Internet Engineering Task Force
INTERNET-DRAFT Anura Jayasumana
draft-jayasumana-reorder-density-01.txt Nischal M. Piratla
Abhijit A. Bare
Tarun Banka
Colorado State University
July 2003
Expires: December 2003

Reorder Density Function - Metric for packet reordering measurement

Status of this memo

Abstract

Out-of-order arrival of packets can significantly degrade the performance of many TCP-based, VoIP-based and Video-based applications. There is a need for a metric that can meaningfully, accurately and unambiguously characterize reordering. This memo proposes a new metric called Reorder Density function (RD), which can give an in-depth view of the reordering present in any packet sequence. This well-defined metric can also be used to evaluate

effects of protocol and hardware implementations on packet reordering. The memo also provides an algorithm to compute the reorder density function followed by some illustrative examples.

1. Introduction and Motivation

Out-of-order arrival of packets is a common phenomenon on the Internet. Major cause of reordering of packets is the local parallelism present in network routers and switches. This parallelism is caused due to different load balancing algorithms used in routers and switches. Packets can also be reordered due to different queuing schemes within the networking equipment itself. Packet reordering leads to degradation of the performance of the applications. For example, perceived quality of voice degrades if a VoIP application receives packets out of order. Once we are able to quantify the degree of reordering in arriving packet streams, it is possible to predict the effects of reordering on applications that are sensitive to reordering, and perhaps even compensate for reordering. This can further help us in evaluating network protocols with respect to packet reordering.

Until now, the percentage of out-of-order packets has been used as a metric for characterizing reordering. However, this metric is vague and lacks in detail. There is also no uniform definition for the the degree of reordering of an arrived packet. For example, consider two packet sequences (1,3,4,2,5) and (1,4,3,2,5). It is possible to interpret the reordering of packets differently in this case, for example [1],

(i) Packets 2, 3 and 4 are out ûof order in both cases.
(ii) Only packet 2 is out of order in the first sequence, while packets 2 and 3 are out of order in the second.
(iii) Packets 3 and 4 are out of order in both the sequences.
(iv) Packets 2, 3 and 4 are out of order in the first sequence, while packets 4 and 2 are out ûof order in the second sequence.

In essence, the percentage of out-of-order packets is subject to interpretation and it cannot capture the reordering unambiguously and, hence, accurately. Thus, there is a need for a more precise and complete definition.

Taking any of the above sequences, if buffers are available to store the packets 3 and 4 while waiting for packet 2, it is possible to recover from the reordering. However, there may be cases where the application requirement is such that the arrival of packet 2 after this delay renders it useless. While one can argue that a good packet reordering measurement scheme should capture such effects, a counter argument can also be made that packet reordering should be measured strictly with respect to the order of delivery and should be application independent.

In this memo, we define a metric called Reorder Density function (RD). RD is the normalized form of a histogram of the occupancy of a

hypothetical buffer that would allow the recovery from out-of-order delivery of packets.

In addition to providing a consistent percentage reordering measure, RD can also be used to compute the percentages corresponding to different degrees of reordering. Next sections explain the concept of the reorder density function (RD).

2. Definitions of terms used

Some important terms are defined, which will help us describe the Reorder Density Function (RD).

2.1 Out-of-order packet:

When a packet other than the expected packet arrives, it is considered as an out-of-order packet, provided it is not a duplicate of an already received packet.

2.2 Buffer Occupancy (D):

An arrived packet with a sequence number greater than the expected packet is considered to be stored in a buffer. Note that this is only a hypothetical buffer that we use to define RD. At any packet arrival instant, the buffer occupancy (assuming one buffer per packet) is therefore the number of such out-of-order packets. If the newly arrived packet is out of order, it will occupy the buffer as well. For example, for the sequence of packets (1,2,4,5,3), the buffer occupancy value, when the packet with the sequence number 4 arrives is 1 because it arrived when 3 was expected. Similarly, the buffer occupancy becomes 2 when the packet with the sequence number 5 arrives, as both packets 3 and 4 have to be held in the buffer. When packet 3 arrives, the occupancy becomes zero as it is no longer necessary to hold packets 4 and 5 to recover from reordering. (The term buffer occupancy was called displacement in [1].)

