT at 4.002 : SID: 0 RATE: 500000 BUCKET: 5000 SENDER: 0
3  RESVCONF EVENT at 7.702 : SID: 10 SENDER: 0 NODE: 1
3  RESVERROR EVENT at 9.003 : SID: 0 CODE: 1 VALUE: 0 NODE: 0
```

The example script can be found in the directory "tcl/ex/rsvp/". The filename is "rsvp_conf.tcl".

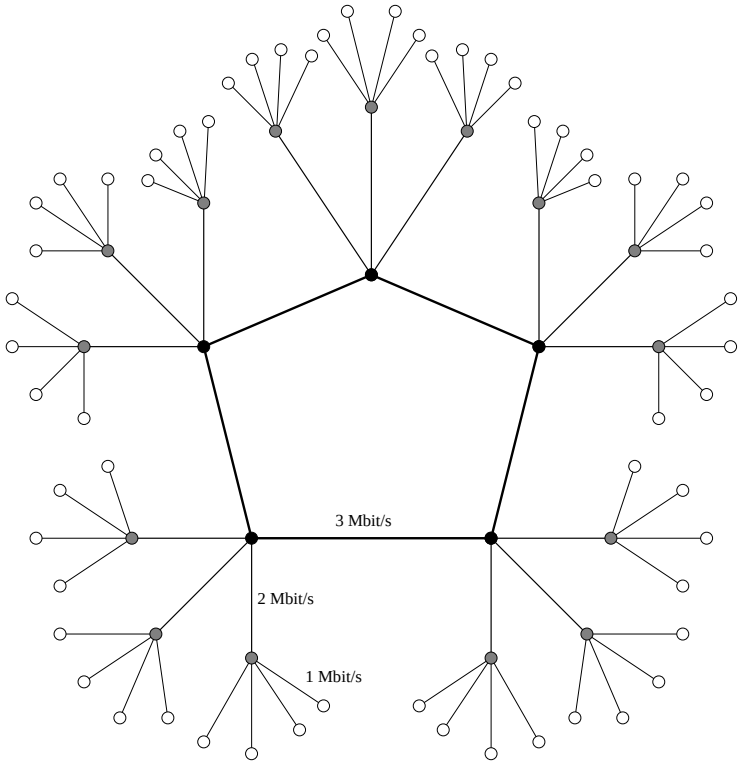

Figure 7: The scenario for the example in section 5.4

5.4 Large simulations

For this section, a "stress test" was performed with RSVP/ns on a topology with 80 nodes. This topology is depicted in figure 7. The 5 black nodes in the middle which are connected with 3Mbit/s links represent the backbone. The grey nodes connect the white end nodes with this backbone.

When the simulation starts, best-effort flows are created from each end node to 15 randomly selected destination nodes. These flows are exponential on/off (Expoo) flows. This means that bursts are sent where the burst and idle times are chosen from exponential distributions (with an average of 1s and 0.5s for the burst time and the idle time respectively). The packet sizes are constant during the lifetime of a flow, but from flow to flow, the size is chosen from an exponential distribution with the average 500 bytes. The peak rate of all flows is 64kbit/s. The flow lifetimes and the waiting times before a new flow is spawned after a flow's lifetime is over are also exponentially distributed with an average of 200 seconds and 100 seconds respectively. Since each flow spawns only one successor, the maximum for the number of best-effort flows that originate from an end node is 15, so the maximum number of best-effort flows in the whole network is 900.

Each end node also sends up to 4 RSVP flows. These flows are sent with a constant bitrate of 64kbit/s. The distributions for the flow lifetime and all other values are the same as for the best-effort flows. Before an RSVP flow is sent, path messages are sent to the receiver, which automatically triggers a reservation request from the receiver. When the reservation is established, the sender creates a traffic source and begins to send. The maximum of RSVP flows in the whole network is 240, so the highest number of flows that can appear in the network at any given time is 1140.

