THESIS

IMPACT OF RESEQUENCING BUFFER DISTRIBUTION ON PACKET REORDERING

Submitted by

Raghunandan Mandyam Narasiodeyar

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2011

ABSTRACT

IMPACT OF RESEQUENCING BUFFER DISTRIBUTION ON PACKET REORDERING

Packet reordering in Internet has become an unavoidable phenomenon wherein packets get displaced during transmission resulting in out of order packets at the destination. Resequencing buffers are used at the end nodes to recover from packet reordering. This thesis presents analytical estimation methods for “Reorder Density” (RD) and “Reorder Buffer occupancy Density” (RBD) that are metrics of packet reordering, of packet sequences as they traverse through resequencing nodes with limited buffers. During the analysis, a “Lowest First Resequencing Algorithm” is defined and used in individual nodes to resequence packets back into order. The results are obtained by studying the patterns of sequences as they traverse through resequencing nodes. The estimations of RD and RBD are found to vary for sequences containing different types of packet reordering patterns such as Independent Reordering, Embedded Reordering and Overlapped Reordering. Therefore, multiple estimations in the form of theorems catering to different reordering patterns are presented. The proposed estimation models assist in the allocation of resources across intermediate network elements to mitigate the effect of packet reordering. Theorems to derive RBD from RD when only RD is available are also presented. Just like the resequencing estimation models, effective RBD for a given RD are also found to vary for different packet reordering patterns, therefore, multiple theorems catering to different patterns are presented. Such RBD estimations would be useful for allocating resources based on certain QoS criteria wherein one of the metrics is RD. Simulations driven by Internet measurement traces and random sequences are used to verify the analytical results. Since high degree of packet reordering is known to affect the quality of applications using TCP and UDP on the Internet, this study has broad applicability in the area of mobile communication and networks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

The Internet has witnessed tremendous growth during the past decade. More people tend to use Internet today than ever before. The Internet has become ubiquitous due to vast information that is easily available to the users at their fingertips. Some reasons for such a growth have been increase in connectivity through different devices, high speeds and bandwidth aggregation across networks. Figure 1-1 shows a graph displaying the number of Internet users (in millions) across different regions in the world.

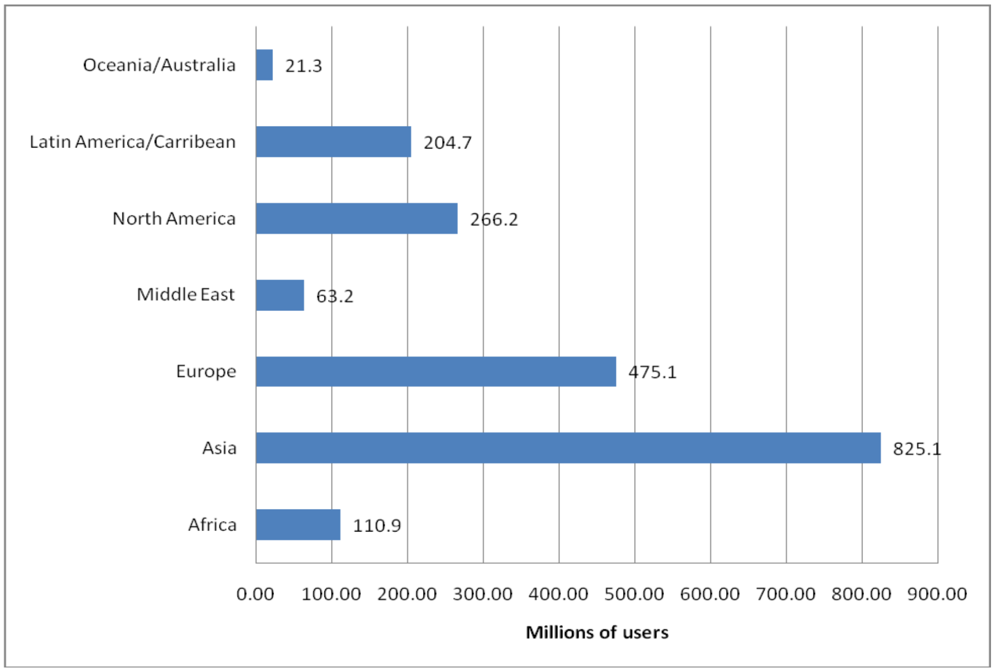


Figure 1-1. Graph showing the number of Internet users in 2010 [64]

From Figure 1-2, it can be seen that major continents such as North America, Austrlia and Europe have high (>50%) Internet penetration levels. However, the Internet penetration levels are still low in populous continents such as Asia, Africa and Middle East.

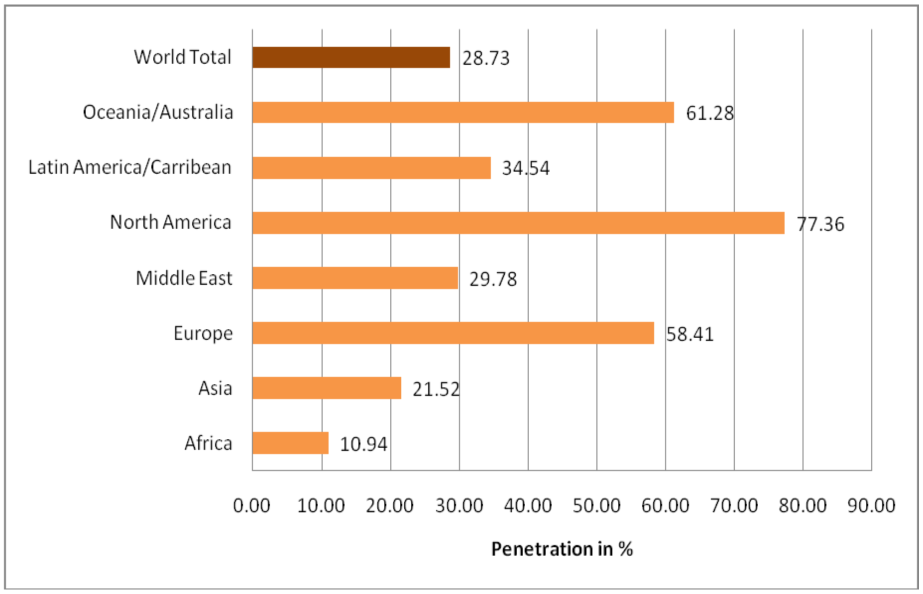


Figure 1-2. Graph showing the Internet penetration in 2010 [64]

With increase in the penetration levels across different regions, more businesses will tend to invest in providing quality Internet service. The gap between offering just connectivity and offering "good" connectivity would start gaining priority resulting in additional focus on Quality of Service (QoS). One of the main issues that could affect the QoS in future could be the issue of packet reordering.

Packet reordering is a common occurrence in the Internet wherein a sequence of packets transmitted from a source arrives at the destination in a different order. Packet reordering is found to have five major causes: packet-level multipath routing, route fluttering, inherent parallelism in modern high-speed routers, link-layer retransmissions and route forwarding lulls [3][7][60]. Irrespective of its cause, packet reordering has a significant detrimental effect in many contexts. Video traffic over UDP is found to perform poorly in situations of high packet reordering [1]. Wireless networks are more prone to problems due to high levels of packet reordering in link layer retransmissions [1]. The output quality of broadband digital television system becomes intolerable beyond a certain threshold of reordering [54][47]. Packet reordering combined with packet error is found to severely degrade the transmission performance in Low Earth Orbit (LEO) satellite networks [54]. Due to the increased parallelism in modern networks and the demands of high performance applications, recovery from packet reordering will consume increasing resources both at end nodes and routers [5]. New applications and protocol designs should be robust to both packet reordering and packet loss by treating them equal [14][18].

While some applications are capable of operating with reordered packets, many of the applications require packets to be put back in order, i.e., resequenced. While this happens in the transport layer when TCP is used, many real-time multimedia type applications handle it at the application level. Often, out-of-order packets significantly degrade the performance of applications, even when TCP or application level re-sequencing is used [57][17]. As a result, many modern router architectures involve extra hardware to minimize the induced reordering. Examples of such hardware include input buffers to ensure that packets from the same flow are forwarded to the same processing element [55] or output buffers to ensure that packets from a flow are released from the router in more or less the same order in which they arrive at the router [19]. These two approaches, involving tracking packet flows, can mitigate the packet reordering introduced within the routers to a certain extent. However, they have no effect on packet reordering due to other causes such as route fluttering. Furthermore, the two approaches are not scalable as the number of flows increase and they also tend to add to the end-to-end latency [5]. Packet reordering phenomenon and its impact on applications has been evaluated in detail in [31][47][54]. Measurement and quantification of packet reordering is addressed in [3][25], and techniques and metrics for long-term monitoring of packet reordering is addressed are [60].

1.1 Contribution

This thesis focuses on the resources required for recovery from packet reordering. Whether recovery is done at end-nodes, such as with application level buffering or TCP level resequencing, or if it is done at intermediate routers, such as with input or output buffering, the recovery inevitably requires buffers. Thus adding input/output buffering to a router may be viewed as transferring some of the recovery resources to intermediate nodes. From a buffer allocation standpoint, this is the first research to provide formal results relating to the impact of resequencing buffer distribution on packet reordering. Analytical results quantifying the impact on packet reordering as they flow through resequencing nodes having a limited number of buffers were derived. The results were verified using simulations. The approach and results are useful for buffer allocation within individual router nodes to keep packet reordering within acceptable levels. With present scaling trends in network link speeds and processor speeds, packet reordering will become an increasingly difficult problem to deal with [5][14]. It may even be considered to be an important Quality of Service (QoS) parameter for Internet service. This research lays the theoretical foundation for allocating recovery buffer resources in a distributed network.

1.2 Thesis outline

In Chapter 2, background about Packet Reordering, its measurement and the concept of reordering patterns are discussed. Chapter 3 focuses on the impact of distribution of resequencing buffers on packet reordering and Chapter 4 offers derivation of Reorder Buffer-Occupancy Density (RBD) from Reorder Density (RD). Chapter 5 concludes the research and Chapter 6 briefly discusses probable future work and applications of RBD.

CHAPTER 2. BACKGROUND

Packet reordering is a phenomenon in the Internet wherein a sequence of packets transmitted from a source, say (1, 2, 3, 4, 5, 6) arrives at the destination in a different order, say (1, 3, 4, 2, 5, 6) where packet 2 arrives two packet arrival instances later. Here, packet 2 is considered to be reordered with respect to the sequence at the source. The effects of packet reordering are known to be severe and it is now considered to be naturally prevalent within the Internet [21].

This chapter gives an introduction to packet reordering by way of mentioning the causes of packet reordering in Section 2.1, effects of packet reordering in Section 2.2, importance of understanding packet reordering in Section 2.3, measuring packet reordering in Section 2.4 followed by Section 2.5 that mentions about common packet reordering patterns seen in the Internet.

2.1 Causes of packet reordering

Packet reordering was initially considered to be a pathological network behavior [24]. Later studies have proved that packet reordering occurs because of specific reasons in the network. Some of them include

a) Heterogeneous network interfaces or bandwidth aggregation [26] [14]

   When trying to achieve bandwidth and increased fault tolerance in wireless networks, the heterogeneity of the paths and interfaces can cause packet ordering.

b) Handoff techniques [46]

   Soft handoff techniques in 4G networks during downward vertical hand off can lead to plenty of reordered packets.

c) Route fluttering [38]

   It refers to the rapidly variable routing when packets are sent from source to the destination.

d) Ad hoc routing [16]

   It refers to the ad hoc manner in which packets are routed between computing devices in a mobile ad hoc network.

e) Diffserv scheduling [22] [8][9]

   The non-conformant packets in the case of excess flow due to negotiated constraints are either dropped or given lower priority which could result in reordering.

f) Retransmissions on wireless links in TCP [6]

   Wireless links are known to perform poorly when packets are considered to be lost which invariably happens due to reordering.

g) Packet striping in layer 2 and layer 3 [7][2]

   A packet can follow multiple paths within a device or logical link. An example could be packets traveling between same source and destination taking different paths through the switch.

Apart from the aforementioned causes, some of the other reasons could include broken equipment, route pauses or route forwarding lulls, inherent parallelism in modern high speed routers [5], link layer retransmissions and packet level multipath routing [30].

2.2 Effects of packet reordering

From the end user perspective, packet reordering is known to cause poor performance in video traffic over UDP in situations of high packet reordering. Despite increased Internet load and technology advancements, the UDP traffic reordering measurements are consistent with prior studies conducted during 1990s [51]. UDP-based applications such as VoIP which use small packet sizes are known to be affected by reordering [51]. Wireless networks are more prone to problems due to high levels of packet reordering in link layer retransmissions [1]. Packet Reordering is known to have many effects on reliable transport layer sequenced protocols such as TCP which can falsely assume reordering to be packet loss [21]. This results in unnecessary retransmissions and activation of the congestion control algorithms causing severe drop in TCP throughput.

Other reliable transmission protocols as SCTP also suffer resequencing delay due to asynchronous packet arrivals at the receiver as a result of packet reordering [62] causing deterioration of performance of some delay sensitive applications.

The output quality of broadband digital television system becomes intolerable beyond a certain threshold of reordering [47]. Packet reordering combined with packet error is found to severely degrade the transmission performance in Low Earth Orbit (LEO) satellite networks [53]. Packet Reordering tends to induce fast retransmissions and unnecessary reduction of the congestion window in the mobile Stream Control TCP (mSCTP) vertical handover [27].

2.3 Importance of understanding packet reordering

Because of the detrimental effects of packet reordering on TCP and UDP, lots of effort is aligned towards negating the effect of packet reordering. Considerable emphasis is placed on maintaining packet order while designing new switch architectures [33]. Detailed reordering models have been proposed to address the lack of evaluations using real protocol implementations and good models of packet reordering [21]. New well-structured, area-efficient, and high speed hardware architectures for packet re-sequencing have been proposed to combat local parallelism within links or switches [52].

Novel TCP variants have been proposed that adapt TCP to wireless networks by using more reliable signals of packet loss and network overload for activating packet retransmission and congestion response separately [29]. Concepts such as locality buffering have been proposed to resequence packets by clustering packets with the same destination port [37]. New load balancing algorithms with flow chopping have been proposed to avoid packet reordering [10]. Studies indicate that new applications and protocol designs should be robust to both packet reordering and packet loss by treating them equal [8]. New flow based models have been proposed to achieve efficient bandwidth utilization and packet order preservation [44]

In the radio access technology for the next generation, standardization is happening for Long Term Evolution (LTE) and Ultra Mobile Broadband (UMB). During this process, studies are conducted keeping in mind the overhead for avoiding packet reordering [61]. Simple modifications to the TCP protocol are done to improve its performance to mobile-induced packet reordering [48]. Enhancements are made to TCP sender algorithm to combat packet reordering that may occur due to vertical handoffs from a slow to fast access link [11].

Novel Network Coding based retransmission methods considering packet reordering delay have been proposed for wireless LANs [50]. QoS negotiation schemes have been proposed for bandwidth aggregation schemes for real time video streaming in next generation networks with focus on minimizing reordering delay and associated packet loss rate [14]. New algorithms such as Uniform Fine-grain Frame Spreading algorithm have been proposed to avoid packet reordering throughout the load balancing switches [12]

All studies and research seem to indicate that packet reordering is a grave concern and needs to be addressed at different levels or layers.

2.4 Measuring packet reordering

To measure packet reordering, the following metrics are available – Type-P-Reordered-Ratio-Stream, Type-P-Packet-Reordering-Extent-Stream, Type-P-Packet-Late-Time-Stream, reordered packet ratio, reordering extent, reordering late time offset, reordering byte offset, reordering free-runs [35], percentage of reordered packets, Reorder Density (RD), Reorder Buffer Occupancy Density (RBD), reorder extent, n-reordering,

Since Reorder Density (RD) and Reorder Buffer Occupancy Density (RBD) have superior attributes compared to other metrics such as simplicity, low sensitivity to packet loss and duplication, low evaluation complexity, robustness and broad range of applications [25], packet reordering analysis in this research is done based on these metrics only.

2.4.1 Reorder Density (RD)

RD is defined as the distribution of displacements of packets from their original positions, normalized with respect to the number of packets [3]. Packets that are in order would have zero displacement. Packets that are early would have negative displacement and late packets would have positive displacements.

2.4.2 Reorder Buffer Occupancy Density (RBD)

RBD is the normalized histogram of the occupancy of a hypothetical buffer that would allow the recovery from out-of-order delivery of packets [25]. If the packet that just arrived is early, it is buffered until it can be released in order. The occupancy of this hypothetical buffer, after the arrival of every packet is used as a measure of reordering.

2.4.3 Terms used to define RD and RBD

The following terms are used to formally define RD and RBD –

i) Receive Index (RI):

Consider a packet sequence (1, 2, .. , N) transmitted over a network. The receive index, RI is a value assigned to a packet as and when it arrives at its destination according to the order of the arrival such as (1, 2, …). In the absence of reordering, the sequence number of the packet and the receive index are the same for every packet. Earliness or lateness of packets can be computed using RI and sequence number.

ii) Out-of-order packet:

When the RI of a packet is not equal to the sequence number of the packet, such a packet is considered to be an out-of-order packet.

iii) Displacement (D):

It is defined as the difference between the RI of the packet and the sequence number of the packet i.e., RI[i] – i. Therefore, an early packet would have a negative displacement and a late packet would have a positive displacement.

iv) Displacement Threshold (DT):

It is a threshold on the displacement of packets that allows the metric to classify a packet as lost or duplicate.

v) Displacement Frequency (FD):

The displacement frequency FD [k] is the number of arrived packets with a displacement of k, where k's range lie between +DT and -DT

vi) Reorder Density (RD):

In relative terms, RD is defined as the distribution of the displacement frequencies FD [k], normalized with respect to N where N is the length of the received sequence

vii) Expected Packet (E):

A packet with sequence number 'p' is considered to be an expected packet if 'p' is the largest sequence number such that all packets with sequence numbers less than 'p' have already arrived

viii) Buffer Occupancy (B):

An arrived packet with a sequence number greater than that of the expected packet is considered to be stored in a hypothetical buffer sufficiently long to permit recovery from reordering. At a given instant of packet arrival, buffer occupancy is equal to the number of out-of-order packets in the buffer including the newly arrived one.

ix) Buffer-Occupancy Threshold (BT):

It is a threshold on the maximum size of the hypothetical buffer that is used for recovery from reordering.

x) Buffer-Occupancy Frequency (FB):

The buffer occupancy frequency, FB [k] is the number of arrival instances after which the buffer occupancy takes the value of k.

xi) Reorder Buffer-Occupancy Density (RBD):

In relative terms, RBD is the buffer occupancy frequencies normalized by the total number of non – duplicate packets.

2.4.4 Illustration to explain Reorder Density

Consider a sequence (1, 2, 3, 4, 5, 6) sent from a source. If one of the packets in the sequence gets reordered, then the resultant sequence could be (1, 3, 4, 5, 2, 6) at the destination. The tables 2.1 and 2.2 illustrate the computation of RD for an arbitrary sequence that does not have duplicates or losses.

Table 2.1. Computation of Displacement Frequency for an arbitrary sequence

Arrived Sequence	1	3	4	5	2	6
RI	1	2	3	4	5	6
D	0	-1	-1	-1	3	0
FD [D]	1	1	2	3	1	2

Table 2.2. Computation of Reorder Density (RD)

D	-1	0	3
FD [D]	3	2	1
RD [D]	0.5	0.333	0.167

From the aforementioned tables, it can be observed that the percentage of packets that are early by 1 arrival instance is 50%, packets that are in order is 33.33% and 16.7% packets are late by 3 arrival instances.

2.4.5 Illustration to explain Reorder Buffer Occupancy Density

Consider the same example that was discussed in Section 2.4.4 wherein at the destination, the packets arrive in the order (1, 3, 4, 5, 2, 6). The tables 2.3 and 2.4 illustrate the computation of RBD for an arbitrary sequence that does not have duplicates or losses.

Table 2.3. Computation of Buffer-Occupancy Frequency for an arbitrary sequence

Arrived Sequence	1	3	4	5	2	6
E	1	2	2	2	2	6
B	0	1	2	3	4	0
FB [B]	1	1	1	1	1	2

Table 2.4. Computation of Reorder Buffer Occupancy Density (RBD)

B	0	1	2	3	4
FB [B]	2	1	1	1	1
RBD [B]	0.333	0.167	0.167	0.167	0.167

From the aforementioned tables it can be observed that the hypothetical buffer had 0 packets during 33.33% of instances; 1 packet, 2 packets and 3 packets during 16.7% of instances respectively.

2.5 Packet reordering patterns

To effectively understand the different reordering patterns that exist across the Internet, it is essential to understand the idea of primary (or p-event) and secondary (or s-event) events [41].

Consider a sequence (1, 3, 4, 2, 5, 8, 6, 7), where packet 2 is late by two positions and packet 8 is early by two positions. Due to the lateness of packet 2, packets 3 and 4 are early by one position each, and similarly because of the earliness of packet 8, packets 6 and 7 are late by one position each. If ***dm*** is the displacement, then for ***dm* > *0***, the next ***dm*** packets are early and for ***dm* < *0***, the previous ***dm*** packets would be late. In this case, the packet with sequence number

m is said to be associated with the primary event (p-event) while the affected ***dm*** packets have undergone secondary events (s-events).

Generally, the effects of reordering are localized. Packet sequences can be partitioned into **reordered segments** (RS) such that the reordering is localized within RS. The reordering events due to packets within RS are not propagated beyond it i.e., if a p-event is part of the RS, then the corresponding s-events will also have to be part of the same RS.

In the sequence (1, 3, 4, 2, 5, 8, 6, 7), the possible RS' are (1, 3, 4, 2) and (5, 8, 6, 7). A RS that cannot be partitioned further is defined as **minimal reordered segment** (MRS) [41].

Based on the number of p-events within a MRS, reordering events may be classified into

i) Independent Reordering (IR)

A MRS containing a single p-event is called an Independent Reordering event. Refer Figure 2-1.

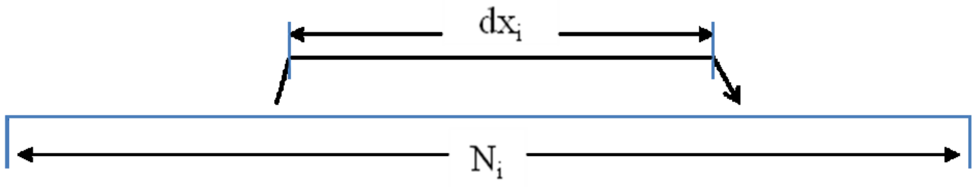


Figure 2-1. Independent Reordering Event

ii) Embedded Reordering (ER)

A MRS containing 2 p-events wherein one p-event is embedded within the other p-event is called Embedded Reordering event. Refer Figure 2-2

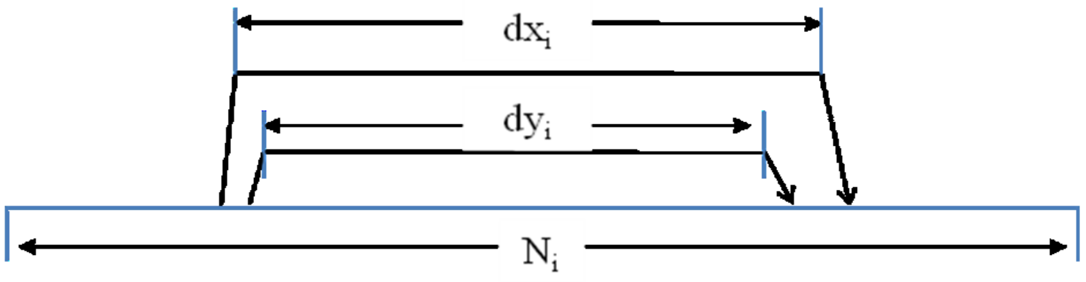


Figure 2-2.Embedded Reordering Event

iii) Overlapped Reordering (OR)

A MRS containing 2 p-events wherein one p-event overlaps with the other p-event is called an Overlapped Reordering event. Refer Figure 2-3

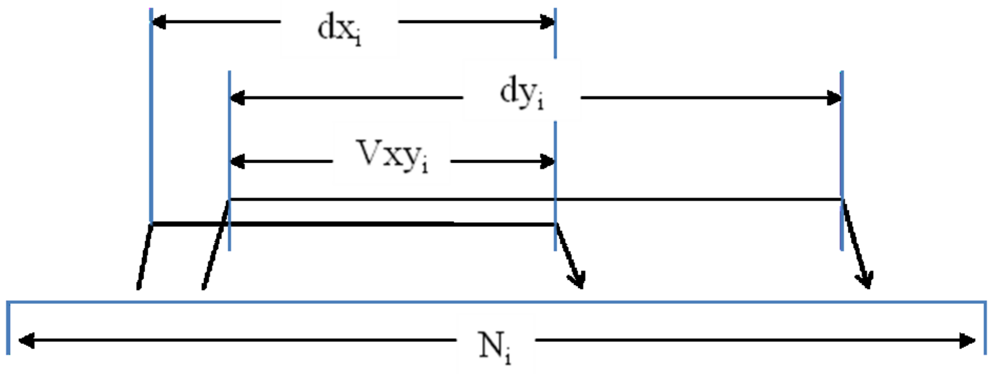
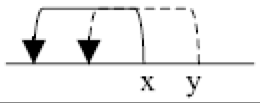
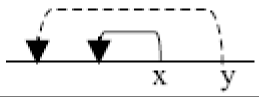
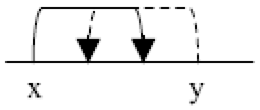
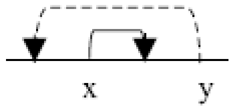
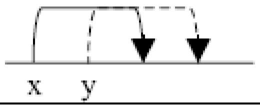
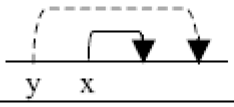
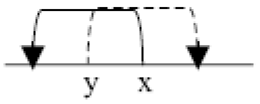
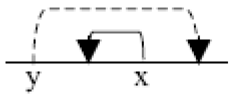


Figure 2-3. Overlapped Reordering Event

Since every p-event can either be late or early, 8 different combinations within overlapped and embedded patterns are possible as shown in Table 2.5.

Table 2.5. Different p-events in basic overlapped and embedded reordering [41]

Overlapped	Embedded
Earliness overlaps with earliness: $d_x < 0,\ d_y < 0$ 	Earliness encloses earliness: $d_x < 0,\ d_y < 0$ 
Earliness overlaps with lateness: $d_x > 0,\ d_y < 0$ 	Earliness encloses lateness: $d_x > 0,\ d_y < 0$ 
Lateness overlaps with lateness: $d_x > 0,\ d_y > 0$ 	Lateness encloses lateness: $d_x > 0,\ d_y > 0$ 
Lateness overlaps with earliness: $d_x < 0,\ d_y > 0$ 	Lateness encloses earliness: $d_x < 0,\ d_y > 0$ 

CHAPTER 3. IMPACT OF DISTRIBUTION OF RESEQUENCING BUFFERS ON PACKET REORDERING

Packet reordering and the motivation to study the problem have already been explained in Chapters 1 and 2. This chapter focuses on one of the effective ways of mitigating packet reordering. The analysis presented in this chapter uses RBD and RD to measure packet reordering.

While some applications are capable of operating with reordered packets, many other applications require packets to be put back in order, i.e., resequenced. When TCP is used as the communication protocol, the resequencing happens in the transport layer, but many real-time multimedia type applications handle it at the application level. Out-of-order packets often significantly degrade the performance of applications even when TCP or application level resequencing is used [16][57]**.** As a result, many modern router architectures include extra hardware to minimize the induced reordering. Examples of such hardware include input buffers to ensure that packets from the same flow are forwarded to the same processing element [55] or output buffers to ensure that packets from a flow are released from the router in more or less the same order in which they arrive at the router [19]. These two approaches that involve tracking packet flows can mitigate the packet reordering introduced within the routers to a certain extent. However, they have no effect on packet reordering due to other causes such as route fluttering [38]. Furthermore, the two approaches are not scalable as the number of flows increase, and also add to the end-to-end latency [5]. The phenomenon of packet reordering and its impact on applications have been evaluated in detail in [54]. Measurement and quantification of packet reordering is addressed in [25] and the techniques and metrics for long-term monitoring of packet reordering are addressed in [60].

This chapter focuses on the resources required for recovery from packet reordering. Due to the increased parallelism in modern networks and the demands of high performance applications, recovery from packet reordering is known to consume increasing resources both at end nodes and routers [3]. Regardless of whether the recovery is done at end - nodes (application level buffering and TCP level resequencing), or at intermediate routers (with input or output buffering), the recovery inevitably requires buffers. Thus adding input/output buffering to a router may be viewed as transferring some of the recovery resources to intermediate nodes. From a buffer allocation standpoint, this is one of the first few researches ([28] also proposes in-network buffers) to provide formal results analyzing the impact of resequencing buffer distribution on packet reordering. Analytical results are derived quantifying the impact on packet reordering as they flow through resequencing nodes containing a limited number of buffers. The results are verified using simulations. The approach and results are useful for buffer allocation at individual router nodes to keep packet reordering within acceptable levels.

With the current scaling trends in network link speeds and processor speeds, packet reordering will become an increasingly difficult problem to deal with [14][41]. It may even be considered as an important Quality of Service (QoS) parameter for network related services. This chapter lays the theoretical foundation for allocating recovery buffer resources in a distributed network (refer Figure 3-1).

An integral part of such a buffer distribution mechanism with resequencing capability is the idea of resequencing algorithm and its design. The resequencing algorithm used for the research is discussed initially followed by the impact of such a distribution mechanism on packet reordering measured using Reorder Density (RD) and Reorder Buffer Occupancy Density (RBD).

