

Some extensions to enhance the scalability of the RSVP protocol

Differences between two drafts:
draft-tommasi-rsvp-enhan-scalab-01.txt
draft-ietf-rsvp-refresh-reduct-01.txt
and new proposals

Franco Tommasi (tommasi@ilenic.unile.it)
Simone Molendini (molendini@ultra5.unile.it)
University of Lecce – ITALY

Slides for RSVP Working Group Meeting
in Washington, November, 9th 1999

Content

INTRODUCTION

CLD

INTRODUCTION
COMPRESSION CODES
TWO EXAMPLES
WHEN TO USE IT

TRIGGER ACK'S

FORMAT DIFFERENCES

WEAK REFRESH

WHAT IS IT?

STRONG REFRESH

WHAT IS IT?
AN EXAMPLE

BUNDLING PROBLEMS

URGENT BIT
BUNDLING DELAY UPPER BOUND

A FRAMEWORK ABOUT MESSAGES

Introduction

THIS PRESENTATION WILL HIGHLIGHT EXISTING DIFFERENCES BETWEEN TWO DRAFTS:

DRAFT-TOMMASI-RSVP-ENHAN-SCALAB-01.TXT

AND

DRAFT-IETF-RSVP-REFRESH-REDUCT-01.TXT.

AND SUGGEST SOME FURTHER DEVELOPMENT

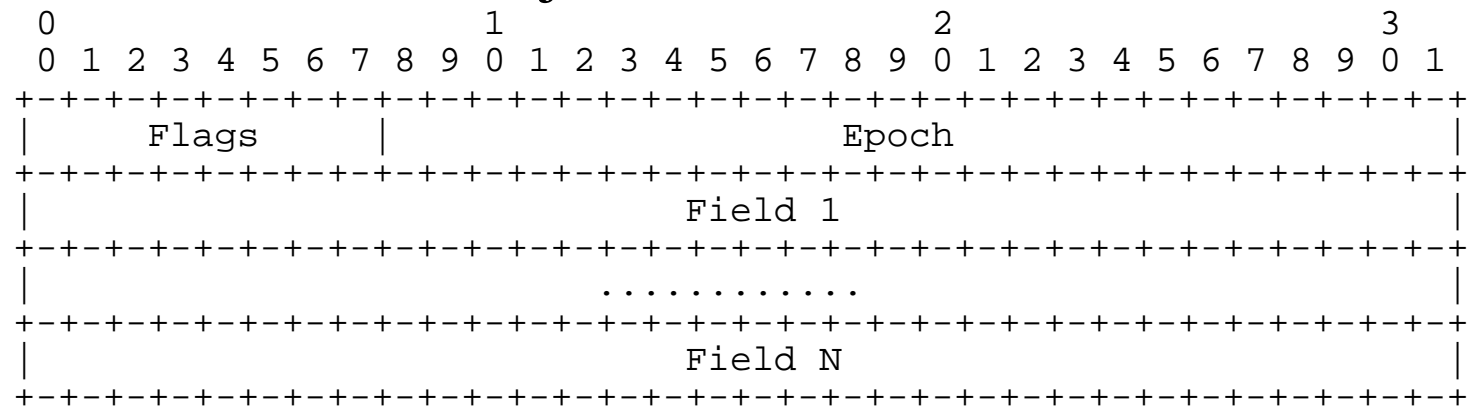
CLD - Introduction

We would like to introduce a basic concept we will extensively use in the following.

The concept is that of a COMPRESSED_LIST_DESCRIPTOR (CLD).

CLD is an RSVP object used to efficiently communicate long lists of 31-bit numbers.

