

Ensuring Acceptable QoS by Detecting MLPPP Fragment Loss in Low Density Traffic

RFC1990 describes a mechanism, the “M” algorithm, for detecting MLPPP fragment loss. This mechanism detects a lost fragment only when all of the links in an MLPPP bundle have received a fragment with a sequence number greater than that of the fragment that is lost. The RFC does not state how the M algorithm should be implemented to guarantee an acceptable level of QoS. Using only the RFC1990 M algorithm in periods of low density traffic can lead to a relatively large amount of time passing before a fragment loss is detected. This can have a detrimental effect on real-time traffic such as voice or video.

This application note explains how the RFC1990 M algorithm can be augmented to detect fragment loss in a way that guarantees an acceptable QoS. It is relevant for Freescale Semiconductor’s MLPPP-enabled, QUICC Engine™ products (MPC8360E, MPC8568E, and MPC8569E).

Contents

1. Low Density Traffic Problem Statement	2
2. Loss Detection and Flushing the WBD	3
3. Conclusion	10
4. Revision History	11

1 Low Density Traffic Problem Statement

Consider the following scenario: a 3G wireless base station (BTS) is connected to the radio network controller (RNC) via 16 E1 lines. Late in the evening the density of traffic drops and there is only one voice call active on the BTS. For the user to experience acceptable voice quality, a packet (typically) needs to be received every 20 ms. [Figure 1](#) shows this scenario.

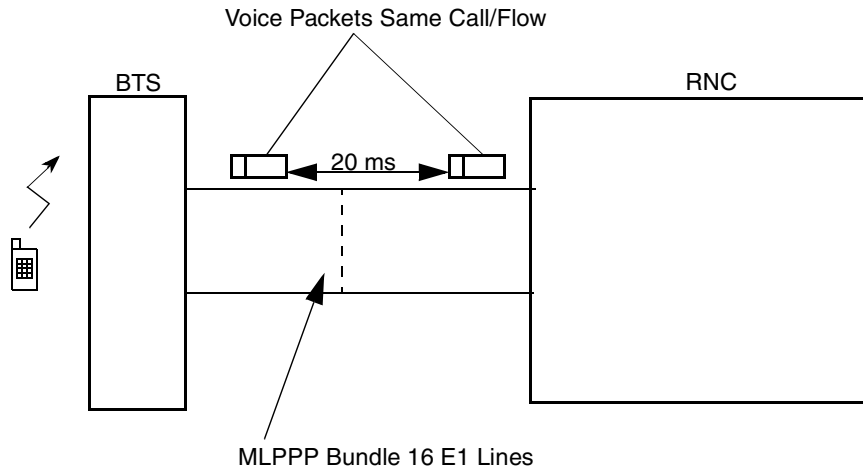


Figure 1. 16 E1 MLPPP Low Density Traffic Problem

Every 20 ms, a voice packet is transferred over the bundle from the RNC to the BTS. It is acceptable to lose 2–3 back-to-back packets; more than this becomes audible to the end user. A 20 ms delay means there is no possibility of packets being transmitted back-to-back. There is a large “idle” gap between consecutive packets on the bundle. Fragment loss detection latency can become an issue in this system. To illustrate the point, it is assumed, as described in RFC1990, that the RNC sends the voice packets round robin over the 16 links in the bundle. For ease of illustration, it is also assumed that there is no fragmentation; each MLPPP “fragment” has its B (Begin) and E (End) bits set to 1.

[Table 1](#) shows when the M algorithm detects fragment loss when there are four E1 links in the bundle.

Table 1. Fragment Loss Detection Example Using RFC1990 M Algorithm

MLPPP Sequence Number			
E1_1	E1_2	E1_3	E1_4
0	1	2	3
4	LOST	6	7
8	9 (Fragment loss detected here)	—	—

Because the packets are transmitted round robin in sequence, they are also received round robin in sequence. [Table 1](#) shows that the first four fragments are received correctly. Then E1_1 receives fragment with sequence number 4. Sequence number 5 is lost on E1_2, so this link’s latest sequence number remains 1. Fragments go round robin until E1_2 again. No fragment loss is detected because it is feasible that the

fragment with sequence number 5 could appear on E1_2. Only when the fragment containing sequence number 9 is fully received on E1_2 is the fragment loss detected. In theory, as fragments 6, 7, and 8 have $B = E = 1$, they can be passed up to the next layer; however, this is system-dependent and is very complex to manage. It is likely that these fragments will be buffered until either sequence number 5 appears or fragment loss is detected. This is the behavior in the MLPPP microcode's WBD; the fragments are buffered until fragment loss is detected.