2.3 Occupancy Threshold (DT):

This parameter defines the tolerance of the application to the maximum allowed displacement of a packet. It can also be viewed as the maximum size of the hypothetical buffer. If an out-of-order packet needs to be stored in the hypothetical buffer already filled to the value of occupancy threshold, the currently expected packet is considered to be delayed more than the tolerance and hence, is assumed to be lost. The threshold may be chosen such that even if the packet ultimately arrives after the threshold, it is of no use to the application. Many factors influence the selection of the occupancy threshold value, for example, the transport layer protocol (UDP or

TCP), the amount of redundant information sent to recover from losses, and whether the sequence of packets belong to a time-sensitive application or not.

In case of a VoIP application, for example, with a bit-rate of 128 kbps and packet size of 200 bytes, DT value can be determined as follows. Assume that the application can wait at most 50 ms for an expected packet, and that the packets arrive at constant rate. That means within 50 ms, the application can receive (128*1000*0.05)/(200*8) i.e. 4 packets. Therefore, the occupancy threshold should be kept at 4.

In case of TCP, a lost or delayed packet will be retransmitted and will reach the destination. So the value of the DT should be kept at least equal to the size of the receiving window on the receiver side.

2.4 Frequency of Occurences (F)

At the arrival of each packet the buffer occupancy may take any value 'i' ranging from 0 to DT. The frequency of occurrence F[i] is the number of times the occupancy takes the value of 'i'.

2.5 Definition of RD

RD is defined as the distribution of all frequencies of occurrence, F[i], normalized with respect to the total number of occurrences, i.e, sum(F[i]) for all i in [0, DT].

3. Algorithm to compute reorder density (RD) function

This section describes an algorithm to compute the reorder density function. Without loss of generality, the description assumes that the sequence numbers start at 1 and increment by 1 for each in-order packet.

```
# E : Next expected sequence number.
# S : Sequence number of the packet just arrived.
# D : Current buffer occupancy.
# DT : Occupancy threshold.
# F[i] : Frequency of occurrence of D = i.
# RD[i] : Normalized frequency for D = i.
# in_buffer(N) : True if the packet with sequence number N is
  already stored in the buffer.
```

==========

1. Initialize E = 1,D = 0 and F[i] = 0 for all values of i (0 <= i <= DT).

2. Do the following for each arrived packet.

```
If (in_buffer(S) || S < E) /*Do nothing*/;
```

```
          /* Case a: S is a duplicate or delayed packet. Discard the
            packet.*/
```

```
          ElseIf (S == E)
          /* Case b: Expected packet has arrived.*/
          {
             E = E + 1;
             While (in_buffer(E))
             {
                D = D - 1; /* Free buffer occupied by E.*/
                E = E + 1; /* Expect next packet.*/
             }
             F[D] = F[D] + 1; /* Update frequency for buffer occupancy
                                 D.*/
          } /* End of ElseIf (S == E)*/

          ElseIf (S > E)
          /*Case c: Arrived packet has a sequence number higher
            than expected.*/
          {
             If (D < DT)
             /* Store the arrived packet in a buffer.*/
                D = D + 1;
             Else
             /* Expected packet is delayed beyond the DT. Treat it as
                lost.*/
             {
                 Repeat
                 {
                    E = E + 1;
                 }
                 Until (in_buffer(E) || E == S);

                 While (in_buffer(E) || E == S)
                 {
                       D = D - 1;
                       E = E + 1;
                 }
             }
             F[D] = F[D] + 1; /* Update frequency for buffer occupancy
                                 D.*/
          } /* End of ElseIf (S > E)*/
```

3. Normalize F[i] to get RD[i] for all values of i (0 <= i <= DT) using

$$RD[i] = \frac{F[i]}{\text{Sum}(F[j] \text{ for } 0 <= j <= DT)}$$

The algorithm starts with the initialization of D to 0 and E to 1. Let S be the sequence number of an arrived packet.