In this chapter, Section 3.1 deals with the resequencing algorithm, Section 3.2 deals with the impact of resequencing buffers on Reorder Density and Section 3.3 with the impact of resequencing buffers on Reorder Buffer Occupancy Density. Verification results are presented in Section 3.4 followed by Conclusion in Section 3.5.

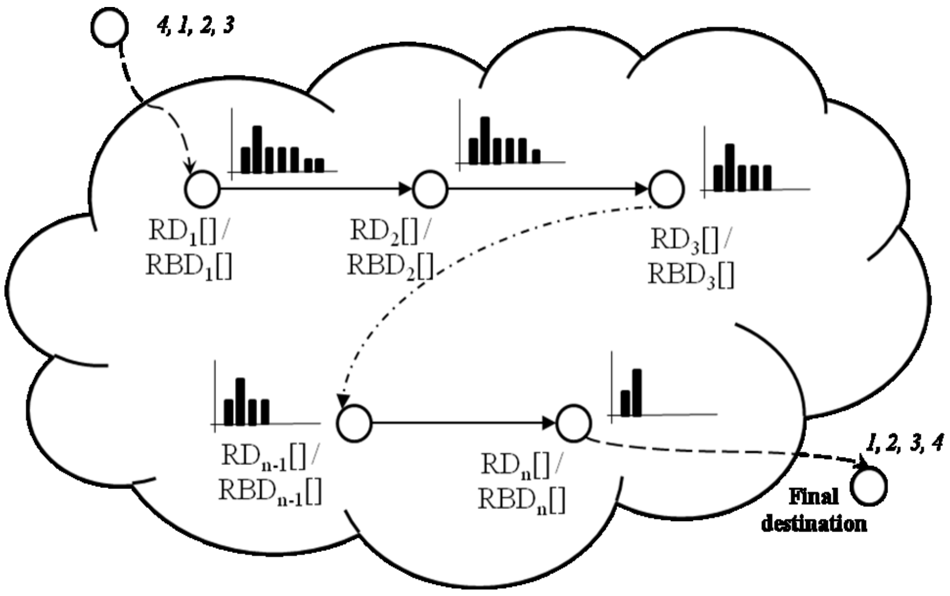


Figure 3-1. High-level description of the problem

3. 1 Resequencing algorithm

A "resequencing node" can be referred to as "a node that attempts to put the out of order packets back in order, i.e., resequence them." If there is sufficient number of buffers available, it is possible to put the entire sequence of packets back in order. However, if the number of buffers available in a single node is insufficient, packets can still be resequenced thereby reducing the amount and degree of reordering. The most efficient approach for resequencing is based on the concept of releasing lowest numbered packet first as shown in Figure 3-2.

Without loss of generality, assume that the packets are numbered 1, 2, and 3 and so on in a sequential manner. The parameter *expected_num* keeps track of next expected in-sequence packet. As and when a packet arrives, its packet number is compared with the *expected_num*. On receipt of a packet with the same packet number, *expected_num* is incremented by 1 and the packet is released. Furthermore, any packets in the buffer corresponding to the new value of *expected_num* are released while updating *expected_num*. If the sequence number of the packet that arrived does not correspond to *expected_num*, it is inserted into the buffer. If there is no room in the buffer, the packet with lowest sequence number (amongst those in the buffer and the packet that just arrived) is released while updating *expected_num*. Figure 3-3 explains the working of the lowest first algorithm when two resequencing buffers are used.

```
START
Initialize expected_num  to 1
While (packets arrive at the node)
{
        If (expected_num equals current_packet_num)
        {
                Release packet into output queue
                Increment expected_num by 1
                While (expected_num in resequencing buffer)
                {
                        Release that packet into output queue
                        Increment expected_num by 1
                }
        }
        Else if (resequencing buffer is not full)
        {
                Store packet in resequencing buffer
        }
        Else // resequencing buffer is full
        {
                Select packet with lowest sequence number amongst all
                packets in resequencing buffer

                If(selected_packet_num less than current_packet_num)
                {
                        Release selected packet into the output queue
                        Store current packet in the buffer
                }
                Else
                {
                        Release current packet into the output queue
                }
        }
}
END
```

Figure 3-2. Lowest First Resequencing Algorithm

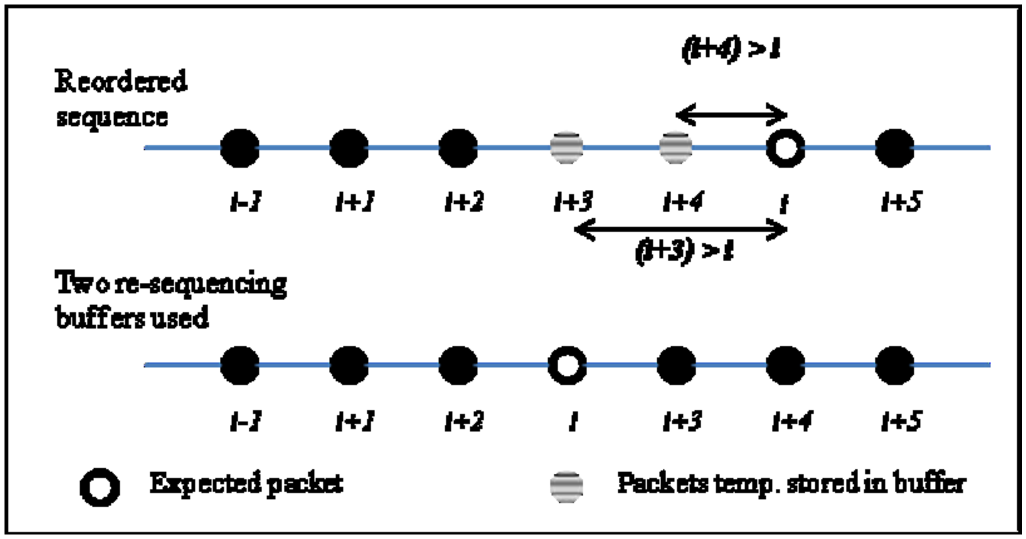


Figure 3-3. Resequencing algorithm explained when two buffers are available

3. 2 Impact of Resequencing Buffers on Reorder Density

In this and the following sections, the impact of resequencing buffers on packet reordering as a stream of packets flow through resequencing nodes is analyzed. As already mentioned in Chapter 2, the degree and nature of packet reordering can be characterized by RD and RBD. Therefore, the objective is to derive the variation of RD and RBD as packets flow through resequencing nodes as illustrated in Figure 3-1. Some commonly used notations in this chapter are summarized in Table 3.1.

In this section, the study and analysis of the impact of resequencing buffers on Reorder Density is provided.

3. 2. 1 Concatenation property of Reorder Density

Based on the definition of reorder density, the following can be inferred:

The non – zero value of RD [X] when $X < 0$ represents packets that are early by X position(s). Similarly, non – zero values of RD [X] when $X > 0$ represents packets that are late by X position(s). RD [X] = 0 corresponds to packets that are in order (neither late nor early).

Table 3.1. Common notations used in this chapter

Symbol	**Description**
b	Capacity of resequencing buffers expressed in terms of number of packets
N	Total number of packets in the sequence (Refer Figure 3-4)
dx_n	**For Independent Reordering** Represents the displacement of the late packet in the ***n***th reordering pattern in a given sequence **For Embedded Reordering** Represents the displacement of the

	• inner late packet when lateness embeds lateness • outer early packet when earliness embeds lateness • inner early packet when lateness embeds earliness • inner early packet when earliness embeds earliness in the nth reordering pattern **For Overlapped Reordering** Represents the displacement of one of the late packets when lateness overlaps lateness
dy_n	**For Embedded Reordering** Represents the displacement of the • outer late packet when lateness embeds lateness • inner late packet when earliness embeds lateness • outer late packet when lateness embeds earliness • outer early packet when earliness embeds earliness in the nth reordering pattern **For Overlapped Reordering** Represents the displacement of one of the late packets when lateness overlaps lateness
RD' [i]	The output value of reorder density of a sequence at a resequencing node
N	Numbers of instances of a given reordering pattern
Vxy_n	Length of overlap between two overlapping packets in the ***n***th Overlapped Reordering pattern (Refer Figure 3-4)
RD [$\hat{a}_{i}$i, N_i]	Reorder Density of a IR, ER or OR sub-sequence
N_i	Number of packets in the ***i***th sub-sequence
S_{RD}	The set of indices of reorder density of the sequence whose values are non – zero
S_{RBD}	The set of indices of reorder buffer occupancy density of the sequence whose values are non – zero
i	Input displacement of the packets in a given sequence
p	$p \rightarrow f(i, b)$ p is the correction factor that is applied to the displacement, i,

	in the sequence by a system of b resequencing buffers
j, k, m	Addition, Subtraction or Multiplicative factors used in the theorems

Consider a sequence of packets containing multiple reordering patterns. We partition the sequence into multiple sub-sequences of length N_i ($i = 1, 2\ldots$), where each sub-sequence contains no more than one reordering pattern. The reordering pattern in the *i*th segment is characterized by $\hat{\boldsymbol{a}}_{i}$i, where $\hat{a}_i$ represents $\{dx_i\}$ for an independent reordering pattern, $\{dx_i, dy_i\}$ for an embedded reordering pattern and $\{dx_i, dy_i, Vxy_i\}$ for an overlapped reordering pattern. N_i is the number of packets in a single sub-sequence. When a number of such patterns form a large sequence and when every pattern (and its surrounding packets) is considered as a sub-sequence, their Reorder Density is represented as RD [$\hat{\boldsymbol{a}}_{i}$1, N_i]

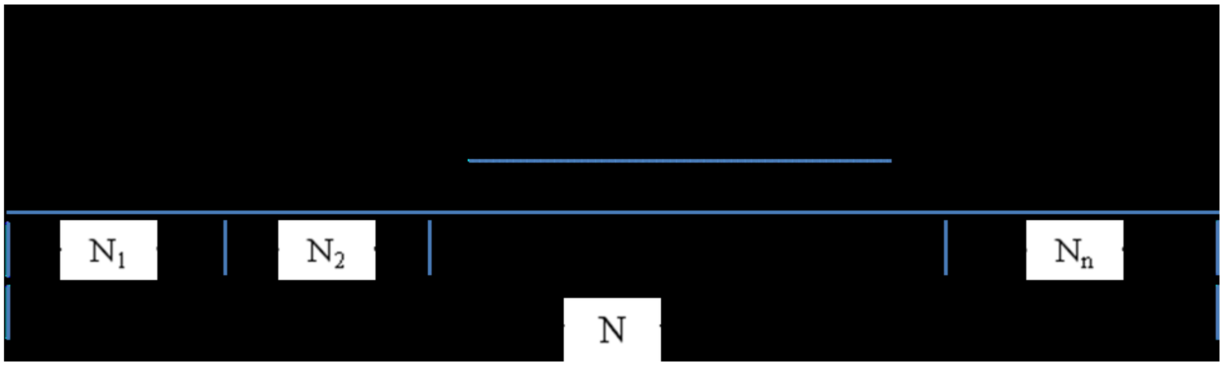


Figure 3-4. A sequence containing more than one reordering pattern

Theorem 1

The reorder density of a sequence containing ***n*** reorder patterns is given by

$$RD_N = \left(\sum_{i=1}^{n} \mathbf{RD}\ [\hat{\boldsymbol{a}}_i, N_i] * N_i \right) / N \tag{1}$$

Proof

RD of a given subsequence is the normalized histogram of the displacement values of the packets in a given sequence. As shown in Figure 3-4, when a number of such independent sequences combine to form a larger sequence, the packets with same displacement values cause RD to add up. Hence every RD value in a given sub-sequence is de-normalized with the number of packets in that sub-sequence (N_i), their total sum computed and normalized with N where N can also be represented as $\sum_{i=1}^{n}(N_i)$.

Q. E. D

3.2.2 Reorder Density of Independent Reordering Patterns

An Independent Reordering occurs when a single packet is either late or early. Consider the sequence (*i-1, i, i+1, i+2, i+3, i+4, i+5*). If *packet i* is late by 4 positions, then the resulting sequence would be (*i-1, i+1, i+2, i+3, i+4, i, i+5*) as shown in Figure 3-5. On using the lowest first resequencing algorithm with a single resequencing buffer, the resultant new sequence would be (*i-1, i+1, i+2, i+3, i, i+4, i+5*).

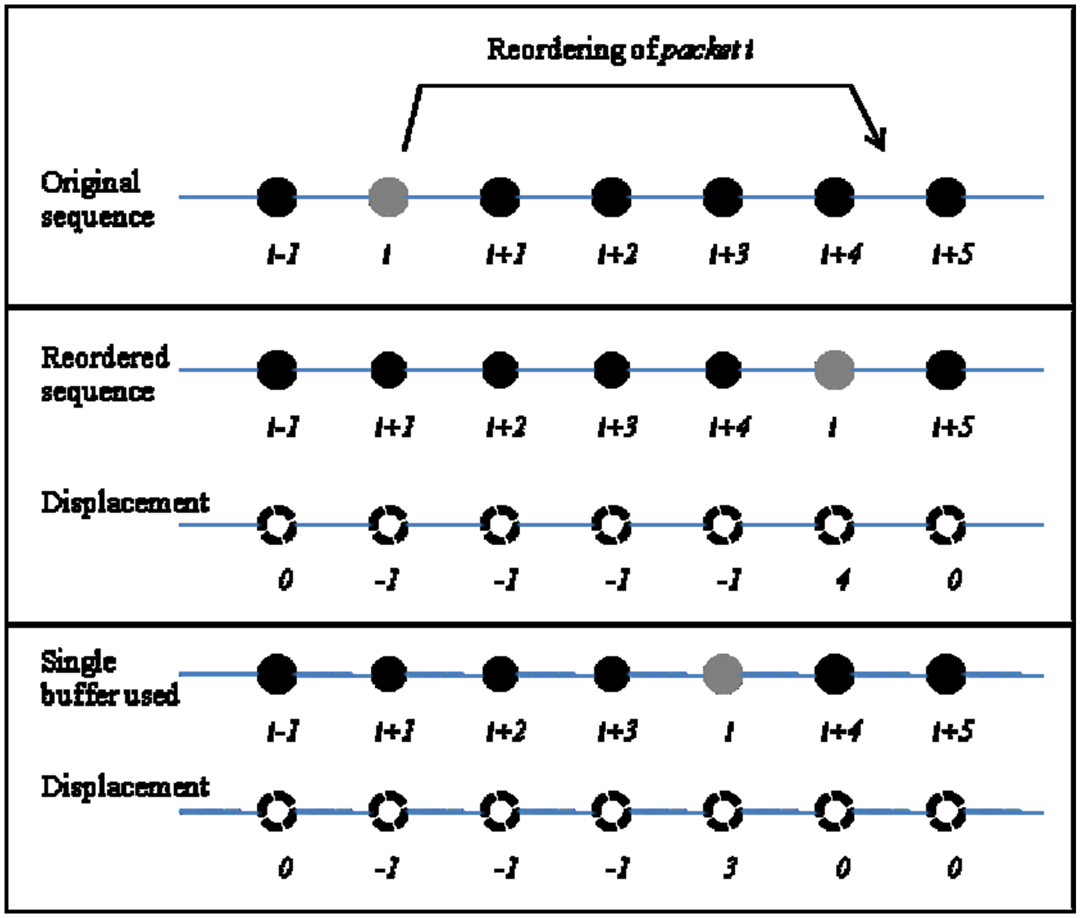


Figure 3-5. Change in displacement values of a sequence containing an independent reordering pattern

After the usage of the resequencing algorithm with a single buffer, there is an improvement in orderliness of sequence resulting in change of the displacement values. The task on hand is to understand the displacement pattern and hence estimate reorder density (RD) when an arbitrary number of such resequencing buffers ‘b’ are used. RD of a pattern before using the lowest first resequencing algorithm is denoted by **RD [i].** The RD of the resultant sequence after applying the resequencing algorithm with ‘b’ buffers is denoted by **RD’ [i].**

Without loss of generality, let us assume that if the set $\boldsymbol{S_{dx}}$ represents the set of all packets that are displaced by ***d*x** positions, then by definition of RD,

$RD[dx] = |S_{dx}| / N$

(2)

Theorem 2

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of late independent reordering pattern that has traversed through a resequencing node with ***b*** buffers is given by

$RD'[i] = j$ **for $b \geq dx$**

$\forall\ i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(3)

$RD'[i - p] = RD[i] + j\,(b / N)$ **for $b < dx$**

$\forall\ i \in S_{RD}$,

where $j = +1$ and $p = 0$ when $i = 0$

$j = -1$ and $p = 0$ when $i = -1$

$j = 0$ and $p = b$ when $i = dx$

(4)

<u>Proof</u>

It is known that, in a sequence, if a packet ***p*** is late by ***dx*** positions, then ***dx*** packets would be early by 1 position as part of secondary events of the late packet.

Case 1: $b \geq dx$

When the available buffer is at least equal to the displacement of the late packet, the entire sequence is back in order.

Therefore, the number of packets with 0-displacement would be

$|S_0'| = N$

By definition,

$RD'[0] = |S_0'| / N$

$= N / N$

=> **$RD'[0] = 1$**

Case 2: $b < dx$

On expanding the equation **(4)**,

a) $RD'[0] = RD[0] + (b / N)$

b) $RD'[-1] = RD[-1] - (b / N)$

c) $RD'[dx - b] = RD[dx]$

When a reordered sequence is traversed by a resequencing node containing ***b*** buffers, the resultant displacement of the late packet ***p*** will be no greater than ***dx - b***. As a consequence, ***dx - b*** packets will be early by 1 position and ***b*** packets will be back in order.

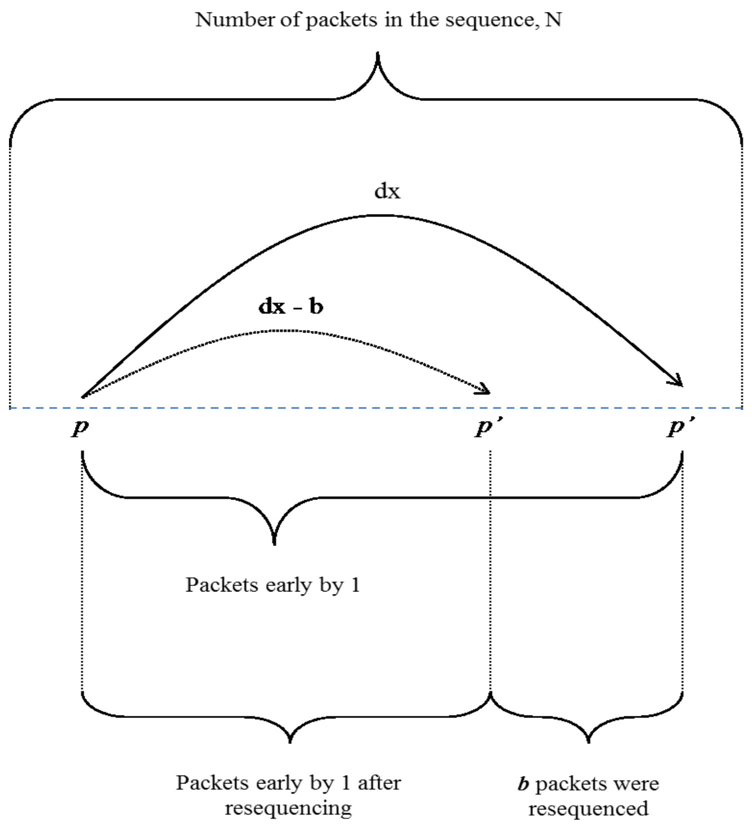

Number of packets in the sequence, N
dx
dx - b
p
p'
p'
Packets early by 1
Packets early by 1 after resequencing
b packets were resequenced

Figure 3-6. Displacement of the late packet 'p' after resequencing

Therefore,

$$|S_0'| = |S_0| + b \qquad \text{(i)}$$

$$|S_{-1}'| = |S_{-1}| - b \qquad \text{(ii)}$$

By definition,

$$RD'[0] = |S_0'| / N$$

$$= (|S_0| + b) / N \qquad \text{from (i)}$$

$$= (|S_0| / N) + (b/N)$$

=> $\mathbf{RD'[0] = RD[0] + (b/N)}$

Similarly,

$$RD'[-1] = |S_{-1}'| / N$$

$= (|S_{-1}| - b) / N$ from (ii)

$= (|S_{-1}| / N) - (b/N)$

=> $\mathbf{RD'[-1] = RD[-1] - (b/N)}$

Also, the displacement of the late packet would now be ***dx – b***, therefore

$dx' = dx - b$

Since the displacement of the late packet has changed,

$|S_{dx'}| = |S_{dx-b}|$

Dividing by N

$|S_{dx'}| / N = |S_{dx-b}| / N$

By definition of RD and from **(2),**

$\mathbf{RD[dx'] = RD[dx - b]}$

Q. E. D

Theorem 3

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of an early independent reordering pattern that has traversed through a resequencing node with ***b*** buffers is given by

$RD'[i] = j$ for $b \geq 1$

$\forall\ i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(5)

Proof

The lowest first resequencing algorithm is such that the packet whose sequence number doesn't match the expected sequence number is buffered. While traversing through a sequence containing a single early packet, the algorithm, on encountering an early packet buffers it until its actual position is reached when the packet gets released. This means that a single buffer is enough to put sequences with a single early packet back into order.

Therefore,

$|S_0'| = N$

By definition,

$RD'[0] = |S_0'| / N$

$= N / N$

=> **$RD'[0] = 1$**

Q. E. D

3. 2. 3 Reorder Density of Embedded Reordering Patterns

Embedded Reordering occurs when a single early or late packet event gets embedded within another late or early packet event. Consider the sequence (*i-3, i-2, i-1, i, i+1, i+2, i+3, i+4, i+5*) as shown in the Figure 3-7. If packets *i* and *i-2* are late by 2 and 6 positions respectively in an embedded fashion, then the resulting sequence would be (*i-3, i-1, i+1, i+2, i, i+3, i+4, i-2, i+5*). If the lowest first resequencing algorithm with a single buffer is used, the resultant sequence would be (*i-3, i-1, i+1, i, i+2, i+3, i-2, i+4, i+5*) as shown in Figure 3-7.

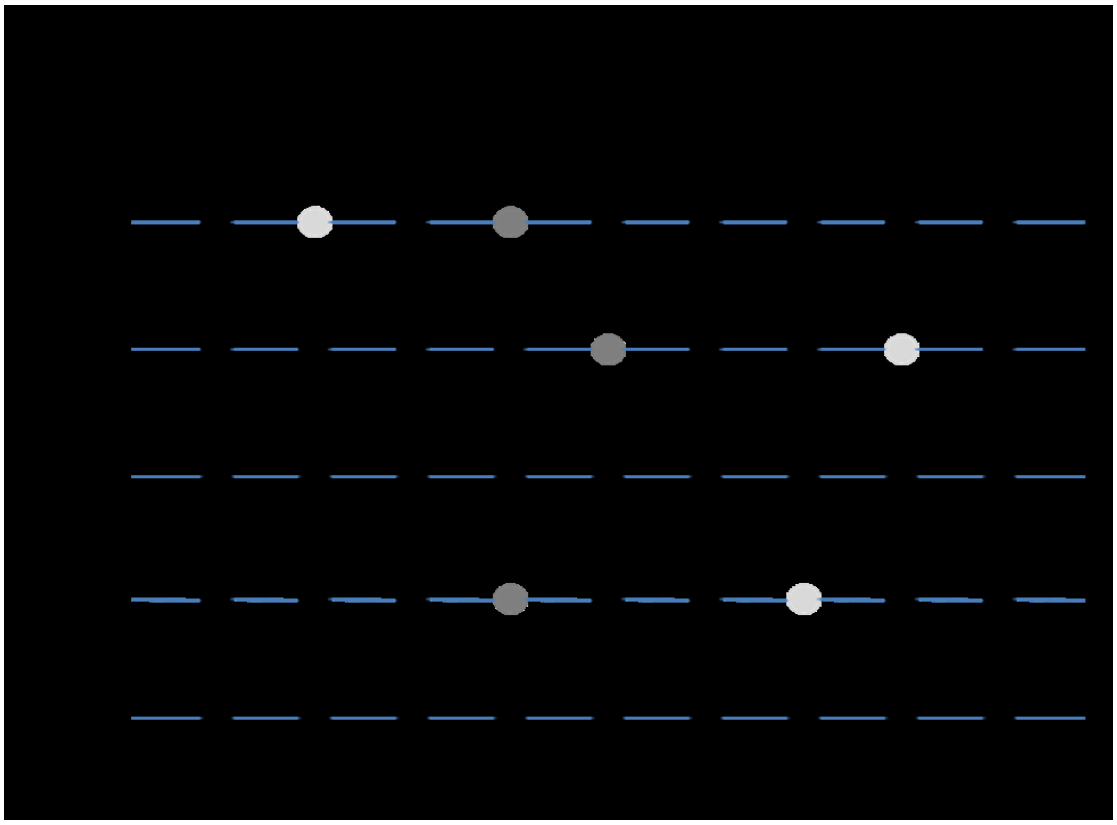


Figure 3-7. Change in displacement values of a sequence containing an embedded reordering pattern

Theorem 4

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of embedded reordering pattern formed by a late event whose displacement is ***dx*** embedded within another late event whose displacement is ***dy*** after it has traversed through a resequencing node with ***b*** resequencing buffers is given by

RD’ [i] = j **for b ≥ dy**

$\forall\, i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(6)

$RD'[i-p] = RD[i] + j(b/N)$ **for $b < dx$**

$\forall\, i \in S_{RD}$

where $j = +1$ and $p = 0$ when $i = 0$

$j = 0$ and $p = 0$ when $i = -1$

$j = -1$ and $p = 0$ when $i = -2$

$j = 0$ and $p = b$ when $i > 1$

(7)

$RD'[i-p] = RD[i] + j(b/N) + (k/N)$ **for $b = dx$**

$\forall\, i \in S_{RD}$,

where $j = +1$, $k = +1$ and $p = 0$ when $i = 0$

$j = 0$, $k = 0$ and $p = 0$ when $i = -1$

$j = -1$, $k = 0$ and $p = 0$ when $i = -2$

$j = 0$, $k = 0$ and $p = b$ when $i > b$

(8)

$RD'[i-p] = RD[i] + j(b/N) + (k/N)$ **for $b = dx + 1$**

$\forall\, i \in S_{RD}$,

where $j = +1$, $k = 0$ and $p = 0$ when $i = 0$

$j = 0$, $k = 1$and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = b$ when $i > b$

(9)

$RD'[i-p] = RD[i] + j(b/N) + k/N$ **for $b > (dx + 1)$ and $b < dy$**

$\forall\, i \in S_{RD}$,

where $j = +1$, $k = 0$ and $p = 0$ when $i = 0$

$j = -1$, $k = 1$ and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = b$ when $i > b$

(10)

Proof

Case 1: $b \geq dy$

When the available buffer is equal to the greater displacement amongst the late packets, the entire sequence is back in order.

Therefore,

$$|S_0'| = N \qquad \text{(i)}$$

By definition,

$$RD'[0] = |S_0'| / N$$

$$= N / N$$

=> **$RD'[0] = 1$**

Case 2: $b < dx$

On expanding the equation (7),

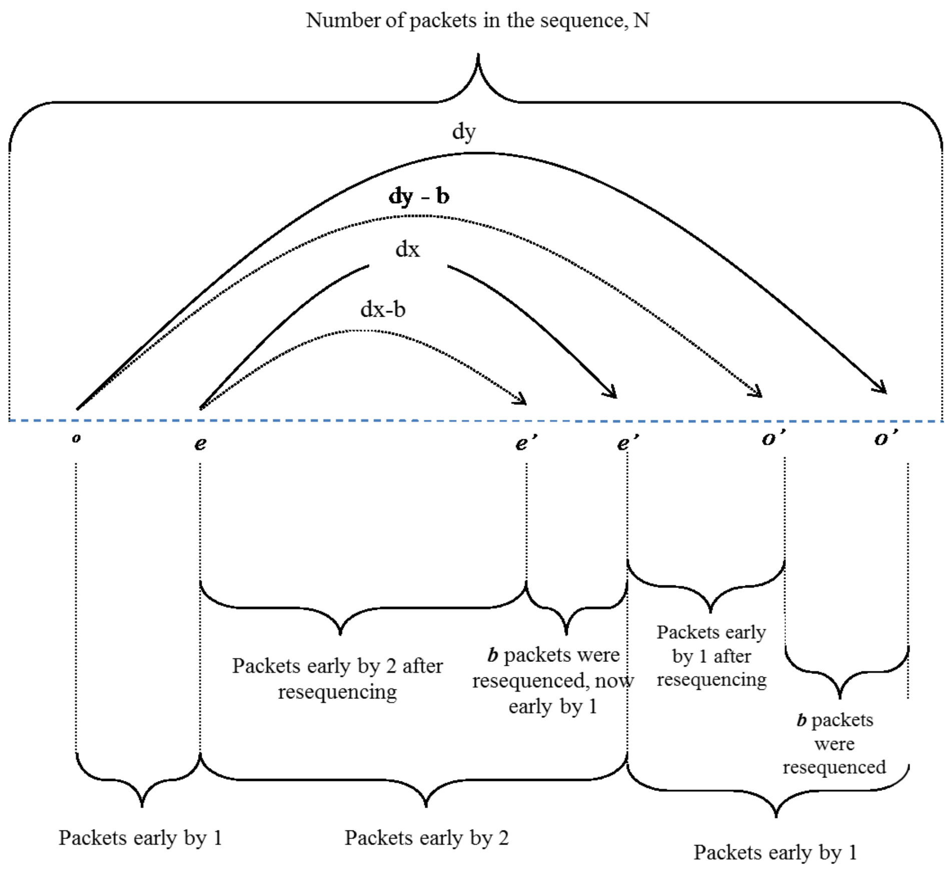
a) $RD'[0] = RD[0] + (b / N)$

b) $RD'[-1] = RD[-1]$

c) $RD'[-2] = RD[-2] - (b / N)$

d) $RD'[dx] = RD[dx - b]$

e) $RD'[dy] = RD[dy - b]$



Number of packets in the sequence, N
dy
dy - b
dx
dx-b
o
e
e'
e'
o'
o'
Packets early by 2 after resequencing
b packets were resequenced, now early by 1
Packets early by 1 after resequencing
b packets were resequenced
Packets early by 1
Packets early by 2
Packets early by 1

Figure 3-8. Displacement of the late packets after resequencing

When a reordered sequence is traversed by a resequencing node containing ***b*** buffers, the resultant displacement of the late packet ***p*** will be no greater than ***dx - b***. As a consequence, ***dx - b*** packets will be early by 1 position and ***b*** packets will be late by 1 position.