It took 4 fragments to detect the loss, which at first glance seems reasonable. However, because there is a 20-ms interval between each fragment, it took 80 ms to detect the loss. Fragment 6 is delayed by 60 ms, 7 by 40 ms, and 8 by 20 ms. This may result in audible disturbance on the voice call.

If there are 16 links in the bundle, then it would take 320 ms to detect the loss, causing significantly more disturbance. This example is simplified and can be much worse in reality if any of the following are true:

- The peer does not transmit the fragments strictly in round robin fashion
- There are multiple fragment loss events

2 Loss Detection and Flushing the WBD

To overcome the issue described in [Section 1, “Low Density Traffic Problem Statement,”](#) the MLPPP microcode's WBD must be flushed when a fragment is lost and the traffic density is low. There are two distinct elements to the solution, as follows:

1. Detecting the fragment loss
2. Triggering the flush

2.1 Fragment Loss Detection

To ensure that fragments are pushed out of the WBD before the delay impacts QoS, the user must monitor for fragment loss, and when necessary, issue the WBD flush command. The following parameters in the CPT are used in the algorithm:

- RX_MAX—highest sequence number of a fragment received on this bundle
- RX_LSQN—oldest missing fragment's sequence number; or, if no fragments are missing, then $RX_MAX + 1$ (this is essentially the next expected sequence number)

The basic idea is to define an array with a length that takes into account the time out period, for example, if the check is done every 1 ms, and the time-out period for the system is 10 ms then the array should have 10 elements. This allows a packet delayed in the WBD for 10 ms to be flushed. When the timer period has expired the maximum sequence number received on this bundle is written to the current element in the array. The array index is then incremented to the next entry.

Each element is checked in turn, meaning the current element is checked, and then on the next timer tick the next element is checked, and so on. Therefore, after one full wrap of the array, and thus one full system timeout period, the first element is checked again. That is, if the array holds N elements, then after N timer ticks the element is checked. When it is checked, the array element contains the highest sequence number received on this bundle one system time-out period ago. If the value of the element is greater than the sequence number of the oldest hole ($CPT[RX_LSQN]$), then a WBD flush command should be issued.

The array elements should be initialized with a number greater than the sequence number range (for example, UNDEFINED_SN = 0x80000000). This undefined value is used to handle the first time through the array and is programmed at initialization only.

An example of how to monitor for this situation is described in the following steps:

1. Record the highest sequence number (new_high) received in the timer period (that is, save RX_Max from Class Parameter Table (CPT)).
2. Get the old highest sequence number (old_high) for this time-out period (for example, in the last 10 ms) by reading the value in the current array index.
3. Set the current array element to the new high sequence number, that is, value from step 1.
4. Increment the array index.
5. Check if the end of the array has been reached, and if so, wrap around.
6. Check if the WBD flush is required. First the user should check if the old_high value from step 2 is equal to UNDEFINED_SN, and if so, no action is required and step 7 can be skipped.
7. This step uses RX_LSQN from the CPT. The user should check if the old_high value from step 2 is greater than RX_LSQN, and if so, the WBD flush command should be called as described in [Section 2.2, “Triggering a WBD Flush.”](#) If it is not greater than RX_LSQN, then no action is required.

These steps are summarized in the code shown in [Example 1](#).

Example 1. Code for Handling Flushing of WBD

```

old_high = flush_array[current_idx];    /* get highest SN from previous timeout period */
flush_array[current_idx] = p_CptPtr->rxMax; /* copy CPT[RX_MAX] to array*/
current_idx += 1;

temp = old_high - p_CptPtr->rxLSQN; /* check if old_high > RX_LSQN (signed int)*/
if (high_idx > arraySize) current_idx=0; /* Handle wraparound */
if (old_high == UNDEFINED_SEQNO) {
    /* Do nothing as this is the initialization value*/
} else if (temp > 0)/* check if flush is required */ {
    flushWBD(old_high);
} else {
    /* No action */
}

```

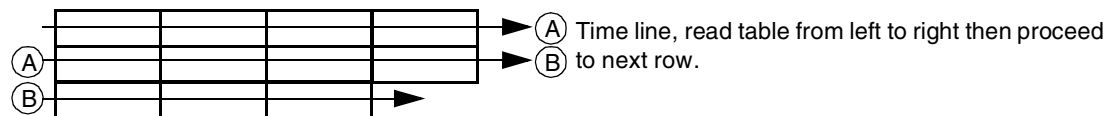
An example scenario for detecting WBD timeout is shown in [Table 2](#).