If S has been received previously or delayed subject to the occupancy threshold condition (case a), it is discarded.

If S is the expected packet (case b), E is incremented by 1 (i.e. the next packet in the sequence is now expected). If the packet with new E, i.e., the next packet in the sequence has already arrived, it need not be held in the buffer any more (it can be used by the application). So the buffer occupancy value is reduced by 1 and E is incremented by 1. This is repeated till all the in-sequence waiting packets are removed.

If the received packet with the sequence number S is not the expected packet, two cases are possible. First case is when S is higher than E (case c), i.e., received packet is an out-of-order packet. If the buffer occupancy is less than the occupancy threshold, the packet with the sequence number E can still be expected. The value of the buffer occupancy is incremented, because the newly arrived packet needs to be stored in the hypothetical buffer. On the other hand, if the buffer occupancy is equal to the occupancy threshold, the currently expected packet E is assumed to be lost and E is incremented repeatedly till E reaches the sequence number of a packet that has been already received. This packet can now be removed from the hypothetical buffer giving space to the newly arrived packet. E is incremented further to check if there are any packets with higher sequence numbers already arrived and waiting, similar to what is done in the S=E case (in case b).

The frequency value for the new value of the buffer occupancy is incremented as shown in the algorithm.

Once the algorithm deals with all the packets and the frequency F[D] is computed, for all the values of D, the F[D] values are normalized to get the density with respect to D. This function is called the Reorder Density function.

4. Examples

We consider a few different sequences to exemplify the above algorithm.

a. Case of no packet loss:

Consider a sequence of 5 packets (1,4,2,5,3) with DT = 10.

Table 1 and 2 show the computation steps when RD algorithm is

applied
   to above sequence.

Table 1: Reorder Histogram computation steps					
E	1	2	2	3	3
S	1	4	2	5	3
D	0	1	1	2	0
F[D]	1	1	2	1	2

(E,S,D,F[D] as described in section 3)

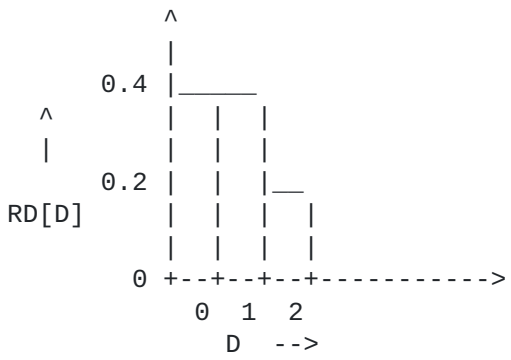
The last row (F[D]) represents the current frequency of occurrence of the buffer occupancy D. The final set of values for F[D] are shown in table 2.

When the first packet with the sequence number S=1 arrives, it is same as the expected sequence number E=1, resulting in the buffer occupancy D=0. Next, when the packet S=4 arrives instead of the expected packet E=2, the buffer occupancy D becomes 1. After receiving the packet with the sequence number 2, the buffer occupancy D is still 1, since the packet 3 that is expected now is not yet received. Packet 4 continues to occupy a buffer. Only one buffer is needed and hence D = 1. On receiving the packet with the sequence number 5, the buffer occupancy D becomes 2. Finally, when we receive the packet with sequence number 3, all the packets up to the sequence number 5 have been received. Thus the buffers can be released and hence the buffer occupancy D becomes 0. The reorder density function (RD) is derived by normalizing reorder histogram in Table 1 as follows:

Table 2: Reorder Density Function (RD)			
D	0	1	2
F[D]	2	2	1
RD[D]	0.4	0.4	0.2

(D,F[D],RD[D] as described in section 3)

Graphical representation of above RD is as follows:



b. Case of packet loss:

Consider a sequence of 6 packets (1,2,4,5,6,7) with DT = 3.

Tables 3 and 4 show the computation steps, when the RD algorithm is applied to the above sequence.