From [41] and Chapter 2, it is known that the ***dx – b*** packets prior to the embedded packet are early by 2 positions. Therefore, when ***dx – b*** packets are late by 1 position, it results in the reduction in the number of packets that are early by 2 positions

i.e.,

$$|S_{-2}'| = |S_{-2}| - b \qquad \text{(ii)}$$

Dividing by N

$|S_{-2}'| / N \quad = (|S_{-2}| / N) – (b / N)$

By definition of RD and from **(2),**

$RD' [-2] \quad = RD [-2] - (b / N)$

But, the resequenced packets would now be part of the secondary event of the outer late packet. Therefore, they would now be early by 1 position

i.e.,

$|S_{-1}'| \quad = |S_{-1}| + b$ (iii)

However, the outer late packet also gets resequenced resulting in ***dy – b*** packets coming late by 1 position and ***b*** packets coming late by 1 position. Therefore, the number of packets that are early by 1 position would now be

$|S_{-1}'| \quad = |S_{-1}| + b – b$

$|S_{-1}'| \quad = |S_{-1}|$ (iv)

This also means that these packets would now be back in order resulting in

$|S_0'| \quad = |S_0| + b$ (v)

Dividing (iv) and (v) by N

$|S_{-1}'| / N \quad = |S_{-1}| / N$

$|S_0'| / N \quad = (|S_0| / N) + (b / N)$

By definition of RD and from **(2),**

$RD' [-1] \quad = RD [-1]$

$RD' [0]= RD [0] + (b / N)$

Since the displacement of both late packets would be no greater than ***dx – b*** and ***dy – b***, we have

$|S_{dx'}| \quad = |S_{dx – b}|$ (vi)

$|S_{dy'}| = |S_{dy-b}|$ (vii)

Dividing (vi) and (vii) by N

$|S_{dx'}| / N = |S_{dx-b}| / N$

$|S_{dy'}| / N = |S_{dy-b}| / N$

By definition of RD and from **(2),**

RD [dx'] = RD [dx – b]

RD [dy'] = RD [dy – b]

Case 3: b = dx

On expanding the equation (8), we get

a) RD' [0] = RD [0] + (b / N) + (1/N)

b) RD' [-1] = RD [-1]

c) RD' [-2] = RD [-2] - (b / N)

d) RD' [dy] = RD [dy - b]

The proofs for (b), (c) and (d) are similar to the ones in Case 2, please refer to **Case 2:** (b), (c) and (e) respectively for their proofs.

When the number of buffers available is exactly equal to ***dx***, the embedded late packet gets resequenced completely. Therefore, there is an addition to the number of packets that are back in order

i.e.,

$|S_0'| = |S_0| + b + 1$ (viii)

Dividing (viii) by N,

$|S_0'| / N = |S_0| / N + b / N + 1/ N$

By definition of RD and from **(2),**

RD’ [0]= RD [0] + (b / N) + (1/N)

Case 4: b = dx + 1

On expanding the equation (9), we get

a) RD’ [0]= RD [0] + (b / N)

b) RD’ [-1] = RD [-1] + (1/N)

c) RD’ [dy] = RD [dy - b]

When the number of buffers available is one more than the displacement of the embedded late packet, the embedded late packet, that is completely resequenced, becomes part of the secondary event of the outer late packet. Therefore an addition to the number of packets that is early by 1 position

i.e.,

$$|S_{-1}'| = |S_{-1}| + 1 \qquad \text{(ix)}$$

Dividing (ix) by N,

$$|S_{-1}'| / N = |S_{-1}| / N + 1/ N$$

By definition of RD and from **(2),**

RD’ [-1] = RD [-1] + (1/N)

Also, we know that, when a reordered sequence is traversed by a resequencing node containing ***b*** buffers, the resultant displacement of the late packet ***p*** will be no greater than ***dy - b***. As a consequence, ***dy - b*** packets will be early by 1 position and ***b*** packets will be late by 1 position.

When the packets that are early by 1 position become late by 1 position, it results in net displacement of 0, therefore

$|S_0'| = |S_0| + b$ (x)

Dividing (ix) by N,

$|S0'| / N = |S_0| / N + b/ N$

By definition of RD and from **(2),**

RD' [0]= RD [0] + (b / N)

Refer to Case 2: (e) for the proof for (c).

Case 5: b > (dx + 1) and b < dy

On expanding the equation (10), we get

a) RD' [0]= RD [0] + (b / N)
b) RD' [-1] = RD [-1] – (b / N) + (1/N)
c) RD' [dy] = RD [dy - b]

When the available number of buffers is greater than ***dx + 1***, the sequence behaves like an independent reordering pattern.

From Theorem 2, Case 2, we have

RD' [-1] = RD [-1] – (b / N)

By definition of RD and **(2)**, we have

$|S_{-1}'| / N = |S_{-1}| / N - (b / N)$

Multiplying by N,

$|S_{-1}'| = |S_{-1}| - b$ (xi)

The embedded late packet is completely resequenced and it contributes as secondary event packets that are early by 1 position to the outer late packet event.

Therefore, (xi) is now

$|S_{-1}'| = |S_{-1}| - b + 1$

Dividing by N,

$$|S_{-1}'| / N \qquad = |S_{-1}| / N - b / N + 1 / N$$

By definition of RD and **(2)**, we have

$$\mathbf{RD' [-1] \qquad = RD [-1] - (b / N) + (1/N)}$$

Q. E. D

Theorem 5

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of embedded reordering pattern formed by an early event whose displacement is ***dx*** embedded within a late event whose displacement is ***dy*** after it has traversed through a resequencing node with ***b*** resequencing buffers is given by

$RD'[i] = j$ **for $b \geq dy$**

$\forall\ i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(11)

$RD'[i - p] = RD[i] + \{j(abs(dx) - 1) + k\}/N$ **for $b = 1$**

$\forall\ i \in S_{RD}$,

where $j = -1$, $k = +1$ and $p = 0$ when $i = 0$

$j = +1$, $k = 0$ and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = b$ when $i > 1$

(12)

$RD'[i - p] = RD[i] + \{j(abs(dx) - 1) + k - j(b - 1)\}/N$

for $b > 1$ and $b < dy$

$\forall\ i \in S_{RD}$,

where $j = -1$, $k = +1$ and $p = 0$ when $i = 0$

$j = +1$, $k = 0$ and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = b$ when $i > 1$

(13)

Proof

Case 1: $b \geq dy$

When the available buffer is equal to the displacement of the late packet, ***dy***, the entire sequence is back in order.

Therefore,

$|S_0'| = N$ (i)

By definition,

$RD'[0] = |S_0'| / N$

$= N / N$

=> **$RD'[0] = 1$**

Case 2: $b = 1$

By expanding the equation (12), we get

a) $RD'[0] = RD[0] - abs(dx) / N + 2 / N$

b) $RD'[-1] = RD[-1] + (abs(dx) - 1/ N)$

c) $RD'[dy - b] = RD[dy]$

We already know that a single buffer is enough to resequence an early packet completely back in order. Since we have an early event being embedded within a late event, the combination of the secondary events of the embedded early packet and the outer late packet would result in net displacement of zero. This is because one secondary event causes packets to come in early by 1 position and the other would cause it to come in late by 1 position.

Therefore, when the embedded early event is completely resequenced, the number of packets with zero displacement would reduce by a factor of the absolute displacement of the early packet minus one (because of the embedded nature of the reordering) also causing a corresponding increase in the number of packets that would now be early by 1 position

This implies

$|S_0'| = |S_0| - (\text{abs}(dx) - 1)$ (ii)

$|S_{-1}'| = |S_{-1}| + (\text{abs}(dx) - 1)$ (iii)

But, outer late packet also gets resequenced by 1 position resulting in an addition of one packet to zero displacement set

Therefore, by adding 1 to (ii), we get

$|S_0'| = |S_0| - (\text{abs}(dx) - 1) + 1$ (iv)

On simplifying it further we get

$|S_0'| = |S_0| - \text{abs}(dx) + 2$ (v)

Dividing (v) by N,

$|S_0'| / N = |S_0| / N - \text{abs}(dx) / N + 2 / N$

By definition of RD and (2), we therefore have

$RD'[0] = RD[0] - \text{abs}(dx) / N + 2 / N$

Similarly, dividing (iii) by N,

$|S_{-1}'| / N = |S_{-1}| / N + (\text{abs}(dx) - 1) / N$

By definition of RD and (2), we therefore have

$RD'[-1] = RD[-1] + (\text{abs}(dx) - 1/ N)$

For (c), refer to Theorem 2, Case 2: (c)

Case 3: b > 1 and b < dy

By expanding the equation (13), we get

a) RD' [0] = RD [0] – abs (dx) / N + 1 / N + b / N

b) RD' [-1] = RD [-1] + (abs (dx) – b / N)

c) RD' [dy - b] = RD [dy]

When the number of resequencing buffers available is greater than 1 and less than ***dy***, the behavior is similar to that of an independent reordering pattern.

From (iv), we have

$$|S_0'| = |S_0| - (\text{abs}(dx) - 1) + 1$$

Since (- (abs (dx) – 1) + 1) is constant after a single buffer usage, let us assume C1 = - (abs (dx) – 1)

$$|S_0'| = |S_0| + C1 \qquad \text{(vi)}$$

Using the concepts proved in Theorem 2, Case 2: (a), we therefore get

$$|S_0'| = |S_0| + C + (b')$$

Since we have already used a single buffer and are dealing with that using the constant C1,

b' = (b – 1)

Therefore,

$$|S_0'| = |S_0| + C1 + (b - 1) \qquad \text{(vii)}$$

On substituting the constant C1 and simplifying (vii) further, we get

$$|S_0'| = |S_0| + 1 + b \qquad \text{(viii)}$$

Dividing (viii) by N

$$|S_0'| / N = |S_0| / N + 1 / N + b / N$$

By definition of RD and (2), we therefore have

RD' [0] = RD [0] – abs (dx) / N + 1 / N + b / N

From (iii), we have

$|S_{-1}'| = |S_{-1}| + (\text{abs}(dx) - 1)$

Since (abs (dx) – 1) is constant after a single buffer usage, let us assume C2 = (abs (dx) – 1). Therefore, we get

$|S_{-1}'| = |S_{-1}| + C2$

Using the concepts proved in Theorem 2, Case 2: (b), we therefore get

$|S_{-1}'| = |S_{-1}| + C2 - b'$ (ix)

Since we have already used a single buffer and are dealing with that using the constant C2,

$b' = (b - 1)$ (x)

Combining (xi) and (x), we get

$|S_{-1}'| = |S_{-1}| + C2 - (b - 1)$ (xi)

On substituting the constant C2 and simplifying (xi) further, we get

$|S_{-1}'| = |S_{-1}| + \text{abs}(dx) - b$ (xii)

Dividing (xii) by N

$|S_{-1}'| / N = |S_{-1}| / N + (\text{abs}(dx) - b) / N$

By definition of RD and (2), we therefore have

$\mathbf{RD'[-1] = RD[-1] + (abs(dx) - b / N)}$

For (c), refer to Theorem 2, Case 2: (c)

Q. E. D

Theorem 6

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of embedded reordering pattern formed by a late event whose displacement is ***dy*** embedded within an early event whose displacement is ***dx*** after it has traversed through a resequencing node with ***b*** resequencing buffers is given by

$RD'[i] = j$ **for $b \geq dy$**

$\forall\ i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(14)

$RD'[i - p] = k(RD[i]) + \{j(dy - 1) + k(abs(dx) - dy + 1)\}/N$ **for $b = 1$**

$\forall\ i \in S_{RD}$,

where $j = -1$, $k = +1$ and $p = 0$ when $i = 0$

$j = +1$, $k = 0$ and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = b$ when $i > 1$

(15)

$RD'[i - p] = k(RD[i]) + \{j(dy - 1) + k(abs(dx) - dy + 1) - j(b - 1)\}/N$

for $b > 1$ and $b < dy$

$\forall\ i \in S_{RD}$,

where $j = -1$, $k = +1$ and $p = 0$ when $i = 0$

$j = +1$, $k = 0$ and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = b$ when $i > 1$

(16)

<u>Proof</u>

Case 1: $b \geq dy$

When the available buffer is equal to the displacement of the late packet, ***dy***, the entire sequence is back in order.

Therefore,

$|S_0'| = N$ (i)

By the definition of RD and (2), we therefore have

$$\begin{aligned} RD'[0] &= |S_0'| / N \\ &= N / N \end{aligned}$$

=> **RD' [0] = 1**

Case 2: b = 1

By expanding the equation (15), we get

a) $RD'[0] = RD[0] + (2 - 2dl + abs(dx)) / N$
b) $RD'[-1] = (dy - 1) / N$
c) $RD'[dy - b] = RD[dy]$

As discussed in the proof for Theorem 5, packets that are embedded within the outer packet event tend to be affected by the secondary events of both the embedded packet and the outer packet. Therefore, there will be a few packets with zero displacement. When the early packet completely gets resequenced, there would be a decrease in the number of packets whose displacement would be zero by a factor of ***(dy – 1)***. The negative one corresponds to embedded-ness of the late packet.

Therefore,

$|S_0'| = |S_0| - (dy - 1)$ (ii)

But, we also know that there might be some packets that are solely affected by the outer early event only. When the early packet gets resequenced, the associated secondary event packets would also get resequenced causing an increase in the number of packets whose displacement in zero.

Therefore (ii) now becomes

$$|S_0'| = |S_0| - (dy - 1) + \text{abs}(dx) \qquad \text{(iii)}$$

But, the embedded late packet and the packets corresponding to its secondary event still exist causing resulting in a decrease in the number of zero displacement packets, therefore

$$|S_0'| = |S_0| - (dy - 1) + \text{abs}(dx) - (dy - 1) \qquad \text{(iv)}$$

On simplifying (iv), we get

$$|S_0'| = |S_0| + 2 - 2dl + \text{abs}(dx) \qquad \text{(v)}$$

Dividing (v) by N

$$|S_0'| / N = |S_0| / N + (2 - 2dl + \text{abs}(dx)) / N$$

By the definition of RD and (2), we therefore have

$$\mathbf{RD'[0] = RD[0] + (2 - 2dl + abs(dx)) / N}$$

Also, we know that a late packet with a displacement ***dy*** would have ***dy*** packet prior to it that are early by 1 position. Therefore

$$|S_{-1}'| = (dy - 1) \qquad \text{(vi)}$$

The negative one corresponds to the embedded-ness of the late packet.

Dividing (vi) by N

$$|S_{-1}'| / N = (dy - 1) / N$$

By the definition of RD and (2), we therefore have

$$\mathbf{RD'[-1] = (dy - 1) / N}$$

For (c), refer to Theorem 2, Case 2: (c)

Case 3: b > 1 and b < dy

By expanding the equation (16), we get

a) $RD'[0] = RD[0] + (b - 2dl + abs(dx))/N$

b) $RD'[-1] = (dy - b)/N$

c) $RD'[dy - b] = RD[dy]$

Once the early packet is completely resequenced, the resequencing pattern tends to be similar to the case of independent reordering. From Theorem 5, Case 3, we know that the number of packets in the zero displacement set (S_0) and 1 position early displacement set (S_{-1}) can be considered to be a constant post single buffer usage.

Equations (v) and (vi) are

$$|S_0'| = |S_0| + 2 - 2dl + abs(dx) \qquad (v)$$

$$|S_{-1}'| = (dy - 1) \qquad (vi)$$

By combining the proofs from Theorem 2, Case 2 with (v) and (vi), we get

$$|S_0'| = |S_0| + 2 - 2dl + abs(dx) + b' \qquad (vii)$$

$$|S_{-1}'| = (dy - 1) - b' \qquad (viii)$$

Since we have already used a single buffer

$$b' = b - 1 \qquad (ix)$$

By using (ix) in (vii) and (viii) and by simplifying, we get

$$|S_0'| = |S_0| + b - 2dl + abs(dx) \qquad (x)$$

$$|S_{-1}'| = (dy - b) \qquad (xi)$$

Dividing (x) and (xi) by N,

$|S_0'| / N =$ $|S_0| / N + (b - 2dl + abs(dx)) / N$

$|S_{-1}'| / N$ $=$ $(dy - b) / N$

By the definition of RD and (2), we therefore have

RD' [0] = RD [0] + (b – 2dl + abs (dx))/ N

RD' [-1] = (dy – b) / N

For (c), refer to Theorem 2, Case 2: (c)

Q. E. D

***Theorem** 7*

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of embedded reordering pattern formed by an early event with displacement ***dx*** embedded within another early event with displacement ***dy*** after it has traversed through a resequencing node with ***b*** resequencing buffers is given by

RD' [i] = j **for b ≥ 2**

$\forall$ i $\in$ S_{RD}

where j = 1 when i = 0

j = 0 when i ≠ 0

(17)

RD' [i - p] = {j (N) + k (abs (dx) + 1) - j}/ N **for b = 1**

$\forall$ i $\in$ S_{RD},

where j = +1, k = -1 and p = 0 when i = 0

j = 0, k = +1 and p = 0 when i = 1

$j = 0, k = 0$ and $p = -1$ when $i = dx$

(18)

<u>Proof</u>

Case 1: b ≥ 2

The lowest first algorithm is such that early packets can be resequenced with a single buffer. Therefore, when a sequence containing 2 early packets are sent through a resequencing node with 2 buffers, the entire sequence is back in order

Refer to the proof for Theorem 2: Case 1.

Case 2: b = 1

By expanding the equation (18), we get

a) $RD'[0] = (N - abs(dx) - 2) / N$

b) $RD'[1] = (abs(dx) + 1) / N$

c) $RD'[dx - 1] = RD[dx]$

When a single buffer is used, the outer early packet gets resequenced completely resulting in the existence of inner embedded early packet and its corresponding secondary events only.

We also know that when a packet p is early by dx positions, there will at most be dx packets that are late by 1 position because of secondary events

Therefore

$|S_1'| = abs(dx)$ (ii)

But since the embedded early packet is affected by the secondary events of the outer early packet, the actual displacement would be 1 less.

Adding 1 to (ii)

$|S_1'| = \text{abs}(dx) + 1$ (iii)

Dividing (iii) by N,

$|S_1'| / N = (\text{abs}(dx) + 1) / N$

By the definition of RD and (2), we therefore have

RD' [1] = (abs (dx) + 1) / N

Since the remaining packets would be back on order, the number of packets with zero displacement is

$|S_0'| = N - (\text{abs}(dx) + 1) - 1$ (iv)

Here, (abs (dx) + 1) represents the secondary events of the embedded early packet and 1 represents the early packet.

On simplifying (iv), we get

$|S_0'| = N - \text{abs}(dx) - 2$ (iv)

Dividing (iv) by N,

$|S_0'| / N = (N - \text{abs}(dx) - 2) / N$ (v)

By the definition of RD and (2), we therefore have

RD' [0] = (N – abs (dx) – 2) / N

Also, the embedded early packet which was affected by the secondary events of the outer early packet is now unaffected resulting in its true displacement which is 1 less

i.e.,

$dx' = dx - 1$

$|S_{dx}'| = |S_{dx-1}|$ (vi)

Dividing (vi) by N, we get

$$|S_{dx}'| / N \quad = \quad |S_{dx-1}| / N$$

By definition of RD and (2), we therefore have

$$\mathbf{RD'\ [dx-1]} \quad = \quad \mathbf{RD\ [dx]}$$

Q. E. D

3. 2. 4 Reorder Density of Overlapped Reordering Patterns

Overlapped Reordering occurs when a late or early packet event overlaps with another late or early packet event. Consider the sequence (*i-3, i-2, i-1, i, i+1, i+2, i+3, i+4, i+5*) as shown in the Figure 3-9.

If packets *i* and *i-2* are late by 3 and 4 positions respectively, then the resulting sequence would be (*i-3, i-1, i+1, i-2, i+2, i+3, i+4, i, i+5*). If the lowest first resequencing algorithm with a single buffer is used, the resultant sequence would be (*i-3, i-1, i-2, i+1, i+2, i+3, i, i+4, i+5*) as shown in Figure 3-9.

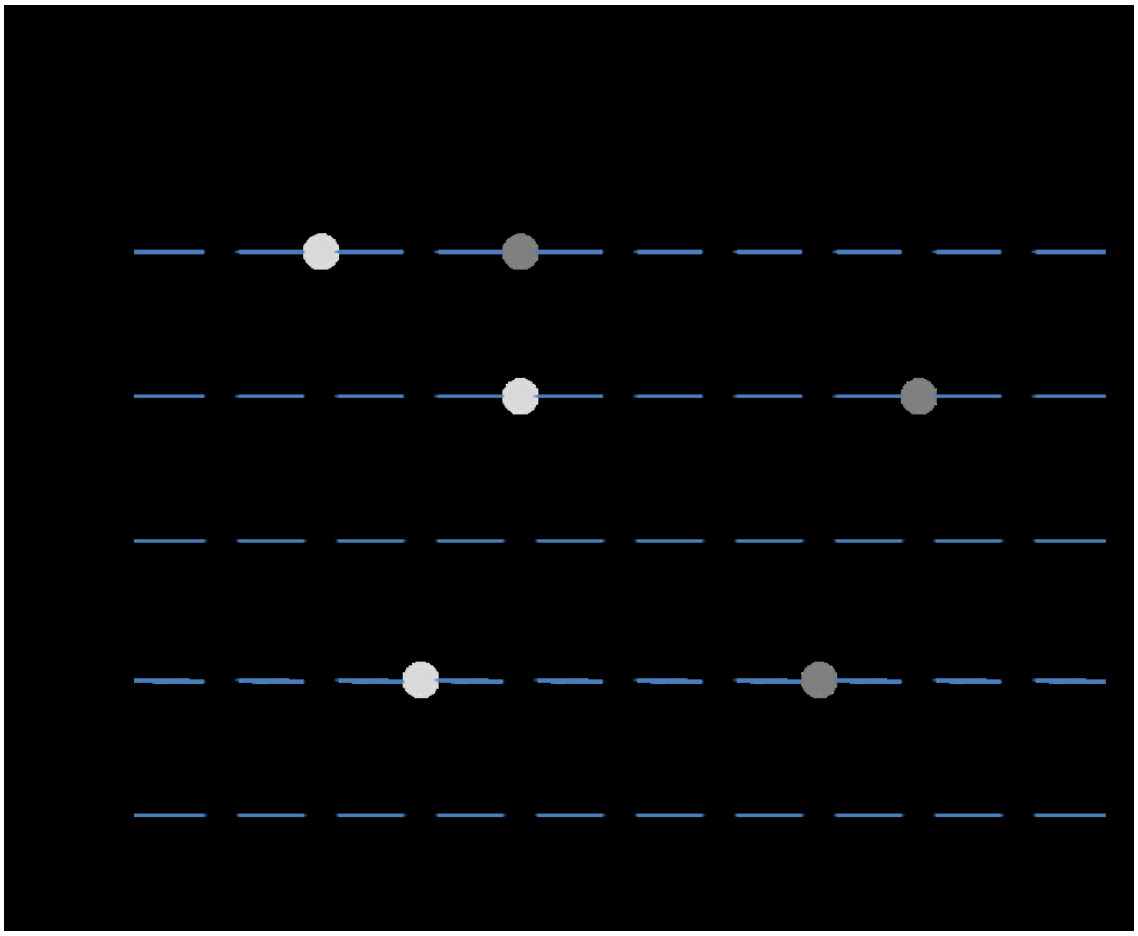


Figure 3-9. Change in displacement values of a sequence containing an overlapped reordering pattern

Theorem 8

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of overlapped reordering pattern formed by a late event overlapping with another late event that has traversed through ***b*** resequencing buffers is given by

$RD'[i] = j$ **for $b \geq \max(dx, dy)$**

$\forall\ i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(19)

$RD'[i - p] = RD[i] + j(b / N)$ **for $b \leq Vxy$**

$\forall\ i \in S_{RD}$,

where $j = +1$ and $p = 0$ when $i = 0$

$j = 0$ and $p = 0$ when $i = -1$

$j = -1$ and $p = 0$ when $i = -2$

$j = 0$ and $p = b$ when $i > 1$

(20)

$RD'[i - p] = RD[i] + \{j(Vxy) + k(b - Vxy)\} / N$

for $b > Vxy$ and $b < \min(dx, dy)$

$\forall\ i \in S_{RD}$,

where $j = +1$, $k = +2$ and $p = 0$ when $i = 0$

$j = 0$, $k = -2$ and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = b$ when $i > b$

(21)

$RD'[i - p] = RD[i] + \{j(Vxy + 1) + k(b - Vxy)\} / N$ **for $b = \min(dx, dy)$**

$\forall\ i \in S_{RD}$,

where $j = +1$, $k = +2$ and $p = 0$ when $i = 0$

$j = 0$, $k = -2$ and $p = 0$ when $i = -1$

$j = 0$, $k = 0$ and $p = 0$ when $i > b$

(22)

$RD'[i - p] = RD[i] + \{j(Vxy + 1) + k((\min(dx,dy) - Vxy) + (k/2)(b - \min(dx,dy))\} / N$

for $b > \min(dx, dy)$ and $b < \max(dx, dy)$

$\forall i \in S_{RD}$,

where $j = +1, k = +2$ and $p = 0$ when $i = 0$

$j = 0, k = -2$ and $p = 0$ when $i = -1$

$j = 0, k = 0$ and $p = 0$ when $i > b$

(23)

<u>Proof</u>

Case 1: $b \geq \max(dx, dy)$

When the number of buffer available is equal to or greater than the highest displacement of one of the late packets, the entire sequence is back in order.

Refer to the proof for Theorem 2: Case 1.

Case 2: $b \leq Vxy$

By expanding equation (20), we get

a) $RD'[0] = RD[0] + (b / N)$
b) $RD'[-1] = RD[-1]$
c) $RD'[-2] = RD[-2] - (b / N)$
d) $RD'[dx] = RD[dx - b]$
e) $RD'[dy] = RD[dy - b]$

From [43], we know that the length of overlap is the number of packets with a displacement of -2 (2 positions early) since packets in the region of overlap are affected by both the late packet events

Therefore,

$Vxy = |S_{-2}|$

We also know that, when a reordered sequence is traversed by a resequencing node containing ***b*** buffers, the resultant displacement of the late packet ***p*** will be no greater than ***dx - b***. As a consequence, ***dx - b*** packets will be early by 1 position and ***b*** packets will be back in order.

Until the region of overlap diminishes, there exists at least one packet that would be early by 2 positions. Therefore, we have

$$|S_0'| = |S_0| + b \qquad \text{(i)}$$

$$|S_{-2}'| = |S_{-2}| - b \qquad \text{(ii)}$$

Dividing (i) and (ii) by N,

$$|S_0'| / N = |S_0| / N + b / N \qquad \text{(iii)}$$

$$|S_{-2}'| / N = |S_{-2}| / N - b / N$$

By definition of RD and [41], we therefore have

$$\mathbf{RD'\,[0] = RD\,[0] + (b / N)}$$

$$\mathbf{RD'\,[-2] = RD\,[-2] - (b / N)}$$

For proofs for (d) and (e), refer to Theorem 2, Case 2: (c)

Case 3: b > Vxy and b < min (dx, dy)

By expanding equation (21), we get

a) $RD'\,[0] = RD\,[0] + Vxy / N + 2\,(b - Vxy) / N$

b) $RD'\,[-1] = RD\,[-1] - 2\,(b - Vxy) / N$

c) $RD'\,[dx] = RD\,[dx - b]$

d) $RD'\,[dy] = RD\,[dy - b]$

When the available buffer is greater than Vxy, the region of overlap would have completely diminished resulting in 2 independent late reordering patterns. Therefore, the number of zero displacement packets increases by twice the buffer size and the number of packets that would be early by 1 position decreases by twice the buffer size.

Substituting for b = Vxy in equation (iii), we get

$|S_0'| / N = |S_0| / N + Vxy / N$ (iv)

Since b ≤ Vxy has already been accounted for in (iv), we have

$|S_0'| / N = |S_0| / N + Vxy / N + 2b' / N$ (v)

But,

$b' = b - Vxy$ (vi)

Using (vi) in (v)

$|S_0'| / N = |S_0| / N + Vxy / N + 2 (b - Vxy) / N$ (vii)

By definition of RD and [41], we therefore have

RD' [0] = RD [0] + Vxy / N + 2 (b – Vxy) / N

Similarly, after the region of overlap diminishes, there only exist packets that are early by 1 position or late packets. Therefore,

$|S_{-1}'| = |S_{-1}| - 2b'$ (viii)

Using (vi) in (viii) and dividing by N,

$|S_{-1}'| / N = |S_{-1}| / N - 2(b - Vxy) / N$

By definition of RD and [41], we therefore have

RD' [-1] = RD [-1] - 2 (b – Vxy) / N

For proofs for (c) and (d), refer to Theorem 2, Case 2: (c)

Case 4: b = min (dx, dy)

By expanding equation (22), we get

a) RD' [0] = RD [0] + (Vxy + 1) / N + 2 (b – Vxy) / N

b) RD' [-1] = RD [-1] - 2 (b – Vxy) / N

c) RD' [dx] = RD [dx – b]

d) RD' [dy] = RD [dy – b]

The value of RD remains the same as Case 3 for all indices except 0 since one of the late packets would now be resequenced completely.