Table 2. Fragment Loss Detection in Practice

Element 0: Before After	RX_MAX RX_LSQN	Element 1: Before After	RX_MAX RX_LSQN	Element 2: Before After	RX_MAX RX_LSQN	Element 3: Before After	RX_MAX RX_LSQN
<input type="checkbox"/> 0	UND 0	<input type="checkbox"/> 1	UND 1	<input type="checkbox"/> 2	UND 2	<input type="checkbox"/> 3	UND 3
<input type="checkbox"/> 4	0 4	<input checked="" type="checkbox"/> 5 4 (5 LOST)	1 4	<input type="checkbox"/> 6	2 6	<input type="checkbox"/> 7	3 7
<input type="checkbox"/> 8	4 8	<input type="checkbox"/> 9	4 9	<input type="checkbox"/> 10	6 10		5 FLUSH

Notes:

Denotes sequence number received always 'just' before the timeout expiry in this example



[Table 2](#) details an example using an array with four elements. This example is working on the traffic pattern shown in [Table 1](#). The table is read from left to right and then proceeds to the next row. It assumes that each element is checked every 1 ms and a fragment is received every 1 ms. It also assumes that the fragments are received in incremental sequence number order. In terms of how to interpret each entry in the table, consider the following entry found in the first two columns of [Table 2](#).

0	4
4	5

- Element 0 Before = 0—this is the old high (step 2).
- Element 0 After = 4—this is the new high (CPT[RX_MAX] read in step 1).
- RX_MAX = 4—this is CPT[RX_MAX] the highest sequence number received on this bundle at this point in time.
- RX_LSQN = 5—this is the next expected sequence number, because no fragments are lost at this point in time and there is no hole in the WBD.

The example in [Table 2](#) detects fragment loss in a similar time frame to the RFC M algorithm when the transmission and reception is round robin. In [Table 2](#), the fragment with sequence number 5 is lost. The example shows that the fragment number 6 is delayed in the WBD by 4 ms regardless of which links the fragments are received on.

For example, if all the fragments are received on one link in the bundle (remember the bundle in this example contains four links), the M algorithm does not detect any fragment loss. The above example detects the fragment loss in a predictable manner regardless of which links in the bundle the fragments are received on.

Table 3 illustrates the original example of a fragment received every 20 ms. This time the array holds ten entries and is checked every 1 ms. Therefore, the system timeout period is 10 ms; this is the maximum amount of time a packet can be delayed in the WBD. The results are shown in Table 3.

Table 3. Fragment Loss Detection Time with 20 ms Traffic

	0	1	2	3	4	5	6	7	8	9	RX_LSQN	RX_MAX
Array Element Before Array Element After	U ¹ 0	U 0	U 0	U 0	U 0	U 0	U 0	U 0	U 0	U 0	1	0
Array Element Before Array Element After	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	1	0
Array Element Before Array Element After	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	2	1
Array Element Before Array Element After	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	2	1
Array Element Before Array Element After	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	3	2
Array Element Before Array Element After	2 2	2 2	2 2	2 2	2 2	2 2	2 2	2 2	2 2	2 2	3	2
Array Element Before Array Element After	2 3	2 3	2 3	2 3	2 3	2 3	2 3	2 3	2 3	2 3	4	3
Array Element Before Array Element After	3 3	3 3	3 3	3 3	3 3	3 3	3 3	3 3	3 3	3 3	4	3
Array Element Before Array Element After	3 4	3 4	3 4	3 4	3 4	3 4	3 4	3 4	3 4	3 4	5	4
Array Element Before Array Element After	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	5	4
Array Element Before Array Element After	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	5 LOSS	4
Array Element Before Array Element After	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	4 4	5	4
Array Element Before Array Element After	4 6	4 6	4 6	4 6	4 6	4 6	4 6	4 6	4 6	4 6	5	6
Array Element Before Array Element After	6 6	Fragment loss detected in first element in this sweep through the array. WBD flush issued.									5	6

Note:

¹ Undefined, the initialization value of the array (0x80000000).