Table 3: Reorder Histogram computation steps

E	1	2	3	3	3	3
S	1	2	4	5	6	7
D	0	0	1	2	3	0
F[D]	1	2	1	1	1	3

(E,S,D,F[D] as described in section 3)

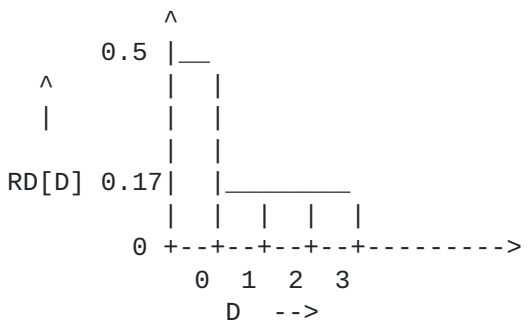
When a packet with the sequence number 4 is received, the expected packet E is 3. So the buffer occupancy D increases by 1. When the packets with the sequence numbers 5 and 6 arrive, D increases to 2 and then to 3 respectively. The buffer occupancy is now equal to the occupancy threshold DT=3. Therefore, when the packet 7 is received, we no longer expect the packet with the sequence number 3 to arrive and assume that it is lost. We can now use all the waiting packets (4,5,6 and 7), reducing the buffer occupancy to 0. The reorder density function (RD) is derived by normalizing the reorder histogram in Table 3 as follows:

Table 4: Reorder Density Function (RD)

D	0	1	2	3
F[D]	3	1	1	1
RD[D]	0.5	0.17	0.17	0.17

(D,F[D],RD[D] as described in section 3)

Graphical representation of above RD is as follows.



c. Case of Duplicate packets:

Consider a sequence of 6 packets (1,3,2,3,4,5) with DT = 5.

Tables 5 and 6 show the computation steps when the RD algorithm is applied to the above sequence.

Table 5: Reorder Histogram computation steps

E	1	2	2	4	4	5
S	1	3	2	3	4	5
D	0	1	0	-	0	0
F[D]	1	1	2	-	3	4

(E,S,D,F[D] as described in section 3)

In the above sequence, duplicate packets are received by the destination. The RD algorithm ignores the arrivals of the duplicate packets.

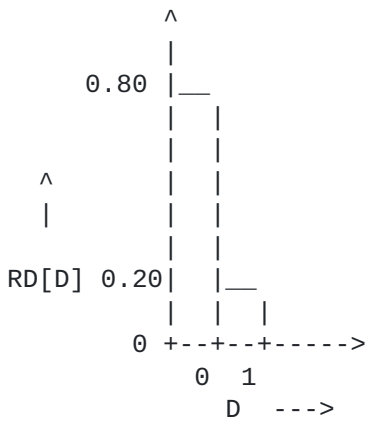
The reorder density function (RD) is derived by normalizing reorder histogram in Table 5 as follows:

Table 6: Reorder Density Function (RD)

D	0	1
F[D]	4	1
RD[D]	0.80	0.20

(D,F[D],RD[D] as described in section 3)

Graphical Representation of RD is as follows:



5. Simple metrics derived from RD

While the reorder density can provide a detailed picture of the degree of reordering present in a sequence of packets, there may be instances, where a simpler metric is needed to compare sequences. The following parameters derived from the reorder density may be used as simpler metrics for packet reordering. Reference [1] shows that these simpler metrics can effectively capture the relative degrees of reordering of packet in sequences effectively.

5.1 90th percentile of RD

This parameter is the buffer occupancy value, such that 90 % of the arrived packets have buffer occupancy less than this value.

5.2 Mean and Standard Deviation of RD

Mean and standard deviation of the buffer occupancy values of the arrived packets may be used as simple metrics.

6. Current Schemes

Currently, the percentage of out-of-order packets is the most commonly used packet reordering metric. With the percentage reorder metric, the information provided by the metric is purely for information only. For example, consider two sequences at the receiver end (2,3,4,5,1) and (2,1,3,4,5). Taking the definition of late arrival as reordered packet [2], in both the cases the percentage reordering is 20. However, it is obvious that the reordering in the second sequence is more acceptable than the first one as the recovery

from the packet reordering is much easier in the former case. This metric is a significant simplification and is not useful in the recovery from reordering.