Therefore, by adding 1 to (vii), we get

$|S_0'| / N = |S_0| / N + (Vxy + 1) / N + 2 (b – Vxy) / N$ (ix)

By definition of RD and [41], we therefore have

RD' [0] = RD [0] + (Vxy + 1) / N + 2 (b – Vxy) / N

Case 5: b > min (dx, dy) and b < max (dx, dy)

By expanding equation (23), we get

a) RD' [0] = RD [0] + (Vxy + 1) / N + 2 (min (dx,dy) – Vxy) / N + (b – min (dx,dy)) / N

b) RD' [-1] = RD [-1] - 2 (min (dx,dy) – Vxy) / N - (b – min (dx,dy)) / N

c) RD' [dy] = RD [dy – b]

From Cases 2 – 4, we have the following

$|S_0'| / N = |S_0| / N + (Vxy + 1) / N + 2(\min (dx, dy) – Vxy) / N$ (x)

$|S_{-1}'| / N = |S_{-1}| / N - 2(\min (dx, dy) – Vxy) / N$ (xi)

In the aforementioned equations, (min (dx, dy) – Vxy) represents the buffer range presented in Case 4. One of the late packets is completely resequenced on using min (dx, dy) buffers, only a single late packet event and its associated secondary event packet exist. Since effect of buffers $\leq$ min (dx, dy) has already been computed in (x) and (xi), the remaining buffer can be computed as

$$b' = b - \min(dx, dy) \qquad \text{(xii)}$$

By combining the proofs from Theorem 2, Case 2 and (x) and (xi), we get

$$|S_0'| / N = |S_0| / N + (Vxy + 1) / N + 2(\min(dx, dy) - Vxy) / N + b' / N \qquad \text{(xiii)}$$

$$|S_{-1}'| / N = |S_{-1}| / N - 2(\min(dx, dy) - Vxy) / N + b' / N \qquad \text{(xiv)}$$

Using (xii) in (xiii) and (xiv),

$|S_0'| / N = |S_0| / N + (Vxy + 1) / N + 2(\min(dx, dy) - Vxy) / N + (b - \min(dx, dy) / N$

$|S_{-1}'| / N = |S_{-1}| / N - 2(\min(dx, dy) - Vxy) / N + (b - \min(dx, dy) / N$

By definition of RD and [41], we therefore have

RD' [0] = RD [0] + (Vxy + 1) / N + 2 (min (dx,dy) – Vxy) / N + (b – min (dx,dy)) / N

RD' [-1] = RD [-1] - 2 (min (dx,dy) – Vxy) / N - (b – min (dx,dy)) / N

For the proof for (c), refer to Theorem 2, Case 2: (c)

Q. E. D

Theorem 9

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of overlapped reordering pattern formed by a late event overlapping with an early event (or an early event overlapping with a late event) that has traversed through ***b*** resequencing buffers is given by

RD' [i] = j **for $b \geq \max(S_{RD})$**

$\forall\, i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(24)

$RD'[i - p] = RD[i] + \{j\,[abs(\min(S_{RD})) - Vxy] + k(Vxy)\} / N$ **for b = 1**

$\forall\, i \in S_{RD}$,

where $j = +1, k = -1$ and $p = 0$ when $i = 0$

$j = 0, k = +1$and $p = 0$ when $i = -1$

$j = 0, k = 0$ and $p = b$ when $i > 1$

(25)

$RD'[i - p] = RD[i] + \{j\,[abs(\min(S_{RD})) - Vxy] + k(Vxy) - k(b - 1)\} / N$

for b > 1 and b < max(S_{RD})

$\forall\, i \in S_{RD}$,

where $j = +1, k = -1$ and $p = 0$ when $i = 0$

$j = 0, k = +1$and $p = 0$ when $i = -1$

$j = 0, k = 0$ and $p = b$ when $i > 1$

(26)

<u>Proof</u>

Case 1: b ≥ max (S_{RD})

When the number of buffers available is equal to or greater than the maximum value of displacement of the late packet, the entire sequence is back in order.

Refer to the proof for Theorem 2: Case 1.

Case 2: b = 1

By expanding the equation (25), we get

a) RD' [0] $= RD[0] + (abs(\min(S_{RD})) - Vxy) / N - Vxy / N$

b) RD' [-1] $= RD[-1] + Vxy / N$

c) RD' [dx] $= RD[dx - b]$

We know that a single buffer is sufficient to completely resequence an early packet back in order resulting in the diminishing of the region of overlap. When a late packet event overlaps with an early packet event or vice versa, the region of overlap contains packets that would have zero displacement because of the secondary event effects of both the late and the early packets. Therefore, the number of packets with zero displacement would now be

$$|S_0'| = |S_0| - Vxy \qquad \text{(i)}$$

The secondary events of the early packet outside the region of overlap would also get negated resulting in more number of packets with zero displacement. Therefore, (i) is now

$$|S_0'| = |S_0| - Vxy + (abs(\min(S_{RD}) - Vxy) \qquad \text{(ii)}$$

Dividing (i) by N

$$|S_0'| / N = |S_0| / N - Vxy / N + (abs(\min(S_{RD}) - Vxy) / N$$

By definition of RD and [41], we therefore have

RD' [0] $\mathbf{= RD[0] + (abs(\min(S_{RD})) - Vxy) / N - Vxy / N}$

When the early packet gets resequenced, the packets in the region of overlap would now be affected by the late packet event only resulting in those packets being early by 1 position. Therefore,

$$|S_{-1}'| = |S_{-1}| + Vxy \qquad \text{(iii)}$$

Dividing (iii) by N

$$|S_{-1}'| / N = |S_{-1}| / N + Vxy / N$$

By definition of RD and [41], we therefore have

RD' [-1] = RD [-1] + Vxy / N

For the proof for (c), refer to Theorem 2, Case 2: (c)

Case 3: b > 1 and b < max (S_{RD})

By expanding the equation (26), we get

a) RD' [0] = RD [0] + (abs (min (S_{RD}))) – Vxy) / N – Vxy / N + (b – 1) / N

b) RD' [-1] = RD [-1] + Vxy / N + (b – 1) / N

c) RD' [dx] = RD [dx – b]

Similar to proofs for earlier Theorems, we have already accounted for the change in packets with zero displacement and -1 displacement when a single buffer is used.

Therefore, (ii) would be

$$|S_0'| = |S_0| - Vxy + (abs (min (S_{RD}) – Vxy) + (b – 1) \qquad (iv)$$

Also, (iii) would be

$$|S_{-1}'| = |S_{-1}| + Vxy – (b – 1) \qquad (v)$$

Dividing (iv) and (v) by N

$$|S_0'| / N = |S_0| / N - Vxy / N + (abs (min (S_{RD}) – Vxy) / N + (b – 1) / N$$

$$|S_{-1}'| / N = |S_{-1}| / N + Vxy / N – (b – 1) / N$$

By definition of RD and [41], we therefore have

RD' [0] = RD [0] + (abs (min (S_{RD}))) – Vxy) / N – Vxy / N + (b – 1) / N

RD' [-1] = RD [-1] + Vxy / N + (b – 1) / N

For the proof for (c), refer to Theorem 2, Case 2: (c)

Q. E. D

Theorem 10

In the absence of duplicates and losses, the reorder density of a sequence with a single instance of overlapped reordering pattern formed by an early event overlapping with another early event that has traversed through ***b*** resequencing buffers is given by

RD’ [i] $= j$ **for $b \geq 2$**

$\forall\ i \in S_{RD}$

where $j = 1$ when $i = 0$

$j = 0$ when $i \neq 0$

(27)

RD’ [i – p] $= \{j\,(N) + k\,(Vxy) + m\,(1)\}/\,N$ **for $b = 1$**

$\forall\ i \in S_{RD}$,

where $j = +1$, $k = -1$ and $m = 1$ when $i = 0$

$j = 0$, $k = +1$ and $m = 0$ when $i = 1$

$j = 0$, $k = 0$ and $m = 1$ when $i = -Vxy$

(28)

Proof

Case 1: $b \geq 2$

Refer to the proof for Theorem 7, Case 1.

Case 2: $b = 1$

By expanding (28), we get

a) RD’ [0] $= (N - Vxy - 1) / N$
b) RD’ [1] $= Vxy / N$

c) RD' [-Vxy] = 1

When the lowest first algorithm is parsing the sequence with a single buffer, at the beginning of the region of overlap, both the early packets are compared, the packet with the lower sequence number of the two packets is released and the other is completely resequenced. This causes one of the packets to get resequenced partially resulting in the net displacement the early packet to be equal to the length of the overlap.

Therefore,

$|S_1'| = Vxy$ (i)

$|S_{-Vxy}'| = 1$ (ii)

Dividing (i) and (ii) by N

$|S_1'| / N = Vxy / N$

$|S_{-Vxy}'| / N = 1 / N$

By definition of RD and [41], we therefore have

RD' [1] = Vxy / N

RD' [-Vxy] = 1

It is also known that the sequence would now contain packets with 0 – displacement, 1-displacement and -Vxy displacement.

We have

$N = |S_1'| + |S_0'| + |S_{-Vxy}'|$

$\Rightarrow |S_0'| = N - |S_1'| - |S_{-Vxy}'|$

Using (i) and (ii) in the aforementioned equation,

$|S_0'| = N - Vxy - 1$ (iii)

Dividing (iii) by N

$|S_0'| / N \quad = \quad (N - Vxy - 1) / N$

By definition of RD and [41], we therefore have

$\mathbf{RD' [0] \quad = \quad (N - Vxy - 1) / N}$

Q. E. D

3. 3 Impact of Resequencing Buffers on Reorder Buffer Occupancy Density

In this section, we study and analyze the impact of resequencing buffers on Reorder Buffer Occupancy Density.

Property 1

At a given instance, the resequencing buffer holds at most ***max (B_m, b)*** packets whose sequence numbers are the largest amongst the sequence numbers of the packets encountered while traversing the sequence thus far. ***B_m*** is the highest value of ***i*** when ***RBD [i]*** is non-zero, and ***b*** is the capacity of the resequencing buffer.

Proof

Based on the definition of reorder buffer occupancy density [43], it is known that the highest value of ***i*** when ***RBD [i]*** is non-zero is the total buffer requirement for placing the packet back into order. The lowest first algorithm releases the packet with the lowest sequence number after having compared with the current packet number resulting in all higher numbered packets getting accumulated in the buffer. Therefore, the buffer can at max hold ***max (B_m, b)*** number of packets.

Q. E. D

Lemma

Consider a sequence with RBD such that

RBD [i] = 0 for $i > B_m$

In such a sequence, a packet with sequence number ***i*** cannot be preceded by more than $\boldsymbol{B_m}$ number of packets whose sequence numbers are greater than ***i***.

Proof

Buffer occupancy is the occupancy of the hypothetical buffer in anticipation of the expected packet [25]. $\boldsymbol{B_m}$ is the highest value of buffer occupancy obtained from the reorder buffer occupancy density graph of the sequence. This implies that there would be at max $\boldsymbol{B_m}$ number of packets in the buffer in the anticipation of the expected packet that is displaced the most in the sequence.

RBD [i] = 0 for $i > B_m$

⇨ i > displacement of the late packet

OR

⇨ Sequence number of i > Sequence numbers of the preceding packets prior to i

Q. E. D

Theorem 11

If the highest value of ***i*** for which ***RBD[i] = 0*** for the sequences at the input and output of resequencing node with b buffers be B_m and B_m', then

$$B_m' = 0 \qquad \textbf{for } b \geq B_m$$

$$B_m' = B_m - b \qquad \textbf{for } b < B_m \qquad \textit{(28)}$$

(29)

<u>Proof</u>

Consider a sequence (*i-3, i-2, i-1, i, i+1, i+2, i+3, i+4, i+5*). If packet ***i*** in this sequence arrives late resulting in a reorder event, the buffer occupancy of the sequence would be non – zero and the maximum value of buffer occupancy would be $\boldsymbol{B_m}$. From the Lemma, we know that this is the same as displacement of the packet from the initial position.

Case 1: $\mathbf{b \geq B_m}$

When the number of resequencing buffers available is equal to or greater than the total number of buffers required to put the entire sequence back into order, the late packet gets resequenced completely resulting in an in order sequence. Since $\boldsymbol{B_m}$ is the highest value of ***i*** when RBD [i] is non – zero, we therefore have

$$\mathbf{Bm = 0}$$

Case 2: $\mathbf{b < B_m}$

From Case 1, we know that when the Bm resequencing buffers are available, the entire sequence is back in order. But, when b (where b < Bm) buffers are available, the sequence gets partially resequenced.

We know that,

If $\boldsymbol{B_m}$ buffers are required to reduce the displacement by $\boldsymbol{B_m}$ positions, we can say that ***b*** buffers are sufficient to reduce the displacement by ***b*** positions.

Therefore,

the new displacement of the late packet would be (Bm – b)

⇨ $\mathbf{B_m' = B_m - b}$

Q.E.D

Theorem 12

In the absence of duplicates and losses, the reorder buffer occupancy density of a sequence that has traversed through a resequencing node with ***b*** buffers is given by

$$\text{RBD'}[0] = 1 \qquad \textbf{for } \mathbf{b \geq B_m} \tag{30}$$

$$\text{RBD'}[i - p] = j\,(\text{RBD}[i]) + k\,(\textstyle\sum_{i=0}^{b} \textbf{RBD}[i]) \qquad \textbf{for } \mathbf{b < B_m}$$

$\forall\, i \in S_{RBD}$ where,

$j = 0, k = +1$ and $p = 0$ when $i = 0$

$j = +1, k = 0$ and $p = b$ when $i > b$

$$\tag{31}$$

Proof

Case 1: $\mathbf{b \geq B_m}$

When the number of resequencing buffers available is equal to or greater than the maximum number of buffers required to put the entire sequence back in order, the sequence is back in order. From the definition of RBD [43], for an in-order sequence, we know that

$$\mathbf{RBD'[0] = 1}$$

Case 2: $\mathbf{b < B_m}$

By expanding equation (31), we get

a) $\text{RBD'}[0] = \sum_{i=0}^{b} \textbf{RBD}[i]$

b) RBD' [b' - b] = RBD [b] where $b' \in \{x : RBD[x] > 0 \text{ and } x > b\}$

From Chapter 2 and [41], we know that every packet that is reordered has a secondary event associated with it. RBD is the normalized value of buffer occupancy of the hypothetical buffer used to store packets that are affected by secondary events [25]. Therefore, when packets get resequenced back into order, the packets affected by the secondary events are also back in order thereby reducing the buffer occupancy and correspondingly increasing the 0 – buffer occupancy or RBD [0].

Since the buffer available is less than the total requirement, the displacement of the primary event packet would reduce by ***b*** causing reduction in buffer occupancy by ***b***. Therefore,

RBD' [0] = RBD [0] + RBD [1] + … + RBD [b]

⇨ **RBD' [0]** = $\sum_{i=0}^{b}$ **RBD [i]**

Also, the displacement(s) of the primary event packet(s) reduces by ***b***. This means that the hypothetical buffer required to resequence these packets would be reduced by ***b***. Therefore,

RBD' [b' - b] = RBD [b] where $b' \in \{x : RBD[x] > 0 \text{ and } x > b\}$

Q. E. D

Inference

A sequence emerging from a resequencing node with (b_1 + b_2) buffers is same as the same sequence emerging from a cascade of two nodes, one with b_1 resequencing buffers and the other with b_2 resequencing buffers.

Explanation

The eventual sum of resequencing buffers used for a sequence assumes much more importance rather than the distribution pattern of the resequencing buffers across nodes (referred to as resequencing nodes). If a sequence requires $\boldsymbol{B_m}$ resequencing buffers to get back into order, the $\boldsymbol{B_m}$ buffers can be distributed across $\boldsymbol{B_m}$ number nodes each with a single resequencing buffer. The eventual resultant sequence after it passes through the $\boldsymbol{B_m}$ nodes would be an in-order sequence with RBD [0] and RD [0] having unity values just like it is mentioned in Figure 3-1.

3.4 Verification and Analysis

The theorems presented in this chapter were verified using two sets of data:

a) Internet traces available in the CNRL repository [63]

b) Sequences generated using random probability distribution

The entire verification and analysis process can be better understood with the help of the Figure 3-10. The verification is broken into smaller modules namely

i) Data interpreter

The Internet traces are served as an input to this module. The Internet traces available in the CNRL repository [63] are *tcpdump* files which are based on relative sequence numbers. Every packet has a beginning byte number and an ending byte number. Since this data cannot be directly used as sequence numbers, they had to be interpreted in a different manner. This module parses the data and re-interprets them into sequence numbers of the form (1, 2, 3....N). Also, special filtering is done to make sure traces with lost or duplicate packets are discarded since the theorems are based on such assumptions.

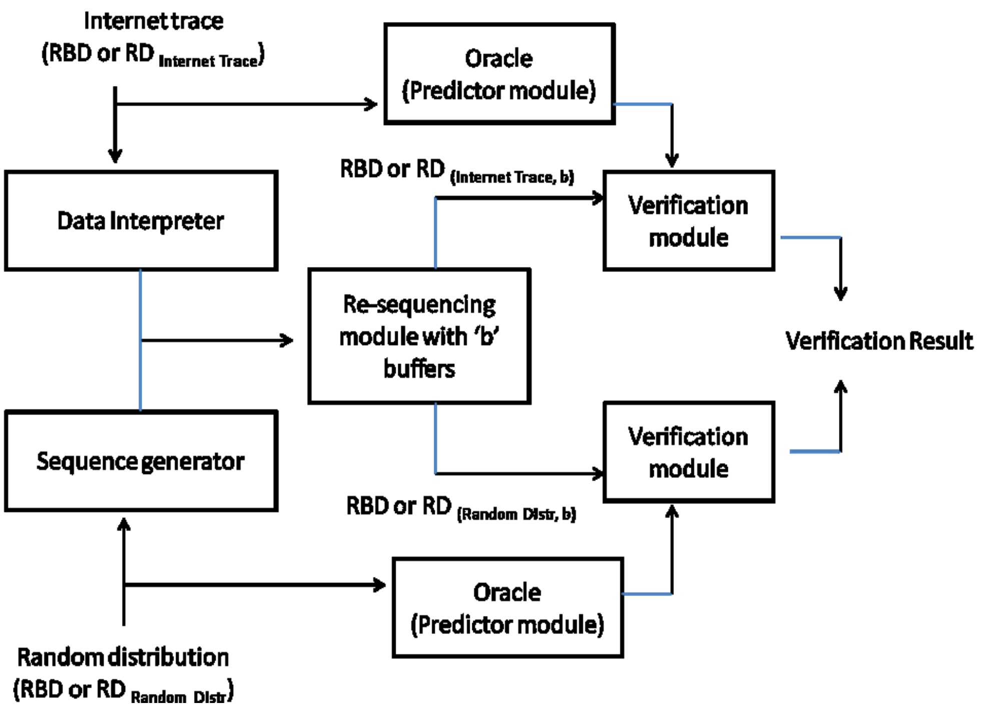


Figure 3-10. Different modules used for verification and analysis

ii) Sequence generator

When the formulae for predicting RD (and RBD) were initially designed, they had to be verified using a simpler approach and hence the creation of such a module. Based on the kind of pattern, the sequence generator takes random values as input to generate a sequence of packets which serves as input to the Resequencing module.

iii) Resequencing module with 'b' buffers

This module consists of the lowest first resequencing algorithm. It accepts a sequence of packets as input and outputs a new sequence of packets as output after resequencing them using 'b' buffers where 'b' would vary based on the maximum number of buffers required for resequencing.

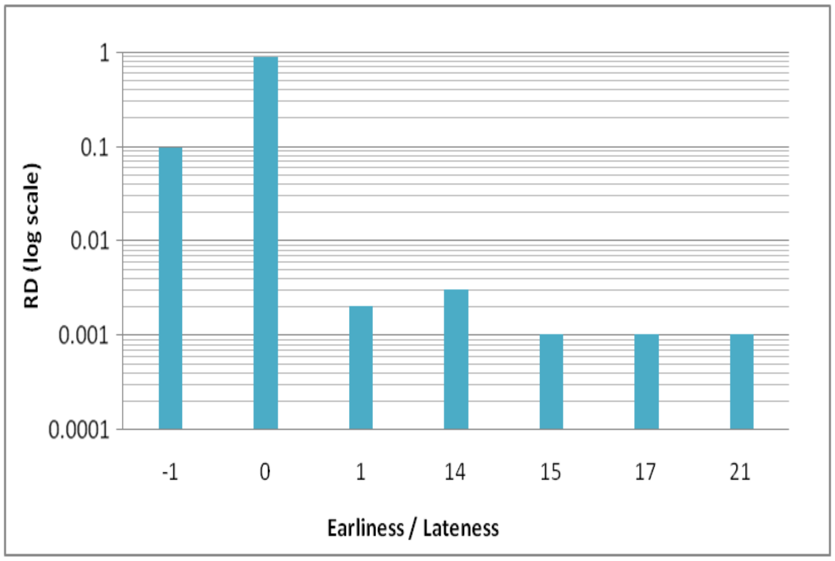
iv) Oracle or predictor module

This forms the crux of the verification process. RD (or RBD) of a sequence (could be an Internet trace or a sequence that was generated) is served as input to this module. All the theorems are mathematically represented in the form of modules. Based on the kind of pattern, an appropriate module would be selected and this output is sent to the verification module.

v) Verification module

This module compares the values of RD (or RBD) generated by the Oracle and the resequencing module and outputs the result based on the consistency in comparison. Most modules have been developed using *C* as the programming language and the *gcc* compiler. The Internet traces had to be processed before sending as input to the Data interpreter and this processing was done using *Perl* since it can be made to understand a number of different patterns.

The CNRL repository [63] contains a number of traces with every trace consisting of one or more different patterns. To illustrate one of the verification results for this chapter, the one with the maximum number of reordering patterns with displacement greater than 1 was selected namely "***0.14.30.4.106.www.olympus.co.jp.dmp***" [63]. The displacements and their probability distribution in the sequence were analyzed. The initial RD and RBD graphs shown in Figure 3-11 were obtained using scripts for RFC 5236 [25] at [63]. Note that the vertical axes of all the RD and RBD graphs are based on log scale. The measured RD value was obtained using Perl scripts available at [63] for RFC 5236 [25].



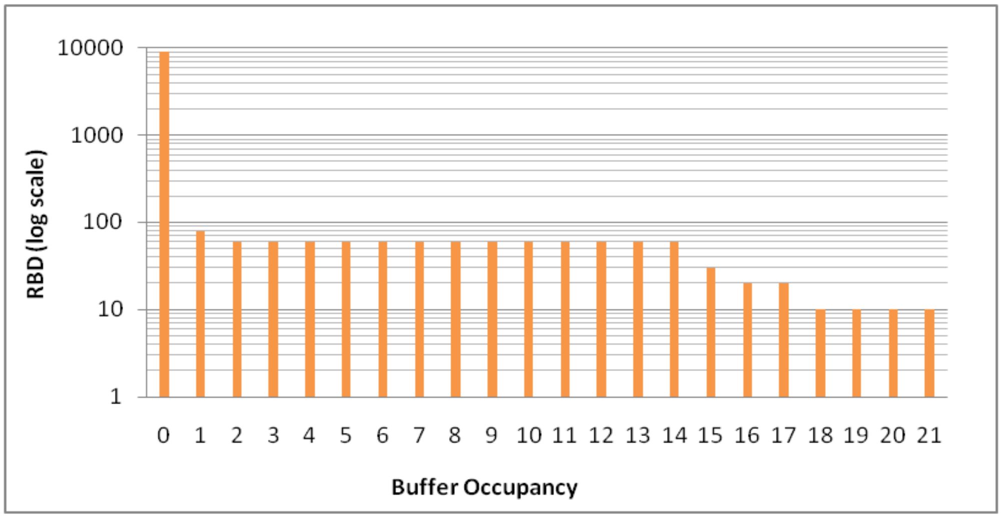


Figure 3-11. Initial graphs of RD and RBD obtained from one of the traces.

A single trace contains 1000 packets obtained over different time intervals from a number of sites. The selected sequence of 1000 packets was passed through Data interpreter module, then through the resequencing module containing first 5 resequencing buffers and then 15 resequencing buffers to obtain 2 new sequences. The RD and RBD of these sequences were measured. The “Measured: columns in Figure 3-12 represent the RD and RBD graphs after using

5 resequencing buffers. Similarly, the “Measured” columns in Figure 3-13 represent the graphs after using 15 resequencing buffers. The “Theoretical” columns for RD and RBD for 5 and 15 resequencing buffers in Figure 3-12 and Figure 3-13 respectively were obtained using the RD and RBD obtained initially and the theoretical results presented in this chapter.

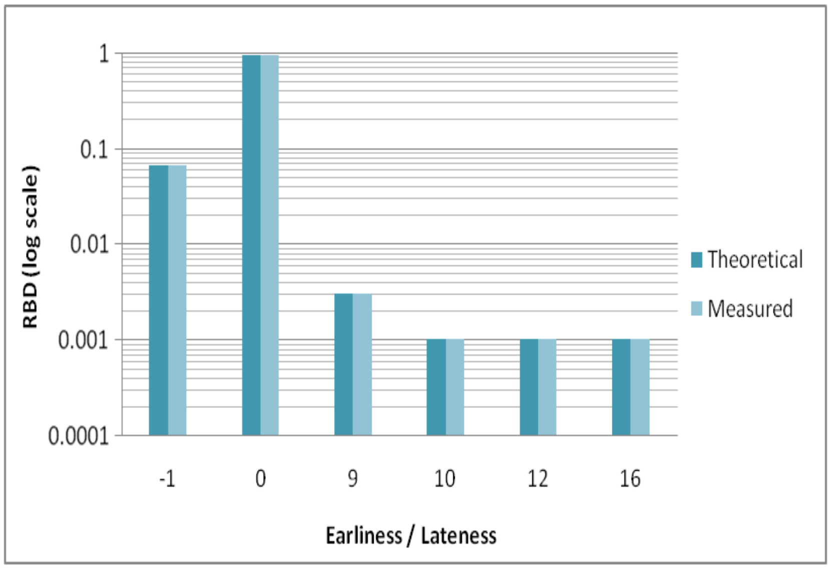


Figure 3-12. Graph of RD after using 5 resequencing buffers

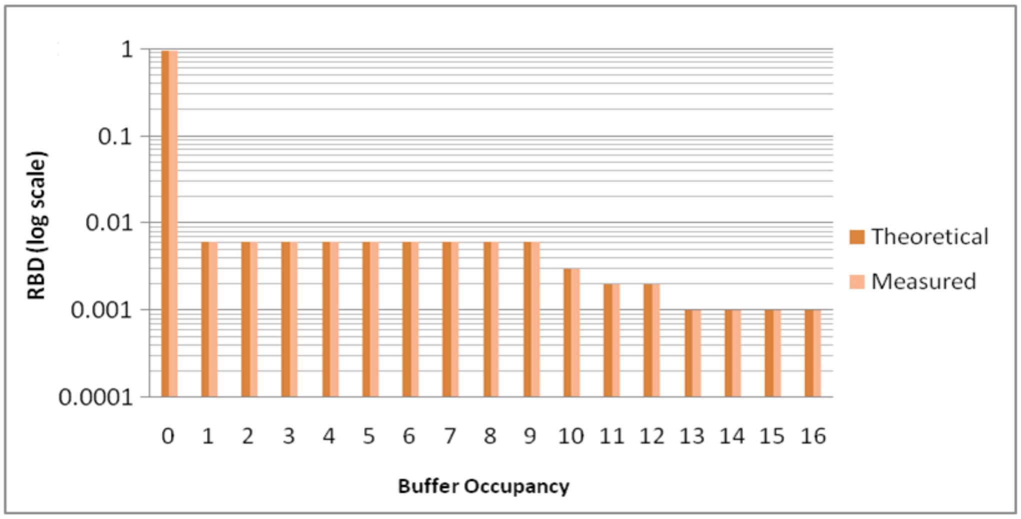


Figure 3-13. Graph of RBD after using 5 resequencing buffers

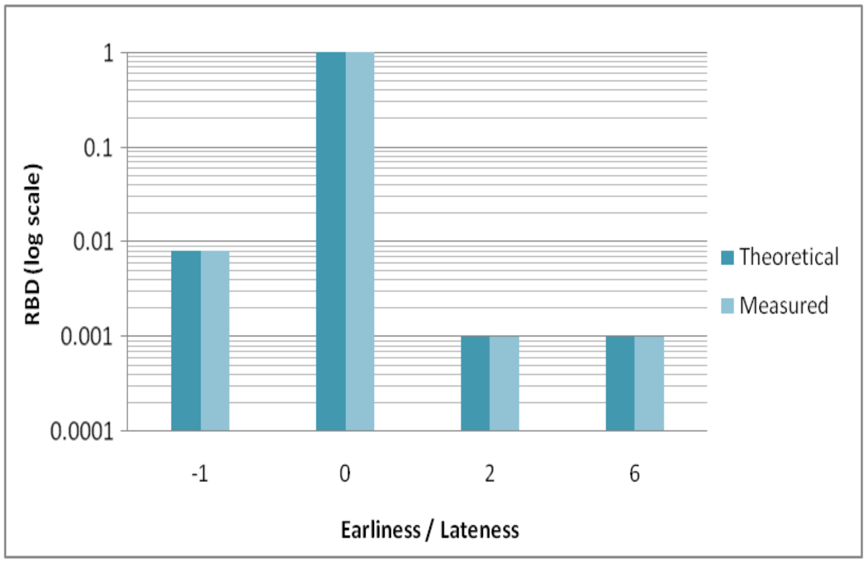


Figure 3-14. Graph of RD after using 15 resequencing buffers

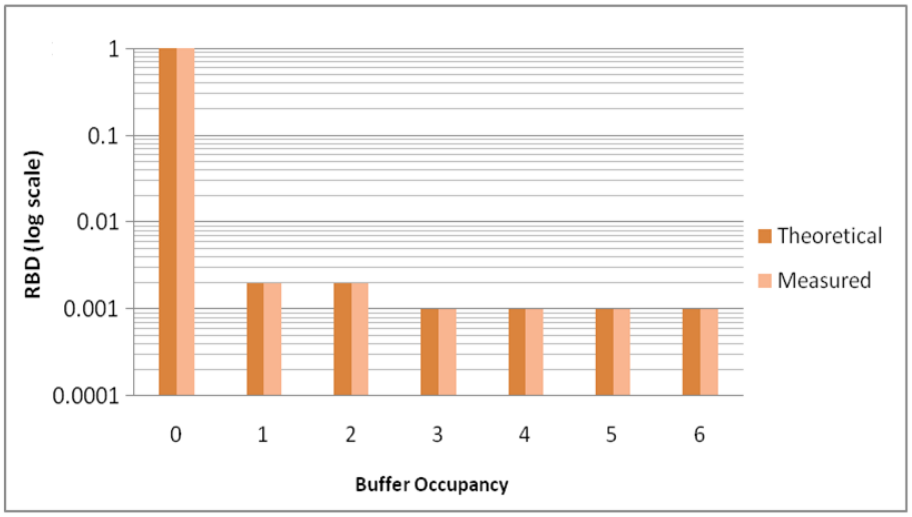


Figure 3-15. Graph RBD after using 15 resequencing buffers

As can be seen, the theoretical and measured values of RD and RBD for a given number of resequencing buffers are equal thereby verifying the theorems. Note the improvement in the orderliness of the sequences.