The format of the object is:

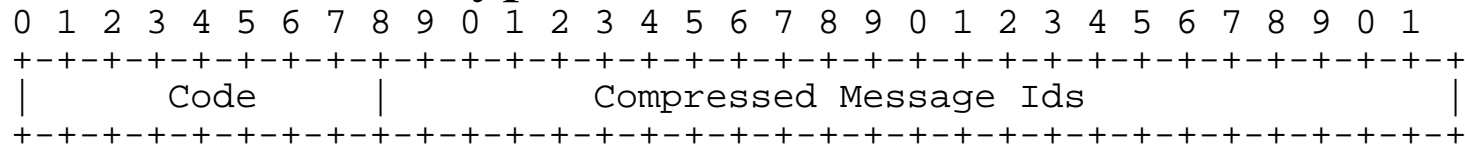


CLD – Compression Codes

A field may be a Message_ID field (Type 1):



or a Code field (Type 2):



where names and meaning of codes are (numeric constant TBA):

3xWORD: defines 3 IDs per words, whose values differ no more than 256 between each other

6xWORD: defines 6 IDs per words, whose values differ no more than 16 between each other

12xWORD: defines 12 IDs per words, whose values differ no more than 4 between each other

RANGE: defines the number of consecutive existing IDs

BITMAP: defines a number of words, whose bits are Yes/No flags

CLD – Two Examples

Example 1:

The following COMPRESSED_LIST_DESCRIPTOR object:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  1 2 3
  0 1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      Flags      |                               Epoch                               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0|                                     49|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      BITMAP      |                               2                               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0 1 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1 1 1 1 1|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|1 1 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 1 0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Describes the Message_IDs list: 49, 51, 52, 57, 58, 59, 62, 64, 66, 68, 70, 71, 76, 77, 78, 79, 80, 81, 82, 83, 84, 87, 89, 91, 93, 95, 96, 101, 102, 103, 104, 107, 109, 111, 112

5 words instead of 36 words

Example 2:

The following one:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      Flags      |                               Epoch                               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0|                                     49|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      RANGE      |                               5                               |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      12XWORD     | 2 | 3 | 2 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 3 | 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0|                                     30101|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      3XWORD     |      12      |          121          |          0          |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Describes the Message_IDs list: 49, 50, 51, 52, 53, 54, 56, 59, 61, 62, 63, 64, 65, 67, 68, 69, 72, 73, 30101, 30113, 30234

6 words instead of 22 words

CLD – When To Use It

We propose to use a CLD wherever possible in the RSVP refresh/ack architecture.

Its use grants huge savings in packet sizes (some initial rough calculation indicates typical reductions of five to ten times; the reduction increases with the number of states).

Its extensive use contributes to **COMPACTNESS** and **ROBUSTNESS** of the protocol/implementation architecture.

Where CLD can be used:

- 1) Acks to Trigger messages contained in Bundle messages;
- 2) Refresh messages
- 3) Ack to Refresh Messages

ACKs to Refresh messages can be simply compacted by the introduction of a **LIST_ID** object (more on this later).

ACKs to trigger messages - Format differences

What is the difference between
Berger's draft and ours?

FORMAT OF BERGER'S ACK TO TRIGGERS:

```
<ACK Message> ::= <Common Header> [ <INTEGRITY> ]  
                <MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>  
                [ <MESSAGE_ID_ACK> ... ] [ <MESSAGE_ID_NACK> ... ]
```

FORMAT OF OUR ACKs TO TRIGGERS:

```
< Ack Message > ::= < Common Header > [ < INTEGRITY > ]  
                  < COMPRESSED_LIST_DESCRIPTOR >
```

That is, we use a **COMPRESSED_LIST_DESCRIPTOR** object instead of a list of **MESSAGE_IDS** objects (N+2 words instead of 3*N words).

WEAK REFRESH – What is it?

We use the CLD object to compact refresh messages.

A cheap and easy way to get ACKs even for refresh messages is to prepend the CLD with a LIST_ID object.

This LIST_ID object is simply returned back to the sender as an ACK to the refresh. This prevent state expiration for loss of K refreshes.

Berger`s draft doesn`t propose compression and LIST_IDs.

In this way a receiver has no way to detect which are the no more valid states (it has to let them time-out thus wasting resources – we are dealing congested situations).

A way to solve the problem is STRONG REFRESH.

STRONG REFRESH – What is it?

In the STRONG REFRESH procedure, all the states to be refreshed are sent in many packets with the same LIST_ID object and increasing numbering in the SEQUENCE_# objects.

The receiver has the way to know exactly if a state it considers active is still valid.

The receiver can easily acknowledge the whole sequence just sending the LIST_ID back. If some packet get lost the receiver can send an ACK up to the sequence number it received.

STRONG REFRESH – An Example

Let us try to explain with an example.