Each row in Table 3 is a 10-ms chunk of time. Because a fragment is received every 20 ms, it takes two rows to receive consecutive fragments. It is assumed that the fragment is received between check 9 and wrapping again to check 0, thus the RX_LSQN and RX_MAX are the values used in each check within

one row. [Table 3](#) shows that the fragment with sequence number 6 is received and stays inside the WBD for ten checks of the array. A fragment loss is detected after 10 ms, which is much less than the 60 ms when four links are in the bundle, or 320 ms when 16 links are in the bundle. This therefore results in a more robust solution to fragment loss handling.

2.1.1 Choosing a System Timeout Period

The system timeout period should take into account the following factors:

1. Transmission time difference between largest and smallest frame
2. Links' differential delay

2.1.1.1 Frame Size Transmission Delay

The transmit time difference of these fragments should be used to help set the system time out period. Assuming that the largest packet in the system is 1536 bytes and the smallest is 64 bytes, this gives the following transmit time:

- 64-byte fragment = 250 μ s
- 1536-byte fragment = 6 ms

For this example, assume the bundle consists of two E1 links and that all 32 1-byte timeslots on the E1 carry data for the MLPPP link. If a 1536-byte fragment starts on one link at the same time as a burst of 64-byte fragments on the other link, the system timeout must be configured to take this into account. [Table 4](#) demonstrates system behavior when the system level timeout is poorly defined. It assumes a 1-ms array element check within a four element array (system time out of 4 ms). In [Table 4](#) the 1536-byte fragment uses sequence number 0 and the 64-byte fragments use sequence numbers 1, 2, 3 etc; sequence number 1 is lost.

Table 4. Poorly Defined System Level Timeout

	Element 0	RX_LSQN RX_MAX	Element 1	RX_LSQN RX_MAX	Element 2	RX_LSQN RX_MAX	Element 3	RX_LSQN RX_MAX
Element Before	U ¹	0	U	0	U	0	U	0
Element After	4	4 ²	8	8	12	12	16	16
Element Before	3	0	Fragment loss is detected by the algorithm, but wrongfully so.					
Element After	20	20						

Note:

- ¹ Undefined, the initialization value of the array (0x80000000).
- ² In each 1ms interval 4 of the 64 byte frames are received, the 64 byte frames start with sequence number 1.

[Table 4](#) shows that the fragment loss is detected too early. In each 1-ms interval, four frames have been received, and because the fragment zero has yet to be fully received RX_LSQN, is configured to zero. Because the array has four elements, each checked every 1 ms, the fragment loss is detected after 4 ms. However, because the 1536-byte fragment has not been fully received, it is not known whether the fragment with sequence number 1 will appear on the same link as the large packet.

This scenario should be detected by the RFC1990 M algorithm because it occurs when all of the bandwidth on the bundle is being fully used. The mechanism for detecting fragment loss in this document is designed for fragment loss detection in low density traffic. It is therefore recommended that the system level timeout is programmed to a value equal to or greater than the transmission time of the largest fragment or frame (plain PPP and LCP frames are transferred over the same links as fragments) possible in the system.

2.1.1.2 Links' Differential Delay

A property of bundling multiple E1 links is that each link has its own delay characteristics. The propagation time for links on the same bundle can be different for each link. Figure 2 shows an example.