3.5 Conclusion

In this chapter, analytical results that can be used to estimate the impact of resequencing buffers on packet reordering were presented. This is one of the first researches to analyze the impact of resequencing nodes on degree and extent of packet reordering. With the presented results, the variation of reorder density and reorder buffer occupancy density as a packet stream goes through a resequencing node can be characterized. The proposed theorems were verified using simulation study based on the traces available at [63] and random probability distribution and were found to be accurate. Such estimations give valuable insight about the quantity of resources (mostly buffers) required to resequence packets in reordered sequences. With increase in network speeds and parallelism across network components and bandwidth aggregation, packet reordering is bound to play a much more important role in the future.

Internet service providers might even have to include packet reordering metric as one of their QoS parameters which means that across the Internet, a number of Internet Service Providers (ISPs) will have to estimate and sync their QoS parameters. Router manufacturers already include extra hardware for mitigating packet reordering [55]. This chapter presented the theoretical basis for estimation of buffer resources to meet target goals and to evaluate the impact of size and placement of buffers on reordering in packet flows.

CHAPTER 4. REORDER BUFFER OCCUPANCY DENSITY FROM REORDER DENSITY

This chapter focuses on derivation of Reorder Buffer Occupancy Density given the Reorder Density of a sequence. The motivation to work on such a problem arose out of multiple reasons: when network designers model computer networks, considerable emphasis is placed on resource constraints since hardware in network components isn’t cheap and one of the more important resource considerations could be the buffer required to store and forward packets as and when they arrive, resequence reordered packets, etc.

Reorder Density (RD) provides an estimate about the degree of displacement of early or late packets in a sequence. While designing the hardware required to mitigate the effects of packet reordering, Reorder Buffer Occupancy Density (RBD) provides better understanding about the resources required and related constraints. By definition [25], RBD is the histogram of the occupancy, normalized with respect to the number of packets, of the hypothetical buffer that is used to resequence the packets.

In the future, Reorder Density (RD) could be considered as one of the Quality of Service (QoS) parameters since that seems more appropriate compared to RBD. Under such design considerations, it would make sense to derive RBD since RBD provides an estimate of the hypothetical buffer required to resequence packets that are out of order. This would enable network designers to maintain particular reorder levels in commercial networks.

In this chapter, Section 4.1 explains the theorems and associated proofs for deriving RBD from RD, Section 4.2 provides the Verification and Analysis of the theorems and Section 4.3 concludes the chapter.

4.1 Reorder Buffer Occupancy Density (RBD) from Reorder Density (RD)

The problem of deriving RBD from RD can be explained using Figure 4-1. As can be seen from the figure, design of computer networks for a particular RD becomes a simple problem when we have the theoretical model to derive RBD from RD.

From the earlier chapters, we know that every sequence that has some degree of reordering can have one or more of the following patterns:

a) Independent Reordering

b) Embedded Reordering

c) Overlapped Reordering

These patterns have already been explained with great detail in Chapter 2. The high level description of the problem is represented in Figure 4-1. From the figure, it can be seen that to model computer networks, RBD is essential and to obtain RBD, either a sequence can be parsed to estimate RBD or the RD of the sequence can be used to obtain the RBD.

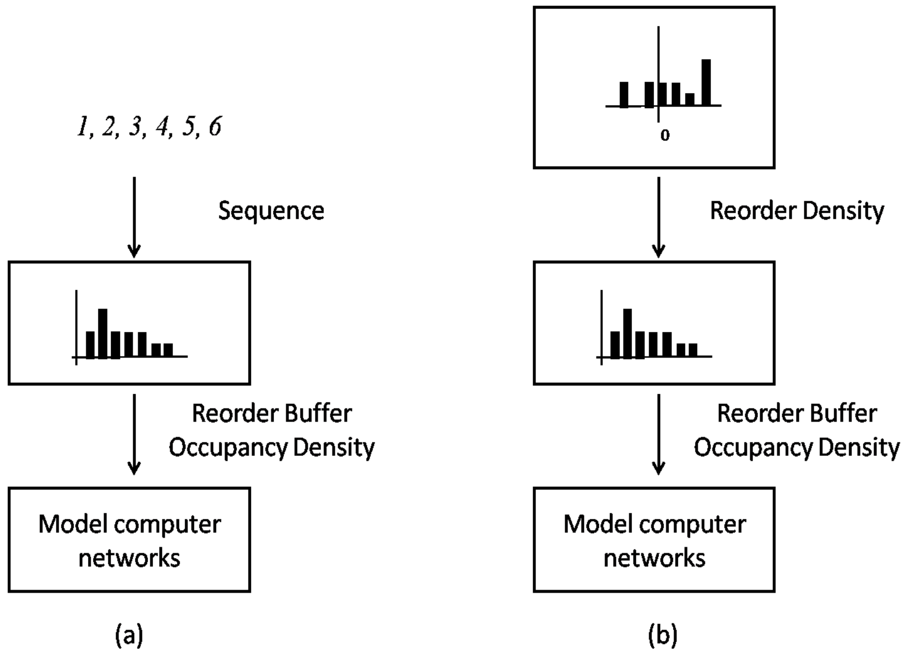


Figure 4-1. High level description of the problem

Table 4.1. Common notations used in this chapter

Symbol	**Description**
Vxy	Length of the region of overlap between two reordered packets
dx, dy	Displacement values of the reordered packets in the sequence
i, j	Additive, Subtractive or Multiplicative factors used in the theorems
RBD [i]	Reorder Buffer occupancy Density of a sequence
RD [i]	Reorder Density of a sequence
N	Number of packets in the sequence
$S_{(RD,L)}$	The set of indices of reorder density of the sequence whose values are greater than zero corresponding to late packets in the sequence. It may be represented as $\{x \mid x > 0, RD[x] > 0\}$

$S_{(RD,E)}$	The set of indices of reorder density of the sequence whose values are lesser than zero corresponding to early packets in the sequence. It may be represented as {x | x < 0, RD [x] > 0}

4.1.1 Independent Reordering Patterns

An Independent Reordering pattern occurs when a single packet is either late or early. Consider the sequence *(i-1, i, i+1, i+2, i+3, i+4, i+5)*. If *packet i* is late by 4 positions, then the resulting sequence would *be (i-1, i+1, i+2, i+3, i+4, i, i+5)*. Similarly, if *packet i+4* is early by 3 positions, then the resulting sequence would be *(i-1, , i+4, i+1, i+2, i+3, i+5)*.

Theorem 1

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an independent reordering pattern caused by a late packet event can be represented as

$$RBD[i] = j / N$$

where

$$j = 1 \quad \text{when } i \geq 1 \text{ and } i \leq dx$$

$$j = (N - dx) \quad \text{when } i = 0$$

$$\forall\, dx \in S_{(RD,L)}$$

(1)

Proof

On expanding the equation (1), we get

a) $RBD[0] = (N - dx) / N$

b) RBD [1 .. dx] = 1 / N

Since the only element in the set ***S*$_{(RD,L)}$** is the displacement value of the late packet, ***dx*** is equal to the value of the sole element in that set. Reorder Buffer Occupancy Density (RBD) provides an estimate of a hypothetical buffer that is used to buffer packets until the expected packet is encountered [43]. The calculation of RBD is done as follows: While traversing the sequence, if the sequence number of the encountered packet doesn't match the sequence number of the expected packet, then it is assumed to be stored in the hypothetical buffer. The buffer occupancy of this hypothetical buffer after traversing the entire sequence determines the RBD of the sequence.

Thus, if a packet arrives late by ***dx*** positions, the displacement of the packet would be ***dx*** and the buffer occupancy of the hypothetical buffer would keep increasing from 1 till ***dx*** at every arrival instance of the new packet until the sequence number of the expected packet corresponds with the sequence number of the late packet.

Since the indices of RBD correspond to the buffer occupancy, we therefore have

RBD [1 .. dx] = 1 / N

Also, since RBD is normalized, the remaining values correspond to the instances when the buffer occupancy was zero and therefore

RBD [0] = (N – dx) / N

Q. E. D

Theorem 2

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an independent reordering pattern caused by an early packet event can be represented as

$$RBD[i] = j / N$$

where

$$\begin{aligned} j &= dy && \text{when } i = 1 \\ j &= (N - dy) && \text{when } i = 0 \\ & && \forall\, dy \in \{abs(y) \mid y \in S_{(RD,E)}\} \end{aligned}$$

(2)

<u>Proof</u>

On expanding the equation (2), we get

a) $RBD[1] = dy / N$

b) $RBD[0] = (N - dy) / N$

Since the only element in the $\boldsymbol{S_{(RD,E)}}$ is the negative displacement of the early packet, ***dy*** will be equal to the absolute value of the sole element in that set.

From the definition of RBD [25], we know that the packet whose sequence number doesn't correspond with the expected sequence number is assumed to be buffered in the hypothetical buffer and the buffer occupancy of this hypothetical buffer determines the values of RBD.

While traversing a sequence consisting of a packet that is early by ***dy*** positions, the early packet would be buffered since the sequence number of the early packet doesn't correspond with the sequence number of the expected packet resulting in the buffer occupancy being 1. This packet would remain buffered until the expected sequence number corresponds to the sequence number of the buffered packet and upon the arrival of such an instance; the buffered packet is released into the sequence.

Thus the number of instances when buffer occupancy would have a value of 1 would be ***dy*** which is the displacement of the early packet. Therefore,

RBD [1] = dy / N

During the remaining packet arrival instances, the buffer occupancy would be zero. Therefore,

RBD [0] = (N – dy) / N

Q. E. D

4.1.2 Embedded Reordering Patterns

An embedded reordering pattern occurs when an early or late packet event is embedded within another early or late packet event. Consider the sequence *(i-1, i, i+1, i+2, i+3, i+4, i+5)*. If *packet i* is late by 4 positions and packet i + 1 is late by 2 positions, then the resulting sequence would *be (i-1, i+2, i+3, i+1, i+4, i, i+5)* resulting in a sequence with embedded reordering pattern.

Theorem 3

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an embedded reordering pattern caused by a late packet event embedded within another late packet event can be represented as

RBD [i] = j / N

where

$j = 1$ when $i \geq 1$ and $i \leq dx$

$j = (N - dx)$ when $i = 0$

$\forall\ dx \in \max(S_{(RD,L)})$

(3)

<u>Proof</u>

By expanding the equation (3), we get

a) RBD [1... dx] = 1 / N

b) RBD [0] = (N - dx) / N

In the case of an embedded reordering pattern formed by a late packet event embedded within another late packet event, the set $\boldsymbol{S_{(RD,L)}}$ would contain the displacement value of the late packets. The value of ***dx*** would be the maximum value of the 2 elements in the set since the outer late packet tends to be the packet with the expected sequence.

While traversing such a sequence, the sequence number of the expected packet would increment to the sequence number of the outer late packet thus causing the buffer occupancy to would keep increasing from 1 to ***dx***. During this process, the late packet that is embedded within the outer late packet event will also be buffered and released just like the other packets and therefore,

RBD [1… dx] = 1 / N

Since the remaining values correspond with zero buffer occupancy, we have

RBD [0] = (N - dx) / N

Q. E. D

Theorem 4

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an embedded reordering pattern caused by an early packet event embedded within a late packet event can be represented as

$$RBD[i] = j / N$$

where

$j = 1$ when $i \geq 1$ and $i \leq dx$

$j = (N - dx)$ when $i = 0$

$\forall\ dx \in S_{(RD,L)}$

(4)

<u>Proof</u>

Please refer to the proof for Theorem 3.

Q. E. D

Theorem 5

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an embedded reordering pattern caused by a late packet event embedded within an early packet event can be represented as

$$RBD[i] = j / N$$

where

$j = 1$ when $i = 2$ to dx

$j = (1 + dy - dx)$ when $i = 1$

$j = (N - dy)$ when $i = 0$

$\forall\ dx = \max(S_{(RD,L)})$ and $dy = \mathrm{abs}(\min(S_{(RD,E)}))$

(5)

Proof

By expanding the equation (5), we get

a) RBD [2 ... dx] = 1 / N

b) RBD [1] = (1 + dy – dx) / N

c) RBD [0] = (N - dy) / N

The values of ***dx*** and ***dy***, given the RD of the sequence, can be computed by calculating the absolute displacements of the late and early packets respectively which happens to be the highest and lowest values of the indices of RD.

While traversing a sequence consisting of an embedded reordering pattern caused by a late packet event embedded within an early packet event, the early packet is first encountered and stored in the hypothetical buffer causing buffer occupancy of 1. The minimum buffer occupancy would be 1 until the expected sequence number corresponds with the sequence number of the stored early packet {proved in Theorem 2}.

During the sequence traversal, if a packet is late, packets get buffered resulting in an increase in the buffer occupancy {proved in Theorem 1}. In this case, since the buffer already contains the early packet, once the expected sequence number is equal to the sequence number of the late packet; more packets are buffered until the actual late packet is encountered in the sequence. Therefore,

RBD [2 ... dx] = 1 / N

This is similar to the RBD values of an independent reordering pattern consisting of a single late packet. Since one extra packet is buffered, the index of RBD is offset by 1. Once the late packet is encountered all the packets except for the early packet are released causing the buffer occupancy of the hypothetical buffer to have a value of 1 for a few more instances. Since the displacement of the early packet is dy and the late packet would now be part of secondary events caused by the early packet, we therefore have

RBD [1] = (1 + dy – dx) / N

Since the hypothetical buffer during the remaining instances would have no occupancy, we have

RBD [0] = (N - dy) / N

Q. E. D

Theorem 6

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an embedded reordering pattern caused by an early packet event embedded within another early packet event can be represented as

$$RBD[i] = j / N$$

where

$j = (dx - dy - 1)$ when $i = 1$

$j = (dy + 1)$ when $i = 2$

$j = (N - dx)$ when $i = 0$

$\forall\ dx = abs(min_1(S_{(RD,E)}))$ and $dy = abs(min_2(S_{(RD,E)}))$

(6)

<u>Proof</u>

By expanding the equation (6), we get

a) $RBD[1] = (dx - dy - 1) / N$

b) $RBD[2] = (dy + 1) / N$

c) RBD [0] = (N - dx) / N

Given the RD of a sequence, the values of ***dx*** and ***dy*** can be computed from all the indices of RD that are non-zero. ***dx*** is the lowest index amongst all the indices and ***dy*** is the second lowest index.

We already know that the hypothetical buffer would have a single packet occupancy (buffer occupancy of 1) until the sequence number of the expected packet corresponds with the sequence number of the early packet {proved in Theorem 2}.

When an additional early packet is encountered during the traversal, the hypothetical buffer would contain one additional packet resulting in buffer occupancy of 2 packets. Since the early packet is embedded within, the displacement is actually offset by 1 and therefore we have

RBD [2] = (dy + 1) / N

Once the sequence number of the expected packet corresponds with the sequence number of the embedded early packet, it is released resulting in single packet buffer occupancy again. Therefore,

RBD [1] = (dx – dy – 1) / N

Since the hypothetical buffer during the remaining instances would have no occupancy,

RBD [0] = (N - dx) / N

Q. E. D

4.1.3 Overlapped Reordering Patterns

An overlapped reordering pattern occurs when an early or late packet event overlaps with another early or late packet event. Consider the sequence (*i-3, i-2, i-1, i, i+1, i+2, i+3, i+4, i+5*). If packets *i* and *i-2* are late by 3 and 4 positions respectively, then the resulting sequence would be (*i-3, i-1, i+1, i-2, i+2, i+3, i+4, i, i+5*) resulting in a sequence with an overlapped reordering event.

Lemma 1

In the absence of losses and duplicates, the length of the region of overlap between two overlapping late events in a sequence containing ***N*** packets can be can be represented as

$$\mathrm{Vxy} = \mathrm{RD}\,[-2] * N \qquad (7)$$

<u>Proof</u>

From the definition of RD [25], it is known that the secondary events of late packets result in a negative displacement of 1 for the packets that get displaced. When 2 such late events overlap, the packets in the region of overlap are part of the secondary events of both the events. This causes a negative displacement of 2. Since the sequences under consideration are assumed to have single reordering patterns, the sum of the instances of the displacement that correspond with -2 would be equal to the length of the overlap. Since RD is a normalized value, it is multiplied by N to de-normalize to obtain the overlap length

Q. E. D

Lemma 2

In the absence of losses and duplicates, the length of the region of overlap between a late event and an early event (or an early event and a late event) in a sequence containing ***N*** packets can be represented as

$$\mathrm{Vxy} = dx + dy - 2 - \left(\sum_{i=1}^{n} \mathbf{RD}\,[k] * N_i\right)$$

$\forall\ dx = \max\{S_{(RD,L)}\}$ and $dy = \text{abs}\{\min\{S_{(RD,E)}\}\}$

$\forall\ k \in \{x \mid x > 0, RD[x] > 0\}$

(8)

<u>Proof</u>

From the definition of RD [25], it is known that the secondary events of early packets result in a positive displacement of 1 for the packets that get displaced. It is also known that the secondary events of late packets result in a negative displacement of 1 for the packets that get displaced. When a late event overlaps with an early event (or an early event overlaps with a late event), the packets in the region of overlap would be part of both the events. In the region of overlap, the positive and negative displacements caused by the secondary events of both the packets cancel each other out resulting in zero displacement.

Lemma 3

In the absence of losses and duplicates, the length of the region of overlap between two overlapping early events in a sequence containing ***N*** packets can be can be represented as

$$Vxy = RD[2] * N$$

(9)

<u>Proof</u>

From the definition of RD [25], it is known that the secondary events of early packets result in a positive displacement of 1 for the packets that get displaced. When 2 such early events overlap, the packets in the region of overlap are part of the secondary events of both the events. This causes a positive displacement of 2. Since the sequences under consideration are assumed to have single reordering patterns, the sum of the instances of the displacement 2 would be equal to the length of the overlap. Since RD is a normalized value, it is multiplied by N to de-normalize to obtain the overlap length

Q. E. D

Theorem 7

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an overlapped reordering pattern caused by a late packet event overlapping with another late packet event can be represented as

$$RBD[i] = j / N$$

where

$$j = (N - dx - dy + Vxy - 1) \quad \text{when } i = 0$$

$$j = 1 \quad \text{when } i \geq 1 \text{ and } i < Vxy$$

$$j = 1 \quad \text{when } i > dx \text{ and } i \leq dy$$

$$j = 2 \quad \text{when } i \geq Vxy \text{ and } i \leq dx$$

$$\forall\ dx = \max_1(S_{(RD,L)}) \text{ and } dy = \max_2(S_{(RD,L)})$$

(10)

<u>Proof</u>

By expanding equation (10), we get

a) $RBD[1 \ldots (Vxy - 1)] = 1 / N$

b) $RBD[(dx+1) \ldots dy] = 1 / N$

c) $RBD[Vxy \ldots dx] = 2 / N$

d) $RBD[0] = (N - dx - dy + Vxy - 1) / N$

Given the value of RD, the values of ***dx*** and ***dy*** can be computed as the highest and the second highest displacement amongst the set of indices of RD.

From Lemma 1, we have

$Vxy = RD[-2] * N$

While traversing a sequence consisting of a late packet event overlapping with another late packet event, the hypothetical buffer gets filled until the sequence number of the expected packet corresponds with the sequence number of the first late packet. At this point, all the packets preceding the secondary events of the overlapping late event are released and ones corresponding with the overlapping late event are still buffered resulting in the hypothetical buffer containing the same number of packets during 2 instances.

Therefore, the number packets corresponding with the secondary events of the overlapping late event is equal to the length of overlap or

$RBD[Vxy] = 2/N$

On continuing the traversal, we find that the hypothetical buffer continues to store packets until the second late packet is encountered. Since there would be more instances when the buffer occupancy would be 2,

$\mathbf{RBD[Vxy \ldots dx] = 2/N}$

Beyond this point the buffer would have single buffer occupancy just like earlier, therefore

$\mathbf{RBD[1 \ldots (Vxy - 1)] = 1 / N}$

$\mathbf{RBD[Vxy \ldots dx] = 2 / N}$

Since the hypothetical buffer during the remaining instances would have no occupancy,

$\mathbf{RBD[0] = (N - dx - dy + Vxy - 1) / N}$

Q. E. D

Theorem 8

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an overlapped reordering pattern caused by a late packet event overlapping with an early packet event (or an early packet event overlapping with a late packet event) can be represented as

$$RBD[i] = j / N$$

where

$$j = (N - dx - dy + Vxy) \quad \text{when } i = 0$$

$$j = (dy - Vxy + 1) \quad \text{when } i = 1$$

$$j = 1 \quad \text{when } i \geq 2 \text{and } i \leq dx$$

$$\forall\ dx = \max\{S_{(RD,L)}\} \text{ and } dy = \text{abs}(\min\{S_{(RD,E)}\})$$

(11)

Proof

A late event causes the occupancy of the buffer to gradually increase until the sequence number of the expected packet matches with the sequence number of the current packet. Also, it is known that an early packet gets buffered until the sequence number of the expected packet corresponds with sequence number of the early packet. The length of overlap can be estimated using Lemma 2.

When an early event overlaps with a late event, the early packet gets buffered causing single buffer occupancy. This remains constant until the traversal reaches the secondary events of the late packet i.e., when we encounter the region of overlap. At this point, the other packets start getting buffered causing a gradual increase in the buffer occupancy until the late packet is encountered. Therefore, the number of instances when the hypothetical buffer would have single buffer occupancy would be {dy – Vxy + 1} resulting in RBD [2] to have a value of {dy – Vxy + 1} / N.

Since we have an early event overlapping with this late event, the early packet would be buffered resulting in an increase of RBD [1]. Hence, the value of RBD [1] can be obtained by subtracting the length of overlap by the absolute value of the displacement of the early packet. The remaining instances would correspond to zero buffer occupancy and hence {N – dx – dy + Vxy}/N.

When a late event overlaps with an early event, the gradual increase in the occupancy of the hypothetical buffer happens initially followed by the single buffer occupancy owing to the late packet. Therefore, the same equations hold good.

Q. E. D

Theorem 9

In the absence of losses and duplicates, the reorder buffer occupancy density of a sequence consisting of an overlapped reordering pattern caused by an early packet event overlapping with another early packet event can be represented as

$$RBD[i] = j / N$$

where

$$j = (N - dy - dx + 1) \quad \text{when } i = 0$$

$$j = (dy + dx - Vxy - 1) \quad \text{when } i = 1$$

$$j = Vxy \quad \text{when } i = 2$$

$$\forall\ dx = abs(min_1(S_{(RD,E)})) \text{ and } dy = abs(min_2(S_{(RD,E)}))$$

(12)

Proof

Since a single buffer is sufficient to resequence an early packet, for two early packets two buffers would be sufficient to do the same. Therefore, the possible indices of RBD would be 0, 1 and 2. Also, the length of overlap can be estimated using Lemma 3 which is equal to RD [2] * N.

While traversing the sequence, the first early packet gets buffered causing single buffer occupancy for dx number of instances. However, since another early packet event overlaps with this event, on encountering the second early packet, the buffer occupancy increases causing two packets to be stored in the hypothetical buffer. The buffer occupancy would remain constant with two packets until the first early packet gets released. The number of instances during the traversal of overlapped region corresponds to Vxy and therefore RBD [2] has a value of Vxy / N.

The number of instances when we have single buffer occupancy would be equal to the sum of the absolute displacement of the two early packets subtracted by the length of the overlap reduced by 1. The reduction by 1 is to factor out the overlapping early packet event.

Since the hypothetical buffer during the remaining instances would have no occupancy, RBD_n [0] has a value of {N – dy – dx + 1} / N.

Q. E. D

4.2 Verification and Analysis

The theorems presented in this chapter were verified using two sets of data - Internet traces available in the CNRL repository [63] and sequences generated using random probability distribution.

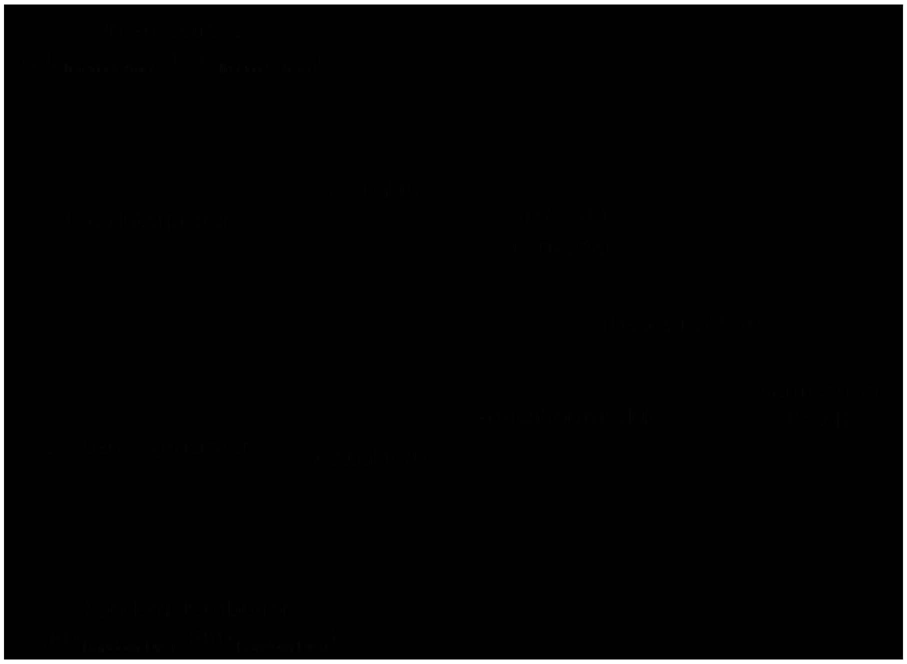


Figure 4-2. Verification process in deriving RBD from RD

The verification process can be better understood with the help of Figure 4-2. The verification can be broken into smaller modules namely

a) **Data interpreter**

The details about this module have already been presented in Chapter 3.

b) **Sequence Generator**

The details about this module have already been presented in Chapter 3.

c) **RD to RBD Converter**

This forms the crux of the verification process. RD of a sequence (could be an Internet trace or a sequence that was generated) is served as input to this module. All the theorems are mathematically represented in the form of modules. Based on the kind of pattern, an appropriate module would be selected and this output is sent to the verification module

d) **Verification module**

This module is slightly different from the one mentioned in Chapter 3 though the purpose of both the modules seem the same. It compares the theoretical RBD and the actual RBD to output the result of the verification process.

4.2.1 Illustrations

In this section, illustrations are provided to explain the derivation of RBD from RD in a clear manner. Since there are separate theorems and derivations for different reordering patters, this section is similar to the earlier ones.

a) Independent Reordering – Late event

Consider a reordered sequence (1, 2, 4, 5, 6, 7, 8, 9, 3, 10) where packet 3 is displaced by +6 positions (late). The computation of RD for this sequence is done in Table 4.2 and Table 4.3.