“CLASSICAL” WEAK/SUMMARY REFRESH

Local state: IDs = {1,2,4}

Received with Srefresh / WeakRefresh: ID = 1,2,5

Refreshed 1,2

Clarification message about 5

4? Nothing done. Waits until timeouts

STRONG REFRESH

Let suppose now that the IDs are received all together: a full list match can be executed, and ALL differences are identified.

Local state: IDs = {1,2,4}

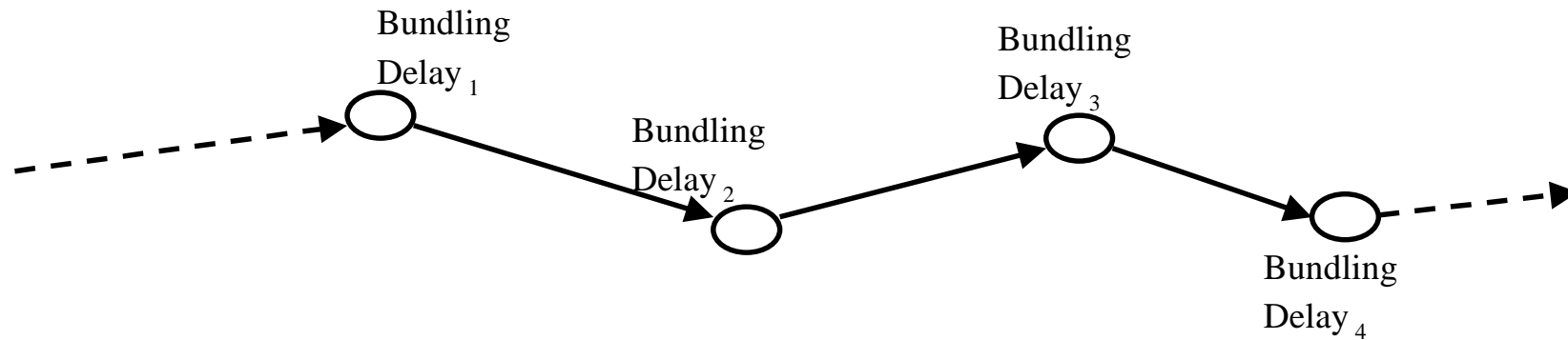
Received with StrongRefresh: ID {1,2,5}

Refreshed 1,2

Clarification message about 5

Clarification message about 4

Bundling problems - Urgent bit



Delay without Bundling.

$$\text{End-to-end RTT} = \sum_i \text{RTT}_i$$

RTT_i includes all latencies (RSVP and not) when no bundling is performed.

Delay with Bundling.

$$\text{End-to-end RTT} = \sum_i \text{RTT}_i + \sum_i \text{Bundling Delay}_i$$

RTT_i are the same as the previous case

Bundling problems - Urgent bit

Question: Will all applications be forced to always accept the additional delay caused by bundling?

Answer: An application must be able to set an **urgent bit** in its trigger messages to communicate it doesn't want it to be delayed for bundling or any other optional (future) processing.

Problems? None. When an Urgent bit is detected, just stop bundling and ship what is already in the send buffer.

Bundling problems – Upper bound

- If an application is given the chance to set an urgent bit (or even if not), it should be given a way to compute an upper bound to what to expect from the bundling delay.
- An upper bound of some sort would prevent implementors from introducing excessive delays.

AN UPPER BOUND CAN BE:

A) **FIXED**

It may be a problem with different links/evolving technologies

B) **ADAPTIVE**

In this case adaptivity can be linked to:

- **link`s bandwidth**

(no computation, extends to new technologies, not very precise; proposed now)

- **link`s RTT**

(more precise, not trivial to compute, extends to new technologies; draft proposal)

- **number of states**

(easy to compute, extends to new technologies; correlation with effective delay TB assessed; doubtful)

Bundling problems – Upper bound

THERE IS AN OPTIMAL VALUE FOR EACH LINK.

Long bounds damage responsiveness.

Short bounds prevent bundling.

"LONG" AND "SHORT" ARE LINK DEPENDENT.

**BEYOND THE ADAPTIVE CHOICE A WORST CASE DELAY
FOR EACH LINK COULD BE PROVIDED**

(100 ms could be a reasonable value)