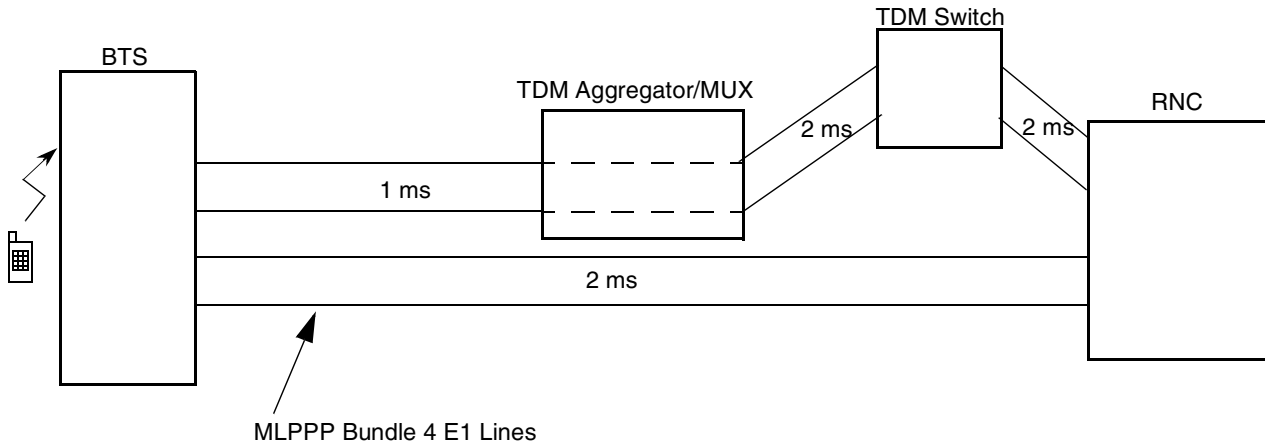


Figure 2. E1 Bundling with Different Propagation Characteristics

The propagation delay differs because two links are connected directly to the RNC and two links are connected to the RNC via aggregation or switching equipment. The two links with the point-to-point connection to the RNC have a propagation delay of 2 ms. The other two links have a propagation delay of 5 ms (1 ms propagation delay from BTS to TDM aggregator + 2 ms propagation delay from aggregator to switch + 2-ms propagation delay to RNC). The differential delay is therefore 3 ms between the links.

This is important because the largest packet in the system can be received on the link with the largest propagation delay. Therefore, the links' differential delay must also be factored into the configuration of the system level timeout value. It is recommended that the user configures the system-level timeout value using Equation 1.

$$\text{System Level Time Out} = \text{Largest Packet Tx Time} + \text{Links' Differential Delay} \quad \text{Eqn. 1}$$

2.2 Triggering a WBD Flush

To initiate a flush of the packets delayed in the WBD, the software must initialize two PPP microcode structures in the following order:

1. Initialize the PPP flush WBD command descriptor
2. Initialize the PPP real time (RT) command register

The PPP RT command register is located at offset 0xFC from the MCC Global PRAM Base. The PPP flush WBD command descriptor is located at an offset from the MURAM base pointed to by the PPP real time (RT) command register.

2.2.1 PPP RT Command Register

The PPP RT command register, shown in [Figure 3](#), is a user-initialized parameter.

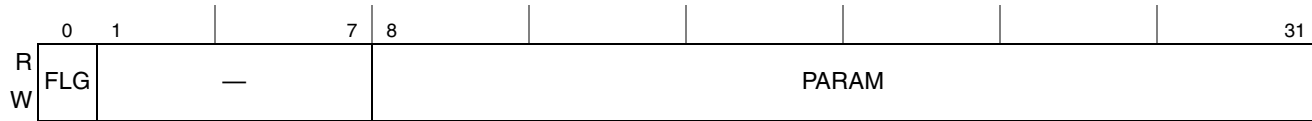


Figure 3. PPP RT Command Register

[Table 5](#) describes the PPP RT command register fields.

Table 5. PPP RT Command Register Field Descriptions

Bits	Name	Description
0	FLG	Command semaphore flag. Set by the host and cleared by the QUICC Engine module. 0 The QUICC Engine module is ready to receive a new command. 1 The PPP RT command register contains a command that the QUICC Engine module is currently processing. The QUICC Engine module clears this bit at the end of the command processing. When the FLG bit is cleared, the host can safely issue a new command.
1–7	—	Reserved.
8–31	PARAM	Command parameter. Configure this to an offset from the MURAM where the PPP Flush WBD Command Descriptor resides.

NOTE

The PPP Flush WBD command descriptor must not be changed in any way while the PPP RT command register FLG field is equal to 1.

2.2.2 PPP Flush WBD Command

This RunTime command starts the process of flushing the WBD for a given class. The host is required to provide a sequence number (FLS_SN) that is used as the new minimum of the Class. The QUICC Engine flushes all fragments, in sequence, with a sequence number less than FLS_SN. The QUICC Engine treats fragments not yet received with a sequence number less than FLS_SN as lost.