Table 4.2. Computation of FD, IR – Late packet

Arrived Sequence	1	2	4	5	6	7	8	9	3	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	0	-1	-1	-1	-1	-1	-1	6	0
FD [D]	1	2	1	2	3	4	5	6	1	3

Table 4.3. Computation of RD, IR – Late packet

D	-1	0	6
FD [D]	6	3	1
RD [D]	0.6	0.1	0.1

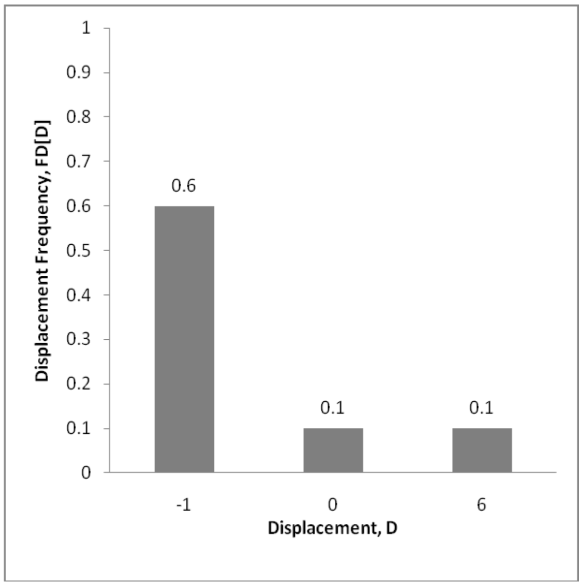


Figure 4-3. RD graph for an IR event caused by a late packet

From the Table 4.3, we have the following

RD [-1]=	0.6	
RD [0]	=	0.1
RD [6]	=	0.1

From ***Theorem 1***, we have

i.	RBD [0]	=	(N – dx) / N
ii.	RBD [1 .. dx]	=	1 / N

Here, N = 10 and dx = 6. Therefore, the computed values of RBD using the theorem would be

RBD [0]	=	0.4
RBD [1 … 6]	=	0.1

The computation of RBD using actual procedures is shown in Table 4.4 and Table 4.5.

Table 4.4. Computation of FB, IR – Late packet

Arrived Sequence	1	2	4	5	6	7	8	9	3	10
E	1	2	3	3	3	3	3	3	3	10
B	0	0	1	2	3	4	5	6	0	0
FB [B]	1	2	1	1	1	1	1	1	3	4

Table 4.5. Computation of RBD, IR – Late packet

B	0	1	2	3	4	5	6
FB [B]	4	1	1	1	1	1	1
RBD [B]	0.4	0.1	0.1	0.1	0.1	0.1	0.1

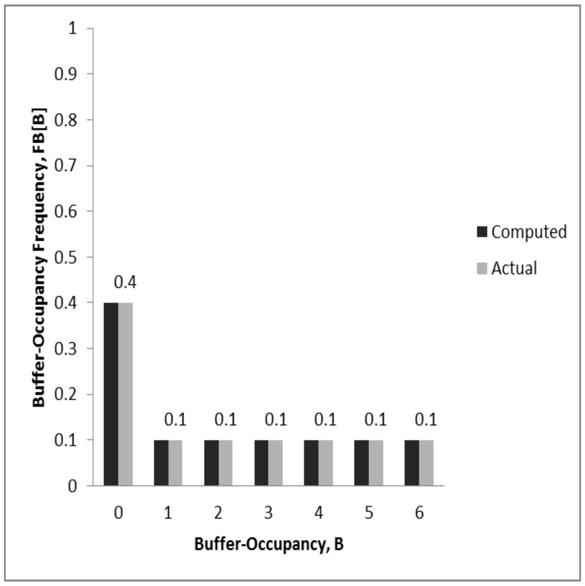


Figure 4-4. RBD graph showing the actual and computed values for an IR event caused by a late packet

From the Figure 4-4, we see that the computed and actual values of RBD are same for an IR event caused by a late packet.

b) Independent Reordering – Early event

Consider a reordered sequence (1, 2, 9, 3, 4, 5, 6, 7, 8, 10) where packet 9 is displaced by -6 positions (early). The computation of RD for this sequence is done in Table 4.6 and Table 4.7.

Table 4.6. Computation of FD, IR – Early packet

Arrived Sequence	1	2	9	3	4	5	6	7	8	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	0	-6	1	1	1	1	1	1	0
FD [D]	1	2	1	1	2	3	4	5	6	3

Table 4.7. Computation of RD, IR – Early packet

D	-6	0	1
FD [D]	1	3	6
RD [D]	0.1	0.3	0.6

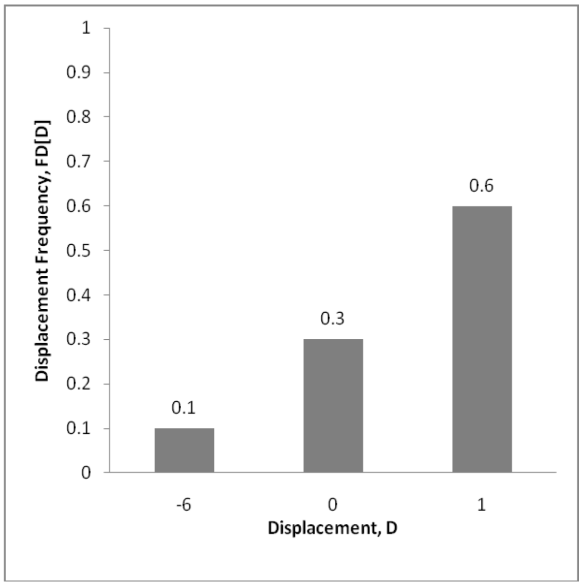


Figure 4-5. RD graph for an IR event caused by an early packet

From the Table 4.7, we have the following

RD [-6] = 0.1

RD [0] = 0.3

RD [1] = 0.6

From ***Theorem 2***, we have

i. RBD [1] = dy / N

ii. RBD [0] = (N – dy) / N

Here, N = 10 and dy = 6. Therefore, the computed values of RBD using the theorem would be

RBD [1] = 0.6

RBD [0] = 0.4

The computation of RBD using actual procedures is shown in Table 4.8 and Table 4.9

Table 4.8. Computation of FB, IR – Early packet

Arrived Sequence	1	2	9	3	4	5	6	7	8	10
E	1	2	3	3	3	3	3	3	3	10
B	0	0	1	1	1	1	1	1	0	0
FB [B]	1	2	1	2	3	4	5	6	3	4

Table 4.9. Computation of RBD, IR – Early packet

B	0	1
FB [B]	4	6
RBD [B]	0.4	0.6

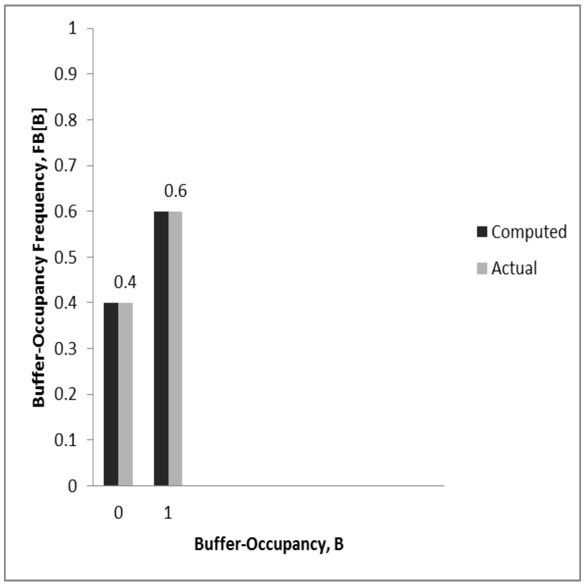


Figure 4-6. RBD graph showing the actual and computed values for an IR event caused by an early packet

From the Figure 4-6, we see that the computed and actual values of RBD are same for an IR event caused by an early packet.

c) Embedded Reordering – Lateness embeds Lateness

Consider a reordered sequence (1, 3, 4, 6, 7, 5, 8, 9, 2, 10) where packets 2 and 5are displaced by +7 (late) and +1 positions (late) respectively. The computation of RD for such a sequence is done in Table 4.10 and

Table 4.11.

Table 4.10. Computation of FD, ER - Lateness embeds Lateness

Arrived Sequence	1	3	4	6	7	5	8	9	2	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	-1	-1	-2	-2	1	-1	-1	7	0
FD [D]	1	1	2	1	2	1	3	4	1	2

Table 4.11. Computation of RD, ER - Lateness embeds Lateness

D	-2	-1	0	1	7
FD [D]	2	4	2	1	1
RD [D]	0.2	0.4	0.2	0.1	0.1

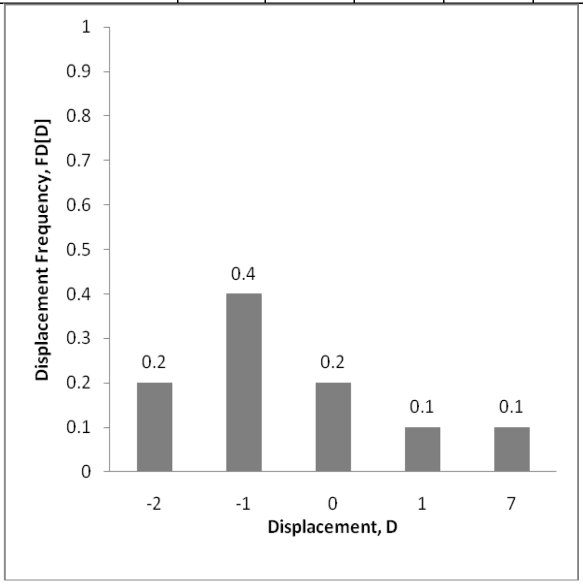


Figure 4-7. RD graph for an ER event caused by a late packet event embedded within a late packet event

From the

Table 4.11, we have the following

RD [-2] = 0.2; RD [-1] = 0.4; RD [0] = 0.2; RD [1] = 0.1; RD [7] = 0.1

From ***Theorem 3***, we have

i. RBD [1 … dx] = 1 / N

ii. RBD [0] = (N – dx) / N

Here, N = 10 and dx = 7. Therefore, the computed values of RBD using the theorem would be

RBD [1… 7] = 0.1

RBD [0] = 0.3

The computation of RBD using actual procedures is shown in

Table 4.12and Table 4.13.

Table 4.12. Computation of FB, ER - Lateness embeds Lateness

Arrived Sequence	1	3	4	6	7	5	8	9	2	10
E	1	2	2	2	2	2	2	2	2	10
B	0	1	2	3	4	5	6	7	0	0
FB [B]	1	1	1	1	1	1	1	1	2	3

Table 4.13. Computation of RBD, ER - Lateness embeds Lateness

B	0	1	2	3	4	5	6	7
FB [B]	3	1	1	1	1	1	1	1
RBD [B]	0.3	0.1	0.1	0.1	0.1	0.1	0.1	0.1

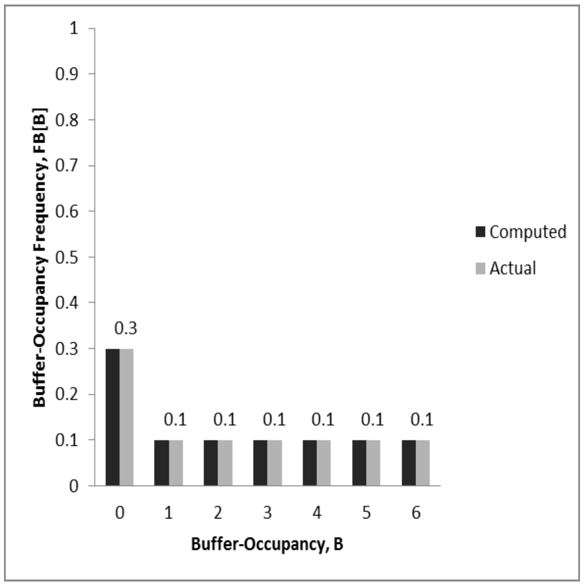


Figure 4-8. RBD graph showing the actual and computed values for an ER event caused by a late packet event embedded within another late packet event

From Figure 4-8, we see that the computed and actual values of RBD for an embedded sequence formed by a late event embedded within another late event are the same.

d) Embedded Reordering – Lateness embeds Earliness

Consider a reordered sequence (1, 3, 4, 8, 5, 6, 7, 9, 2, 10) where packets 2 and 8 are displaced by +7 (late) and -4 positions (early) respectively. The computation of RD for this sequence is done in Table 4.14 and Table 4.15.

Table 4.14. Computation of FD, ER - Lateness embeds Earliness

Arrived Sequence	1	3	4	8	5	6	7	9	2	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	-1	-1	-4	0	0	0	-1	7	0
FD [D]	1	1	2	1	2	3	4	3	1	5

Table 4.15. Computation of RD, ER - Lateness embeds Earliness

D	-4	-1	0	7
FD [D]	1	3	5	1
RD [D]	0.1	0.3	0.5	0.1

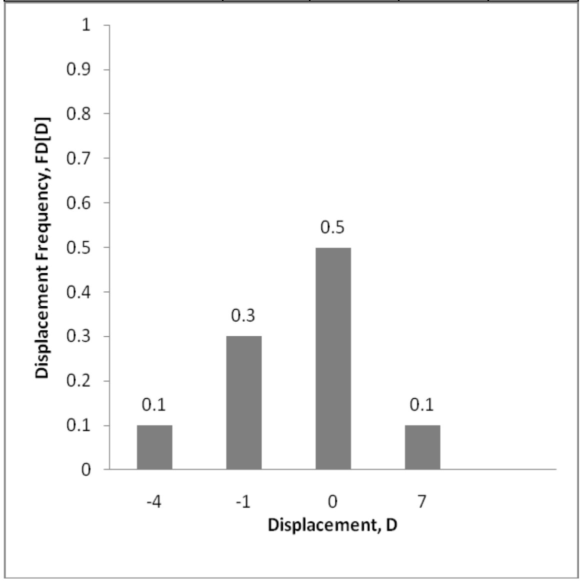


Figure 4-9. RD graph for an ER event caused by an early packet event embedded within a late packet event

From the Table 4.15, we have the following

RD [-4] = 0.1; RD [-1] = 0.3; RD [0] = 0.5; RD [7] = 0.1

From ***Theorem 4***, we have

i. RBD [1 … dx] = 1 / N

ii. RBD [0] = (N – dx) / N

Here, N = 10 and dx = 7. Therefore, the computed values of RBD using the theorem would be

RBD [1 … 7] = 0.1

RBD [0] = 0.3

The computation of RBD using actual procedures is shown in Table 4.16 and Table 4.17

Table 4.16. Computation of FB, ER - Lateness embeds Earliness

Arrived Sequence	1	3	4	8	5	6	7	9	2	10
E	1	2	2	2	2	2	2	2	2	10
B	0	1	2	3	4	5	6	7	0	0
FB [B]	1	1	1	1	1	1	1	1	2	3

Table 4.17. Computation of RBD, ER - Lateness embeds Earliness

B	0	1	2	3	4	5	6	7
FB [B]	3	1	1	1	1	1	1	1
RBD [B]	0.3	0.1	0.1	0.1	0.1	0.1	0.1	0.1

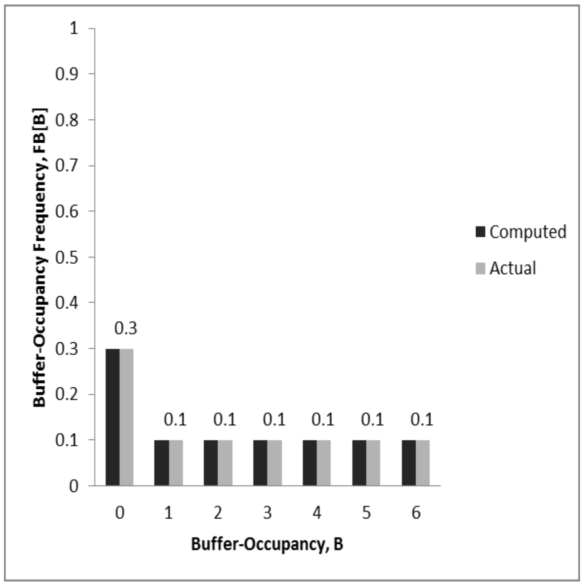


Figure 4-10. RBD graph showing the actual and computed values for an ER event caused by an early packet event embedded within a late packet event

From Figure 4-10, we see that the computed and actual values of RD for an embedded reordering sequence formed by an early event embedded within a late event are the same.

e) Embedded Reordering – Earliness embeds Lateness

Consider a reordered sequence (1, 9, 2, 4, 5, 6, 3, 7, 8, 10) where packets 3 and 9 are displaced by +4 (late) and -7 positions (early) respectively. The computation of RD for this sequence is done in Table 4.18 and Table 4.19.

Table 4.18. Computation of FD, ER - Earliness embeds Lateness

Arrived Sequence	1	9	2	4	5	6	3	7	8	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	-7	1	0	0	0	4	-1	1	0
FD [D]	1	1	1	2	3	4	1	1	2	5

Table 4.19. Computation of RD, ER - Earliness embeds Lateness

D	-7	-1	0	1	4

FD [D]	1	1	5	2	1
RD [D]	0.1	0.1	0.5	0.2	0.1

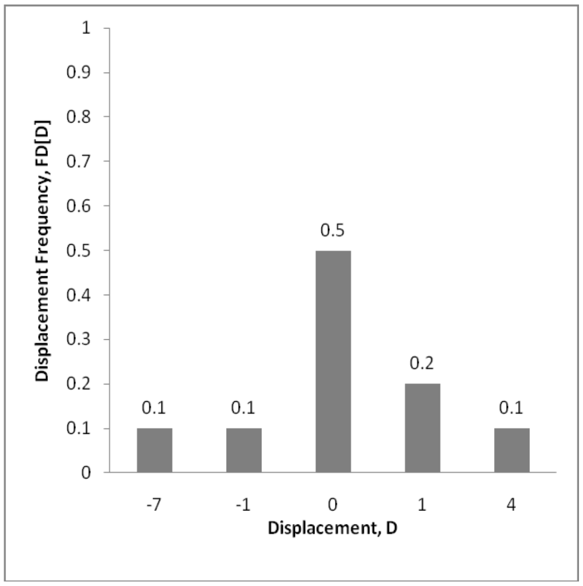


Figure 4-11. RD graph for an ER event caused by a late packet event embedded within an early packet event

From the Table 4.19, we have the following

RD [-7] = 0.1; RD [-1] = 0.1; RD [0] = 0.5; RD [1] = 0.2; RD [4] = 0.1

From ***Theorem 5***, we have

i. RBD [2 ... dx] = 1 / N

ii. RBD [1] = (1 + dy– dx) / N

iii. RBD [0] = (N - dy) / N

Here, N = 10, dx = 4 and dy = 7. Therefore, the computed values of RBD using the theorem would be

RBD [2 … 4] = 0.1

RBD [1] = 0.4

RBD [0] = 0.3

The computation of RBD using actual procedures is shown in

Table 4.20 and Table 4.21

Table 4.20. Computation of FB, ER - Earliness embeds Lateness

Arrived Sequence	1	9	2	4	5	6	3	7	8	10
E	1	2	2	3	3	3	3	7	8	10
B	0	1	0	2	3	4	1	1	1	0
FB [B]	1	1	2	1	1	1	2	3	4	3

Table 4.21. Computation of RBD, ER - Earliness embeds Lateness

B	0	1	2	3	4
FB [B]	3	4	1	1	1
RBD [B]	0.3	0.4	0.1	0.1	0.1

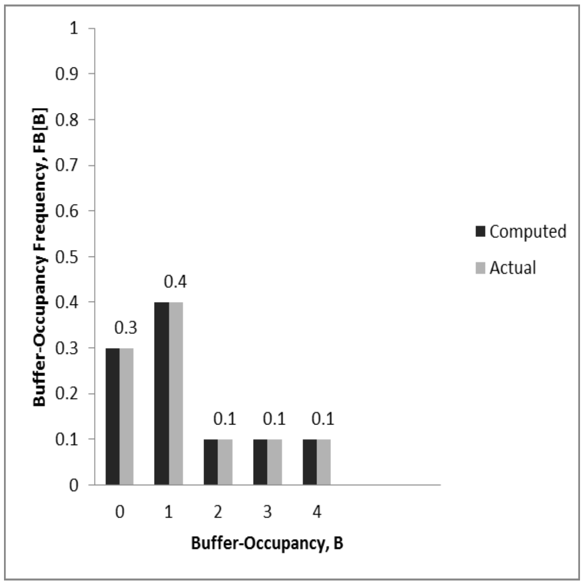


Figure 4-12. RBD graph showing the actual and computed values for an ER event caused by a late packet event embedded within an early packet event

From Figure 4-12, we see that the actual and computed values for an embedded reordering event formed by a late event embedded within an early event are the same.

f) Embedded Reordering – Earliness embeds Earliness

Consider a reordered sequence (1, 9, 2, 6, 3, 4, 5, 7, 8, 10) where packets 6 and 9 displaced by -2 (early) and -7 (early) positions respectively. The computation of RD for this sequence is done in Table 4.22 and Table 4.23.

Table 4.22. Computation of FD, ER - Earliness embeds Earliness

Arrived Sequence	1	9	2	6	3	4	5	7	8	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	-7	1	-2	2	2	2	1	1	0
FD [D]	1	1	1	1	1	2	3	2	3	2

Table 4.23. Computation of RD, ER - Earliness embeds Earliness

D	-7	-2	0	1	2
FD [D]	1	1	2	3	3
RD [D]	0.1	0.1	0.2	0.3	0.3

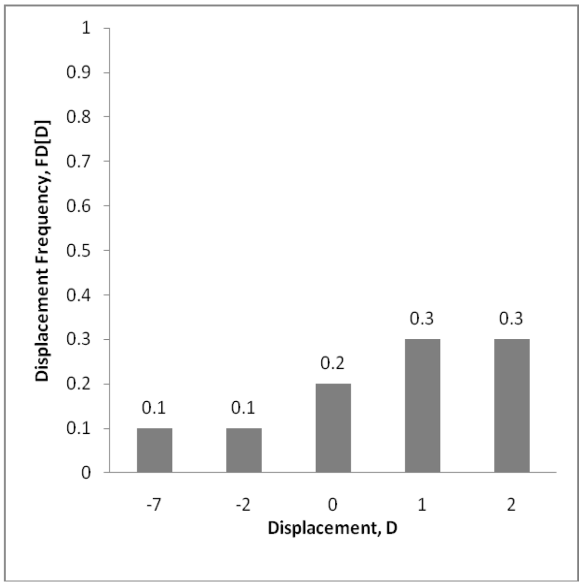


Figure 4-13. RD graph for an ER event caused by an early packet event embedded within another early packet event

From the Table 4.23, we have the following

RD [-7] = 0.1; RD [-2] = 0.1; RD [0] = 0.2; RD [1] = 0.3; RD [2] = 0.3

From ***Theorem 6***, we have

i.	RBD [1]	=	(dx – dy – 1) / N
ii.	RBD [2]	=	(dy + 1) / N
iii.	RBD [0]	=	(N - dx) / N

Here, N = 10, dx = 7 and dy = 2. Therefore, the computed values of RBD using the theorem would be

RBD [1]	=	0.4
RBD [2]	=	0.3
RBD [0]	=	0.3

The computation of RBD using actual procedures is shown in Table 4.24 and Table 4.25

Table 4.24. Computation of FB, ER - Earliness embeds Earliness

Arrived Sequence	1	9	2	6	3	4	5	7	8	10
E	1	2	2	3	3	4	5	7	8	10
B	0	1	1	2	2	2	1	1	0	0
FB [B]	1	1	2	1	2	3	3	4	2	3

Table 4.25. Computation of RBD, ER - Earliness embeds Earliness

B	0	1	2
FB [B]	3	4	3
RBD [B]	0.3	0.4	0.3

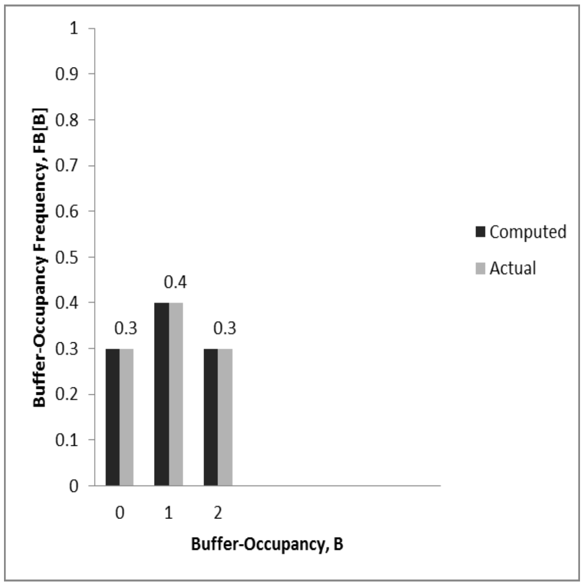


Figure 4-14. RBD graph showing the actual and computed values for an ER event caused by an early packet event embedded within another early packet event

From Figure 4-14 we see that the actual and computed values of an embedded reordering event formed by an early event embedded within another early event are the same.

g) Overlapped Reordering – Lateness overlaps Lateness

Consider a reordered sequence (1, 3, 5, 6, 7, 2, 8, 9, 4, 10) where packets 2 and 4 are displaced by +4 and +5 positions respectively. The computation of RD for this sequence is done in Table 4.26 and Table 4.27.

Table 4.26. Computation of FD, OR - Lateness overlaps Lateness

Arrived Sequence	1	3	5	6	7	2	8	9	4	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	-1	-2	-2	-2	4	-1	-1	5	0
FD [D]	1	1	1	2	3	1	2	3	1	2

Table 4.27. Computation of RD, OR - Lateness overlaps Lateness

D	-2	-1	0	4	5
FD [D]	3	3	2	1	1
RD [D]	0.3	0.3	0.2	0.1	0.1

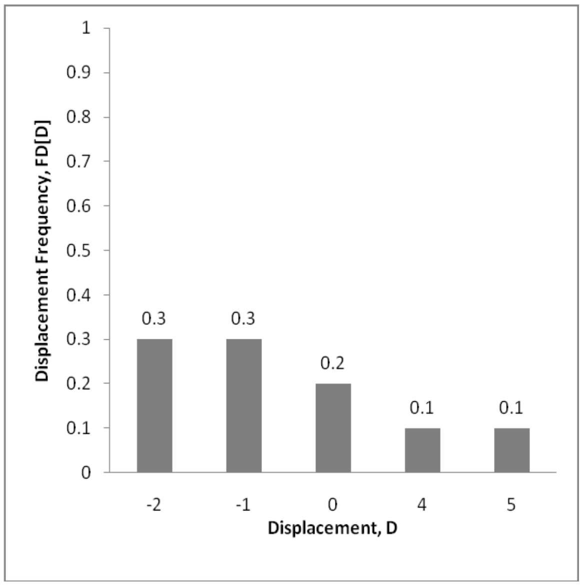


Figure 4-15. RD graph for an OR event caused by a late packet event overlapping within another late packet event

From the Table 4.27, we have the following

RD [-2] = 0.3; RD [-1] = 0.3; RD [0] = 0.2; RD [4] = 0.1; RD [5] = 0.1

From ***Theorem** 7*, we have

i.	RBD [1 … (Vxy – 1)]	=	1 / N
ii.	RBD [(dx+1) … dy]	=	1 / N
iii.	RBD [Vxy … dx]	=	2 / N
iv.	RBD [0]	=	(N –dx –dy + Vxy – 1) / N

Here,

N = 10, dx = 4, dy = 5

From Lemma 1, we have

Vxy = 3

Therefore, the computed values of RBD using the theorem would be

RBD [1 ... 2]	=	0.1
RBD [3… 4]	=	0.2
RBD [5]	=	0.1
RBD [0]	=	0.3

The computation of RBD using actual procedures is shown in Table 4.28 and Table 4.29.

Table 4.28. Computation of FB, OR - Lateness overlaps Lateness

Arrived Sequence	1	3	5	6	7	2	8	9	4	10
E	1	2	2	2	2	4	4	4	4	10
B	0	1	2	3	4	3	4	5	0	0
FB [B]	1	1	1	1	1	2	2	1	2	3

Table 4.29. Computation of RBD, OR - Lateness overlaps Lateness

B	0	1	2	3	4	5
FB [B]	3	1	1	2	2	1

RBD [B]	0.3	0.1	0.1	0.2	0.2	0.1

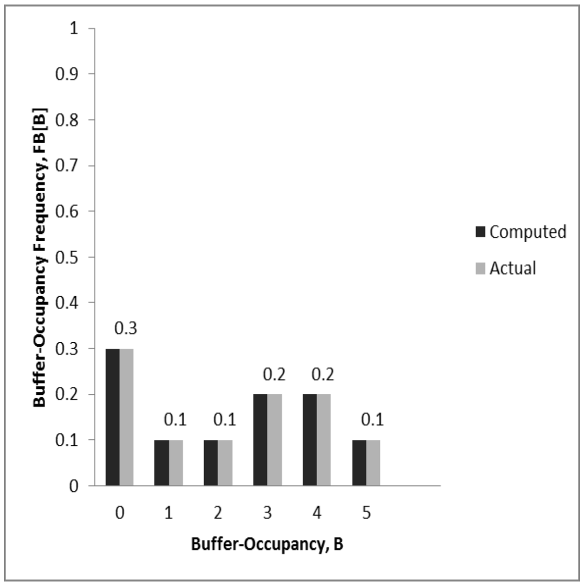


Figure 4-16. RBD graph showing the actual and computed values for an OR event caused by a late packet event overlapping within another late packet event

From Figure 4-16, we see that the actual and computed values of an overlapped reordering event formed by a late event overlapping with another late event are the same.

h) Overlapped Reordering – Lateness overlaps Earliness

Consider a reordered sequence (1, 3, 8, 4, 5, 2, 6, 7, 9, 10) where packets 2 and 8 get displaced. The computation of RD for this sequence is done in Table 4.30 and Table 4.31.