The goal of this scenario is to test the functionality of RSVP/ns in such a large network where the nodes in the backbone have to maintain a high number of constantly changing path and reservation

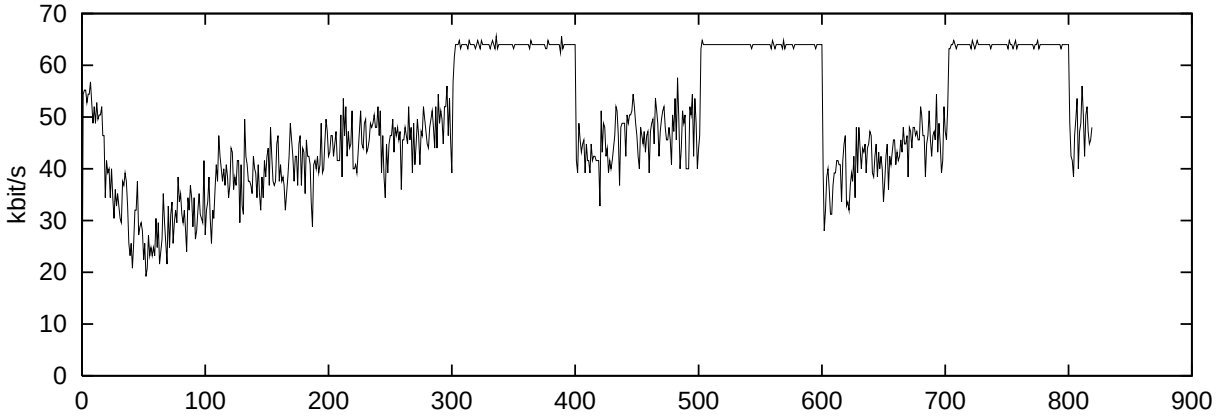

Figure 8: The result of the measurements described in section 5.4

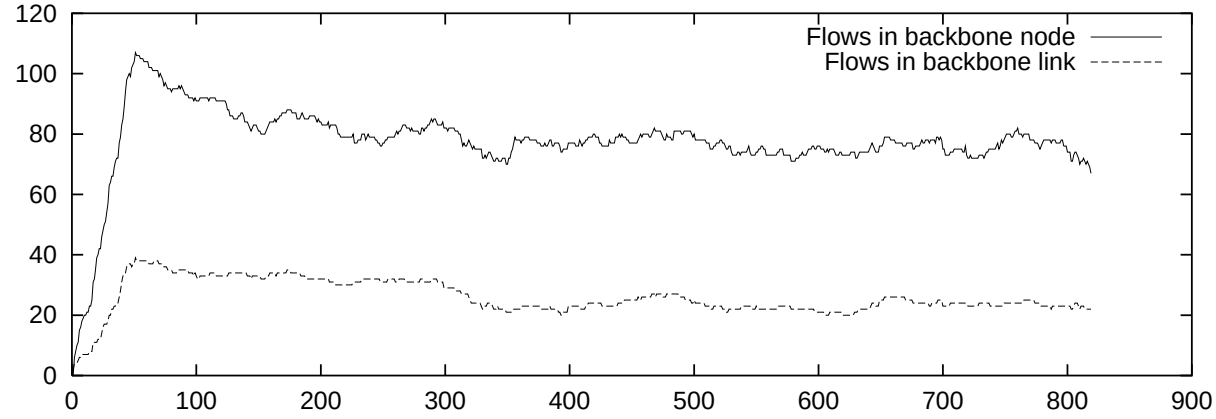

Figure 9: The number of flows in an arbitrarily chosen backbone node and backbone link

states and where the end nodes create a "background noise", both of best-effort packets and packets for reserved traffic. To achieve this goal, a test flow with 64kbit/s and a packet size of 100bytes is sent from one arbitrarily chosen node to another node on the opposite side of the network. At certain times (from 300s-400s, 500s-600s and 700s-800s), reservations are set up for this flow. The rate for the flow is measured by a simple monitor object at the receiver node. The result of these measurements is shown in picture 8.

Picture 8 clearly shows how the reservations affect the flow and protect it from packet losses, keeping it constantly at 64kbit/s for the duration of the reservation. Figure 9 shows the number of flows with reservations in an arbitrarily chosen node in the backbone and an arbitrarily chosen link in the backbone. It can be seen how the simulation does not reach an entirely stable state before 200 seconds have passed. The fourth measured value in this scenario was the number of dropped packets for reserved flows in the same backbone link. This value was constantly zero throughout the whole simulation.

The example script can be found in the directory "tcl/ex/rsvp/". The filename is "rsvp_large.tcl". It took about 90 minutes to run this simulation on a Sun Ultra 2 with a 296 Mhz processor.

6 Disclaimer

This version of RSVP/ns can be seen as a "snapshot" of an ongoing project. Full correctness of this version can not be guaranteed and it is the user's own responsibility to verify and validate the correctness of simulations with RSVP/ns. The user is also expected to read the copyright notice and disclaimer in the RSVP/ns source files before using RSVP/ns.

References

[Ben96] J.C.R. Bennett and H. Zhang, "WFQ: Worst-case Fair Weighted Fair Queueing", INFOCOM 1996, March 1996

[Dem89] Demers, A., Keshav, S., Shenker, S., "Analysis and simulation of a fair queueing algorithm", Proceedings of ACM SIGCOMM 89

[Jain91] Jain, R., "The Art of Computer Systems Performance Analysis", John Wiley & Sons, 1991

[Jam97] Jamin, S., Shenker, S.J., Danzig, P.B., "Comparison of Measurement-based Admission Control Algorithms for Controlled-Load Service", Proc. IEEE INFOCOM '97, April 97

[NS] Fall, K., Varadhan, K., "ns Notes and Documentation", April 1998, "http://www-mash.cs.berkeley.edu/ns/nsDoc.ps.gz"

[RFC1633] Braden, R., Clark, D., Shenker, S., "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997

[RFC2205] Braden, R., Zhang, L., Berson, S., Herzog, S., Jamin, S., "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification", RFC 2205, September 1997

[RFC2209] Braden, R., Zhang, L., "Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules", RFC 2209, September 1997

[RFC2210] Wroclawski, J., "The Use of RSVP with IETF Integrated Services", RFC 2210, September 1997

[RFC2211] Wroclawski, J., "Specification of the Controlled-Load Network Element Service", RFC 2211, September 1997

[VINT] Webpage of the VINT project, "http://netweb.usc.edu/vint/"