N-reordering [3] is a metric where an expected packet is 1-reordered, 2-reordered and so on till it arrives. If a packet arrives after 40 positions from its expected position then it is 40-reordered. Two examples are listed in Appendix A to show the difference between reorder density and N-reordering. These examples how that N-reordering is much more susceptible to delayed packets as it cannot treat them as lost when their useful life is over, whereas with RD this is taken care of using threshold.

Reordering offset[4] is another metric to measure reordering. In this metric the packet is not reordered until it arrives. However, a duplicate packet is considered as a reordered packet. Unlike RD, this metric is not orthogonal to duplication of packets. Appendix B uses a few few example sequences to compare Reordering offset and RD.

7. Security Considerations

This document does not define any protocol. The metric definition per se is believed to have no security implications.

8. IANA Considerations

This document requires nothing from the IANA.

9. References

1. T. Banka, A. A. Bare, A. P. Jayasumana, "Metrics for Degree of Reordering in Packet Sequences", Proc. 27th IEEE Conference on Local Computer Networks, Tampa, FL, Nov. 2002.

2. V.Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," Ph.D. dissertation, U.C. Berkeley, 1997, ftp://ftp.ee.lbl.gov/papers/vp-thesis/dis.ps.gz.

3. S. Shalunov, "Definition of IP Packet Reordering Metric", Internet Draft, <draft-shalunov-reordering-definition-02.txt>, March 2003.

4. A. Morton, L. Ciavattone, G. Ramachandran, S.Shalunov and J. Perser, "Packet Reordering Metric for IPPM", Internet Draft, <draft-ietf-ippm-reordering-03.txt>, June 2003.

10. Authors' Addresses

Anura Jayasumana <anura@engr.colostate.edu>
Nischal M. Piratla <nischal@engr.colostate.edu>
Abhijit A. Bare <abhijit@cs.colostate.edu>
Tarun Banka <tarunb@cs.colostate.edu>

Computer Networking Research Laboratory,
Department of Electrical and Computer Engineering,
Colorado State University,
Fort Collins, CO 80523.

Expiration Date: December 2003

11. Appendix A

Example 1:For the sequence <1,2,3,4,5,2,1>

RD output:

D	0	1	2	3
F[D]	5	0	0	0
RD[D]	1.00	0.00	0.00	0.00

N-reordering output:
1-reordering = 33.333333%
2-reordering = 40.000000%
3-reordering = 50.000000%
4-reordering = 33.333333%
5-reordering = 50.000000%
no 6-reordering

1-reordering = 2
2-reordering = 2
3-reordering = 2
4-reordering = 1
5-reordering = 1
no 6-reordering

In this example, the N-reordering algo shows that there is 5-reordering. If you look at the sequence there are two duplicate packets namely, seq#s 2 & 1. In RD, the F[D] does not exist for D>0 i.e., there is no reordering. As one can see, the sequence has no reordering.

Example 2: For Sequence:
<1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,2>

RD output with DT = 5:

D	0	1	2	3	4	5
F[D]	35	1	1	1	1	1
RD[D]	0.875	0.025	0.025	0.025	0.025	0.025

N-reordering output:

1-reordering = 2.500000%
2-reordering = 2.564103%
3-reordering = 2.631579%
4-reordering = 2.702703%
5-reordering = 2.777778%
6-reordering = 2.857143%
7-reordering = 2.941176%
8-reordering = 3.030303%
9-reordering = 3.125000%
10-reordering = 3.225806%
11-reordering = 3.333333%
12-reordering = 3.448276%
13-reordering = 3.571429%
14-reordering = 3.703704%
15-reordering = 3.846154%
16-reordering = 4.000000%
17-reordering = 4.166667%
18-reordering = 4.347826%
19-reordering = 4.545455%
20-reordering = 4.761905%
21-reordering = 5.000000%
22-reordering = 5.263158%
23-reordering = 5.555556%
24-reordering = 5.882353%
25-reordering = 6.250000%
26-reordering = 6.666667%
27-reordering = 7.142857%
28-reordering = 7.692308%
29-reordering = 8.333333%
30-reordering = 9.090909%
31-reordering = 10.000000%
32-reordering = 11.111111%
33-reordering = 12.500000%
34-reordering = 14.285714%
35-reordering = 16.666667%