Table 4.30. Computation of FD, OR - Lateness overlaps Earliness

Arrived Sequence	1	3	8	4	5	2	6	7	9	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	-1	-5	0	0	4	1	1	0	0

FD [D]	1	1	1	2	3	1	1	2	4	5

Table 4.31. Computation of RD, Overlapped Reordering, Late overlaps Early

D	-5	-1	0	1	4
FD [D]	1	1	5	2	1
RD [D]	0.1	0.1	0.5	0.2	0.1

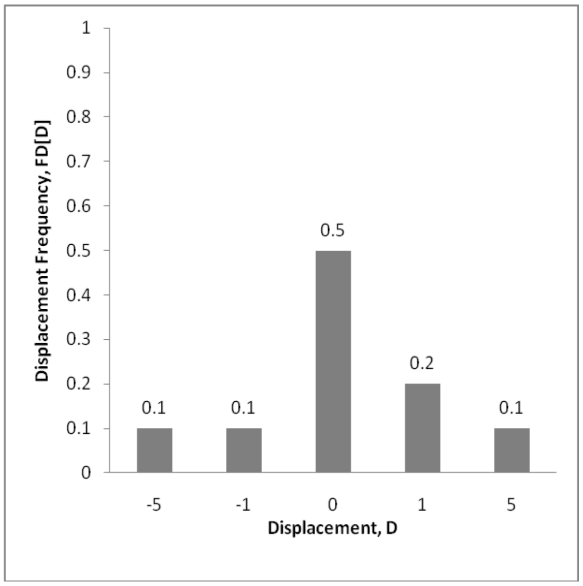


Figure 4-17. RD graph for an OR event caused by a late packet event overlapping within an early packet event

From the Table 4.31, we have the following

RD [-5] = 0.1; RD [-1] = 0.1; RD [0] = 0.5; RD [1] = 0.2; RD [4] = 0.1

From ***Theorem 8***, we have

i. RBD [1] = (dy – Vxy + 1) / N

ii. RBD [2 … dx] = 1 / N

iii. RBD [0] = (N – dx – dy + Vxy) / N

Here, N = 10, dx = 4 and dy = 5.

From Lemma 2, we have

Vxy = 2

Therefore, the computed values of RBD using the theorem would be

RBD [1]	=	0.4
RBD [2 … 4]	=	0.1
RBD [0]	=	0.3

The computation of RBD using actual procedures is shown in Table 4.32 and Table 4.33

Table 4.32. Computation of FB, OR - Lateness overlaps Earliness

Arrived Sequence	1	3	8	4	5	2	6	7	9	10
E	1	2	2	2	2	2	6	7	9	10
B	0	1	2	3	4	1	1	1	0	0
FB [B]	1	1	1	1	1	2	3	4	2	3

Table 4.33. Computation of RBD, OR - Lateness overlaps Earliness

B	0	1	2	3	4
FB [B]	3	4	1	1	1
RBD [B]	0.3	0.4	0.1	0.1	0.1

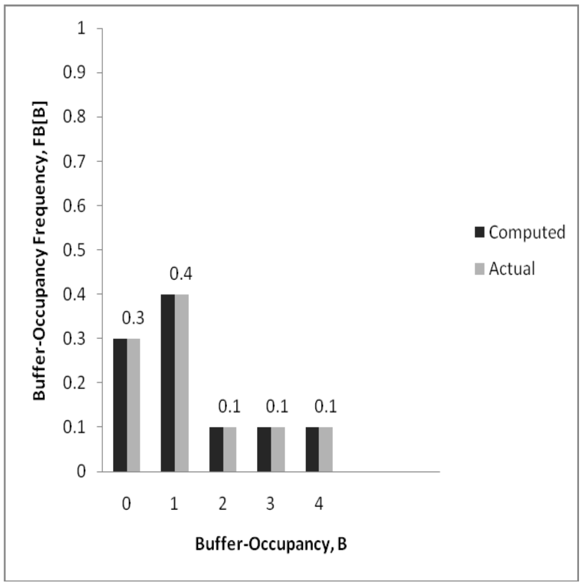


Figure 4-18. RBD graph showing the actual and computed values for an OR event caused by a late packet event overlapping within an early packet event

From Figure 4-18, we see that the actual and computed values of an overlapped reordering event formed by a late event overlapping with an early event are the same.

i) Overlapped Reordering – Earliness overlaps Earliness

Consider a reordered sequence (1, 2, 4, 5, 6, 7, 8, 9, 3, 10) where packet 3 is displaced by +6 positions (late). The computation of RD for this sequence is done in Table 4.34 and Table 4.35.

Table 4.34. Computation of FD, OR - Earliness overlaps Earliness

Arrived Sequence	1	6	2	3	9	4	5	7	8	10
RI	1	2	3	4	5	6	7	8	9	10
D	0	-4	1	1	-4	2	2	1	1	0
FD [D]	1	1	1	2	2	1	2	3	4	2

Table 4.35. Computation of RD, OR - Earliness overlaps Earliness

D	-4	0	1	2
FD [D]	2	2	4	2
RD [D]	0.2	0.2	0.4	0.2

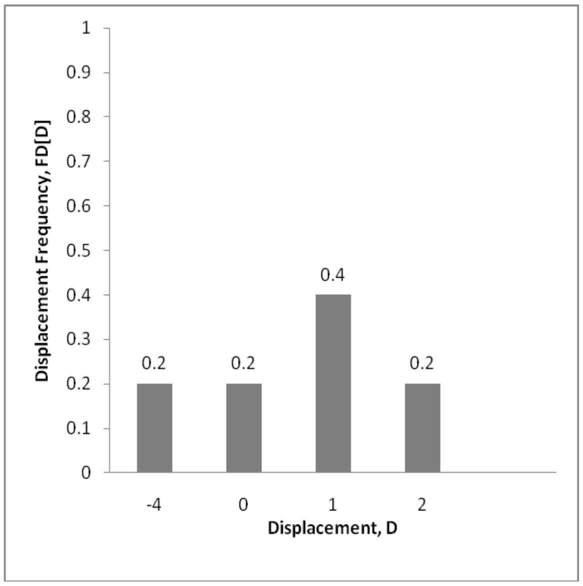


Figure 4-19. RD graph for an OR event caused by an early packet event overlapping within another early packet event

From the Table 4.35, we have the following

RD [-4] = 0.2; RD [0] = 0.2; RD [1] = 0.4; RD [2] = 0.2

From ***Theorem 9***, we have

i.	RBD [2]	=	Vxy / N
ii.	RBD [1]	=	(dy + dx - Vxy – 1) / N
iii.	RBD [0]	=	(N – dy – dx + 1) / N

Here, N = 10, dx = 4 and dy = 4.

From Lemma 3, we have

Vxy = 2

Therefore, the computed values of RBD using the theorem would be

RBD [2]	=	0.2
RBD [1]	=	0.5

RBD [0] = 0.3

The computation of RBD using actual procedures is shown in Table 4.36 and Table 4.37

Table 4.36. Computation of FB, OR - Earliness overlaps Earliness

Arrived Sequence	1	6	2	3	9	4	5	7	8	10
E	1	2	2	3	4	4	5	7	8	10
B	0	1	1	1	2	2	1	1	0	0
FB [B]	1	1	2	3	1	2	4	5	2	3

Table 4.37. Computation of RBD, OR - Earliness overlaps Earliness

B	0	1	2
FB [B]	3	5	2
RBD [B]	0.3	0.5	0.2

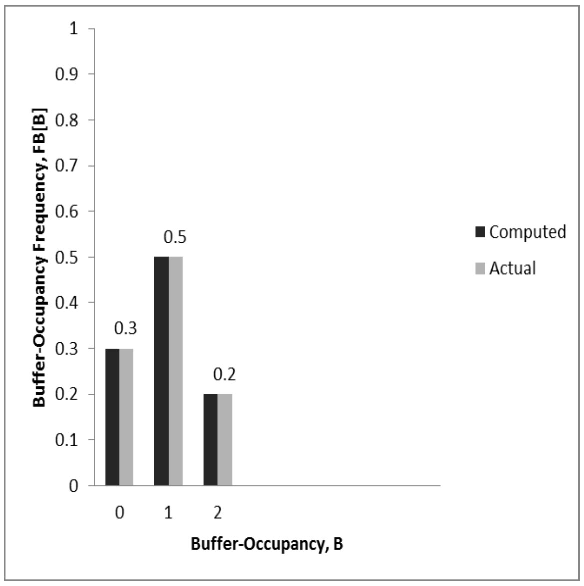


Figure 4-20. RBD graph showing the actual and computed values for an OR event caused by an early packet event overlapping within another early packet event

From Figure 4-20, we see that the actual and computed values of an overlapped reordering event formed by an early event overlapping with another early event are the same.

4.3 Conclusion

Based on the theorems presented in this chapter, it can be seen that the Reorder Buffer-Occupancy Density can be calculated using the Reorder Density of a sequence. The theorems were verified using Internet traces available at the CNRL repository [63] and random sequences. The results of the verification process showed that given the kind of reordering pattern, the RD of the sequence and the number of packets in the sequence, one can easily obtain the RBD of the sequence using these theorems.

While designing network architectures and components, RBD serves as a more useful metric to allocate resources required for negating packet reordering. But, it is more likely that RD will be used as a metric in QoS given the probability of displacement frequency that RD provides. Therefore, the research provided in this chapter would serve as a useful tool to measure RBD from RD, thereby enabling the usage of at least RD if not RBD.

CHAPTER 5. CONCLUSION

In this research, analytical results to estimate the impact of re-sequencing buffers on packet reordering have been presented. This is one of the first such researches to analyze the impact of re-sequencing nodes on degree and extent of packet reordering. With the presented results, the variation of reorder density and reorder buffer occupancy density as a packet stream goes through a resequencing node can be characterized. The proposed theorems were verified using simulation study based on the traces available at [63] and were found to be accurate. Such estimation gives valuable insight about the quantity of resources required to resequence packets in reordered sequences. With increase in network speeds and parallelism across network components, packet reordering is bound to play a much more important role in the future. Internet service providers might even have to include packet reordering metric as one of their QoS parameters which means across the Internet a number of such ISPs will have to estimate and sync their QoS parameters. Router manufacturers already include extra hardware for mitigating packet reordering. This research presents the theoretical basis for estimation of buffer resources to meet target goals and to evaluate the impact of size and placement of buffers on reordering in packet flows.

CHAPTER 6. FUTURE WORK

The research holds good for sequences with a single type of reordering pattern whose reordering pattern in known in advance. In future, the theorems can be extended to accommodate multiple instances of specific reordering patterns. Theorems can be devised to obtain the Reorder Density (RD) of a single instance of reordering pattern, given the overall RD of a sequence containing multiple instances of different reordering patterns. The impact of distribution of resequencing buffers on such sequences containing multiple instances of reordering patterns can be studied in great detail to emulate real-life implementations.

The theorems assume the existence of resequencing mechanism when all packets from a single flow (or sequence) converge onto a single network node and continue to do so thus enabling resequencing. But, in reality, this could be an interesting problem since the packets from a single flow (or sequence) may or may not converge much before reaching the destination. The effect of latency on getting the packets back into order is also an interesting problem.

The primary purpose of Reorder Density (RD) and Reorder Buffer-Occupancy Density (RBD) is to measure packet reordering in computer networks. When one considers the larger perspective of these metrics which basically measure reordering and quantify the effect in terms of buffers, we find that they have more applications than one could have ever thought. Reordering is a common phenomenon across the world. Any pattern that follows a sequence can undergo reordering thereby affecting the status quo.

6.1 Packet Reordering in Mobile Networks

One of the reasons for the tremendous growth of Internet has been the availability of Internet connectivity on mobile devices. The Internet has become ubiquitous due to vast information that is easily available to the users at their fingertips. All this is happening despite the fact that mobile connectivity suffers when compared to broadband service, in terms of quality of service. When more and more users start using mobile devices for connecting to Internet, the focus would shift from just connectivity to quality of connectivity and when that happens, Internet Service Providers would start to focus on QoS and packet reordering happens to one of the critical things affecting the performance as already seen.

As mentioned earlier, in radio access technology for the next generation, Long Term Evolution and Ultra Mobile Broadband are getting standardized. Studies conducted in these areas are focusing on the overhead involved in avoiding packet reordering. Such studies indicate the importance of tackling the problem of packet reordering and its solutions in the mobile and broadband industry.

6.2 Reordering in areas outside computer networks

The concepts of RD and RBD may be applied to other streams where reordering is a common phenomenon. Medical science is one such area where recent achievements have been accomplished in the study of patterns in the human body such as DNA / RNA sequencing. Manufacturing is another such area where machines are used to carry out systematic and monotonous work. Machines are basically instruction based systems which lack the idea of cognition that a human being does. When multiple machines communicate or depend on each other, sequencing is important and the so-called machines need to adhere to a strict code or pattern. Almost all manufacturing units are sequence based and lack of order in such sequences can result in insurmountable losses.

REFERENCES

[1] C. M. Arthur, D. Girma, D. Harle and A. Lehane, “The Effects of Packet Reordering in a Wireless Multimedia Environment,” Wireless Comm. Systems, pp. 453 – 457, Sep 2004.

[2] C. M. Arthur, A. Lehane and D. Harle, “Keeping Order: Determining the Effect of TCP Packet Reordering,” Networking and Services, 2007 (ICNS’07) pp. 116 – 116, June 2007.

[3] T. Banka, A. A. Bare and A. P. Jayasumana, “Metrics for Degree of Reordering in Packet Sequences,” Local Computer Networks (LCN 2002), pp. 333 – 342, 6-8 Nov. 2002.

[4] A. A. Bare, “Measurement and Analysis of Packet Reordering,” M.S Thesis, Colorado State University, Nov. 2004.

[5] A. A. Bare, A. P. Jayasumana and N. M. Piratla, “On growth Of Parallelism within Routers and Its Impact on Packet Reordering,” Proc. 15th IEEE Workshop on Local and Metropolitan Area Networks, Princeton NJ, pp. 145-150, June 2007.

[6] J. Bellardo and S. Savage, “Measuring Packet Reordering,” Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement (IMW '02), pp. 97-105, 2002.

[7] J. C. R. Bennett, C. Partridge and N. Shectman, “Packet Reordering is Not Pathological Network Behavior,“ IEEE/ACM Trans Networking, pp. 789-798, Dec. 1999.

[8] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim and K. Obraczka, “A New TCP for Persistent Packet Reordering,” IEEE/ACM Trans. Networks, vol. 14(2),pp. 369-382, Apr. 2006.

[9] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim and K. Obraczka, “TCP-PR: TCP for Persistent Packet Reordering,” Distributed Comp. Systems, 2003, pp. 222-231, May 2003.

[10] W. Chao, Z. Xingming, C. Wenping and N. Xiaona, "Load Balancing Algorithm Using Flow Chopping to Avoid Packet Reordering," Information Tech. and Applications, 2009 (IFITA'09), pp. 193-197, May 2009.

[11] L. Daniel, I. Jarvinen and M. Kojo, "Combating Packet Reordering in Vertical Handoff using Cross-layer Notifications to TCP," Wireless Mobile Computing, Networking & Communication (WIMOB '08), pp. 297-303, Oct. 2008.

[12] Y. Dong, D. Wang, N. Pissinou and J. Wang, "Multi-path Load Balancing In Transport Layer," Next Generation Internet Networks, pp. 135-142, May 2007.

[13] K. Evensen, D. Kasper, P. Engelstad, A. Hansen, C. Griwodz and P. Halvorsen, "A Network-Layer Proxy for Bandwidth Aggregation and Reduction of IP Packet Reordering," Local Computer Networks, 2009 (LCN'09), pp. 585-592, Oct. 2009.

[14] J. Feng, Z. Ouyang, L. Xu and B. Ramamurthy, "Packet reordering in high-speed networks and its impact on high-speed TCP variants," Computer Comm, vol. 32(1), Jan 2009.

[15] J. C. Fernandez, T. Taleb, M. Guizani and N. Kato, "Bandwidth Aggregation-Aware Dynamic QoS Negotiation for Real-Time Video Streaming in Next-Generation Wireless Networks," IEEE Trans. On Multimedia, pp. 1082-1093, Oct. 2009.

[16] Z. Fu, B. Greenstein, X. Meng and S. Lu, "Design and Implementation of a TCP-friendly Transport Protocol for Ad Hoc Wireless Networks," Network Protocols, 2002, pp. 216-225, Nov. 2002.

[17] X. Fu, W. Yu, S. Jiang, S. Graham and Y. Guan, "TCP performance in a flow-based mix networks: Modeling and analysis," IEEE Trans. On Parallel and Distributed Systems, vol. 20(5), pp. 695 – 709, May 2009.

[18] L. Gharai, C. Perkins and T. Lehman, "Packet Reordering, High Speed Networks and Transport Protocol Performance," Computer Comm. and Networks (ICCCN 2004), pp. 73 – 78, 11-13 Oct. 2004.

[19] S. Govind, R. Govindarajan and J. Kuri, "Packet Reordering in Network Processors," Parallel and Dist. Proc. Sym. (IPDPS 2007), pp. 1 – 10, 26-30 Mar. 2007.

[20] S. Gunreben and G. Hu, "A Multi-layer Analysis Of Reordering in Optical Burst Switched Networks," IEEE Communication Letters, pp. 1013-1015, Dec. 2007.

[21] P. Hurtig and A. Brunstrom, "Emulation Support for Advanced Packet Reordering Models," Communications (ICC'10), pp. 1-6, May 2010.

[22] G. Iannaccone, S. Jaiswaland C. Diot, "Packet reordering inside the Sprint backbone," Tech. Report, TR01-ATL-062917, Sprint ATL, June 2001.

[23] G. Istrate, A. Hansson and G. Yan, "Packet Reordering Metrics: Some Methodological Considerations," Networking and Services, 2006 (ICNS'06), pp. 4-4, July 2006.

[24] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose and D. Towsley, "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone," IEEE/ACM Trans. On Networking, pp. 54-66, Feb 2007.

[25] A. Jayasumana, N. Piratla, T. Banka, A. Bare and R. Whitner, "Improved Packet Reordering Metrics," IETF RFC 5236.

[26] D. Kaspar, K. Evensen, A. F. Hansen, P. Engelstad, P. Halvorsen and C. Griwodz, "An Analysis of the Heterogeneity and IP Packet Reordering over Multiple Wireless Networks," Computers and Communications, 2009 (ISCC'09), pp. 637-642, July 2009.

[27] D. P. Kim, S. J. Koh and V. Leung, "On the Packet Reordering of mSCTP for Vertical Handover in Heterogonous Wireless Networks," Vehicular Tech. Conf, 2008 (VTC'08), pp. 1-5, Sep 2008.

[28] W. C. Kwon, S. Yoo, J. Um, S. W. Jeong, "In-Network Reorder Buffer To Improve Overall NoC Performance While Resolving the In-Order Requirement Problem," Design, Automation & Test in Europe Conference & Exhibition, 2009, pp. 1058-1063, Apr. 2009.

[29] C. Lai, K. C. Leung and V. O. K. Li, "Enhancing Wireless TCP: A Serialized-Timer Approach," INFOCOM, 2010, pp. 1-5, Mar. 2010.

[30] J. R. Lane and A. Nakao, “Best-Effort Network Layer Packet Reordering in Support of Multipath Overlay Packet Dispersion,” Global Telecom. Conf, 2008 (GLOBECOM’08), pp. 1-6, Dec 2008.

[31] M. Laor and L. Gendel, “The Effect Of Packet Reordering in a Backbone Link on Application Throughput,” IEEE Network, pp. 28 – 36, Sep/Oct 2002.

[32] K. -C Leung, V. O. K. Li and D. Yang, “An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges,” IEEE Trans. On Parallel and Distributed Systems, vol. 18(4), pp. 552 – 535, Apr. 2007.

[33] B. Lin and I. Keslassy, “The Concurrent Matching Switch Architecture,” IEEE/ACM Trans. On Networking, pp. 1330-1343, Aug. 2010.

[34] C. Ma and K. -C. Leung, “Improving TCP Reordering Robustness in Multipath Networks,” Local Computer Networks, 2004, pp. 409-410, Nov. 2004.

[35] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, J. Perser, “Packet Reordering Metrics,” IETF RFC 4737

[36] A. Nehme, W. Phillips and W. Robertson, “The Effect of Re-Ordering and Dropping Packets on TCP over A Slow Wireless Link,” Canadian Conf. On. Elect. and Comp. Engr, 2003 (IEEE CCECE’03), vol. 3, pp. 1555-1558, May 2003.

[37] A. Papadogiannakis, D. Antoniades, M. Polychronakis and E. P. Markatos, “Improving the Performance of Passive Network Monitoring Applications using Locality Buffering,” Modeling, Analysis and Simulation of Comp. and Telecom. Systems, 2007 (MASCOTS’07), pp. 151-157, Oct. 2007.

[38] V. Paxson, “Measurements and Analysis of End-to-End Internet Dynamics,” Ph.D. Dissertation, University of California, Berkeley, Apr. 1997.

[39] N. M. Piratla, “A Theoretical foundation, Metrics and Modeling of Packet Reordering and Methodology of Delay Modeling using inter-packet gaps,” PhD thesis.

[40] N. M. Piratla and A. P. Jayasumana, “Reordering of Packets due to Multipath Forwarding - An Analysis,” Communications, 2006 (ICC’06), pp. 829-834, June 2006.

[41] N. M. Piratla and A. P. Jayasumana, "Metrics for packet reordering - A comparative analysis," Int. Journal of Communication Systems, vol 21(1), pp. 99-113, Jan 2007.

[42] N. M. Piratla, A. P. Jayasumana and T. Banka, “On Reorder Density and its Application to Characterization of Packet Reordering,” Local Computer Networks (LCN 2005), pp. 156 – 165, 17-17 Nov. 2005.

[43] N. M. Piratla, A. P. Jayasumana, A. A. Bare and T. Banka, “Reorder buffer-occupancy density and its application for measurement and evaluation of packet reordering,” Computer Communications, vol. 30(9), pp. 1980-1993, June 2007.

[44] S. Prabhavat, H. Nishiyama, N. Ansari and N. Kato, “On The Performance Analysis Of Traffic Splitting On Load Imbalancing and Packet Reordering Of Bursty Traffic,” Network Infrastructure and Digital Content, 2009 (IC-NIDC’09), pp. 236-240, Nov. 2009.

[45] M. L. Sanni, G. A. Aderounmu, S. A. Bello, A. O. Ajayi, “An Improved Packet Resequencing Model for Selective Repeat Request Protocol In Data Communication Networks,” Computer and Comm., 2007 (ISCC’07), pp. 375-280, July 2007.

[46] T. Shu, M. Liu, Z. Li and K. Zheng, “Network-layer Soft Vertical Handoff Schemes without Packet Reordering,” Local Computer Networks, 2009 (LCN’09), pp. 285-288, Oct. 2009.

[47] S. Spirou, “Packet Reordering Effects on the Subjective Quality of Broadband Digital Television,” Consumer Electronics, 2006. ISCE '06, 2006.

[48] W. Sun, T. Wen and Q. Guo, “A Novel Protocol for Mobile-induced Packet Reordering in Mobile Ad Hoc Networks,” Information Science and Engr., 2008 (ISISE’08), pp. 626-631, Dec. 2008.

[49] T. Taleb, D. Mashima, K. Hashimoto, N. Kato and Y. Nemoto, “On How To Mitigate the Packet Reordering Issue in the Explicit Load Balancing Scheme,” Global Information Infrastructure Symp., 2007 (GIIS’07), pp. 14-19, July 2007.

[50] Y. Tanigawa, J. Kim and H. Tode, “Retransmission Method with Network Coding based on Reordering Delay in Wireless LAN,” Industrial Informatics (INDIN), 2010, pp. 1011-1015, July 2010.

[51] S. P. Tinta, A. E. Mohr, J. L. Wong, “Characterizing End-to-End Packet Reordering with UDP Traffic,” Computers and Comm., 2009 (ISCC’09), pp. 321-324, July 2009.

[52] S. Traboulsi, M. Meitinger, R. Ohlendorf and A. Herkersdorf, “An Efficient Hardware Architecture for Packet Re-sequencing in Network Processors MPSoCs,” Digital System Design, Architectures, Methods and Tools, 2009 (DSD’09), pp. 11-18, Aug. 2009.

[53] G. Wang, J. Wang and L. Liu, “The Solution of Packet Reordering in LEO Satellite Networks,” Communication Software and Networks, 2009 (ICCSN’09), pp. 438-442, Feb. 2009.

[54] G. Wang, J. Wang, L. Liu and G. Chen, “Packet Reordering Analysis for LEO Satellite Networks,” Proc. Int. Conf. on Embedded Software and Systems (ICESS2008), pp. 308 – 313, 29-31 July 2008.

[55] B. Wu, Y. Xu, B. Liu, H. Lu and B. Liu, “A Practical Packet Reordering Mechanism with Flow Granularity for Parallelism Exploiting in Network Processors,” Proc. of the 19th IEEE Int. Parallel and Distributed Proc. Sym. (IPDPS'05), pp. 133.1, 2005.

[56] B. Wu, Y. Xu, B. Liu, H. Lu and X. Wang, “An Efficient Scheduling Mechanism with Flow-Based Packet Reordering in a High-Speed Network Processor,” Proc. IEEE Workshop High Performance Switching and Routing (HPSR), 2005

[57] Y. Xia and D. Tse, “Analysis on packet re-sequencing for reliable network protocols, “Performance Evaluation, vol. 61(4), pp. 299 – 328, Aug 2005.

[58] D. Yang, K. -C. Leung and V. O. K. Li, “Simulation-Based Comparisons of Solutions for TCP Packet Reordering in Wireless Networks,” Wireless Comm. and Networks, 2007 (WCNC’07), pp. 3238-3243, Mar. 2007.

[59] X. Yang, Z. Qing-li, F. Fang-fa, Y. Ming-yan and L. Cheng, “NISAR: An AXI Compliant On-chip NI Architecture Offering Transaction Reordering Processing,” ASIC, 2007 (ASICON’07), pp. 890-893, Oct. 2007.

[60] B. Ye, A. P. Jayasumana and N. M. Piratla, “On Monitoring of End-to-End Packet Reordering over the Internet,” Networking and Services, 2006 (ICNS '06), pp. 3 – 3, 16-18 July 2006.

[61] J. -H. Yun, M. Lee, S. Choi and H. Cha, “Comparison of Handover Schemes for 3GPP Long Term Evolution and 3GPP2 Ultra Mobile Broadband,” Personal Indoor and Mobile Radio Comm., 2008 (PIMRC’08), pp. 1-5, Sep. 2008.

[62] K. Zheng, X. Jiao, M. Liu and Z. Li, “An Analysis of Resequencing delay of Reliable Transmission Protocols over Multipath,” Communications, 2010 (ICC’10), pp. 1-5, May 2010.

[63] Computer Networking Research Laboratory, http://www.cnrl.colostate.edu/.

[64] Internet World Stats, http://www.internetworldstats.com/.