The host should prepare a PPP Flush WBD command descriptor (16 bytes in MURAM) as described in [Table 6](#). The offset in MURAM to this descriptor should be initialized in the CMD field of the PPP RT command register.

Table 6. PPP Flush WBD Command Descriptor

Offset	Name	Size	Field Description
Offset +0x00	—	1 byte	Reserved
	BPT pointer	3 bytes	Should be initialized to the bundle parameter table pointer of the Class.

Table 6. PPP Flush WBD Command Descriptor (continued)

Offset	Name	Size	Field Description
Offset +0x04	—	1 byte	Reserved
	CPT pointer	3 bytes	Should be initialized to the class parameter table pointer.
Offset + 0x08	FLS_SN	4 bytes	This sequence number is used as the new minimum (over the links) of the Class. QUICC Engine flushes the WBD Ring up to, but not including, this sequence number.
Offset + 0x0C	Class_Info	2 bytes	Class information. See Figure 4 .
Offset +0x0E	—	2 bytes	Reserved

The PPP flush WBD command descriptor’s Class_Info field is show in [Figure 4](#).

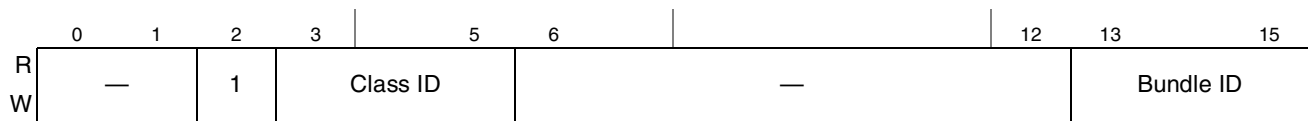


Figure 4. Class Information

This host command can be used when the fragment loss detection algorithm detects that a fragment has been lost. After issuing this host command via the PPP RT command register, newly received fragments with sequence numbers less than the Sequence Number field will be silently discarded.

The Sequence Number field is shown in [Figure 5](#) and [Figure 6](#).

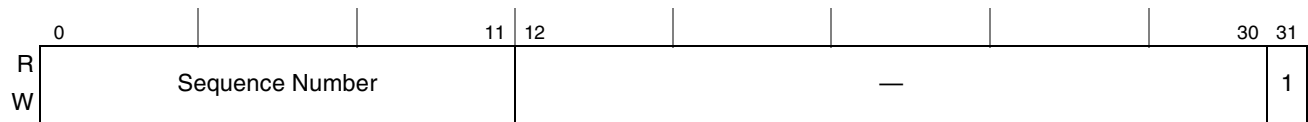


Figure 5. 12-Bit Sequence Number

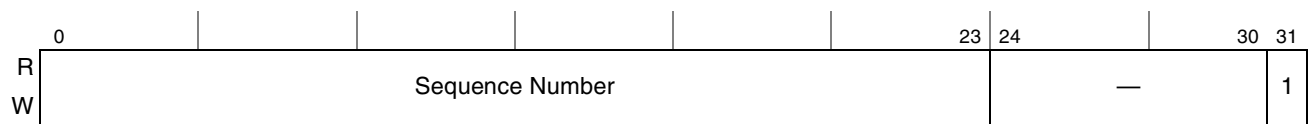


Figure 6. 24-Bit Sequence Number

NOTE

See the *QUICC Engine™ Block Reference Manual with Protocol Interworking* for the most current information on the WBD flush command.

3 Conclusion

The RFC1990 M algorithm works well when packets are transmitted round robin over links in the bundle in periods of high density traffic. However, in periods of low density traffic, or if the packets are not transmitted round robin, the M algorithm cannot guarantee detecting fragment loss in a known time frame. This can have an adverse effect on QoS. The M algorithm coupled with the fragment loss detection

algorithm described in this application note presents a robust and deterministic solution to fragment loss handling regardless of the traffic profile.

4 Revision History

Table 7 provides a revision history for this application note.

Table 7. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	10/2009	<ul style="list-style-type: none"> • Initial public release.

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor
Literature Distribution Center
1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale and the Freescale logo are trademarks or registered trademarks of Freescale Semiconductor, Inc. in the U.S. and other countries. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc., 2009. All rights reserved.