36-reordering = 20.000000%
37-reordering = 25.000000%
38-reordering = 33.333333%
39-reordering = 50.000000%
no 40-reordering

This example clearly shows that N-reordering is much more susceptible to delayed packets as it cannot treat them as lost when their useful life is over, whereas with RD this is taken care of.

12. Appendix B

From <draft-ietf-ippm-reordering-01.txt>

"...Table 1 Example with Packet 4 Reordered,

Sending order(SrcNum@Src): 1,2,3,4,5,6,7,8,9,10

SrcNum @Dst	NextExp	Src Time	Dst Time	Delay	IPDV	Dst Order	Posit. Offset	Late Time
1	1	0	68	68		1		
2	2	20	88	68	0	2		
3	3	40	108	68	0	3		
5	4	80	148	68	-82	4		
6	6	100	168	68	0	5		
7	7	120	188	68	0	6		
8	8	140	208	68	0	7		
4	9	60	210	150	82	8	4	62
9	9	160	228	68	0	9		
10	10	180	248	68	0	10		

Each column gives the following information:

SrcNum Packet sequence number at the Source.
NextExp The value of NextExp when the packet arrived(before update).
SrcTime Packet time stamp at the Source, ms.
DstTime Packet time stamp at the Destination, ms.
Delay 1-way delay of the packet, ms.
IPDV IP Packet Delay Variation, ms
IPDV = Delay(SrcNum)-Delay(SrcNum-1)..."

Reorder Density for the above example:

```
   SrcNum
   @Dst NextExp  Buffer occupancy  Frequency
    1     1          0              F[0] = 1
    2     2          0              F[0]++
    3     3          0              F[0]++
    5     4          1              F[1] = 1
    6     4          2              F[2] = 1
    7     4          3              F[3] = 1
    8     4          4              F[4] = 1
    4     4          0              F[0]++
    9     9          0              F[0]++
   10    10          0              F[0]++
```

Normalized F[i] for all i: F[0] = 0.6, F[1] = 0.1, F[2] = 0.1, F[3] = 0.1, F[4] = 0.1

In this case, if we can set DT = 3, then the table will look like this:

```
   SrcNum
   @Dst Expected  Buffer occupancy Frequency
   1     1          0              F[0] = 1
   2     2          0              F[0]++
   3     3          0              F[0]++
   5     4          1              F[1] = 1
   6     4          2              F[2] = 1
   7     4          3              F[3] = 1
   8     4          0              F[0]++    {after the current
packet's
                                               arrival, packet '4' is
                                               rendered useless!}
   4     9          0              -         {discarded pkt.}
   9     9          0              F[0]++
   10    10         0              F[0]++
```

Normalized F[i] for all i: F[0] = 5/9, F[1] = 1/9, F[2] = 1/9, F[3] = 1/9

Other examples in current metrics are almost similar. However, there are no examples with packet duplication. Here is one for the metric proposed. (Note: Packet '5' is duplicated.)

SrcNum @Dst	NextExp	Buffer Occupancy	Frequency
1	1	0	F[0] = 1
2	2	0	F[0]++
3	3	0	F[0]++
5	4	1	F[1] = 1
6	4	2	F[2] = 1
7	4	3	F[3] = 1
8	4	4	F[4] = 1
4	4	0	F[0]++
5	9	0	- {duplicate packet}
9	9	0	F[0]++

Normalized F[i] for all i: F[0] = 5/9, F[1] = 1/9, F[2] = 1/9, F[3] = 1/9, F[4] = 1/9.

At the arrival of a duplicate packet the buffer occupancy is held the same as the duplicate packet will be discarded and the contents of the buffer remain.