APPENDIX

<u>Map.c</u>

```
#include <stdio.h>

#define SIZE 1005

void get_latest_index(int byte_num[],int *end_num,int *end_index);
int chk_for_all(int byte_num[]);
int get_new_index(int byte_num[], int x);

int main(int argc, const char* argv[] ) {

        FILE *in;
        in = fopen(argv[1],"r");

        if(NULL == in)
        {
                printf("\n Error");
        }
        else
        {
                int byte_num[SIZE];
                int astart[SIZE],aend[SIZE],asize[SIZE];
                int start,end,size;
                int it = 1;
                int i;
                int quit = 1;
                int prev_quit = 0;

                for(i = 0;i<SIZE;i++)
                        byte_num[i] = -1;

                //printf("\n Done with init");
                i = 0;
                while(EOF != fscanf(in,"%d %d %d",&start,&end,&size)){
                        astart[i] = start;
                        aend[i] = end;
                        asize[i] = size;

                        i++;
                }

                int MAX = SIZE;

                //printf("\n Done with reading");
                while(quit && (prev_quit < MAX)){
                        int end_num = -1,end_index = -1;
```

```
		start = astart[it-1];
		end = aend[it-1];

		if(it == 1){
			byte_num[it] = start;
			byte_num[it+1] = end;
		}
		else
			get_latest_index(byte_num,&end_num,&end_index);


		if(start == end_num){
			byte_num[end_index+1] = end;
			//printf("\n bye_num[%d] = %d",end_index+1,end);
			//it = 1;
		}

		if(it == MAX) {
			prev_quit++;
			quit = chk_for_all(byte_num);
			///printf("\n Recursing ..");
			it = 1;
		}
		else
			it++;

	}

	//printf("\n Done with reading");

	/*
	for(i=0;i<1000;i++)
		printf("\n [%d]\t%d",i+1,byte_num[i]);
	*/

	fclose(in);

	int num_of_late = 0;
	int num_of_early = 0;
	int was_it_only_ir = 1;
	int late_by_one=0;

	if(prev_quit < MAX) {
	in = fopen(argv[1],"r");

		printf("\n");
		int ri = 1;
		while(EOF != fscanf(in,"%d %d %d",&start,&end,&size)){
			int nstart = get_new_index(byte_num,start);
			int nend = get_new_index(byte_num,end);

			if((ri-nstart) > 0)
				num_of_late++;
			else if((ri-nstart) < 0)
				num_of_early++;
```

```
                                        if((ri-nstart)< -1)
                                                was_it_only_ir = 0;

                                        if((ri-nstart) == 1)
                                                late_by_one++;

                                        if((ri-nstart) != 0)
                                                printf("\n%d\t%d\t\t%d",nstart,ri,(ri-nstart));
                                        else
                                                printf("\n%d\t%d\t%d",nstart,ri,(ri-nstart));
                                        ri++;
                                }
                                printf("\n<%d:%d",was_it_only_ir,num_of_late);
                                printf(":%d:%d>",num_of_early,late_by_one);
                        }
                        else
                                printf("\n Packets lost ...");
                        fclose(in);
                }
}


int get_new_index(int byte_num[], int x) {
        for(int i = 1;i<=SIZE;i++)
                if(x == byte_num[i])
                        return i;

        return -1;
}
void get_latest_index(int byte_num[],int *end_num,int *end_index) {
        int i;

        for(i=1;i<=SIZE;i++){
                if(byte_num[i] == -1) {

                        *end_num = byte_num[i-1];
                        *end_index = i-1;
                        break;
                }
        }

        //printf ("\n end_num %d\tend_index %d",*end_num,*end_index);
}

int chk_for_all(int byte_num[]){
        int i;

        for(i=1;i<=999;i++)
                if(byte_num[i] == -1)
                        return 1;

        //printf("\n Returns 0");
        return 0;
}
```

find_displacements.pl
```
#!/usr/local/bin/perl -w


$fol = "./indexed/";
opendir(IMD, $fol) || die("Cannot open directory");
@thefiles = readdir(IMD);
closedir(IMD);
foreach $thing (@thefiles) {
    unless ( ($thing eq ".") || ($thing eq ".."))
     {
        $redir = $fol.$thing;
        @filedata = `cat $redir`;

                $num_of_lines = 0;
        foreach $line (@filedata) {
                        #print "$line";
                        if($num_of_lines > 999)
                        {
                                #print "here";

                                if($line =~ m/\<(\d+):(\d+):(\d+):(\d+)\>/) {
                                        $ir = $1;
                                        $late = $2;
                                        $early = $3;
                                        $one_late = $4;

                                        #print "hmmm...";
                                        if($ir == 1 && $late ne 0) {
                                                print "\n Filename : $thing ------ \t";
                                                print "\tLate : $late\tEarly : $early";
                                        }
                                        else {
                                                #print "Doing ..";
                                        }

                                        if(($late+$early) > 100 && $ir == 1) {
                                                $do = `cp ./indexed/$thing ./copied`;
                                                print $do;
                                        }


                                }
                        }
                        $num_of_lines++;
                }

     }
}
```
Allt.pl
```
#!/usr/local/bin/perl -w

$fol1 = "./selected/";
$fol2 = "./toindex/";
$fol3 = "./indexed/";
opendir(IMD, $fol1) || die("Cannot open directory");
```

```
@thefiles = readdir(IMD);
closedir(IMD);
foreach $thing (@thefiles) {
      unless ( ($thing eq ".") || ($thing eq ".."))
       {
              $redir = $fol1.$thing;
              $toindex = $fol2.$thing;
              $indexed = $fol3.$thing;

              @fileoutput = `t.pl $redir > $toindex`;
              @fileoutput1 = `./doit $toindex > $indexed`;
              print "\tD";
       }
}
print "\nDONE !!";
```

<u>extract_data.pl</u>

```
#!/usr/local/bin/perl -w

$fol = "./Japan_txt/";
opendir(IMD, $fol) || die("Cannot open directory");
@thefiles = readdir(IMD);
closedir(IMD);
foreach $thing (@thefiles) {
      unless ( ($thing eq ".") || ($thing eq ".."))
       {
                            print "$thing";
              $redir = $fol.$thing;
              print "$redir";
              @filedata = `cat $redir`;
              foreach $line(@filedata) {

                            #print "$filedata[$line]";

                                          @myLines = `cat $line`;

                                          $prev_start = 0;
                                          $prev_end = 0;
                                          $displacement = 0;
                                          $number_lines = 0;
                                          $start_count = 0;
                                          $reorder_start = 0;

                                          my %reorder_hash = ();


                                          foreach (@myLines) {
                                                        $line = $_;
                                                        #print $line;

                                                        $number_lines++;

                                                        if($line =~ m/(S|.|P) (\d+):(\d+)/) {
                                                                      $start = $2;
                                                                      $end = $3;
                                                                      #print "\n$2\t\t$3";
```

```
                                        if($prev_end ne $start && $prev_end ne $prev_start) {
                                                print "\nReorder $prev_end : $start";
                                                $start_count = 1;
                                                $reorder_hash{$prev_end} = 1;
                                        }

                                        if($start_count eq 1) {
                                                #print "happening";
                                                $displacement++;
                                        }

                                        if(defined $reorder_hash{$start}) {
                                                delete $reorder_hash{$start};
                                                print "\t $start late by $displacement";
                                                $start_count = 0;
                                                $displacement = 0;
                                        }

                                        $prev_start = $start;
                                        $prev_end = $end;
                                }
                        }


                }

        }
}
print "\nDONE !!";
```

<u>Makefile</u>

```
CC= cc

INC=-Iinclude
LIB=-Llib
CFLAGS=

FLAGS= $(INC) $(LIB) $(CFLAGS)

OBJS = debug.o reorder.o resequence.o seqgen.o comparator.o util.o main.o

all: main

main: $(OBJS)
	$(CC) -o m $(OBJS) $(FLAGS)

main.o: main.c
	$(CC) -c main.c $(FLAGS)

util.o: util.c
	$(CC) -c util.c $(FLAGS)

comparator.o : comparator.c
	$(CC) -c comparator.c $(FLAGS)
```

```
reorder.o : reorder.c
		$(CC) -c reorder.c $(FLAGS)

resequence.o : resequence.c
		$(CC) -c resequence.c $(FLAGS)

seqgen.o: seqgen.c
		$(CC) -c seqgen.c $(FLAGS)

debug.o: debug.c
		$(CC) -c debug.c $(FLAGS)

clean :
		rm m
		rm *.o
```

Comparator.c

```
/*
Filename:
comparator.c

Funtionality:
This file contains methods to compute the displacement, predict RD, etc

Author:
Raghunandan Mandyam Narasiodeyar
Computer Network Research Lab
Dept. of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO - 80523
*/

#include <main.h>

/*
Method to compute the displacement
Input arguments:
actual - Pointer to the first NODE element
num_of_packets - Number of packets in the sequence
*/
NODE* compute_disp(NODE *actual,int num_of_packets)
{
		int i=1;
		NODE *temp,*current,*first;
		NODE *tactual = actual;

		temp = (NODE *)malloc(sizeof(NODE));
		first = current = temp;

		while(i != num_of_packets)
		{
				current->info = i-(tactual->info);
				temp = (NODE *)malloc(sizeof(NODE));
				current->ptr = temp;
				current = temp;
```

```
			i++;
			tactual = tactual->ptr;
		}
		current->info = i-(tactual->info);
		current->ptr = NULL;

		return(first);
}

/*
A sub-method used to compute RD
Input arguments:
first - Pointer to the first NODE element
x - element to be inserted
*/

RD_NODE* increment_rd(int x,RD_NODE *first)
{
		RD_NODE *temp;
		RD_NODE *prev;

		int status = FAIL;

		temp = first;

		if(temp == NULL && x != 0)
		{
			RD_NODE* n;
			n = (RD_NODE*)malloc(sizeof(RD_NODE));
			n->d = 0;
			n->disp = 0;
			n->ptr = NULL;

			temp = first = n;
		}

		if(temp == NULL)
		{
			RD_NODE* n;
			n = (RD_NODE*)malloc(sizeof(RD_NODE));
			n->d = x;
			n->disp = 1;
			n->ptr = NULL;

			return n;
		}

		while(temp != NULL)
		{
			if(temp->d == x)
			{
				(temp->disp)++;
				status = SUCCESS;
				break;
			}
			prev = temp;
			temp = temp->ptr;
		}

		if(status != SUCCESS)
		{
			RD_NODE* n;
			n = (RD_NODE*)malloc(sizeof(RD_NODE));
			n->d = x;
			n->disp = 1;
			n->ptr = NULL;
			prev->ptr = n;
		}

		return first;
}

/*
Method used to compute RD
Input arguments:
disp - Pointer to the displacement NODE element
num_of_packets - Number of packets in the sequence
*/

RD_NODE* compute_rd(NODE *disp,int num_of_packets)
{
		RD_NODE *first=NULL;
		RD_NODE *trd;
		NODE *temp;

		temp = disp;

		while(temp != NULL)
		{
			printf("\n iterating through main link list ..");
			first = increment_rd(temp->info,first);
			temp = temp->ptr;
		}

		return first;
}

/*
Method used to predict RD in an embedded reordering pattern formed by
late packet event embedded within another late packet event
Input arguments:
rdfirst - Pointer to the first element of the RD node
num_of_packets - Number of packets in the sequence
num_of_buffers - Number of resequencing buffers to be used
*/

RD_NODE* predict_erlel_rd(RD_NODE *rdfirst,int num_of_packets,int num_of_buffers)
{
		RD_NODE *temp;
		RD_NODE *newtemp;
		RD_NODE *newFirst;
```

```
signed int dx=-1,dy=-1;
temp = rdfirst;

while(temp != NULL)
{
        if(temp->d > 0 && dx == -1)
                dx = temp->d;
        if(temp->d > 0 && dx != -1)
                dy = temp->d;

        temp = temp->ptr;
}

temp = rdfirst;
newFirst = (RD_NODE *)malloc(sizeof(RD_NODE));
newtemp = newFirst;

//printf("\n DX : %d\tDY : %d",dx,dy);

while(temp != NULL)
{
        int minimum = min(dx,dy);
        //printf("\n Minimum : %d Num_of_buffers : %d",minimum,num_of_buffers);
        if(num_of_buffers < minimum)
        {
                //printf("\n num_of_buffers < minimum");
                if(temp->d == 0)
                {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp + num_of_buffers;
                }
                else if(temp->d == -1)
                {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp;
                }
                else if(temp->d == -2)
                {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp - num_of_buffers;
                }
                else if(temp->d > 1)
                {
                        newtemp->d = (temp->d) - num_of_buffers;
                        newtemp->disp = temp->disp;
                }
        }
        else if(num_of_buffers == minimum)
        {
                //printf("\n num_of_buffers == minimum");
                if(temp->d == 0)
                {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp + num_of_buffers + 1;
                }
                else if(temp->d == -1)
```

```
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp;
		}
		else if(temp->d == -2)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp - num_of_buffers;
		}
		else if(temp->d > num_of_buffers)
		{
			newtemp->d = (temp->d) - num_of_buffers;
			newtemp->disp = temp->disp;
		}
	}
	else if(num_of_buffers == minimum + 1)
	{
		//printf("\n num_of_buffers + 1 == minimum");
		if(temp->d == 0)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp + num_of_buffers;
		}
		else if(temp->d == -1)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp + 1;
		}
		else if(temp->d > num_of_buffers)
		{
			newtemp->d = (temp->d) - num_of_buffers;
			newtemp->disp = temp->disp;
		}
	}
	else if(num_of_buffers > (minimum + 1) && num_of_buffers < max(dx,dy))
	{
		//printf("\n num_of_buffers > minimum && num_of_buffers < max(dx,dy)");
		if(temp->d == 0)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp + num_of_buffers;
		}
		else if(temp->d == -1)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp - num_of_buffers + minimum + 2;
		}
		else if(temp->d > minimum)
		{
			newtemp->d = (temp->d) - num_of_buffers;
			newtemp->disp = temp->disp;
		}
	}
	else
	{
		newtemp->d = 0;
```

```
				newtemp->disp = num_of_packets;
				newtemp->ptr = NULL;
				break;
			}

			if(temp->ptr != NULL && newtemp->disp != 0/*&& newtemp->disp != num_of_packets*/)
			{
				RD_NODE *again = (RD_NODE *)malloc(sizeof(RD_NODE));
				newtemp->ptr = again;
				newtemp = again;
			}
			else
				newtemp->ptr = NULL;

			temp = temp->ptr;
		}

		return (newFirst);
}

/*
Method used to predict RD in an overlapped reordering pattern formed by
late packet event overlapping another late packet event
Input arguments:
rdfirst - Pointer to the first element of the RD node
num_of_packets - Number of packets in the sequence
num_of_buffers - Number of resequencing buffers to be used
*/

RD_NODE* predict_orlol_rd(RD_NODE *rdfirst,int num_of_packets,int num_of_buffers)
{
		RD_NODE *temp;
		RD_NODE *newtemp;
		RD_NODE *newFirst;

		signed int dx=-1,dy=-1;
		temp = rdfirst;
		int vxy = 0;

		while(temp != NULL)
		{
			if(temp->d > 0 && dx == -1)
				dx = temp->d;
			if(temp->d > 0 && dx != -1)
				dy = temp->d;

			if(temp->d == -2)
				vxy = temp->disp;

			temp = temp->ptr;
		}

		temp = rdfirst;
		newFirst = (RD_NODE *)malloc(sizeof(RD_NODE));
		newtemp = newFirst;
		int minimum = min(dx,dy);
```

```
printf("\n VXY : %d",vxy);

while(temp != NULL)
{
	//printf("\n Minimum : %d Num_of_buffers : %d",minimum,num_of_buffers);
	if(num_of_buffers <= vxy)
	{
		//printf("\n num_of_buffers < minimum");
		if(temp->d == 0)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp + num_of_buffers;
		}
		else if(temp->d == -1)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp;
		}
		else if(temp->d == -2)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp - num_of_buffers;
		}
		else if(temp->d > 1)
		{
			newtemp->d = (temp->d) - num_of_buffers;
			newtemp->disp = temp->disp;
		}
	}
	else if(num_of_buffers > vxy && num_of_buffers < minimum)
	{
		int j = vxy;
		int k = (num_of_buffers - vxy);

		//printf("\n num_of_buffers < minimum");
		if(temp->d == 0)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp + j + 2*k;
		}
		else if(temp->d == -1)
		{
			newtemp->d = temp->d;
			newtemp->disp = temp->disp - 2*k;
		}
		else if(temp->d > num_of_buffers)
		{
			newtemp->d = (temp->d) - num_of_buffers;
			newtemp->disp = temp->disp;
		}
	}
	else if(num_of_buffers == minimum)
	{
		int j = vxy;
		int k = (num_of_buffers - vxy);
```

```
            //printf("\n num_of_buffers < minimum");
            if(temp->d == 0)
            {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp + j + 2*k + 1;
            }
            else if(temp->d == -1)
            {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp - 2*k;
            }
            else if(temp->d > num_of_buffers)
            {
                        newtemp->d = (temp->d) - num_of_buffers;
                        newtemp->disp = temp->disp;
            }
}
else if(num_of_buffers > (minimum) && num_of_buffers < max(dx,dy))
{
            int j = vxy;
            int k = minimum - vxy;
            int l = num_of_buffers - minimum;

            if(temp->d == 0)
            {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp + j + 2*k + l + 1;
            }
            else if(temp->d == -1)
            {
                        newtemp->d = temp->d;
                        newtemp->disp = temp->disp - (2*k) - l;
            }
            else if(temp->d > num_of_buffers)
            {
                        newtemp->d = (temp->d) - num_of_buffers;
                        newtemp->disp = temp->disp;
            }
}
else
{
            newtemp->d = 0;
            newtemp->disp = num_of_packets;
            newtemp->ptr = NULL;
            break;
}

if(temp->ptr != NULL && newtemp->disp != 0/*&& newtemp->disp != num_of_packets*/)
{
            RD_NODE *again = (RD_NODE *)malloc(sizeof(RD_NODE));
            newtemp->ptr = again;
            newtemp = again;
}
else
            newtemp->ptr = NULL;
```

```
				temp = temp->ptr;
		}

		return (newFirst);
}

/*
Method used to predict RD in an embedded reordering pattern formed by
late packet event embedded within a early packet event
Input arguments:
rdfirst - Pointer to the first element of the RD node
num_of_packets - Number of packets in the sequence
num_of_buffers - Number of resequencing buffers to be used
*/

RD_NODE* predict_erlee_rd(RD_NODE *rdfirst,int num_of_packets,int num_of_buffers)
{
		RD_NODE *temp;
		RD_NODE *newtemp;
		RD_NODE *newFirst;

		signed int dearly, dlate;
		temp = rdfirst;
		int vxy = 0;

		while(temp != NULL)
		{
				if(temp->d < -1)
						dearly = temp->d;

				if(temp->d > 0)
						dlate = temp->d;

				temp = temp->ptr;
		}

		temp = rdfirst;
		newFirst = (RD_NODE *)malloc(sizeof(RD_NODE));
		newtemp = newFirst;

		printf("\n de : %d\tdl : %d",dearly,dlate);

		while(temp != NULL)
		{
				if(num_of_buffers == 1)
				{
						int j = dearly + 1;
						int k = num_of_buffers;
						//printf("\n num_of_buffers < minimum");
						if(temp->d == 0)
						{
								newtemp->d = temp->d;
								newtemp->disp = temp->disp + j + k;
						}
						else if(temp->d == -1)
```

```
                        {
                                newtemp->d = temp->d;
                                newtemp->disp = temp->disp - j;
                        }
                        else if(temp->d > 1)
                        {
                                newtemp->d = (temp->d) - num_of_buffers;
                                newtemp->disp = temp->disp;
                        }
                }
                else if(num_of_buffers > 1 && num_of_buffers < dlate)
                {
                        int j = dearly + 1;
                        int k = 1;
                        int m = num_of_buffers-1;

                        //printf("\n num_of_buffers < minimum");
                        if(temp->d == 0)
                        {
                                newtemp->d = temp->d;
                                newtemp->disp = temp->disp + j + k + m;
                        }
                        else if(temp->d == -1)
                        {
                                newtemp->d = temp->d;
                                newtemp->disp = temp->disp - j - m;
                        }
                        else if(temp->d > 1)
                        {
                                newtemp->d = (temp->d) - num_of_buffers;
                                newtemp->disp = temp->disp;
                        }
                }
                else
                {
                        newtemp->d = 0;
                        newtemp->disp = num_of_packets;
                        newtemp->ptr = NULL;
                        break;
                }

                if(temp->ptr != NULL && newtemp->disp != 0/*&& newtemp->disp != num_of_packets*/)
                {
                        RD_NODE *again = (RD_NODE *)malloc(sizeof(RD_NODE));
                        newtemp->ptr = again;
                        newtemp = again;
                }
                else
                        newtemp->ptr = NULL;

                temp = temp->ptr;
        }

        return (newFirst);
}
```

```
/*
Method used to predict RD in an independent reordering pattern
Input arguments:
rdfirst - Pointer to the first element of the RD node
num_of_packets - Number of packets in the sequence
num_of_buffers - Number of resequencing buffers to be used
*/

RD_NODE* predict_rd(RD_NODE *rdfirst,int num_of_packets,int num_of_buffers)
{
        RD_NODE *temp;
        RD_NODE *newRD=copy_rd_node(rdfirst);
        RD_NODE *newtemp = newRD;
        int max_req=0;
        temp = rdfirst;

        while(temp != NULL)
        {
                if(temp->d > max_req)
                        max_req = temp->d;

                temp = temp->ptr;
        }

        temp = rdfirst;

        if(num_of_buffers < max_req)
        {
                while(temp != NULL)
                {
                        if(0 == temp->d)
                        {
                                newtemp->disp = (temp->disp)+num_of_buffers;
                        }
                        else if(-1 == temp->d)
                        {
                                newtemp->disp = (temp->disp)-num_of_buffers;
                        }
                        else
                        {
                                newtemp->d = (temp->d)-num_of_buffers;
                        }

                        temp = temp->ptr;
                        newtemp = newtemp->ptr;
                }
        }
        else
        {
                while(temp != NULL)
                {
                        if(0 == temp->d)
                        {
                                newtemp->disp = num_of_packets;
                                newtemp->ptr = NULL;
                        }
                        temp = temp->ptr;

                }
        }
        return (newRD);
}

int get_num_mrs(NODE *disp)
{
        NODE *temp;
        int mrs = 0;
        temp = disp;

        while(temp != NULL)
        {
                if(temp->info > 0)
                        mrs++;

                temp = temp->ptr;
        }

        return mrs;
}
```

Debug.c

```
/*
Filename:
debug.c

Funtionality:
This file contains methods required to debug or print the values of RD

Author:
Raghunandan Mandyam Narasiodeyar
Computer Network Research Lab
Dept. of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO - 80523
*/

#include <main.h>

/*
Method used to display the contents of the list
Input:
first - Pointer to the first element of the NODE
*/
void display_list(NODE *first)
{
        NODE *temp;
        temp = first;

        while(temp != NULL)
```

```
          {
                    printf("%4d ",temp->info);
                    temp = temp->ptr;
          }
          printf("\n");
}

/*
Method used to display the contents of the RD list
Input:
rdfirst - Pointer to the first element of the NODE
num_of_packets - Number of packets in the sequence
*/

void display_rd_node(RD_NODE *rdfirst,int num_of_packets)
{
          RD_NODE *temp;
          temp = rdfirst;

          while(temp != NULL)
          {
                    printf("\n RD[%d]\t%d/%d",temp->d,temp->disp,num_of_packets);
                    temp = temp->ptr;
          }

}


/*
Method used to display the contents of the RD list
Input:
first - Pointer to the first element of the NODE
num_of_packets - Number of packets in the sequence
*/
void display_rd(RD_NODE *first,int num_of_packets)
{
          RD_NODE *trd = first;

          printf("\n|---------------------------|");
          printf("\n|\tReorder Density");
          printf("\n|---------------------------|");

          while(trd != NULL)
          {
                    printf("\n\tRD[%d]\t%d/%d ",trd->d,trd->disp,num_of_packets);
                    trd = trd->ptr;
          }
          printf("\n|---------------------------|");
}
```

Reorder.c

```
/*
Filename:
reorder.c
```

```
Funtionality:
This file contains the method(s) to reorder based on a particular set of input

Author:
Raghunandan Mandyam Narasiodeyar
Computer Network Research Lab
Dept. of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO - 80523
*/

#include <main.h>


NODE* reorder(NODE *first,int info,int displacement)
{
        NODE *temp;
        NODE *prev;
        NODE *store;

        int d = 0;
        int swap = 0;

        prev = temp = first;

        if(displacement > 0)
        {
                while(temp != NULL)
                {
                        if(temp->info == info)
                        {
                                store = temp;

                                if(prev != temp)
                                {
                                        prev->ptr = temp->ptr;
                                        temp = temp->ptr;
                                }
                                else
                                {
                                        first = temp->ptr;
                                        temp = temp->ptr;
                                }

                                swap = 1;
                        }

                        if(swap)
                        {
                                d++;
                                if(d == displacement+1)
                                {
                                        prev->ptr = store;
                                        store->ptr = temp;
                                        break;
```

```
            }
            if(temp->ptr == NULL)
                swap = 0;
        }

        prev = temp;
        temp = temp->ptr;
    }

    if(!swap)
    {
        prev->ptr = store;
        store->ptr=NULL;
    }
}
else if(displacement < 0)
{
    int ptr = 0;

    while(temp != NULL)
    {
        ptr++;
        if(temp->info == info)
        {
            store = temp;

            if(prev != temp)
            {
                prev->ptr = temp->ptr;
                temp = temp->ptr;
            }
            else
            {
                first = temp->ptr;
                temp = temp->ptr;
            }
            break;
        }
        prev = temp;
        temp = temp->ptr;
    }

    prev = temp = first;
    int ptr1 = 0;

    while(temp != NULL)
    {
        //printf("\n ptr1 %d\tptr %d\tdis %d",ptr1,ptr,displacement);
        // since displacement is -ve .. it is added here
        if((ptr + displacement) == ptr1)
        {
            if(prev != temp)
            {
                prev->ptr = store;
                store->ptr = temp;
            }
```

```
                                        else
                                        {
                                                store->ptr = first;
                                                first = store;
                                        }
                                        break;
                                }
                                ptr1++;
                                prev = temp;
                                temp = temp->ptr;
                        }
                }

                return first;
        }
```

Resequence.c

```
/*
Filename:
resequence.c

Funtionality:
This file contains the method(s) to resequence packets based on
the lowest sequence number first algorithm

Author:
Raghunandan Mandyam Narasiodeyar
Computer Network Research Lab
Dept. of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO - 80523
*/

#include <main.h>

#define MAX_NUM 99999

int next_expected(NODE *vfirst,int x)
{
        NODE *temp;
        int max = -1;
        temp = vfirst;

        while(temp->info != x && temp != NULL)
        {
                //printf("%d\t",temp->info);
                if(temp->info > max)
                        max = temp->info;

                temp = temp->ptr;
        }

        return max+1;
}
```

```
void mark_flag(int x,NODE *first)
{
        NODE *temp;
        temp = first;

        while(temp != NULL)
        {
                if(temp->info == x)
                {
                        temp->parsed = 1;
                        break;
                }
                temp = temp->ptr;
        }
}

int get_expected(NODE *first)
{
        NODE *temp = first;

        while(temp != NULL)
        {
                if(temp->parsed == -1)
                        return temp->info;

                temp = temp->ptr;
        }

        return -1;
}

NODE* resequence(NODE* first,int num_of_packets)
{
        int expected = 1;
        int store = MAX_NUM;
        NODE *temp = first;
        NODE *vtemp = first;

        NODE *bfirst = create_orig_seq(num_of_packets);

        while(temp != NULL)
        {
                //printf("\n expected : %d\ttemp->info : %d\tstore : %d",expected,temp->info,store);
                if(expected == temp->info)
                {
                        vtemp->info = temp->info;
                        mark_flag(vtemp->info,bfirst);
                        vtemp = vtemp->ptr;

                        if(store != MAX_NUM && store < temp->info)
                        {
                                vtemp->info = store;
                                mark_flag(vtemp->info,bfirst);
                                store = MAX_NUM;
                        vtemp = vtemp->ptr;
                        expected = get_expected(bfirst);

                    }
                    else
                        expected++;
                }
                else
                {
                    if(store < temp->info)
                    {
                        vtemp->info = store;
                        mark_flag(vtemp->info,bfirst);
                        vtemp = vtemp->ptr;
                        store = MAX_NUM;

                    }
                    if(store > temp->info && store != MAX_NUM)
                    {
                        vtemp->info = temp->info;
                        vtemp = vtemp->ptr;
                    }
                    else
                        store = temp->info;

                }

                temp = temp->ptr;
        }

        return first;
}


/*
NODE* resequence(NODE* first)
{
        NODE *temp,*prev;
        NODE *vtemp,*vfirst;
        int store=EMPTY;
        int expected=1;

        int alreadyParsed[MAX_PACKETS];

        for(int i = 1;i<=MAX_PACKETS;i++)
                alreadyParsed[i] = FALSE;

        vfirst = vtemp = temp = first;

        while(temp != NULL)
        {
                if(expected != temp->info)
                {
                    if(store == EMPTY && expected < temp->info)
                    {
                        store = temp->info;
                        printf("\n Stored %d",store);
                }
                else if(expected > temp->info)
                {
                        vtemp->info = temp->info;
                        vtemp = vtemp->ptr;
                }
                else
                {
                        if(store < temp->info)
                        {
                                vtemp->info = store;
                                vtemp = vtemp->ptr;
                                store = temp->info;
                                printf("\n Stored %d",store);
                        }

                }
        }
        else
        {
                int x;
                vtemp->info = temp->info;
                x = vtemp->info;
                vtemp = vtemp->ptr;
                expected = next_expected(vfirst,x);
                printf("\n\t\t\t\t\t\t EXPECTED = %d",expected);
                if(expected == store)
                {
                        vtemp->info = store;
                        vtemp = vtemp->ptr;
                        store = EMPTY;
                        expected = next_expected(vfirst,x);
                        //if(EMPTY == store)
                                expected+=2;
                        printf("\n\t\t\t\t\t\t EXPECTED = %d",expected);
                }

        }

        if(temp->ptr == NULL && -1 != store)
                vtemp->info = store;


        temp = temp->ptr;
}

return vfirst;

}

*/
```

Seqgen.c

```
/*
```

```
Filename:
seqgen.c

Funtionality:
This file contains the method(s) to create a sequence of packets
based on the number of packets required

Author:
Raghunandan Mandyam Narasiodeyar
Computer Network Research Lab
Dept. of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO - 80523
*/

#include <main.h>


NODE* create_orig_seq(int num_of_packets)
{
	int i=1;
	NODE *temp,*current,*first;

	temp = (NODE *)malloc(sizeof(NODE));
	first = current = temp;

	while(i != num_of_packets)
	{
		current->info = i;
		current->parsed = -1;
		temp = (NODE *)malloc(sizeof(NODE));
		current->ptr = temp;
		current = temp;
		i++;
	}
	current->info = i;
	current->ptr = NULL;

	return(first);
}
```

<u>Util.c</u>

```
/*
Filename:
util.c

Funtionality:
This file contains the method(s) that act as utilities
for the entire verification and simulation process

Author:
Raghunandan Mandyam Narasiodeyar
Computer Network Research Lab
Dept. of Electrical and Computer Engineering
```

```
Colorado State University
Fort Collins, CO - 80523
*/

#include <main.h>

int max(int x,int y)
{
        return (x > y ? x : y);
}

int min(int x,int y)
{
        return (x < y ? x : y);
}

RD_NODE* copy_rd_node(RD_NODE *first)
{
        RD_NODE *temp;
        //RD_NODE *ntemp=NULL;
        RD_NODE *nrdfirst=NULL;
        RD_NODE *prev;

        temp = first;

        while(temp != NULL)
        {
                if(nrdfirst == NULL)
                {
                        RD_NODE *t;
                        t = (RD_NODE*)malloc(sizeof(RD_NODE));
                        t->d = temp->d;
                        t->disp = temp->disp;
                        t->ptr=NULL;
                        nrdfirst = t;
                        prev = t;
                }
                else
                {
                        RD_NODE *next;
                        next = (RD_NODE*)malloc(sizeof(RD_NODE));
                        next->d = temp->d;
                        next->disp = temp->disp;
                        next->ptr = NULL;
                        prev->ptr = next;
                        prev = next;
                }

                temp = temp->ptr;

        }
        return nrdfirst;
}

NODE* copy_node(NODE *first)
{
        NODE *temp;
        //RD_NODE *ntemp=NULL;
        NODE *nrdfirst=NULL;
        NODE *prev;

        temp = first;

        while(temp != NULL)
        {
                if(nrdfirst == NULL)
                {
                        NODE *t;
                        t = (NODE*)malloc(sizeof(NODE));
                        t->info = temp->info;
                        t->ptr=NULL;
                        nrdfirst = t;
                        prev = t;
                }
                else
                {
                        NODE *next;
                        next = (NODE*)malloc(sizeof(NODE));
                        next->info = temp->info;
                        next->ptr = NULL;
                        prev->ptr = next;
                        prev = next;
                }

                temp = temp->ptr;

        }
        return nrdfirst;
}
```